frontiers | Research Topics

Snakes on a spaceship—an overview of python in space physics

Edited by
Angeline G. Burrell, John Coxon, K.-Michael Aye, Leslie Lamarche and Sophie A. Murray

Published in
Frontiers in Astronomy and Space Sciences
Frontiers in Physics



FRONTIERS EBOOK COPYRIGHT STATEMENT

The copyright in the text of individual articles in this ebook is the property of their respective authors or their respective institutions or funders. The copyright in graphics and images within each article may be subject to copyright of other parties. In both cases this is subject to a license granted to Frontiers.

The compilation of articles constituting this ebook is the property of Frontiers.

Each article within this ebook, and the ebook itself, are published under the most recent version of the Creative Commons CC-BY licence. The version current at the date of publication of this ebook is CC-BY 4.0. If the CC-BY licence is updated, the licence granted by Frontiers is automatically updated to the new version.

When exercising any right under the CC-BY licence, Frontiers must be attributed as the original publisher of the article or ebook, as applicable.

Authors have the responsibility of ensuring that any graphics or other materials which are the property of others may be included in the CC-BY licence, but this should be checked before relying on the CC-BY licence to reproduce those materials. Any copyright notices relating to those materials must be complied with.

Copyright and source acknowledgement notices may not be removed and must be displayed in any copy, derivative work or partial copy which includes the elements in question.

All copyright, and all rights therein, are protected by national and international copyright laws. The above represents a summary only. For further information please read Frontiers' Conditions for Website Use and Copyright Statement, and the applicable CC-BY licence.

ISSN 1664-8714
ISBN 978-2-8325-2959-1
DOI 10.3389/978-2-8325-2959-1

About Frontiers

Frontiers is more than just an open access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers journal series

The Frontiers journal series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the *Frontiers journal series* operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews. Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the *Frontiers journals series*: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area.

Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers editorial office: frontiersin.org/about/contact

Snakes on a spaceship—an overview of python in space physics

Topic editors

Angeline G. Burrell — Naval Research Laboratory, United States
John Coxon — Northumbria University, United Kingdom
K.-Michael Aye — Freie Universität Berlin, Germany
Leslie Lamarche — SRI International, United States
Sophie A. Murray — Dublin Institute for Advanced Studies (DIAS), Ireland

Citation

Burrell, A. G., Coxon, J., Aye, K.-M., Lamarche, L., Murray, S. A., eds. (2023). *Snakes on a spaceship—an overview of python in space physics*. Lausanne: Frontiers Media SA. doi: 10.3389/978-2-8325-2959-1

Table of contents

07 PyThea: An open-source software package to perform 3D reconstruction of coronal mass ejections and shock waves
Athanasios Kouloumvakos, Laura Rodríguez-García, Jan Gieseler, Daniel J. Price, Angelos Vourlidas and Rami Vainio

22 AuroraX, PyAuroraX, and aurora-asi-lib: A user-friendly auroral all-sky imager analysis framework
M. Shumko, D. Chaddock, B. Gallardo-Lacourt, E. Donovan, E. L. Spanswick, A. J. Halford, I. Thompson and K. R. Murphy

32 The Space Physics Environment Data Analysis System in Python
Eric W. Grimes, Bryan Harter, Nick Hatzigeorgiu, Alexander Drozdov, James W. Lewis, Vassilis Angelopoulos, Xin Cao, Xiangning Chu, Tomo Hori, Shoya Matsuda, Chae-Woo Jun, Satoko Nakamura, Masahiro Kitahara, Tomonori Segawa, Yoshizumi Miyoshi and Olivier Le Contel

48 Single-spacecraft techniques for shock parameters estimation: A systematic approach
D. Trotta, L. Vuorinen, H. Hietala, T. Horbury, N. Dresing, J. Gieseler, A. Kouloumvakos, D. Price, F. Valentini, E. Kilpua and R. Vainio

64 HUXt—An open source, computationally efficient reduced-physics solar wind model, written in Python
Luke Barnard and Mathew Owens

79 The SpacePy space science package at 12 years
Jonathan T. Niehof, Steven K. Morley, Daniel T. Welling and Brian A. Larsen

89 Profiling heliophysics data in the pythonic cloud
Alex K. Antunes, Eric Winter, Jon Duane Vandegriff, Brian A. Thomas and Jeffrey W. Bradford

98 Wave-particle interactions toolset: A python-based toolset to model wave-particle interactions in the magnetosphere
Stelios Tourgaidis and Theodore Sarris

122 Python tools for ESA's Swarm mission: VirES for Swarm and surrounding ecosystem
A. R. A. Smith, M. Pačes and Swarm DISC

132 Modeling geomagnetic induction in submarine cables
Shibaji Chakraborty, David H. Boteler, Xueling Shi, Benjamin S. Murphy, Michael D. Hartinger, Xuan Wang, Greg Lucas and Joseph B. H. Baker

146 GMAG: An open-source python package for ground-based magnetometers
Kyle R. Murphy, I. Jonathan Rae, Alexa J. Halford, Mark Engebretson, Christopher T. Russell, Jürgen Matzka, Magnar G. Johnsen, David K. Milling, Ian R. Mann, Andy Kale, Zhonghua Xu, Martin Connors, Vassilis Angelopoulos, Peter Chi and Eija Tanskanen

158 The "SPectrogram Analysis and Cataloguing Environment" (SPACE) labelling tool
Corentin Louis, Caitriona Jackman, Sam Mangham, Kevin Smith, Elizabeth O'Dwyer, Aaron Empey, Baptiste Cecconi, Adam Boudouma, Philippe Zarka and Shane Maloney

166 Superposed epoch analysis using time-normalization: A Python tool for statistical event analysis
Samuel D. Walton and Kyle R. Murphy

174 pyDARN: A Python software for visualizing SuperDARN radar data
Xueling Shi, Marina Schmidt, Carley J. Martin, Daniel D. Billett, Emma Bland, Francis H. Tholley, Nathaniel A. Frissell, Krishna Khanal, Shane Coyle, Shibaji Chakraborty, Marci Detwiller, Bharat Kunduri and Kathryn McWilliams

192 sami2py—Overview and applications
Jeff Klenzing, Jonathon M. Smith, Alexa J. Halford, J. D. Huba and Angeline G. Burrell

203 Kamodo's model-agnostic satellite flythrough: Lowering the utilization barrier for heliophysics model outputs
Rebecca Ringuette, Darren De Zeeuw, Lutz Rastaetter, Asher Pembroke, Oliver Gerland and Katherine Garcia-Sage

218 pyGPI5: A python D- and E-region chemistry and ionization model
Stephen R. Kaeppler, Robert Marshall, Ennio R. Sanchez, Diana H. Juarez Madera, Riley Troyer and Allison N. Jaynes

228 GeospaceLAB: Python package for managing and visualizing data in space physics
Lei Cai, Anita Aikio, Anita Kullen, Yue Deng, Yongliang Zhang, Shun-Rong Zhang, Ilkka Virtanen and Heikki Vanhamäki

248 The Lompe code: A Python toolbox for ionospheric data analysis
A. Ø. Hovland, K. M. Laundal, J. P. Reistad, S. M. Hatch, S. J. Walker, M. Madelaire and A. Ohma

264 Solar energetic particle time series analysis with Python
Christian Palmroos, Jan Gieseler, Nina Dresing, Diana E. Morosan, Eleanna Asvestari, Aleksi Yli-Laurila, Daniel J. Price, Saku Valkila and Rami Vainio

277 MAFIAT: Magnetic field analysis tools
D. J. Price, J. Pomoell and E. K. J. Kilpua

283 Daedalus MASE (mission assessment through simulation exercise): A toolset for analysis of *in situ* missions and for processing global circulation model outputs in the lower thermosphere-ionosphere
Theodore E. Sarris, Stelios Tourgaidis, Panagiotis Pirnaris, Dimitris Baloukidis, Konstantinos Papadakis, Christos Psychalas, Stephan Christoph Buchert, Eelco Doornbos, Mark A. Clilverd, Pekka T. Verronen, David Malaspina, Narghes Ahmadi, Iannis Dandouras, Anna Kotova, Wojciech J. Miloch, David Knudsen, Nils Olsen, Octav Marghitu, Tomoko Matsuo, Gang Lu, Aurélie Marchaudon, Alex Hoffmann, Dulce Lajas, Anja Strømme, Matthew Taylor, Anita Aikio, Minna Palmroth, Roderick Heelis, Nickolay Ivchenko, Claudia Stolle, Guram Kervalishvili, Therese Moretto-Jørgensen, Robert Pfaff, Christian Siemes, Pieter Visser, Jose van den Ijssel, Han-Li Liu, Ingmar Sandberg, Constantinos Papadimitriou, Joachim Vogt, Adrian Blagau and Nele Stachlys

313 EXOSpy: A python package to investigate the terrestrial exosphere and its FUV emission
Gonzalo Cucho-Padin, Dolon Bhattacharyya, David G. Sibeck, Hyunju Connor, Allison Youngblood and David Ardila

323 Time-frequency catalogue: JSON implementation and python library
Baptiste Cecconi, Corentin K. Louis, Xavier Bonnin, Alan Loh and Mark B. Taylor

332 Jupiter radio emission probability tool
B. Cecconi, S. Aicardi and L. Lamy

339 The SunPy Project: An interoperable ecosystem for solar data analysis
The SunPy Community , Will T. Barnes, Steven Christe, Nabil Freij, Laura A. Hayes, David Stansby, Jack Ireland, Stuart J. Mumford, Daniel F. Ryan and Albert Y. Shih

349 Solar-MACH: An open-source tool to analyze solar magnetic connection configurations
Jan Gieseler, Nina Dresing, Christian Palmroos, Johan L. Freiherr von Forstner, Daniel J. Price, Rami Vainio, Athanasios Kouloumvakos, Laura Rodríguez-García, Domenico Trotta, Vincent Génot, Arnaud Masson, Markus Roth and Astrid Veronig

360 SwarmFACE: A Python package for field-aligned currents exploration with Swarm
Adrian Blagau and Joachim Vogt

379 Making an executable paper with the Python in Heliophysics Community to foster open science and improve reproducibility
Shawn Polson, Rebecca Ringuette, Lutz Rastaetter, Eric Grimes, Jonathan Niehof, Nicholas A. Murphy and Yihua Zheng

402 SkyWinder: A Python package for flight control, telemetry, and image analysis
Carl Björn Kjellstrand and Bifford P. Williams

415 The pysat ecosystem
R. A. Stoneback, A. G. Burrell, J. Klenzing and J. Smith

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Original Research
PUBLISHED 06 September 2022
DOI 10.3389/fspas.2022.974137

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Stefano Markidis,
KTH Royal Institute of Technology, Sweden
Feng Wang,
Guangzhou University, China

*CORRESPONDENCE
Athanasios Kouloumvakos,
Athanasios.Kouloumvakos@jhuapl.edu

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 20 June 2022
ACCEPTED 04 August 2022
PUBLISHED 06 September 2022

CITATION
Kouloumvakos A, Rodríguez-García L, Gieseler J, Price DJ, Vourlidas A and Vainio R (2022), PyThea: An open-source software package to perform 3D reconstruction of coronal mass ejections and shock waves.
Front. Astron. Space Sci. 9:974137.
doi: 10.3389/fspas.2022.974137

COPYRIGHT
© 2022 Kouloumvakos, Rodríguez-García, Gieseler, Price, Vourlidas and Vainio. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

PyThea: An open-source software package to perform 3D reconstruction of coronal mass ejections and shock waves

Athanasios Kouloumvakos [1]*, Laura Rodríguez-García [2], Jan Gieseler [3], Daniel J. Price [4], Angelos Vourlidas [1] and Rami Vainio[3]

[1]Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD, United States, [2]Universidad de Alcalá, Space Research Group, Alcalá de Henares, Madrid, Spain, [3]Space Research Laboratory, Department of Physics and Astronomy, University of Turku, Turku, Finland, [4]Department of Physics, University of Helsinki, Helsinki, Finland

PyThea is a newly developed open-source Python software package that provides tools to reconstruct coronal mass ejections (CMEs) and shocks waves in three dimensions, using multi-spacecraft remote-sensing observations. In this article, we introduce PyThea to the scientific community and provide an overview of the main functionality of the core software package and the web application. This package has been fully built in Python, with extensive use of libraries available within this language ecosystem. PyThea package provides a web application that can be used to reconstruct CMEs and shock waves. The application automatically retrieves and processes remote-sensing observations, and visualizes the imaging data that can be used for the analysis. Thanks to PyThea, the three-dimensional reconstruction of CMEs and shock waves is an easy task, with final products ready for publication. The package provides three widely used geometrical models for the reconstruction of CMEs and shocks, namely, the graduated cylindrical shell (GCS) and an ellipsoid/spheroid model. It also provides tools to process the final fittings and calculate the kinematics. The final fitting products can also be exported and reused at any time. The source code of PyThea package can be found in GitHub and Zenodo under the GNU General Public License v3.0. In this article, we present details for PyThea's python package structure and its core functionality, and we show how this can be used to perform three-dimensional reconstruction of coronal mass ejections and shock waves.

KEYWORDS

heliophysics, python software package, solar corona, coronal mass ejection, shock wave, remote-sensing solar observations, 3D reconstruction

1 Introduction

Coronal mass ejections (CMEs) are large expulsions of magnetized plasma from the Sun's outer atmosphere, the corona, towards the interplanetary (IP) space, and they are the most spectacular eruptive phenomenon observed in the solar corona and one of the main drivers of space weather. Big CMEs can release a huge amount of plasma and magnetic flux into the IP space (e.g. Chen, 2011; Webb and Howard, 2012). CMEs can also drive shock waves (e.g. Ontiveros and Vourlidas, 2009; Frassati et al., 2019) and can cause very intense geomagnetic storms on Earth, which usually have a significant impact on ground-based and space-borne technological systems (Lakhina and Tsurutani, 2016; Temmer, 2021).

Our knowledge about CMEs and shock waves has significantly increased over the past several decades. The initiation processes, three-dimensional (3D) structure, evolution, and properties of CMEs and shock waves are very important and recurrent areas of heliospheric physics research (e.g. Balmaceda et al., 2020; Balmaceda et al., 2022; Rodríguez-García et al., 2022), since these are the main space weather drivers. In addition, a deeper understanding of the physical processes involved in these phenomena (e.g. Patsourakos and Vourlidas, 2012; Kwon and Vourlidas, 2017; Long et al., 2017) is essential to resolve fundamental scientific questions in heliophysics. This understanding also helps to elucidate the processes involved in other associated solar phenomena. For example, during the most energetic solar events, when fast CMEs and shock waves are observed, particles are accelerated at very high energies (e.g. Kouloumvakos et al., 2019; 2020a). Solar energetic particles (SEP) events are another important element of space weather and pose a significant hazard to the inner heliosphere, especially for those events with large intensity increases and high energy particles (e.g. Gómez-Herrero et al., 2015; Rodríguez-García et al., 2021).

CMEs and shock waves can be observed either by remote sensing instruments or can be measured *in situ*. Coronal transients are nowadays routinely observed in white light (WL) that is scattered by free electrons of the solar corona. These observations are provided by coronagraphs or heliospheric imagers onboard spacecraft located in near-Earth orbit. Observations of the solar corona from the Sun-Earth Connections Coronal and Heliospheric Investigation (SECCHI; Howard et al., 2008) onboard the *Solar Terrestrial Relations Observatory* (STEREO; Kaiser et al., 2008) have provided nearly simultaneous imaging of CMEs and other transients from different viewpoints for more than a decade. The three different perspectives provided by near-Earth spacecraft (e.g. the *Solar Dynamics Observatory* (SDO; Pesnell et al., 2012) and the *Solar and Heliospheric Observatory* (SOHO; Domingo et al., 1995)) and the two STEREO spacecraft allow us to reconstruct CMEs and shocks. However, since 2015, communications with STEREO-B were lost and currently STEREO consists of only one observatory–STEREO-A–that slowly catching up with Earth. Multiple viewpoint observations are essential to study the 3D structure and kinematics of the CMEs and shock waves, alleviating projection effects and reducing the uncertainty when determining the position and kinematics (Mierla et al., 2010).

Numerous previous studies have extensively used and relied on CME and shock wave 3D reconstructions to address top-level scientific questions. Most of the studies focus on examining the geometrical properties and kinematic parameters of CMEs and shock waves as they evolve in the solar corona. These studies contribute to addressing fundamental questions about CME initiation and energetics; the interaction of CMEs with coronal structures, the solar wind magneto-plasma, or other CMEs (CME-CME interaction) (Scolini et al., 2020; Palmerio et al., 2021); physics of collisionless shock waves in the corona (Kwon et al., 2014; Kwon and Vourlidas, 2017); initiation and development mechanisms of shocks (Mancuso et al., 2019); the relation of CMEs-shocks to solar flares; and many other CME or shock related subjects (e.g. Wood et al., 2011; Hess et al., 2020; Rouillard et al., 2020). Some studies have indirectly benefited from 3D reconstructions to address their overarching goals for another topic, other than the physics of CME and shock waves. These studies investigate, for example, the role of CMEs and shock waves to SEP acceleration and release to the heliosphere (e.g. Rouillard et al., 2016; Kouloumvakos et al., 2019, 2020b; Giacalone et al., 2020; Dresing et al., 2022), the wide distribution of SEPs in the heliosphere (e.g. Rouillard et al., 2012; Lario et al., 2014, 2016, 2017; Kouloumvakos et al., 2016; Zhu et al., 2018; Rodríguez-García et al., 2021; Kouloumvakos et al., 2022), and the connection of CMEs and shocks to solar radio emission (e.g. Type II, IV radio bursts; Zucca et al., 2018; Morosan et al., 2020; Kouloumvakos et al., 2021; Jebaraj et al., 2021) and solar gamma-ray bursts (Kouloumvakos et al., 2020a). CME and shock wave reconstruction can also play a critical role in space weather forecasting and nowcasting models. The propagation characteristics and kinematics of IP CMEs are key parameters to evaluate the impact of the events on Earth using space weather modelling.

The most widely used method to perform 3D reconstructions combines a geometrical model with the scraytrace code which is available in the Solarsoft library that uses the Interactive Data Language (IDL) to run it. IDL is a popular programming language in areas of science, however, a paid licence is needed to use it. Additionally, this 3D reconstruction process requires that the user download and process the data before the reconstruction. The users can use pairs or triplets of images from coronagraphs or extreme ultraviolet (EUV) imagers. Within the geometrical models, the graduated cylindrical shell (GCS; Thernisien et al., 2006; Thernisien, 2011) model is widely used to parameterize the 3D structure of a typical flux-rope shape like a CME. For shock waves, a spheroid or an ellipsoid model is extensively used to parameterize them (e.g. Kwon et al., 2014).

The 3D reconstruction is then performed by overlaying, onto each imager, the grid of points parameterized from the geometrical model. The points are usually projected onto the plane-of-sky (POS) when using images from the coronagraphs. Then the user changes the model parameters until a good fit is achieved compared with the observations. The final 3D reconstruction has to reproduce the observed features at each observing point.

In this article, we present PyThea, a newly developed open-source software package that can be used to perform 3D reconstruction of coronal mass ejections and shock waves. The name of the package is inspired by the words *Python* and *Thea*. In Greek mythology, *Thea*, also called Euryphaessa "wide-shining", is the Titaness of sight and the shining light of the clear blue sky and also the mother of Helios (the Sun), Selene (the Moon), and Eos (the Dawn). PyThea is developed and managed by A. Kouloumvakos. The source code of the package is provided in a publicly available GitHub repository[1]. GitHub is a code hosting platform for version control and collaboration. This package is produced in Python and licensed under GPL-3.0 License[2]. The main goal behind this package development is to provide to the community the functionality and tools needed to perform robust 3D reconstructions of CMEs and shock waves in Python programming language, which is one of the most widely used programming languages[3]. Python is a free-of-cost and open-source programming language, it is easy to learn and use, and it has a very active and supportive community.

The scope of this article is to introduce PyThea software package to the heliophysics community and present its main features. This presentation will be generic and independent of the package version. We start with a description of PyThea's python package structure and its core functionality. Then we show details on the application and the constructed Graphical User Interface (GUI). The user can use this application to perform a full analysis of an eruptive event by visualizing the remote sensing observations and performing the CME and shock wave reconstructions. In the last section, we discuss the current state of the package and future development.

2 PyThea software package

2.1 Overview

The PyThea software package is developed to provide the necessary tools to perform 3D reconstruction of CMEs and shock waves and determine their kinematics, using multi-spacecraft remote sensing observations. This package has been built in Python (≥3.8) with an extensive use of libraries available within the Python language ecosystem. PyThea has been tested in Unix based systems, however, PyThea is a platform-independent package. During the development and release process of the package, we follow a semantic versioning, which consists of three-part version numbers: major version, minor version, and patch. At the time of writing this article, PyThea's latest version is v0.6.6. This is primarily done to convey that there is a compatibility between releases. Every new version of PyThea is uploaded and registered in the Python Package Index (PyPI). This is a repository with a list of available software packages produced with the Python programming language.

The latest or specific versions of PyThea can be found and downloaded from PyPI[4] or from the host GitHub repository. PyThea can be installed using pip, which is the package installer for Python and it is usually shipped along with it. The pip command installs PyThea package from PyPI by default. Like most python packages, to use PyThea it is required that some dependent packages have been installed. These dependent packages are installed automatically using pip. An installation using conda is also possible (see details in GitHub). conda is another package and environment management system, which helps to easily install packages and their dependencies. Unlike pip, conda checks the dependencies before the installation and identifies conflicts before they occur. We recommend for less experienced users to create a "virtual environment" in Python or conda[5]. before installing PyThea package. This is an easy process that allows the package to be installed in an isolated environment rather than globally.

We build PyThea using fundamental Python packages for data manipulation, scientific computing, and high level visualization that produce publication-quality figures. NumPy (Harris et al., 2020) and SciPy (Virtanen et al., 2020) are, for example, two foundational Python packages that are used to operate on data and deal with scientific computations. Both packages offer a comprehensive library of mathematical functions that can operate on arrays and matrices. Another example is Matplotlib (Hunter, 2007), which is a comprehensive plotting library for Python. To build the main application and the GUI, we used the Streamlit Python package. This is an open-source Python library that provides an easy way to create web applications. After the installation of the PyThea package, the web application can be launched using only one command in the terminal.

For the scientific data analysis, we use two core packages, Astropy (Astropy Collaboration et al., 2013; Astropy

1 https://github.com/AthKouloumvakos/PyThea

2 GNU General Public License v3.0

3 Python has overtaken C at 1st position in October 2021 in TIOBE Programming Community index. This index is an indicator of the popularity of programming languages.

4 https://pypi.org/project/PyThea/

5 More details of how to manage environments using conda are provided in this link

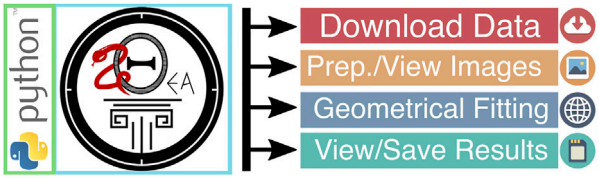


FIGURE 1
Semantic of PyThea's typical workflow for the analysis and reconstruction of an event.

Collaboration et al., 2018) and SunPy (Mumford et al., 2020; The SunPy Community et al., 2020). These packages provide the functionality to read, process, and visualize astronomical data. Astropy includes many libraries that support scientific data analysis. In PyThea, we use two main functionality features from Astropy, the ability to use numbers with associated units (astropy.units) and to represent and transform between different coordinate systems (astropy.coordinates). From SunPy, we use the data search, retrieval, load, and visualization functionality. The most important subpackages are sunpy.net, which is used to retrieve the data, and sunpy.map, which provides the framework to load and visualize the data. Additionally, the sunpy.coordinates subpackage provides support to represent and transform between different coordinate systems used in solar physics.

In Figure 1, we present a typical workflow when analysing and reconstructing an event using PyThea. The first step is the data retrieval from multiple sources. Then, these data are processed and visualized so that the user can fit the geometrical model. The last step is the calculation of the kinematics and saving of the final fittings. In the following sections, we present information for each subprocess.

2.2 Data acquisition and loading

PyThea provides functions to download and load solar data from selected imagers. The process relies on the data acquisition interface of SunPy, named Fido. It is a powerful interface that provides a unified way to perform data search and retrieval from multiple sources simultaneously. The primary data source for PyThea is the Virtual Solar Observatory (VSO; Hill et al., 2009). It is a tool developed to allow access to data from multiple solar data providers and different data sets from space-borne and ground-based instruments. The imaging data is provided in the Flexible Image Transport System (FITS) file format, which is commonly used in the astronomy community. A particular goal for PyThea's data acquisition pipeline is to make the data search and retrieval process an easy and automated task. We envision that users may spend more time on data analysis rather than on data search and download when using PyThea.

PyThea currently provides the option to download and process imaging data from eight different imagers from four different spacecraft (not including PSP and SolO). Multi-viewpoint observations are essential to perform robust CME or shock wave 3D reconstructions. A synoptic list of the imagers and spacecraft whose data can be used in PyThea is given in Table 1. The imaging data also cover a broad range of heights, from the solar corona to the interplanetary space. In PyThea, we use observations from EUV and heliospheric imagers, as well as from coronagraphs. For the EUV imagers AIA and EUVI, for example, PyThea provides the option to analyse images from only one pre-selected wavelength (~19.3 nm). A multi-wavelength implementation is also under consideration. Additionally, imaging data from the two new solar missions, *Parker Solar Probe* (PSP; Fox et al., 2016) and *Solar Orbiter* (SolO; Müller et al., 2020; Zouganelis et al., 2020), are expected to be included in a future release of the PyThea package.

For the selected imagers, Fido searches in VSO for available imaging data and retrieves them automatically. The search is performed at a time interval which is by default 1 hour before and after a user-selected central time. The user can extend the time interval in which the search and download is performed. For every new search, a new data set is downloaded for each of the selected imagers only when the data do not yet exist in the local database. If PyThea's GUI is used for this process, the data update (download and preparation of images) is fully automatic. We discuss PyThea's GUI in more detail in Section 3.

To load FITS files, PyThea uses the sunpy.Map.Map class from SunPy's utilities. This class is another powerful feature provided by SunPy, which is used to load solar imaging data. During the loading process, the sunpy.Map.Map class detects automatically the file type and the associated instrument using information from the FITS header. It also uses the FITS header keywords to determine and interpret the coordinate system of the imaging data, and construct the map metadata. This class provides the functionality needed to create a World Coordinate System (WCS) header from a SkyCoord object and the utilities to perform geometric transformations between different coordinate systems using the WCS interface (e.g. pixel_to_worldand world_to_pixel).

2.3 Imaging processing and visualization

PyThea provides general functions that can be used to filter and prepare the loaded maps (sunpy.map) before visualizing them. The first and most important aspect is to keep the images that meet certain criteria. Filtering the maps helps to load only those images whose quality is the best and can be used for the analysis. These functions remove duplicate images, or images with selected exposure times, dimensions, or polarization angles as far as the coronagraphic images are concerned. The final list of maps is returned as a sunpy.map.MapSequence, which is a chronologically ordered list.

TABLE 1 A list of the available imagers used in PyThea.

Spacecraft	Imager	Wavelength	Height	Type
SOHO	LASCO-C2	WL	—	Coronagraph
	LASCO-C3	WL	—	Coronagraph
STEREO-A/B	EUVI	193 Å	—	Imager
	COR1	WL	—	Coronagraph
	COR2	WL	—	Coronagraph
	HI1	WL	—	Heliospheric Imager
	HI2	WL	—	Heliospheric Imager
SDO	AIA	193 Å	—	Imager
PSP(*)	WISPR	193 Å	—	Heliospheric Imager
SolO (*)	Metis	WL and UV	—	Coronagraph
	Hi	193 Å	—	Heliospheric Imager

(*) Imager not implemented in PyThea yet. Expected to be available in a future release

The second set of functions can be used to prepare the images. This is a basic image processing step that includes four basic preparations: 1) normalization of the exposure time, 2) resampling of the images, and in the case of coronagraphic images, 3) masking the occulter and 4) combining different polarized brightness images that belong to the same polarization sequence. Most of the coronagraphic data are provided as total brightness images, except for STEREO/COR1, which are triplets of polarized brightness images in three different angles. Currently, PyThea (v0.6.6) does not include or provide the functionality to reprocess the FITS files and increase their processing level, e.g. to calibrate the images into physical units or to apply flat field corrections, remove stray light, correct geometric distortion and vignetting, and other corrections. However, the level of the FITS files provided from VSO is adequate to robustly perform the 3D reconstruction. The images are resampled with the use of the superpixel method that is provided by sunpy.map. This method reduces the resolution of the images by combining pixels and thus increasing the signal-to-noise ratio (SNR). The default image size used in PyThea is 512 × 512 pixels, providing fast processing times in the GUI and high SNR, which are essential to track the evolution of faint transients in the corona. The user can change the default image size for every imager separately by changing the configuration file. Additionally, during the processing of the coronagraphic images, we mask out the pixels that are inside the occulting disk since they do not contain any useful data.

During the geometrical fitting process, it is useful to use base or running difference images because it is easier to track the evolution of the CME and shock wavefront. Therefore, a final step of processing is to produce the base difference or running difference images from the plain images by subtracting the previous image or the first image in a sequence, respectively. PyThea provides utilities that can be used to process the images into base or running difference images, and returns the final maps as sunpy.map.MapSequence.

The final maps are plotted as images using the sunpy.map plot function. This function plots the map object using a method equivalent to imshow from matplotlib package (Hunter, 2007) and uses the "nearest neighbor" interpolation method. For plain images, we use false colours to better visualize them. SunPy provides the colourmap for each imager as defined by the instrument teams. For the difference images, we use a reverse grey colourmap and a linear normalization of the data. In Figure 2, we show an example of the images in EUV and WL produced with PyThea. The GCS (green grid in Figure 2) and ellipsoid geometrical model (red grid in Figure 2) are fitted to the CME and the shock wave, respectively, and overlaid to the images.

2.4 The geometrical models

2.4.1 Overview

The fitting of the geometrical models to the observations is performed using near simultaneous images from multiple viewpoints. The final fit is registered to one of the selected images, hence, we term this as "single frame" fitting. Every single frame fitting can be converted into a pandas.DataFrame, which is a data structure, organizing them into a table row. This row contains the geometrical parameter values, the observation time of the image, and the source of the observations for only one fit. The different single frame fittings can be combined into a single table that constitutes the "event model fittings", as we term this. The event model fittings is also a pandas.DataFrame containing information of the same geometrical parameters as the single frame fitting. The format of the event model fittings is similar to a spreadsheet, and it can be used to calculate the kinematics and to store the geometrical fitting to a file.

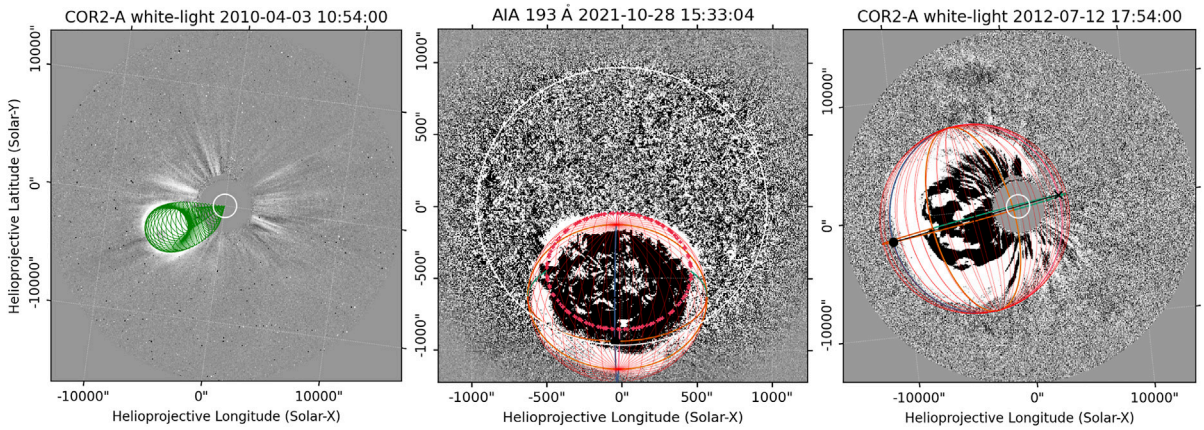


FIGURE 2
Running difference images in WL (right and left panels) and EUV (middle panel) produced with PyThea. The GCS (green grid), ellipsoid and spheroid models (red grid) are overplotted to the images.

Each geometrical model is constructed as a class so that the user can create an object for each model. The models are described by a set of positional and geometrical parameters. The positional parameters are represented by using astropy.SkyCoord in the heliographic Stonyhurst (HGS) or heliographic Carrington (HGC) coordinate system. The origin of both coordinate systems is located at the solar center. The geometrical parameters of every model are described as physical quantities using astropy.units. Additionally, for each model we provide a method that returns the coordinates of the mesh points constructing the model. The points are represented using astropy.SkyCoord in the HGS frame.

2.4.2 The spheroid and ellipsoid model

The ellipsoid or spheroid models are usually used to reconstruct shock waves in 3D. CME-driven shock waves in the corona can be observed as propagating fronts in WL (e.g. Kwon et al., 2013). These fronts has been suggested to form the halo envelope of CMEs (Kwon et al., 2014), so that the outermost halo front part of the CMEs is formed by the propagating shock wave rather than a projection of the CME ejecta. In coronagraphic observations the projection of the large-scale morphology of the WL shock waves is usually seen as an ellipse. In 3D, the ellipsoid model seems a rather good approximation to describe the global large-scale structure of the WL shocks. Additionally, a simpler spheroid model can also be used, when it is better to perform the fitting with a reduced complexity.

First, we start from the definition of the ellipsoid model, which is the most generic case. An ellipsoid is a quadratic closed surface in which all plane cross-sections are either ellipses or circles. It has three mutually perpendicular axes of symmetry that intersect at a center of symmetry, which we will call hereafter the center of the ellipsoid. The three line segments along the axis of symmetry that start from the center of the ellipsoid and end at its surface are called the principal semi-axes of the ellipsoid, and they correspond to the semi-major axis and semi-minor axis of the plane cross-section ellipses.

The implicit equation of an ellipsoid in the Cartesian coordinate system is $x^2/a^2 + y^2/b^2 + z^2/c^2 = 1$, where a, b, and c are the lengths of the three principal semi-axes. In Figure 3, we show an ellipsoid with the three axes of symmetry and the three principal semi-axes. In the case where the three semi-axes of the ellipsoid are equal ($a = b = c$), the surface is a sphere, and if only two semi-axes are equal, the surface is an ellipsoid of revolution, or most commonly called a spheroid. The two cases are also presented in Figure 3. Then, a spheroid is obtained by revolving an ellipse about one of its principal axes, and it has a circular symmetry. When revolving the ellipse about its minor (major) axis, an oblate (prolate) spheroid is formed. Therefore, a spheroid is oblate (prolate) when a and b are equal and also are greater (smaller) than c.

Since the spheroid is a special case of the ellipsoid model, we start by defining the latter first. To define the ellipsoid model, we use three positional and three geometrical parameters. In Table 2, we present synoptically the parameters for each model. The point of reference for the ellipsoid (and spheroid) model is the center of symmetry. In PyThea, this point is represented as a SkyCoord in the HGS coordinate system. In the spherical coordinates, the three positional parameters, r_{center}, ϕ, θ, define the heliocentric distance, latitude, and longitude of the ellipsoid center, respectively. Additionally, we define the primary (first) semi-

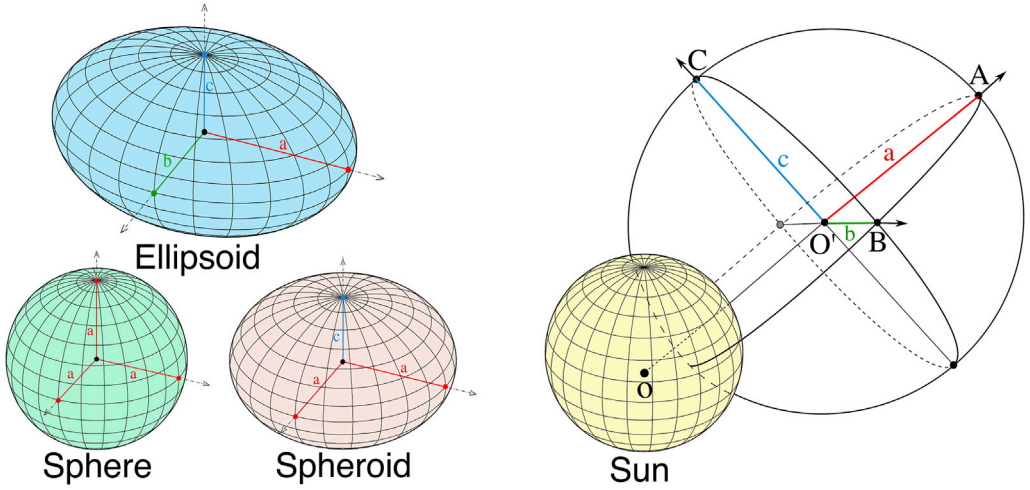


FIGURE 3
A schematic of three different ellipsoids (left), and of an ellipsoid model showing the different axis and parameters (right).

TABLE 2 Geometrical models available in PyThea showing the different parameters used.

Models	Parameters		Definitions
	Positional(+)	Geometrical	
Sphere	ϕ, θ, r_c	a	ϕ: longitude, θ: latitude, a: first principal semi-axes
	ϕ, θ	h, κ	h: apex height (r_{apex}), κ: self-similar constant
Spheroid	ϕ, θ, r_c	a, b	r_c: heliocentric distance, b: second principal semi-axes
	ϕ, θ	h, κ, ϵ	ϵ: eccentricity
Ellipsoid	ϕ, θ, r_c, γ	a, b, c	γ: tilt angle, c: third principal semi-axes
	ϕ, θ, γ	h, κ, ϵ, α	α: aspect ratio
GCS	ϕ, θ, γ	h, κ, α	h: apex height (h_{apex}), κ: aspect ratio, α: half angle

(+) The positional parameters ϕ, θ, r_c define the heliocentric distance, latitude, and longitude of the ellipsoid left.

axis of the ellipsoid model to start from the ellipsoid center and pointing radially outward towards the same direction as the position vector of the ellipsoid center. The other two semi-axis are orthogonal to the primary semi-axis, and their direction depends on the tilt angle, γ, relative to the solar equator. For $\gamma = 0$, one of the semi-axis is coplanar with the solar rotation vector, and the other semi-axis is parallel to the solar equatorial plane. Note that for the spheroid model the tilt angle is not defined since any rotation along the radial semi-axis is trivial as there is a circular symmetry with respect to this axis.

The geometrical parameters of the ellipsoid model are the lengths a, b, and c of the three principal semi-axes (two in the case of the spheroid), respectively. The positional and geometrical parameters are sufficient to fully define the geometry of the ellipsoid model in 3D. PyThea's users can adjust the ϕ, θ, and γ values to match the position and orientation of the ellipsoid with the observed shock in WL and EUV, and the length of the three principal semi-axes to fit the geometry of the shock front in every direction. The ellipsoid model can also be constrained to expand self-similarly. This provides a more convenient way to perform the shock fittings. In this case, the positional and geometrical parameters have to be defined differently. Instead of using the length of the three semi-axes as the geometrical parameters, the user adjust the heliocentric height of the ellipsoid at the apex, and the length of the other two semi-axis are calculated using a self-similar constant (κ), an aspect ratio (α), and the eccentricity (ϵ) of the ellipsoid defined from the cross-sectional ellipse of two semi-axis. The κ parameter is a self-similar constant that is defined as the ratio of the height of the apex to the length of one of the semi-axis ($\kappa = b/(r_{apex} - 1\ R_{\odot})$). This value is proportional to the aspect ratio between the two semi-axis a and b. The OA

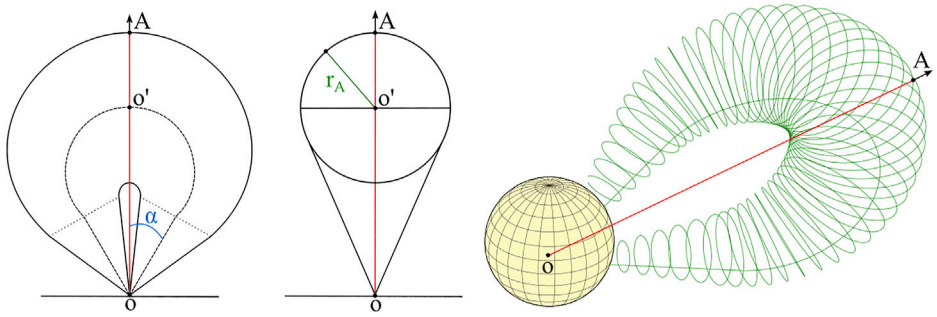


FIGURE 4
Plane sections of the GCS model (left and middle panels) showing the different parameters, and a schematic of the model mesh (right panel).

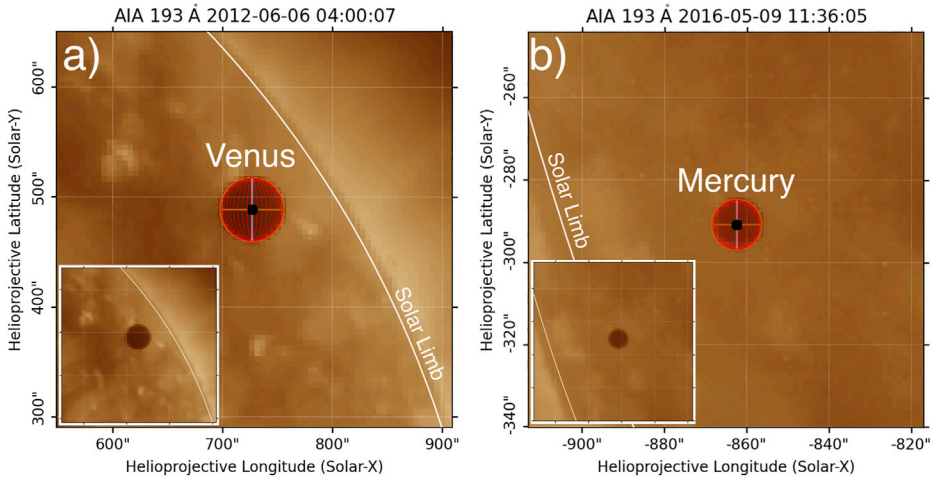


FIGURE 5
Two examples of the geometrical fitting to bodies in the solar system with known locations and dimensions. Panel **(A)** shows the Venus transit on 6 June 2012, and Panel **(B)** the Mercury transit on 9 May 2016, as viewed by AIA at 19.3 nm. The fitted ellipsoid to the planets is shown with the red mesh. The insert images show the AIA observations without the fitting.

line segment shown in Figure 3, is the height of the apex, so $r_{apex} = OA = r_{center} + a$. The α parameter is the second aspect ratio of the ellipsoid, and it is defined as $\alpha = b/c$, while ϵ is the eccentricity, and we define it as follows:

$$\epsilon = \begin{cases} \sqrt{1 - a/b} & a < b \\ -\sqrt{1 - b/a} & a > b \end{cases} \quad (1)$$

In this case, users can adjust any of these four geometrical parameters r_{apex}, κ, α, and ϵ to fit the ellipsoid model to the observed shock front. Changing r_{apex} and keeping constant the other parameters, the expansion is self-similar. For the positional parameters, ϕ, θ, γ are also adjusted by the user, while r_{center} is calculated from $r_{center} = r_{apex} - a$.

2.4.3 The GCS model

The GCS model is an empirical geometrical model of a flux rope defined by Thernisien et al. (2006); Thernisien (2011). It consists of a curved front that is a cylindrical shell forming the main part of the CME–from its "legs" to the apex–and two attached cones that correspond to the legs of the CME. The resulting shape is reminiscent of a croissant, as we show in Figure 4. The model is constrained to expand self-similarly, which it seems to be the case for most of the CMEs at heliocentric heights $> 10\,R_\odot$ (e.g. Balmaceda et al., 2020). Previous studies (Dumbović et al., 2019) have shown that the GCS model reproduces well the large-scale structure of flux rope-like CMEs in the solar corona, and it is widely used to reconstruct their 3D structure.

Following Thernisien et al. (2006), we define the GCS model using three positional and three geometrical parameters. These are given in Table 2. We define as a point of reference for the GCS model the apex center (*A* in Figure 4). This point is represented in PyThea as a SkyCoord in the HGS coordinate system. The primary axis of the GCS model is defined from the solar center and directed towards the apex center. In the spherical coordinates, the three positional parameters (r_{apex}, ϕ, θ) define the heliocentric distance, latitude, and longitude of the flux rope apex center, respectively. Additionally, the flux rope can be tilted relative to the solar equator. For a tilt angle, γ, equal to zero, both legs of the CME are located at the solar equator.

The geometrical parameters of the GCS model are three: 1) the heliocentric height at the apex, h_{apex} (*OA* in Figure 4), 2) the aspect ratio at the apex, κ, which is defined as the ratio *OO*' to r_A and sets the rate of lateral versus radial expansion of the CME, and 3) the half angle, α, which is the angle between the axis of the cone and the primary axis. From these six parameters we can fully define the GCS model, therefore, in PyThea the user can adjust ϕ, θ, and γ to match the flux rope position and orientation with observations, and h_{apex}, κ, and α to fit the geometry.

2.5 Visualizing the geometrical models

To fit the geometrical models to the observed CMEs and shocks, the mesh points constructing the models, have to be projected to the respective images that are used for the reconstruction. For each geometrical model, the coordinates of the mesh points are represented using astropy.SkyCoord in the HGS frame so these points are in world coordinates. To visualize the mesh points we use SunPy functionality and plot_coord which plots an astropy.SkyCoord onto the image (see also astropy.plot_coord). The plot_coord method converts the world coordinates to pixel coordinates and plots them onto the images. The world-to-pixel coordinate transformation is based on the FITS WCS standard. This standard describes the geometric transformations between two sets of coordinates, hence, associating physical values to positions within the FITS dataset. In our case, the FITS WCS standard is used to convert the heliospheric coordinates to pixel coordinates of an image. SunPy supports a broad set of heliospheric coordinate systems (see Thompson, 2006) that extend the Astropy coordinates framework and also allows transformations between the different coordinate systems implemented in both SunPy and Astropy . The accuracy of the coordinate transformations and projections is a critical aspect that ultimately controls the accuracy of this software package. Thankfully, the functionality provided by the SunPy has been extensively tested and agrees with great accuracy with published values in the Astronomical Almanac. In Figure 5, we show two examples of the geometrical fitting to bodies in the Solar System with known locations and dimensions. This fitting provides a good test of the accuracy of the geometrical models and the coordinate transformations and the final visualization. For example, in Panel a, we show the Venus transit on 6 June 2012, as viewed by AIA at 19.3 nm. Using the location and radius of the planet we fit the ellipsoid model to Venus and visualize the result. To calculate the planet's location, we used the Solar System ephemeris file (DE432s.bsp) provided by Jet Propulsion Laboratory and for the radius, we used the mean equatorial radius (R_V = 6051.8 km) provided by NASA's space science data coordinated archive. The resulting fitting match very well with the observations. For Panel b, we used the same method for the Mercury transit on 9 May 2016, as viewed by AIA at 19.3 nm.

2.6 Fittings processing and kinematic plots

The main goal when performing the geometrical 3D fitting of a CME or a shock wave is to reconstruct their 3D structure and determine their position and kinematics with accuracy, minimizing the projection effects. PyThea provides utility functions to further process the produced geometrical 3D fittings. These functions can be used to calculate the kinematics and visualize the temporal evolution of the fitted geometrical parameters.

Using PyThea's utilities, the user can perform a polynomial or spline fitting to the kinematic curves and determine the propagation and expansion speed. For the polynomial fitting, we use the numpy.polyfit function that performs a least squares polynomial fit of any degree to the set of data points. For the spline fit, we use the UnivariateSpline from the scipy.interpolate package (Virtanen et al., 2020). The user can change the degree of the smoothing spline (from unity to five) and the smoothing factor. The uncertainty of the kinematic is computed by the goodness of fit to the height-time data. For both fitting models, the uncertainty is calculated from the standard deviation. To calculate this for the polynomial method, we use the covariance matrix, whereas, for the spline fit it is computed from the residuals of the fit, which is a measure of how well a spline fits the data. An example of the produced height (length)-time and speed-time plots is given in Figure 8 where we perform a full reconstruction for a solar event.

2.7 Save/load the fitting results

The results of the geometrical 3D fittings can be saved and then loaded again to preview or continue the analysis for an event. As we mentioned in Section 2.4.1, the individual frame fittings are stored in the model_fittings class that initializes an object to store all the frame fittings of the geometrical model. This object contains two methods, to_dict() and to_json(), that can be used to return the final fittings of the geometrical model in either a dictionary or JSON (JavaScript Object Notation) file format.

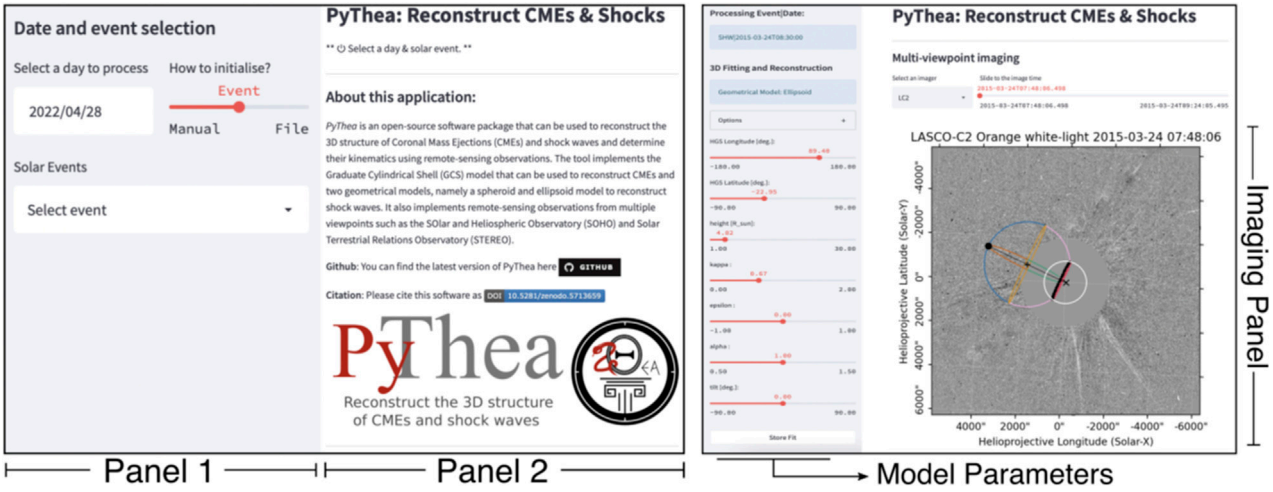


FIGURE 6
Two views of PyThea's web application. The left panel shows the starting page of the application and the right panel shows the main fitting page.

JSON is an open, language-independent standard file format, that uses human-readable text to store data in attribute–value pair format. In Python, the JSON files can be easily imported and converted into a dictionary using the JSON encoder and decoder package. Inside the JSON file, we store information about the date and time of the selected event, the geometrical model used and the parameters of the geometrical model, which is a pandas.DataFrame of the frame fittings, and the parameters used to process the kinematics, that is the selected fitting method and the associated fitting parameters. An example of the exported JSON file is given below:

Listing 1: An example of the exported JSON file.

```
{
 "event_selected": "FLX2p0|2021-01-01T00:00:00",
 "date_process": "2021-01-01T00:00:00.000000",
 "geometrical_model": {
  "type": "GCS",
  "parameters": {
   "hgln": [0.0, ...],
   "hglt": [0.0, ...],
   [...]
  }
 },
 "kinematics": {
  "fit_method": {
   "type": "polynomial",
   "order": 1
  }
 },
 "date_created": "2020-01-01T00:00:00",
 "version": "0.6.6"
}
```

3 The web application

3.1 Graphical user interface

Performing a 3D reconstruction is mainly an interactive process. To reconstruct an event, the user aim to achieve the best fit of a geometrical model to multi-viewpoint coronal observations by adjusting a set of geometrical parameters. Without a GUI, this process is almost impossible to be performed. For that reason, in PyThea we provide a modern application that can be used to perform a full analysis of an event. The GUI of this application has been built based on Streamlit. Figure 6 shows two views of the PyThea's web application. Panel a shows the starting page of PyThea and panel b the main fitting page. The web app consists of two main vertical panels, panels 1 and 2 as labeled in Figure 6. Panel one is used as a placeholder for the user input widgets, while panel two is used for the display of data elements. Additionally, in Figure 7 we show a more detailed view of the four input widgets contained in panel 1.

3.2 Initializing the fitting process

When the web application starts for the first time, the user has to initialize it by choosing one of three different options. The first option is to select the date of the fitting. After this selection, the application searches in the Heliophysics Events Knowledgebase (HEK: Hurlburt et al., 2010) database for registered solar flares and returns them in a dropdown menu. Each option contains information about the flare class and the flare maximum time, and the user can select among the different events. This selection is used to associate the fitting to a flare and give a unique identification label to the final products. The identification label is used mainly for archiving purposes of the final fitting files. If the user does not want to associate the fitting files to a flare or when a solar flare has not been registered in the HEK database at the time of the CME or the shock under



FIGURE 7
Panels **(A–D)** show a detailed view of the panels that appear in PyThea's web application. These panels contain the different input widgets such as sliders, radio buttons, drop boxes, and others that can be used to provide input parameters to the application.

investigation (for example because the flare was located behind the visible disk), the user can select the second option which is to initialize the application manually. This is dome by choosing between different event identification labels (e.g. CME, shock) and select manually the date and time of the solar event under investigation to start the fitting process. The next step is the selection of the geometrical model (i.e. the ellipsoid) that is used for the fitting. After this step, the application is ready to initiate, and the data download and loading process starts. We give further details of this process below. The third option is to provide a fitting file previously processed and the application will automatically initialize. Then the fitting file loads the user can preview or continue the fitting process.

3.3 Imaging data download, load, and process

The application downloads, loads, and processes the imaging data automatically after the selection of the geometrical model. By default only three imagers are loaded in the beginning, however, more instruments can be selected from a list of supported imagers (see Figure 7C, top) incide the application. The application searches in VSO for available imaging data for a selected time interval before and after the event's characteristic time (i.e. the flare maximum). This time interval can be changed during the fitting process (see Figure 7C, bottom). When a new imager is selected or when the time interval changes, the application automatically downloads and processes the new images. The imaging data are displayed in the "Image Panel" shown in Figure 6B. By default the images are processed as running difference images. The application provides also the option to view base difference or plain images (see Figure 7D, top). The main imager of which the data is viewed on the main fitting page can be selected from a dropdown menu. An example is shown in Figure 6B. Using a time slider, the different loaded images can be viewed.

3.4 The geometrical fitting process

At every image the geometrical model is over-plotted with or without the grid of the fitting mesh. The parameters of the geometrical model can be changed by adjusting the parameter sliders (see Figure 7B). Any update of the model parameters results in an automatic update of the location of the model in the images. The fitting process is repeated until there is a good fit of the geometrical model to the observations. For the multi-viewpoint analysis of an event, nearly-simultaneous images from at least three different imagers have to be used and the geometrical model has to have a good fit in more than one image. To perform a multi-viewpoint fitting the application provides two supplementary images that appear on two side-by-side panels below the primary image that is already displayed on the top. The data from the supplementary imagers are the closest in time available images to the main image.

The geometrical fitting to a single image (or triplet of images) can be saved and every new single frame fitting is added to a pandas.DataFrame, which can be used to construct the model_fittings object. This object contains all the information about the geometrical fitting process and is used to export the final results to a JSON file format (see Section 2.7). Additionally,

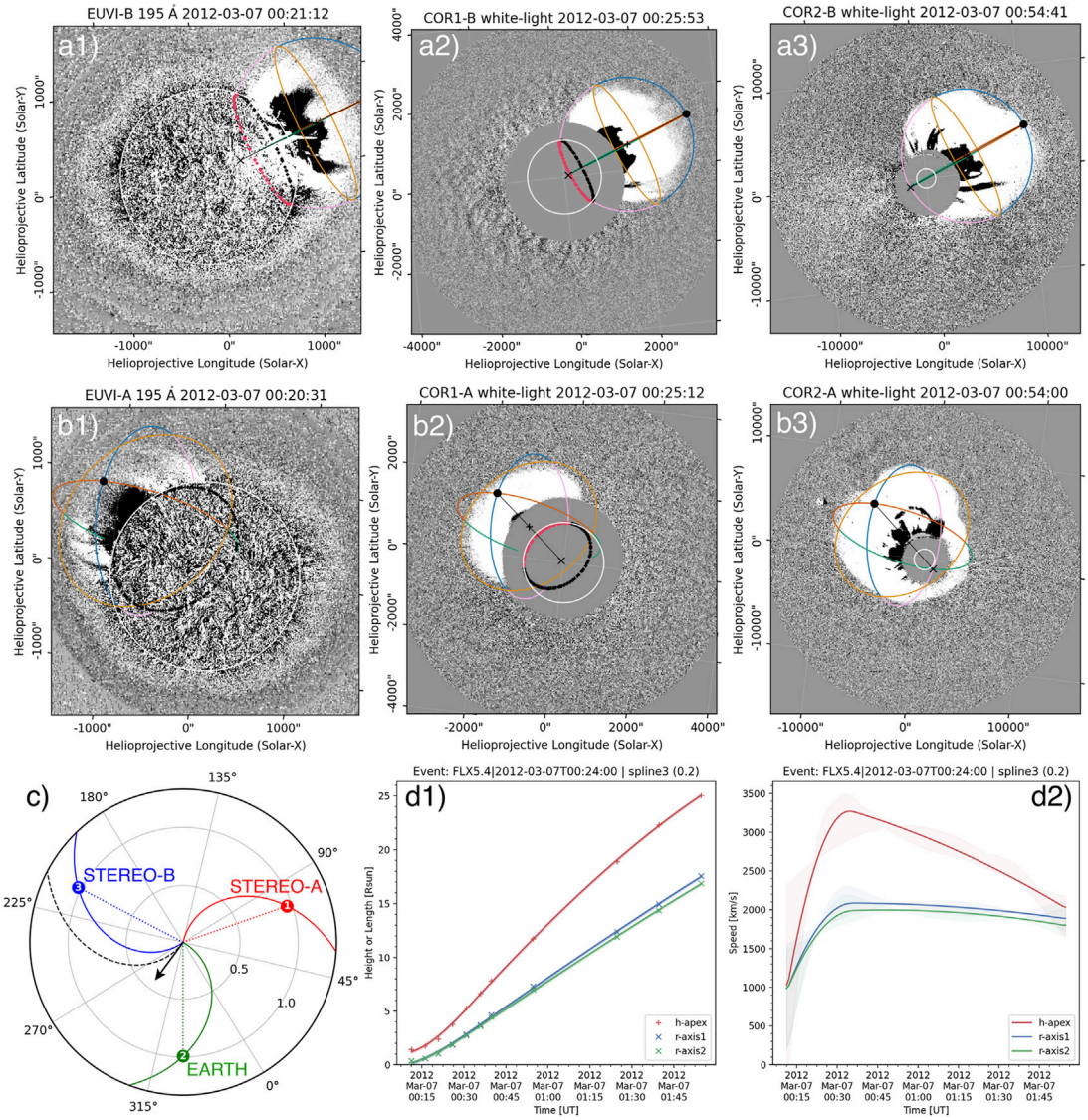


FIGURE 8
Selected results from the shock reconstruction of the 7 March 2012 solar event. Panels **(A1–A3)**, show images from STEREO-A EUVI, COR1, and COR2 where we also show the reconstructed shock wave front. Similar to panels **(B1–B3)** that show nearly simultaneous images from STEREO-B. Panel **(C)** is a view of the ecliptic plane from the ecliptic north showing the relative positions of the STEREO-A/B and Earth in Carrington coordinates, on 7 March 2012 at 00:15 UT. This ecliptic view is produced with the Solar-MACH tool. Panels **(D1, D2)**, show the height and kinematic time profile plots, respectively, of the reconstructed shock wave.

single frame fitting is uniquely stored in the pandas.DataFrame, so when the model parameters change the values in the table update automatically without registering a new record if this record already exists.

During the multiple-viewpoint, multiple-time, fitting process, PyThea provides two options to view and supervise the fitting parameters. The first enables a view of the fittings pandas.DataFrame table where the values of the geometrical model can be previewed. A single-frame fitting can be selected and loaded or deleted. When a single-frame fitting is loaded all the geometrical parameters are loaded to the sliders and the fitting can be revised. The second option enables the processing and visualization of kinematics. Various functions can be fitted in the height-time profiles of the CMEs and the shocks. The kinematic plots are displayed below the imaging panel in the application. These plots can be saved in PNG format during the fitting process. Additionally, the geometrical fittings can be downloaded as JSON files (see Section 2.7) at any time during the fitting process.

3.5 An example shock reconstruction

Using the web application of PyThea we reconstructed the solar event on 7 March 2012 and use this as an example of the product that can be produced with this software package and can lead to significant scientific discoveries as we explained in the Introduction. This solar event was associated with a powerful X5.4 class flare from AR 11429 that was located at N18°E31°. The flare started at 00:02 UT and peaked at 00:24 UT and was accompanied by a bright coronal EUV wave and an ultra-fast CME and a shock wave. The event was also associated with a major SEP event on the same day which was one of the strongest proton events of 2012 and was detected by three spacecraft separated at least 120° apart in longitude. This event has been analysed with great detail by previous studies (Kwon et al., 2014; Kouloumvakos et al., 2016) that showed that a very fast shock wave formed, capable to accelerate and release SEPs in very distant locations in the heliosphere.

In Figure 8, we show a few of the results from the shock reconstruction. Panels a) and b) show remote sensing observations from the two STEREO spacecraft. We project the reconstructed shock wave front onto the event images. For this reconstruction, there were available data from three different viewpoints, of STEREO-A/B and SOHO. The location of the spacecraft and Earth are shown in panel c) of Figure 8 where we show a view of the ecliptic plane from the ecliptic north. Panels d1) and d2), show the kinematic profiles of the shock wave. We have spline-fitted the height (length)-time measurement and then calculated the kinematic curves that are shown on panel d2), from the time derivative of the spline fit. For this calculation, we use the gradient of the height (length)-time measurements. The gradient is computed using a second-order central differences scheme in the interior points and a first-order one-side differences scheme at the boundaries. The derived kinematics agree well, within the uncertainties, with the shock speed values derived in previous studies. We find a maximum speed of ~ 3270 ±200 km/s at the shock apex, while from Figure 10 of Kwon et al. (2014) we estimate that the maximum shock speed is ~3180 km/s. The two results seem to be in good agreement considering the inherent uncertainty of the fitting process which is user dependent and the uncertainty of the final fitting of the kinematics.

4 Discussion and future prospects

In this paper, we present PyThea to the scientific community, which is a newly developed open-source Python software package that provides tools to reconstruct CMEs and shock waves. This package has been fully built in Python, with extensive use of libraries available within this language ecosystem. We showed details of the main functionality of the core software package and also presented the web application that can be used to reconstruct CMEs and shocks waves with reduced complexity, especially in data retrieval and data reduction.

The current version of PyThea is v0.6.6 and can be installed using PyPI. The source code can also be found on GitHub and Zenodo[6]. The development of the PyThea core package has been ongoing for 1 year and uninterrupted since its first release. We hope the community appreciate this effort and that the number of users who explore the capabilities of PyThea keeps growing in the nearly future. We welcome any critical review that would make this package better for the community.

A significant milestone is to reach the standards of version 1.0.0 soon. This would require a higher level of documentation, with guidelines and tutorials and enhanced documentation strings of the functions and classes used in the core package. Additionally, increased coverage and a better level of testing of the various components of the package are necessary and automation of several processes such as doc build, testing, and packaging will be needed for the stable version. Lastly, finalizing the stability of the core package will assure that the minor releases are backwards compatible and that only major releases would have breaking changes or new major features.

Further development of this package to include imaging data from the two new solar missions, PSP and SolO, is considered to be a priority. Additionally, we plan to improve various current features and include new ones. The main focus will be to ease and improve, even further, the reconstruction process and enhance the visualization part. For example, we plan to include features that would allow the user to change the base difference images, select the step of the running difference images, zoom into the images, draw a grid over the solar surface or plot the solar limb visible from Earth perspective for different observers. Further developments in the image processing would also allow us to make the fitting process semi-automatic in future releases. These and other features will be considered in a future release. Currently, no funding is associated directly with the development of the package, however, we will pursue opportunities for financial support to continue maintaining and making significant contributions to PyThea.

Data availability statement

Publicly available datasets were analyzed in this study. The data can be accessed at the Virtual Solar Observatory (http://sdac.virtualsolar.org/cgi/search). These data were processed and analyzed using Python's SunPy package (The SunPy Community et al., 2020).

6 Zenodo is a general-purpose open repository developed under the European OpenAIRE program and is powered by CERN Data Centre.

Author contributions

AK developed PyThea software package. JG, LR-G, and DP have tested PyThea's application and provided critical comments for the development and release of the package. AK prepared the first draft and all the authors involved in the preparation of the final manuscript and critically revised it. All authors have read the paper and approved its submission.

Funding

This study has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101004159 (SERPENTINE). AK acknowledges financial support from the SERPENTINE project that allowed to initiate PyThea's development and from NASA NNN06AA01C (SOSIS Phase-E) contract that allowed further improvements and update of the package to the version used for this study. LR-G acknowledges the financial support by the Spanish Ministerio de Ciencia, Innovación y Universidades FEDER/MCIU/AEI Projects ESP2017-88436-R and PID2019-104863RB-I00/AEI/10.13039/501100011033. JG and RV acknowledge the support of Academy of Finland (FORESAIL, grants 312357 and 336809). AV was supported by NASA 80NSSC19K1261 and LWS 80NSSC19K0069 grants.

Acknowledgments

This research has made use of PyThea v0.6.6, an open-source and free Python package to reconstruct the 3D structure of CMEs and shock waves (Zenodo: https://doi.org/10.5281/zenodo.5713659). This research used version 4.0.0 of the SunPy open source software package (The SunPy Community et al., 2020). The Solar-MACH tool was originally developed at Kiel University, Germany and further discussed within the ESA Heliophysics Archives USer (HAUS) group. SOHO is an international collaboration between NASA and ESA. LASCO was constructed by a consortium of institutions: NRL (United States), MPIA (Germany), LAS (France), and the University of Birmingham (UK). The SECCHI data are produced by an international consortium of the NRL, LMSAL and NASA GSFC (United States), RAL and the University of Birmingham (UK), MPS (Germany), CSL (Belgium), IOTA and IAS (France). The AIA data used here are courtesy of SDO (NASA) and the AIA consortium. We thank the AIA team for easy access to calibrated data.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Balmaceda, L. A., Vourlidas, A., Stenborg, G., and Cyr, St.O. C. (2020). On the expansion speed of coronal mass ejections: Implications for self-similar evolution. *Sol. Phys.* 295, 107. doi:10.1007/s11207-020-01672-6

Balmaceda, L. A., Vourlidas, A., Stenborg, G., and Kwon, R.-Y. (2022). The hyper-inflation stage in the coronal mass ejection formation: A missing link that connects flares, coronal mass ejections, and shocks in the low corona. *Astrophys. J.* 931, 141. doi:10.3847/1538-4357/ac695c

The SunPy Community, Barnes, W. T., Bobra, M. G., Christe, S. D., Freij, N., Hayes, L. A., et al. (2020). The sunpy project: Open source development and status of the version 1.0 core package. *Astrophys. J.* 890, 68. doi:10.3847/1538-4357/ab4f7a

Chen, P. F. (2011). Coronal mass ejections: Models and their observational basis. *Living Rev. Sol. Phys.* 8, 1. doi:10.12942/lrsp-2011-1

Domingo, V., Fleck, B., and Poland, A. I. (1995). The SOHO mission: An overview. *Sol. Phys.* 162, 1–37. doi:10.1007/BF00733425

Dresing, N., Kouloumvakos, A., Vainio, R., and Rouillard, A. (2022). On the role of coronal shocks for accelerating solar energetic electrons. *Astrophys. J. Lett.* 925, L21. doi:10.3847/2041-8213/ac4ca7

Dumbović, M., Guo, J., Temmer, M., Mays, M. L., Veronig, A., Heinemann, S. G., et al. (2019). Unusual plasma and particle signatures at mars and STEREO-A related to CME-CME interaction. *Astrophys. J.* 880, 18. doi:10.3847/1538-4357/ab27ca

Fox, N. J., Velli, M. C., Bale, S. D., Decker, R., Driesman, A., Howard, R. A., et al. (2016). The solar Probe plus mission: Humanity's first visit to our star. *Space Sci. Rev.* 204, 7–48. doi:10.1007/s11214-015-0211-6

Frassati, F., Susino, R., Mancuso, S., and Bemporad, A. (2019). Comprehensive analysis of the formation of a shock wave associated with a coronal mass ejection. *Astrophys. J.* 871, 212. doi:10.3847/1538-4357/aaf9af

Giacalone, J., Mitchell, D. G., Allen, R. C., Hill, M. E., and McNutt, J. (2020). Solar Energetic Particles Produced by a Slow Coronal Mass Ejection at ~0.25 au. *Astrophys. J. Suppl. Ser.* 246, 29. doi:10.3847/1538-4365/ab5221

Gómez-Herrero, R., Dresing, N., Klassen, A., Heber, B., Lario, D., Agueda, N., et al. (2015). Circumsolar energetic particle distribution on 2011 november 3. *Astrophys. J.* 799, 55. doi:10.1088/0004-637x/799/1/55

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Hess, P., Rouillard, A. P., Kouloumvakos, A., Liewer, P. C., Zhang, J., Dhakal, S., et al. (2020). WISPR imaging of a pristine CME. *Astrophys. J. Suppl. Ser.* 246, 25. doi:10.3847/1538-4365/ab4ff0

Howard, R. A., Moses, J. D., Vourlidas, A., Newmark, J. S., Socker, D. G., Plunkett, S. P., et al. (2008). Sun Earth connection coronal and heliospheric investigation (SECCHI). *Space Sci. Rev.* 136, 67–115. doi:10.1007/s11214-008-9341-4

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Hurlburt, N. E., Cheung, M., Schrijver, C., Chang, L., Freeland, S., Green, S., et al. (2010). An introduction to the heliophysics event Knowledgebase. *Am. Astronomical Soc. Meet. Abstr. #216* 216, 40222.

Jebaraj, I. C., Kouloumvakos, A., Magdalenic, J., Rouillard, A. P., Mann, G., Krupar, V., et al. (2021). Generation of interplanetary type II radio emission. *Astron. Astrophys.* 654, A64. doi:10.1051/0004-6361/202141695

Kaiser, M. L., Kucera, T. A., Davila, J. M., Cyr, St.O. C., Guhathakurta, M., and Christian, E. (2008). The STEREO mission: An introduction. *Space Sci. Rev.* 136, 5–16. doi:10.1007/s11214-007-9277-0

Kouloumvakos, A., Kwon, R. Y., Rodríguez-García, L., Lario, D., Dresing, N., Kilpua, E. K. J., et al. (2022). The first widespread solar energetic particle event of solar cycle 25 on 2020 November 29. Shock wave properties and the wide distribution of solar energetic particles. *Astron. Astrophys.* 660, A84. doi:10.1051/0004-6361/202142515

Kouloumvakos, A., Patsourakos, S., Nindos, A., Vourlidas, A., Anastasiadis, A., Hillaris, A., et al. (2016). Multi-viewpoint observations of a widely distributed solar energetic particle event: The role of EUV waves and white-light shock signatures. *Astrophys. J.* 821, 31. doi:10.3847/0004-637X/821/1/31

Kouloumvakos, A., Rouillard, A. P., Share, G. H., Plotnikov, I., Murphy, R., Papaioannou, A., et al. (2020a). Evidence for a coronal shock wave origin for relativistic protons producing solar gamma-rays and observed by neutron monitors at Earth. *Astrophys. J.* 893, 76. doi:10.3847/1538-4357/ab8227

Kouloumvakos, A., Rouillard, A. P., Wu, Y., Vainio, R., Vourlidas, A., Plotnikov, I., et al. (2019). Connecting the properties of coronal shock waves with those of solar energetic particles. *Astrophys. J.* 876, 80. doi:10.3847/1538-4357/ab15d7

Kouloumvakos, A., Rouillard, A., Warmuth, A., Magdalenic, J., Jebaraj, I. C., Mann, G., et al. (2021). Coronal conditions for the occurrence of type II radio bursts. *Astrophys. J.* 913, 99. doi:10.3847/1538-4357/abf435

Kouloumvakos, A., Vourlidas, A., Rouillard, A. P., Roelof, E. C., Leske, R., Pinto, R., et al. (2020b). The solar origin of particle events measured by parker solar Probe. *Astrophys. J.* 899, 107. doi:10.3847/1538-4357/aba5a1

Kwon, R.-Y., Ofman, L., Olmedo, O., Kramar, M., Davila, J. M., Thompson, B. J., et al. (2013). STEREO observations of fast magnetosonic waves in the extended solar corona associated with EIT/EUV waves. *Astrophys. J.* 766, 55. doi:10.1088/0004-637X/766/1/55

Kwon, R.-Y., and Vourlidas, A. (2017). Investigating the wave nature of the outer envelope of halo coronal mass ejections. *Astrophys. J.* 836, 246. doi:10.3847/1538-4357/aa5b92

Kwon, R.-Y., Zhang, J., and Olmedo, O. (2014). New insights into the physical nature of coronal mass ejections and associated shock waves within the framework of the three-dimensional structure. *Astrophys. J.* 794, 148. doi:10.1088/0004-637X/794/2/148

Lakhina, G. S., and Tsurutani, B. T. (2016). Geomagnetic storms: Historical perspective to modern view. *Geosci. Lett.* 3, 5. doi:10.1186/s40562-016-0037-4

Lario, D., Kwon, R. Y., Richardson, I. G., Raouafi, N. E., Thompson, B. J., von Rosenvinge, T. T., et al. (2017). The solar energetic particle event of 2010 august 14: Connectivity with the solar source inferred from multiple spacecraft observations and modeling. *Astrophys. J.* 838, 51. doi:10.3847/1538-4357/aa63e4

Lario, D., Kwon, R. Y., Vourlidas, A., Raouafi, N. E., Haggerty, D. K., Ho, G. C., et al. (2016). Longitudinal properties of a widespread solar energetic particle event on 2014 february 25: Evolution of the associated CME shock. *Astrophys. J.* 819, 72. doi:10.3847/0004-637X/819/1/72

Lario, D., Raouafi, N. E., Kwon, R. Y., Zhang, J., Gómez-Herrero, R., Dresing, N., et al. (2014). The solar energetic particle event on 2013 april 11: An investigation of its solar origin and longitudinal spread. *Astrophys. J.* 797, 8. doi:10.1088/0004-637X/797/1/8

Long, D. M., Bloomfield, D. S., Chen, P. F., Downs, C., Gallagher, P. T., Kwon, R. Y., et al. (2017). Understanding the physical nature of coronal "EIT waves". *Sol. Phys.* 292, 7. doi:10.1007/s11207-016-1030-y

Mancuso, S., Frassati, F., Bemporad, A., and Barghini, D. (2019). Three-dimensional reconstruction of CME-driven shock-streamer interaction from radio and EUV observations: A different take on the diagnostics of coronal magnetic fields. *Astron. Astrophys.* 624, L2. doi:10.1051/0004-6361/201935157

Mierla, M., Inhester, B., Antunes, A., Boursier, Y., Byrne, J. P., Colaninno, R., et al. (2010). On the 3-D reconstruction of Coronal Mass Ejections using coronagraph data. *Ann. Geophys.* 28, 203–215. doi:10.5194/angeo-28-203-2010

Morosan, D. E., Palmerio, E., Pomoell, J., Vainio, R., Palmroth, M., and Kilpua, E. K. J. (2020). Three-dimensional reconstruction of multiple particle acceleration regions during a coronal mass ejection. *Astron. Astrophys.* 635, A62. doi:10.1051/0004-6361/201937133

Müller, D., Cyr, St.O. C., Zouganelis, I., Gilbert, H. R., Marsden, R., Nieves-Chinchilla, T., et al. (2020). The Solar Orbiter mission. Science overview. *Astron. Astrophys.* 642, A1. doi:10.1051/0004-6361/202038467

Mumford, S., Freij, N., Christe, S., Ireland, J., Mayer, F., Hughitt, V., et al. (2020). Sunpy: A python package for solar physics. *J. Open Source Softw.* 5, 1832. doi:10.21105/joss.01832

Ontiveros, V., and Vourlidas, A. (2009). Quantitative measurements of coronal mass ejection-driven shocks from LASCO observations. *Astrophys. J.* 693, 267–275. doi:10.1088/0004-637X/693/1/267

Palmerio, E., Nieves-Chinchilla, T., Kilpua, E. K. J., Barnes, D., Zhukov, A. N., Jian, L. K., et al. (2021). Magnetic structure and propagation of two interacting CMEs from the Sun to saturn. *J. Geophys. Res. Space Phys.* 126, e2021JA029770. doi:10.1029/2021JA029770

Patsourakos, S., and Vourlidas, A. (2012). On the nature and genesis of EUV waves: A synthesis of observations from SOHO, STEREO, SDO, and hinode (invited review). *Sol. Phys.* 281, 187–222. doi:10.1007/s11207-012-9988-6

Pesnell, W. D., Thompson, B. J., and Chamberlin, P. C. (2012). The solar Dynamics observatory (SDO). *Sol. Phys.* 275, 3–15. doi:10.1007/s11207-011-9841-3

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., et al. (2018). The astropy project: Building an open-science project and status of the v2.0 core package. *Astron. J.* 156, 123. doi:10.3847/1538-3881/aabc4f

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., et al. (2013). Astropy: A community Python package for astronomy. *Astron. Astrophys.* 558, A33. doi:10.1051/0004-6361/201322068

Rodríguez-García, L., Gómez-Herrero, R., Zouganelis, I., Balmaceda, L., Nieves-Chinchilla, T., Dresing, N., et al. (2021). The unusual widespread solar energetic particle event on 2013 August 19. Solar origin and particle longitudinal distribution. *Astron. Astrophys.* 653, A137. doi:10.1051/0004-6361/202039960

Rodríguez-García, L., Nieves-Chinchilla, T., Gómez-Herrero, R., Zouganelis, I., Vourlidas, A., Balmaceda, L. A., et al. (2022). Evidence of a complex structure within the 2013 august 19 coronal mass ejection - radial and longitudinal evolution in the inner heliosphere. *Astron. Astrophys.* 662, A45. doi:10.1051/0004-6361/202142966

Rouillard, A. P., Plotnikov, I., Pinto, R. F., Tirole, M., Lavarra, M., Zucca, P., et al. (2016). Deriving the properties of coronal pressure fronts in 3D: Application to the 2012 may 17 ground level enhancement. *Astrophys. J.* 833, 45. doi:10.3847/1538-4357/833/1/45

Rouillard, A. P., Poirier, N., Lavarra, M., Bourdelle, A., Dalmasse, K., Kouloumvakos, A., et al. (2020). Modeling the early evolution of a slow coronal mass ejection imaged by the parker solar Probe. *Astrophys. J. Suppl. Ser.* 246, 72. doi:10.3847/1538-4365/ab6610

Rouillard, A. P., Sheeley, N. R., Tylka, A., Vourlidas, A., Ng, C. K., Rakowski, C., et al. (2012). The longitudinal properties of a solar energetic particle event investigated using modern solar imaging. *Astrophys. J.* 752, 44. doi:10.1088/0004-637X/752/1/44

Scolini, C., Chané, E., Temmer, M., Kilpua, E. K. J., Dissauer, K., Veronig, A. M., et al. (2020). CME-CME interactions as sources of CME geoeffectiveness: The formation of the complex ejecta and intense geomagnetic storm in 2017 early september. *Astrophys. J. Suppl. Ser.* 247, 21. doi:10.3847/1538-4365/ab6216

Temmer, M. (2021). Space weather: The solar perspective. *Living Rev. Sol. Phys.* 18, 4. doi:10.1007/s41116-021-00030-3

Thernisien, A. F. R., Howard, R. A., and Vourlidas, A. (2006). Modeling of flux rope coronal mass ejections. *Astrophys. J.* 652, 763–773. doi:10.1086/508254

Thernisien, A. (2011). Implementation of the graduated cylindrical shell model for the three-dimensional reconstruction of coronal mass ejections. *Astrophys. J. Suppl. Ser.* 194, 33. doi:10.1088/0067-0049/194/2/33

Thompson, W. T. (2006). Coordinate systems for solar image data. *Astron. Astrophys.* 449, 791–803. doi:10.1051/0004-6361:20054262

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2

Webb, D. F., and Howard, T. A. (2012). Coronal mass ejections: Observations. *Living Rev. Sol. Phys.* 9, 3. doi:10.12942/lrsp-2012-3

Wood, B. E., Wu, C. C., Howard, R. A., Socker, D. G., and Rouillard, A. P. (2011). Empirical reconstruction and numerical modeling of the first geoeffective coronal mass ejection of solar cycle 24. *Astrophys. J.* 729, 70. doi:10.1088/0004-637X/729/1/70

Zhu, B., Liu, Y. D., Kwon, R.-Y., and Wang, R. (2018). Investigation of energetic particle release using multi-point imaging and *in situ* observations. *Astrophys. J.* 865, 138. doi:10.3847/1538-4357/aada80

Zouganelis, I., De Groof, A., Walsh, A. P., Williams, D. R., Müller, D., St Cyr, O. C., et al. (2020). The Solar Orbiter Science Activity Plan. Translating solar and heliospheric physics questions into action. *Astron. Astrophys.* 642, A3. doi:10.1051/0004-6361/202038445

Zucca, P., Morosan, D. E., Rouillard, A. P., Fallows, R., Gallagher, P. T., Magdalenic, J., et al. (2018). Shock location and CME 3D reconstruction of a solar type II radio burst with LOFAR. *Astron. Astrophys.* 615, A89. doi:10.1051/0004-6361/201732308

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 26 September 2022
DOI 10.3389/fspas.2022.1009450

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Aleksandar Valjarević,
University of Belgrade, Serbia
Virginia Kalb,
National Aeronautics and Space Administration (NASA), United States

*CORRESPONDENCE
M. Shumko,
msshumko@gmail.com

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 01 August 2022
ACCEPTED 08 September 2022
PUBLISHED 26 September 2022

CITATION
Shumko M, Chaddock D, Gallardo-Lacourt B, Donovan E, Spanswick EL, Halford AJ, Thompson I and Murphy KR (2022), AuroraX, PyAuroraX, and aurora-asi-lib: A user-friendly auroral all-sky imager analysis framework.
Front. Astron. Space Sci. 9:1009450.
doi: 10.3389/fspas.2022.1009450

COPYRIGHT
© 2022 Shumko, Chaddock, Gallardo-Lacourt, Donovan, Spanswick, Halford, Thompson and Murphy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

AuroraX, PyAuroraX, and aurora-asi-lib: A user-friendly auroral all-sky imager analysis framework

M. Shumko[1,2]*, D. Chaddock[3], B. Gallardo-Lacourt[1,4], E. Donovan[3], E. L. Spanswick[3], A. J. Halford[1], I. Thompson and K. R. Murphy

[1]NASA's Goddard Space Flight Center, Greenbelt, MD, United States, [2]Department of Astronomy, University of Maryland, College Park, MD, United States, [3]University of Calgary, Calgary, AB, Canada, [4]Department of Physics, The Catholic University of America, Washington, D.C., United States

Within the context of the Heliophysics System Observatory, optical images of the aurora are emerging as an important resource for exploring multi-scale geospace processes. This capability has never been more critical as we are on the cusp of a new era of geospace research, by which we mean studying the overall system as a *system of systems*. Historically, the patchwork of ground-based instrumentation has required customized solutions for accessing data, assessing data relevance, and then ultimately using each individual network alongside other assets. Here we introduce a new and comprehensive approach for data discovery and utilization for one type of data, namely auroral images. The AuroraX project (https://aurorax.space/) is a cyberinfrastructure platform for the discovery of scientific opportunities with access to optical auroral data. The program has broad objectives, so we focus on one key thread. In particular, we focus on describing the AuroraX platform and its API and web-based tools for all-sky imager (ASI) data. As a practical example, we demonstrate how to identify conjunctions using the AuroraX conjunction finder or PyAuroraX, a Python library that interfaces with the AuroraX platform. We then demonstrate how aurora-asi-lib, a Python library for interacting with and analyzing high-resolution ASI data, can be used for detailed conjunction analysis on a personal computer. Together, these tools enable a rapid and streamlined end-to-end exploration of auroral data.

KEYWORDS

THEMIS, REGO, aurora, all-sky-imager, python, conjunction

1 Introduction

In the domain of space physics (also known as Heliophysics and geospace research) it is becoming clear that we need observations of processes that span a range of space and time scales: from kinetic and fast, to global and relatively slow. Historically, informing our knowledge of the small and fast scales has led to extensive focus on *in-situ point*

measurements in space, and high time and space resolution measurements on the ground, organized around, for example, incoherent scatter radars. For the global picture, our community has looked to, for example, system-level observations provided by imagers such as those carried by NASA's IMAGE spacecraft (Burch, 2000) complimented by information from near-global networks of ground-based instruments such as magnetometers and high frequency radars.

NASA's Time History of Events and Macroscale Interactions during Substorms (THEMIS) mission was launched in 2007 with its prime mission objective being identification of the instability responsible for substorm onset (e.g., Angelopoulos et al., 2008). The mission science necessitated relatively high space and time information about the aurora across a large swath of magnetic local time. This was addressed by including a continent-wide network of All-Sky Imagers (ASIs) as part of the overall mission. In terms of space and time scales, THEMIS-ASI delivered a fundamentally new view of the aurora, and by extension, of geospace dynamics (e.g., Donovan et al., 2006b; Mende et al., 2009; Jones et al., 2013). The key new thing that THEMIS-ASI brought forward was the ability to track the spatio-temporal evolution of small-scale structures that are organized over large distances, providing us our first real look at the mesoscales that Dan Baker once referred to as the *missing middle*.

ASIs and other scientific auroral imaging systems have been around for many decades and have contributed enormously to auroral science. Examples include the very concept of the substorm, introduced by Akasofu (1964), and several descriptions of new phenomena that highlight the tight connections between the magnetosphere and ionosphere, understanding of how the aforementioned mesocales contribute to geospace dynamics at the system level (Nishimura et al., 2010), and unravelling the mystery of the newly discovered STEVE phenomenon (MacDonald et al., 2018). ASIs are only part of our arsenal of tools for observing the aurora. Meridian Scanning photometers (MSPs), such as those operated as part of CANOPUS, provide high quality quantitative measurements of auroral intensities at multiple wavelengths along a scan plan (e.g., Rostoker et al., 1995). As well, global auroral imagers, such as the Wideband Imaging Camera on IMAGE (Burch, 2000), provide a true global picture of geospace as projected along magnetic field lines onto the ionosphere.

Auroral observations provide us what is to date our best view of the so-called *missing middle*. Because of our increasing collective interest of geospace at the system level, and the study of geospace as a system of systems, the *missing middle* is becoming ever more important. Consequently, historical, contemporary, and future auroral observations are increasingly important. However, the data landscape vis-a-viz auroral observations is highly challenging. Instruments have been operated by a large number of groups, in a large number of modes, and the resulting data is available and discoverable to an extent that is not ideal. While the overall data is large in volume, it is this heterogeneity and locations that are really its challenge. The various data formats ultimately stifles system-level research. Similar data formats are important to research groups who, for example, are developing machine learning models to classify auroral phenomena (e.g., Clausen and Nickisch, 2018; Zhong et al., 2020).

In this paper, we introduce the AuroraX project which aims to overcome the above issues by providing key tools to aid auroral researchers. The first tool is the AuroraX website and Virtual Observatory. This website provides various interfaces for quickly visualizing summary data (i.e., keograms, movies, etc.), determining what imagers operated at a given time, and search for conjunctions between numerous ground- and space-based instruments. The second tool is PyAuroraX, a Python library to programmatically interact with the AuroraX platform. The third tool is aurora-asi-lib, a Python all-sky imager library that provides functions to download, load, analyze, and visualize the THEMIS and the Red-line Emission Geospace Observatory (REGO) ASI data on a personal computer. Since each ASI array has unique data quality and caveats that often require visual inspection, these tools analyze the ASI data as-is. We leave the task of assessing the imager-specific data quality to the user (Starck and Murtagh, 2007).

2 AuroraX

The motivating driver behind the development of the AuroraX cyberinfrastructure project is to enable data mining and analysis of existing and future arrays of auroral data, which is accomplished by developing a set of tools specifically designed for exploring auroral observations. With these tools, the project would enable key scientific discoveries and enhance the benefits of the world's auroral instrumentation. This is being accomplished with the development of key systems/standards for uniform metadata generation and search, image content analysis, interfaces to leading international tools, and a community involvement that includes more than 80% of the world's data providers.

The AuroraX website, located at https://aurorax.space, is designed to be the first place to start your auroral analysis. The website currently provides interfaces for: performing conjunction searches, exploring summary data such as keograms and movies, viewing data availability, and documentation about the platform including guides for using the API and client libraries (e.g., PyAuroraX). In the following sections, we explained some of AuroraX's capabilities in detail.

2.1 Data repository

Fundamentally, AuroraX processes a rich database of metadata, which we refer to as the *data repository*. Indeed, AuroraX does not contain any raw data, only metadata and various summary data products. Our goal with this approach is to ease concerns with data ownership and stewardship, which may cause hesitancy to have data publicly available and searchable by AuroraX. In other words, AuroraX is meant to be a centralized data exploration and event discovery platform, and not a raw data repository. This architecture keeps the data repository slim and optimized for the search engine.

The metadata in the AuroraX data repository is currently organized into two categories:

- *Ephemeris* records: Provide location and operational information for a given ground- or space-based instrument.
- *Data product* records: Describe keograms or other summary products (no images are stored in the database, only URLs which are used as unique identifiers).

Ephemeris data are 1-min location records when a ground- or space-based instrument collected data. This allows applications, such as the search engine, to return more useful query results—ones where raw data exists and can be further evaluated by auroral researchers. On the other hand, the *data product* category consists of records describing summary data in the form of, for example, keograms and timelapse movies. These records are accessed via a unique website URL where the data product lives, allowing this data to be served by each organization that produces ASI data.

Both *ephemeris* and *data product* records can contain any number of arbitrary metadata fields (metadata about metadata) which can be used by the search engine to assist with further levels of filtering and data discovery.

2.2 Search engine

The THEMIS ASI array has generated more than 100 terabytes of data since becoming operational in 2005, with the instruments continuing to generate new data every day. Other ASI arrays, such as REGO and Transition Region Explorer (TREx), produce a combined >100 terabytes per year. Even for the experienced scientists, sifting through this large data volume, in search for isolated times with scientific importance, can be time consuming. This process of "event discovery" can be simplified and streamlined by leveraging a database of metadata describing auroral data and its optical content. By combining AuroraX's *data repository* with a search algorithm, we are able to provide the scientific community with a procedure to significantly reduce the amount of time spent searching through these datasets for auroral events.

One of AuroraX's search engine functions is the Conjunction Search. This function is designed to quickly provide periods of time for which two or more ground- or space-based instruments were operating, and were magnetically conjugate. A conjunction search engine with the ability to filter by ASI metadata is the key differentiating factor of AuroraX. As a platform that is built specifically for auroral data, we tailored the search algorithm to consider pertinent information about the instruments and build tools focused on maximizing their scientific contributions.

Conjunction searches can be performed using the AuroraX Conjunction Search webpage (https://aurorax.space/conjunctionSearch/standard), the AuroraX API, or client libraries like PyAuroraX and IDL-AuroraX (for the Python and IDL programming languages, respectively). Figure 1A shows the Conjunction Search web interface with the red rectangle highlighting a series of dropdown menus and filter boxes elements that customize a conjunction search. These include specifying the start and end time, ground instruments and/or spacecraft to find conjunctions between, maximum distance between the instruments (kilometers between magnetic footprints), and conjunction type. Searches can be further refined by using a customizable set of filters on the metadata in the AuroraX *Data Repository*. These filters are very flexible and easy to adjust for each ASI array or spacecraft instrument. Some examples include instrument operating mode, quality flags, and predicted auroral image content based on machine learning models. To see more information about a conjunction, clicking on the *Open* button in Figure 1A leads to a detailed view about a conjunction that we show in Figure 1B. The Conjunction Search also provides pre-loaded searches that serve as examples. One of the examples finds all conjunctions, defined as a <500 km footprint separation, between any THEMIS ASI and any THEMIS spacecraft, when the machine learning model predicted amorphous pulsating aurora with $>95\%$ confidence (Grono et al., 2017).

2.3 Virtual observatory

Besides the conjunction search, AuroraX allows users to easily browse through the summary data. The AuroraX Virtual Observatory provides interactive visualizations and data browsing interfaces to quickly navigate the vast amount of auroral data available in the platform. These interfaces are designed for browsing through the *data repository* in a simple and efficient manner. AuroraX currently has two components to the Virtual Observatory: the Keogramist, and the Event Explorer.

As the name implies, the Keogramist (https://aurorax.space/keogramist) visualizes keograms—a highly compressed summary product for quickly analyzing ASI data. For the unfamiliar reader, a keogram corresponds to a time series representation of the luminosity along a single meridian. Typically, they are assembled by looping over every image and taking a vertical slice

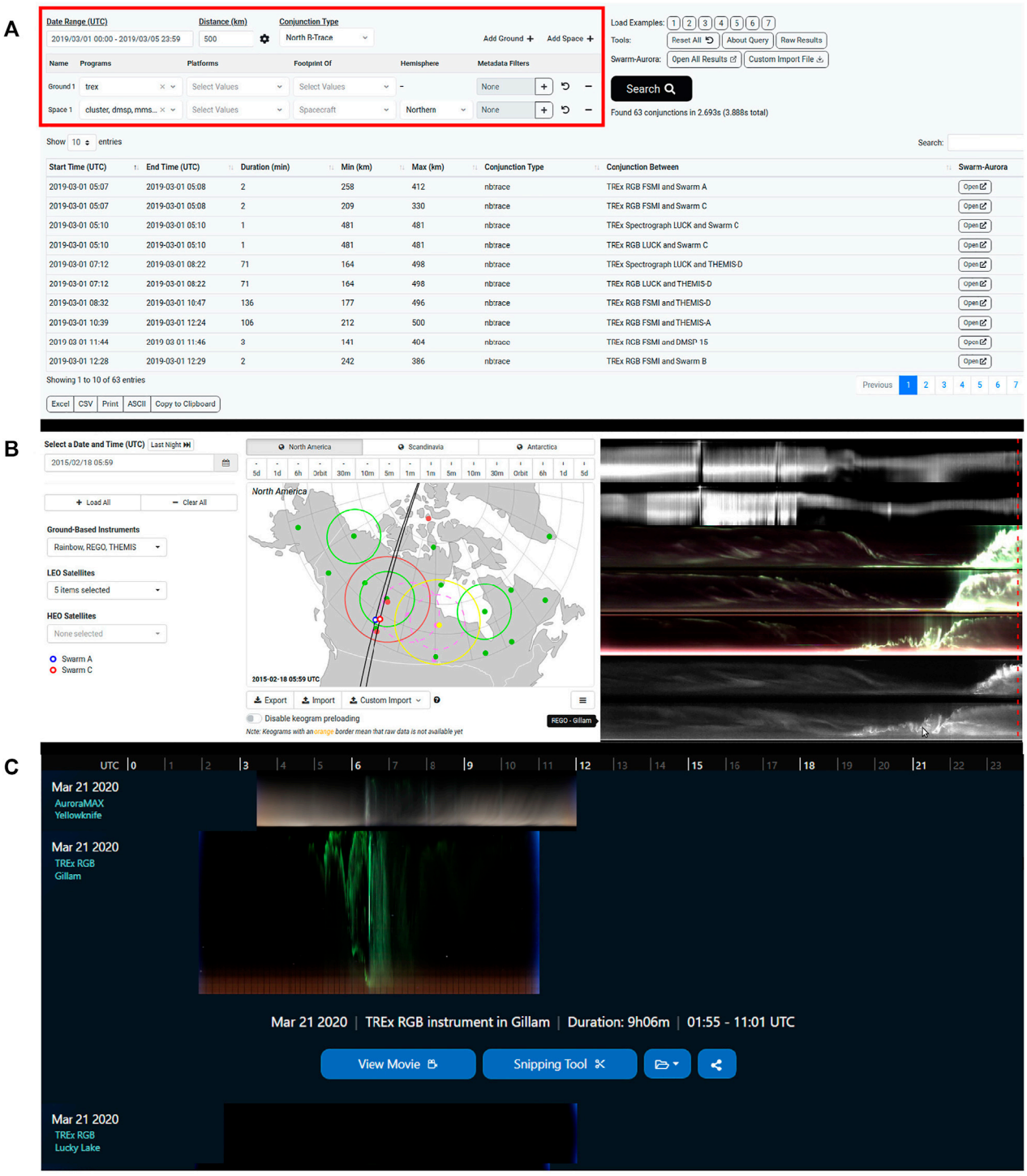


FIGURE 1
The https://aurorax.space/website. Panel **(A)** shows the conjunction finder with the customizable elements highlighted with the red rectangle. Panel **(B)** shows a detailed view of one of the conjunctions. Lastly, Panel **(C)** shows the Keogramist for three TREx ASIs on 21 March 2020. The Gillam keogram is expanded to show further options.

through the center of the image (or through a custom path such a path of a satellite). Objects in the sky such as auroral arcs, pulsating aurora, substorms, clouds, the moon, etc. have unique keogram signatures that allow for a quick interpretation. Keogramist presents keograms from any number of ASI instruments in the AuroraX data repository in a compact and visually-appealing interface. Figure 1C shows an example of the TREx keograms from 21 March 2020. Similarly, when a user identifies a day of interesting auroral activity, they can quickly view additional summary data such as timelapse movies. Keogramist also allows the user to filter the keograms displayed based on the *data product* record metadata fields in the data repository. This could be as simple as the green-channel content in an RGB-based ASI, or more complex such as classifications derived from a machine learning model (e.g., pulsating aurora in the field-of-view).

Equally useful, the second component of the Virtual Observatory is the Event Explorer (https://aurorax.space/eventExplorer). Although this component is still under development, it will allow users to see a 3D visualization of AuroraX ephemeris data with ground-based auroral images projected onto an interactive globe. This tool is designed to assist with visualizing auroral data and evaluating possible conjunctions using a more interactive and global interface. The auroral images are mapped to a 1024 × 512 grid covering − 180° to 180° longitude, and − 90° to 90° latitude (corresponding to 0.33° longitude and 0.35° latitude resolution), and visualized onto the globe. The grid format was first developed by NOAA and used as part of their 30-min auroral prediction OVATION model outputs (Newell et al., 2010; Machol et al., 2012). AuroraX has adopted this grid format to provide a global view of summary ASI data alongside representations of spacecraft geographic positions and magnetic footprints, provided by SSCWeb.

The Swarm-Aurora project was designed to facilitate and drive the use of Swarm (Friis-Christensen et al., 2006) in auroral science, and push Swarm beyond its primary mission objective to become a key instrument in auroral science research (https://swarm-aurora.com). In addition to AuroraX's Virtual Observatory components, the continued development of the Swarm-Aurora website has become a part of AuroraX's priorities. One recent improvement has been the integration of AuroraX with Swarm-Aurora, allowing users to browse Swarm-Aurora using the AuroraX Conjunction Search results. In fact, the example conjunction search query shown in Figures 1A,B was made by Swarm-Aurora. Lastly, architectural design changes were made to Swarm-Aurora to enhance the experience for users around the world, specifically optimizing loading times. Swarm-Aurora now operates on commercial cloud infrastructure in four regions; one in Calgary, two in the United States, and one in Europe. Adding more regions is trivial and can be done as the user base grows and are needed.

2.4 PyAuroraX

An important part of any platform like AuroraX is to allow users to leverage the data for use in their own scientific analyses and applications. To assist with these tasks, we developed software to interact with AuroraX using only a few lines of code. This is enabled by the AuroraX API and subsequent client libraries maintained by the project, one of which is PyAuroraX (https://github.com/aurorax-space/pyaurorax).

PyAuroraX allows users to interact with the AuroraX API and perform *conjunction*, *ephemeris*, and *data product* searches using Python. For example, the pyaurorax. conjunctions.search () function can be used to search AuroraX for conjunctions in the same way as the AuroraX Conjunction Search website. The below Python code shows how to perform a simple search, asking AuroraX to find all conjunctions between several instruments from the THEMIS ASI array and any Swarm spacecraft.

```
import pyaurorax
import datetime

# define search parameters
start = datetime.datetime(2019, 1, 1, 0, 0, 0)
end = datetime.datetime(2019, 1, 3, 23, 59, 59)
ground_params = [
    {
        "programs": ["themis-asi"],
        "platforms": ["fort smith", "gillam"],
    }
]
space_params = [
    {
        "programs": ["swarm"],
        "hemisphere": ["northern"],
    }
]
distance = 500

# perform conjunction search
s = pyaurorax.conjunctions.search(start=start,
                                  end=end,
                                  distance=distance,
                                  ground=ground_params,
                                  space=space_params,
                                  verbose=True)
```

Users can also retrieve other information from AuroraX such as data sources, *ephemeris* records, *data product* records, and data availability. All functions in PyAuroraX are also available for use with IDL programs by using the IDL-AuroraX library (https://github.com/aurorax-space/idl-aurorax).

Documentation and examples are a key part of helping new users learn what is possible and provides a reference for more experienced users. To assist with this and ease the learning curve, AuroraX has developed a documentation website, available at https://docs.aurorax.space, with technical details about the platform, the metadata in it, and the various applications and tools available for use. Extensive examples and code snippets are available in the

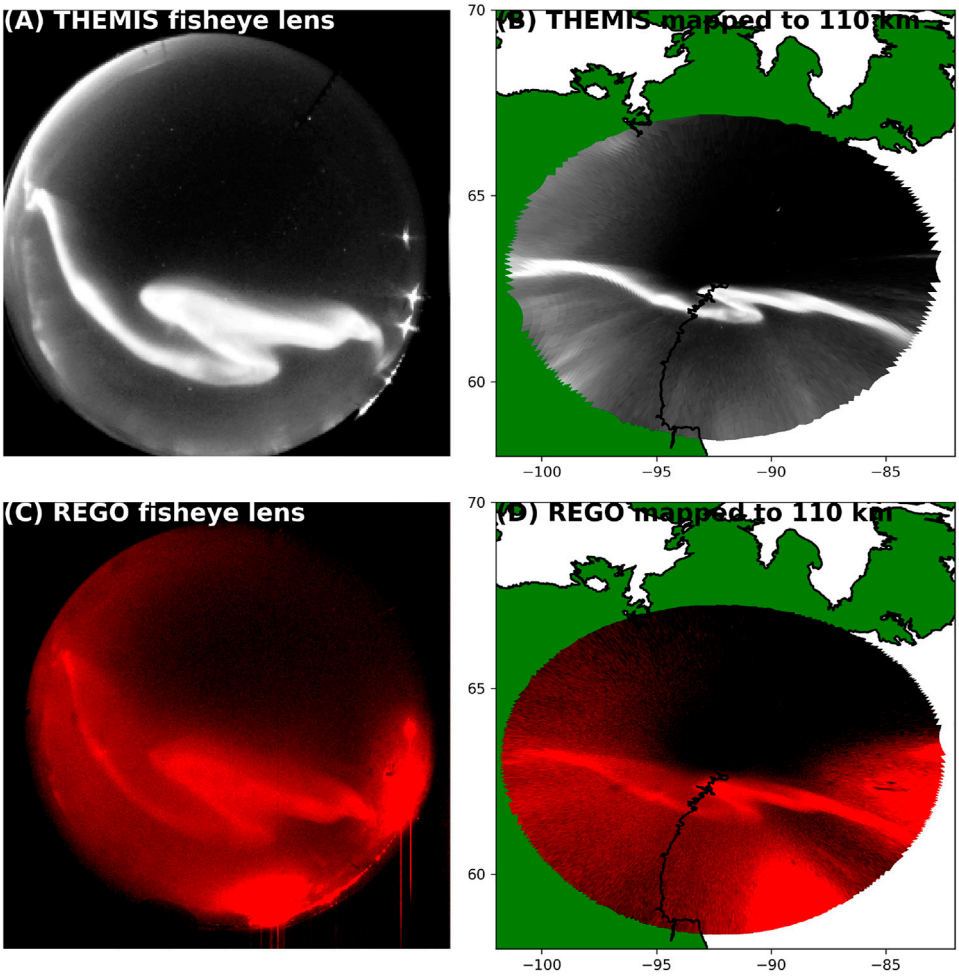


FIGURE 2
An auroral arc observed simultaneously by the REGO and THEMIS imagers at Rankin Inlet, Canada. Panels **(A)** and **(C)** show the fisheye lens view, while panels **(B)** and **(D)** show the same images projected to the 110 km assumed aurora emission altitude. Only the pixels with elevation $>10°$ are plotted.

"Developer Zone" to provide quick and simple uses of key programmatic tasks. The source code for this website is also available on Github alongside other open-source codebases within the AuroraX project (https://github.com/aurorax-space/docs).

3 Analyzing high-resolution ASI data using aurora-asi-lib

The final component of AuroraX is aurora-asi-lib, henceforth referred to, and imported as, asilib. It enables researchers to apply common data analysis tasks to the THEMIS and REGO ASI image data on a personal computer. Here we overview the main functions, while the online documentation at https://aurora-asi-lib.readthedocs.io/contains more examples, a tutorial, and a thorough API reference.

As we tour a few asilib functions, keep in mind that asilib is designed to manage the lower-level tasks. For example, if you want to load the image data via asilib.load_image(), asilib will attempt to download the 1-h Common Data Format (CDF) data if it is not already saved on your computer. Likewise, if you call asilib.plot_keogram(), it will automatically load (and download if necessary) the ASI data before plotting it. For reference, Supplementary Figures S2–S4 were made using the code in a Jupyter Notebook that is provided as Supplementary Material in both the ipynb and pdf formats.

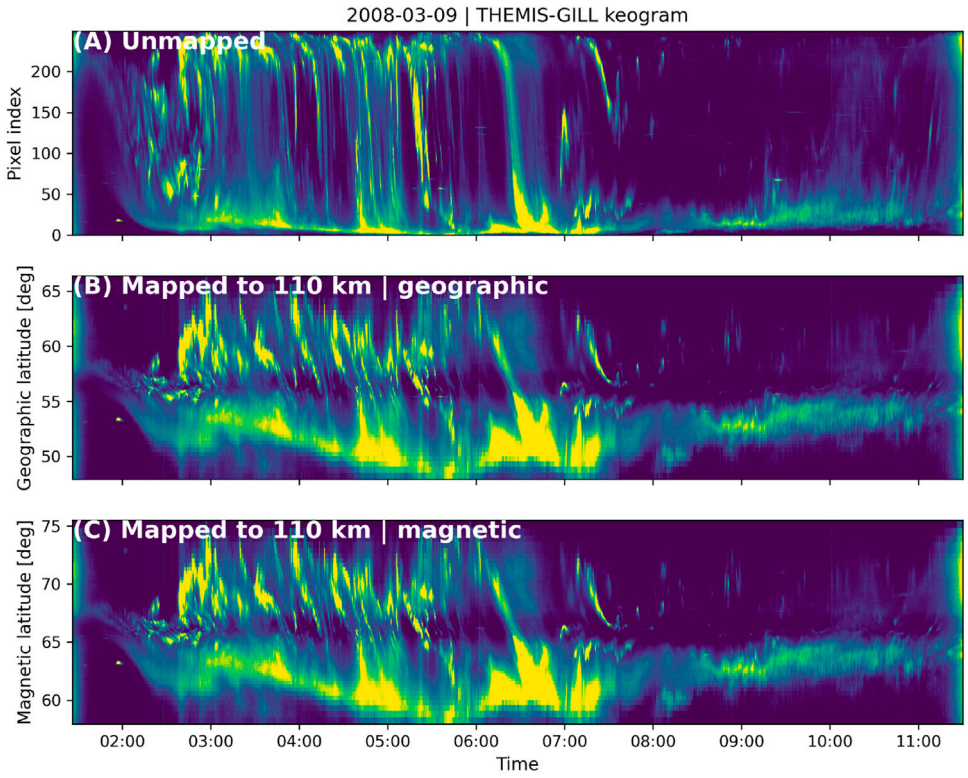


FIGURE 3
A full-night keogram showing the dynamic aurora observed at Gillam, Canada on 9 March 2008. Panel **(A)** shows the unmapped keogram with the pixel index vertical axis, panel **(B)** shows the geographic latitude of the pixels mapped to 110 km altitude. Lastly, panel **(C)** shows the corresponding magnetic latitudes.

3.1 Plotting single images

One common way to visualize all-sky images is with asilib.plot_fisheye(). It plots the raw ASI images oriented with North at the top and East to the right of each image. The term fisheye comes from the fisheye lens that expands the imager's field of view to nearly 180°. Figures 2A,C show an example of an auroral arc observed concurrently by the THEMIS and REGO ASIs stationed at Rankin Inlet (RANK). By default the color map is automatically chosen: black-to-white for THEMIS and black-to-red for REGO. The default color scale is dynamically calculated using percentile logic described in the documentation.

The other common way to visualize images is by projecting the fisheye image onto a geographic map using asilib.plot_map(). asilib uses the skymap files to map each pixel's vertices to a (latitude, longitude) point at an aurora emission altitude, typically assumed 110 km for THEMIS and 230 km for REGO (Donovan et al., 2006a; Liang et al., 2016). Figures 2B,D show the fisheye images mapped to 110 km altitude. By default, pixels that look at elevation $< 10°$ are not mapped due to nearby obstructions and the stretching of pixels closest to the horizon. And lastly, asilib.make_map() provides a default geographic map to project the images onto.

3.2 Keograms

You can make a keogram using the asilib. plot_keogram () function that takes an optional map_alt keyword argument. If map_alt is not provided, the keogram's vertical axis is pixel index, as we show in Figure 3A. If a valid map altitude is provided, the vertical axis is geographic latitude as we show in Figure 3B. Lastly, by providing map_alt and setting aacgm=True, the vertical axis becomes magnetic latitude in the Altitude-adjusted corrected geomagnetic coordinate system (AACGM) (Shepherd, 2014). The latitude transformation between Figure 3A and Figure 3B is substantial—the low elevation pixels observe much wider regions of latitude, compared to the pixels at higher elevations.

3.3 Animating images

asilib allows you to easily animate the fisheye and mapped images using asilib.animate_fisheye() and asilib.animate_map(). It first saves png images in aasilib.config['ASI_DATA_DIR']/animations/images/folder, and then animates them using the ffmpeg library (FFmpeg Developers, 2022). Supplementary

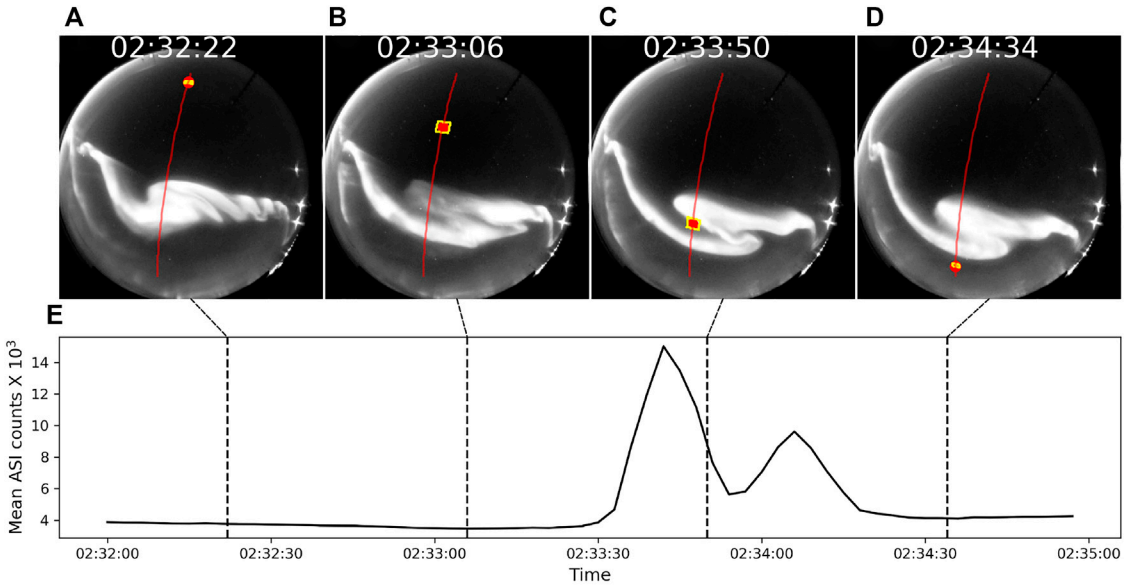


FIGURE 4
A conjunction montage of Supplementary Video S2. Panels **(A–D)** shows the auroral arc evolution and the satellite's location. The red line is the satellite track and the red dot is its instantaneous position. The yellow quadrilateral bounds the pixels inside the 20 × 20 km area surrounding the satellite's 110 km altitude footprint. Lastly, panel **(E)** shows the mean auroral intensity, as a function of time, and inside the yellow quadrilaterals. When the satellite passed through the arc between 2:33:30 and 2:34:15, the mean auroral intensity correspondingly intensified.

Video S1 in the supporting document shows an example animation of auroral streamers.

Animating just the images is somewhat limiting. Thus, asilib also includesasilib.animate_fisheye_generator() and asilib.animate_map_generator() (which are technically coroutines) to allow the user to modify the animations as they are made. This is useful, for example, if you need to draw the satellite's location in each image.

3.4 Conjunction analysis tools

Currently, asilib provides three functions that are useful for analyzing conjunctions: asilib.lla2footprint (), asilib.lla2azel(), and asilib.equal_area().

asilib.lla2footprint() uses IRBEM-Lib [Boscher et al. (2012); requires a separate installation and compilation of the Fortran source code] to trace a satellite's position, in geographic (latitude, longitude, altitude) (LLA) coordinates, along a magnetic field line. This field line is defined using one of the magnetic field models that are supported by IRBEM. The primary use of this function is to map a satellite's location from, for example, 500 km altitude, to its magnetic footprint at the assumed auroral emission altitude (e.g., 110 km for THEMIS or 230 km for REGO as previously mentioned).

The next function is asilib.lla2azel(). This function maps the satellite's footprint location, in LLA coordinates, to the ASI's (azimuth, elevation) coordinates (AzEl) using the skymap files. This function returns both the AzEl coordinates as well as the corresponding pixel indices.

Lastly, asilib.equal_area() calculates a mask of pixels inside an auroral emission area—useful to calculate the mean auroral intensity (or another statistical quantity) in a physical area in the sky. The mask contains 1s inside of the area and numpy.nan outside of it. You then multiply the image with the mask: the pixel intensities outside of the area are then numpy.nan and unchanged inside the area. We chose to use numpy.nan to ensure that the mean of the intensity is correctly applied—it will fail if you call numpy.mean(image*mask), but numpy. nanmean (image*mask) will ignore NaNs and correctly calculate the mean intensity inside the area.

3.5 Analyzing a conjunction

In this example we combine the aforementioned analysis functions to calculate the mean auroral intensity surrounding the footprint of an artificial satellite during a conjunction with a THEMIS ASI. This satellite orbits at a 500-km altitude low Earth orbit. We will ultimately calculate the mean ASI intensity in a 20 × 20 km area at a 110 km altitude and animate the conjunction. Using an artificial satellite allows us to clearly exemplify how any satellite's footprint could be easily analyzed by asilib.

For this example we use the satellite's location in LLA coordinates with time stamps that line up with the ASI times. In reality, the satellite and ASI time stamps are unlikely to line up, so you'll need to align the satellite's and ASI's time stamps.

Our analysis consists of three main steps:

1) Trace the satellite's position along the magnetic field line to 110 km using asilib.lla2footprint().
2) Locate the satellite's footprint in the imager's field of view (azimuth and elevation coordinates) using asilib.lla2azel().
3) Calculate the auroral intensity surrounding the satellite's footprint. We create a 20 × 20 km area mask using asilib.equal_area() and use it to calculate the mean ASI intensity as a function of time (and satellite position).

These steps are implemented in the "Figure 4" section of the asilib_figures notebook.

Supplementary Video S2 shows the result of this conjunction analysis and Figure 4 shows a four-frame montage summarizing the animation. Figures 4A–D show the fisheye lens images at the annotated time stamps. The satellite's footprint path is represented by the red line and the instantaneous footprint by the red dot. The yellow areas show the 20 × 20 km area around the footprint. And lastly, Figure 4E shows the mean auroral intensity time series—clearly showing the auroral arc intensity between 2: 33:30 and 2:34:15.

4 Quality assurance

We developed AuroraX, PyAuoraX, and aurora-asi-lib with usability, reliability, and maintainability at the forefront. Documentation is critically important to the survival and usability of software. The AuroraX documentation is hosted at https://docs.aurorax.space/and the asilib documentation at https://aurora-asi-lib.readthedocs.io/. There you will find installation instructions, examples, comprehensive tutorials, and API references.

The source code for AuroraX, PyAuroraX, and asilib is open source and hosted on GitHub. The two Python libraries are also cataloged on Zenodo and can be installed from the Python Packaging Index (PyPI; install using the *pip* command). On GitHub you can submit an Issue for bugs or feature requests, and contribute with a Pull Request.

To ensure code stability, the codebases for both Python libraries include tests suites that you can run locally and are automatically executed using GitHub Actions every time a change is pushed to the repository. These comprehensive tests check and warn of any software bugs or changes in function behaviour over the course of further development and maintenance.

5 Conclusion

The AuroraX website, PyAuroraX, and aurora-asi-lib tools provide the auroral science community with a simple and robust set of analysis tools to enable system-level auroral science. As we demonstrated, these tools provide an end-to-end analysis solution for using auroral data. We described one such solution: to identify and analyze conjunctions. We showed how you can use the AuroraX Conjunction Search website or PyAuroraX to identify and filter conjunctions between a number of ASIs and spacecraft. We then use aurora-asi-lib to quantify the auroral intensity at the satellite's footprint during a conjunction. This example is just one way that AuroraX can help you quickly sift through an immense volume of ASI data to uncover new physics in a significantly less amount of time than was previously possible.

In the near future we will expand AuroraX's data repository by including more ASI arrays, satellites, and informative metadata filters. Furthermore, TREx and other ASIs will be added to aurora-asi-lib by the current development team and the community. The continued success and usability of AuroraX depends on community contributions. We encourage an open science approach, and look forward to working more broadly within the auroral research community.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary Material, further inquiries can be directed to the corresponding author.

Author contributions

ED, ES, and DC designed and developed the AuroraX platform and PyAuroraX. AH, KM, and IT assisted MS with developing aurora-asi-lib. MS, BG-L, DC, and ED wrote the manuscript. ED, ES, and DC provided the THEMIS and REGO ASI data and expertise. All authors contributed to manuscript revision, read, and approved the submitted version

Funding

MS and BG-L acknowledge the support provided by the NASA Postdoctoral Program at the NASA's Goddard Space Flight Center, administered by Oak Ridge Associated Universities under contract with NASA. BG-L was also supported by the NASA competed Internal Scientist Funding Model (ISFM) on Mesoscale Magnetospheric Dynamic. AH was supported in part by the Goddard ISFM which funds the Space Precipitation Impacts team with grant HISFM21. ED, ES, and DC acknowledge the AuroraX funding sources: the Canadian

Foundation for Innovation (CFI), the Province of Alberta, the Technical University of Denmark (DTU, Swarm DISC Program), and the Canadian Space Agency.

Acknowledgments

The authors acknowledge all of the scientists, technicians, and engineers who designed, built, and maintain these ASI arrays. The UCalgary team acknowledges the AuroraX, REGO ASI and THEMIS ASI funding agencies and thanks them for their continued support. The UCalgary team also acknowledges the University of Alberta and RGP Consulting for their work on AuroraX.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The reviewer VK declared a shared affiliation with the authors MS, BG-L, and AH to the handling editor at the time of review.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fspas.2022.1009450/full#supplementary-material

References

Akasofu, S.-I. (1964). The development of the auroral substorm. *Planet. Space Sci.* 12, 273–282. doi:10.1016/0032-0633(64)90151-5

Angelopoulos, V., McFadden, J. P., Larson, D., Carlson, C. W., Mende, S. B., Frey, H., et al. (2008). Tail reconnection triggering substorm onset. *Science* 321, 931–935. doi:10.1126/science.1160495

Boscher, D., Bourdarie, S., O'Brien, P., Guild, T., Shumko, M., and Brunet, A. (2012). *Irbem-lib library*. doi:10.5281/zenodo.6867552

Burch, J. (2000). Image mission overview. *IMAGE mission* 1–14, 1–14. doi:10.1007/978-94-011-4233-5_1

Clausen, L. B. N., and Nickisch, H. (2018). Automatic classification of auroral images from the oslo auroral themis (oath) data set using machine learning. *J. Geophys. Res. Space Phys.* 123, 5640–5647. doi:10.1029/2018JA025274

Donovan, E., Mende, S., Jackel, B., Frey, H., Syrjäsuo, M., Voronkov, I., et al. (2006a). The themis all-sky imaging array—System design and initial results from the prototype imager. *J. Atmos. Solar-Terrestrial Phys.* 68, 1472–1487. doi:10.1016/j.jastp.2005.03.027

Donovan, E., Mende, S., Jackel, B., Syrjäsuo, M., Meurant, M., Voronkov, I., et al. (2006b). The azimuthal evolution of the substorm expansive phase onset auror. *Proc. 8th Int. Conf. Substorms* 8, 55–60.

FFmpeg Developers (2022). Ffmpeg tool. Available at: https://ffmpeg.org/ (Accessed 02 10, 2022).

Friis-Christensen, E., Lühr, H., and Hulot, G. (2006). Swarm: A constellation to study the earth's magnetic field. *Earth Planets Space* 58, 351–358. doi:10.1186/bf03351933

Grono, E., Donovan, E., and Murphy, K. R. (2017). Tracking patchy pulsating aurora through all-sky images. *Ann. Geophys.* 35, 777–784. doi:10.5194/angeo-35-777-2017

Jones, S., Lessard, M., Rychert, K., Spanswick, E., Donovan, E., and Jaynes, A. (2013). Persistent, widespread pulsating aurora: A case study. *JGR. Space Phys.* 118, 2998–3006. doi:10.1002/jgra.50301

Liang, J., Donovan, E., Jackel, B., Spanswick, E., and Gillies, M. (2016). On the 630 nm red-line pulsating aurora: Red-line emission geospace observatory observations and model simulations. *J. Geophys. Res. Space Phys.* 121, 7988–8012. doi:10.1002/2016ja022901

MacDonald, E. A., Donovan, E., Nishimura, Y., Case, N. A., Gillies, D. M., Gallardo-Lacourt, B., et al. (2018). New science in plain sight: Citizen scientists lead to the discovery of optical structure in the upper atmosphere. *Sci. Adv.* 4, eaaq0030. doi:10.1126/sciadv.aaq0030

Machol, J. L., Green, J. C., Redmon, R. J., Viereck, R. A., and Newell, P. T. (2012). Evaluation of ovation prime as a forecast model for visible aurorae. *Space weather.* 10, 746. doi:10.1029/2011SW000746

Mende, S., Harris, S., Frey, H., Angelopoulos, V., Russell, C., Donovan, E., et al. (2009). "The themis array of ground-based observatories for the study of auroral substorms," in *The THEMIS mission* (Springer), 357.

Newell, P. T., Sotirelis, T., and Wing, S. (2010). Seasonal variations in diffuse, monoenergetic, and broadband aurora. *J. Geophys. Res.* 115. doi:10.1029/2009JA014805

Nishimura, Y., Lyons, L., Zou, S., Angelopoulos, V., and Mende, S. (2010). Substorm triggering by new plasma intrusion: Themis all-sky imager observations. *J. Geophys. Res.* 115, 5166. doi:10.1029/2009ja015166

Rostoker, G., Samson, J., Creutzberg, F., Hughes, T., McDiarmid, D., McNamara, A., et al. (1995). Canopus—A ground-based instrument array for remote sensing the high latitude ionosphere during the istp/ggs program. *Space Sci. Rev.* 71, 743–760. doi:10.1007/bf00751349

Shepherd, S. (2014). Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations. *J. Geophys. Res. Space Phys.* 119, 7501–7521. doi:10.1002/2014ja020264

Starck, J.-L., and Murtagh, F. (2007). *Astronomical image and data analysis*. Berlin, Germany: Springer.

Zhong, Y., Ye, R., Liu, T., Hu, Z., and Zhang, L. (2020). Automatic aurora image classification framework based on deep learning for occurrence distribution analysis: A case study of all-sky image data sets from the yellow river station. *J. Geophys. Res. Space Phys.* 125, e2019JA027590. doi:10.1029/2019JA027590

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 06 October 2022
DOI 10.3389/fspas.2022.1020815

Check for updates

OPEN ACCESS

EDITED BY
Sophie A. Murray,
Dublin Institute for Advanced Studies (DIAS), Ireland

REVIEWED BY
Arnaud Masson,
European Space Astronomy Centre (ESAC), Spain
Daniel Da Silva,
Goddard Space Flight Center (NASA), United States

*CORRESPONDENCE
Eric W. Grimes,
egrimes@igpp.ucla.edu

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 16 August 2022
ACCEPTED 16 September 2022
PUBLISHED 06 October 2022

CITATION
Grimes EW, Harter B, Hatzigeorgiu N, Drozdov A, Lewis JW, Angelopoulos V, Cao X, Chu X, Hori T, Matsuda S, Jun C-W, Nakamura S, Kitahara M, Segawa T, Miyoshi Y and Le Contel O (2022), The Space Physics Environment Data Analysis System in Python. *Front. Astron. Space Sci.* 9:1020815. doi: 10.3389/fspas.2022.1020815

COPYRIGHT
© 2022 Grimes, Harter, Hatzigeorgiu, Drozdov, Lewis, Angelopoulos, Cao, Chu, Hori, Matsuda, Jun, Nakamura, Kitahara, Segawa, Miyoshi and Le Contel. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

The Space Physics Environment Data Analysis System in Python

Eric W. Grimes[1]*, Bryan Harter[2], Nick Hatzigeorgiu[3], Alexander Drozdov[1], James W. Lewis[3], Vassilis Angelopoulos[1], Xin Cao[2], Xiangning Chu[2], Tomo Hori[4], Shoya Matsuda[5], Chae-Woo Jun[4], Satoko Nakamura[4], Masahiro Kitahara[4], Tomonori Segawa[4], Yoshizumi Miyoshi[4] and Olivier Le Contel[6]

[1]Department of Earth, Planetary and Space Sciences, University of California Los Angeles, Los Angeles, CA, United States, [2]Laboratory for Atmospheric and Space Physics, University of Colorado Boulder, Boulder, CO, United States, [3]Space Sciences Laboratory, University of California Berkeley, Berkeley, CA, United States, [4]Institute for Space-Earth Environmental Research, Nagoya University, Nagoya, Japan, [5]Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa, Japan, [6]Laboratoire de Physique des Plasmas (LPP), UMR7648 CNRS/Ecole Polytechnique/UPMC/Université Paris-Sud/Observatoire de Paris, Palaiseau, France

In this article, we describe the free, open-source Python-based Space Physics Environment Data Analysis System (PySPEDAS), a platform for multi-mission, multi-instrument retrieval, analysis, and visualization of Heliophysics data. PySPEDAS currently contains load routines for data from 23 space missions, as well as a variety of data from ground-based observatories. The load routines are built from a common set of general routines that provide access to datasets in different ways (e.g., downloading and caching CDF files or accessing data hosted on web services), making the process of adding additional datasets simple. In addition to load routines, PySPEDAS contains numerous analysis tools for working with the dataset once it is loaded. We describe how these load routines and analysis tools are built by utilizing other free, open-source Python projects (e.g., PyTplot, cdflib, hapiclient, etc.) to make tools for space and solar physicists that are extremely powerful, yet easy-to-use. After discussing the code in detail, we show numerous examples of code using PySPEDAS, and discuss limitations and future plans.

KEYWORDS

heliophysics, space physics, magnetospherc physics, data analysis, data visualization, python

1 Introduction

Dynamically typed, interpreted languages, such as Interactive Data Language (IDL) and Matlab, have exploded in usage for day-to-day data analysis due their ease of use, curated libraries of scientific routines, simplified debugging, and interactive plotting. There is a need for tools tailored toward the Heliophysics community; for example, to provide access to the scientific data from multiple missions in a generic and easy-to-use way, as well as general analysis tools to work with these data. Releasing generic analysis tools has numerous benefits, such as limiting duplication of effort and reducing potential

TABLE 1 Tplot data model.

x	Array containing time values
y	Array containing data values
dy	Array containing error values
v	Array containing Y-axis locations for the values specified in "y" above (spectrograms)
v1, v2, v3	Additional dependencies

TABLE 2 Standard load routine keywords.

Trange	Two-element array containing time range
Suffix	Suffix to append to the tplot variables
get_support_data	Flag to load the "support_data" variables from the CDF files
varformat	String specifying variable format (to limit the variables loaded)
varnames	List of variable names to load
downloadonly	Flag to download the data files without loading into tplot variables
notplot	Flag to load the data into dictionaries instead of creating tplot variables
no_update	Flag to load the local data but not request the remote data
time_clip	Clip the variables down to the exact time range specified in the trange keyword

errors made by researchers. In IDL, this need led to the development of the SolarSoft (SSW)[1] package for solar physicists and the Space Physics Environment Data Analysis System [SPEDAS; Angelopoulos et al., 2019)[2] package for space physicists. The SPEDAS package supports loading, plotting, analysis, and integration of data from several space-based and ground-based observatories, providing a comprehensive environment for space physics data analysis and visualization. While powerful, IDL has numerous limitations, including the high cost of licensing, limited support, difficulty in bridging among different programming languages, as well as conflicts with other libraries caused by the single namespace limitation. The single namespace limitation requires that all IDL functions have unique names, making it nearly impossible to use SPEDAS and other large packages (e.g., SolarSoft) in the same environment without naming conflicts.

Due to these limitations and the increasing popularity of the Python programming language (Burrell et al., 2018), we began the development of an implementation of the IDL SPEDAS package in the Python programming language. The tools in PySPEDAS allow users to access scientific quality data products from numerous space missions and ground observatories with only a few lines of code. PySPEDAS provides a variety of generic analysis tools, from simple operations, such as interpolating magnetic field data, to more complex operations, such as calculating and plotting 2D slices of 3D velocity distribution data from particle instruments.

Development of PySPEDAS is occurring on GitHub at:
https://github.com/spedas/pyspedas

Development of PySPEDAS is occurring in the open to encourage contributions from mission teams as well as the general Heliophysics community.

PySPEDAS releases are distributed using the Python Package Index (PyPI) and can be installed with:

pip install pyspedas

PySPEDAS conforms to the Python in Heliophysics Community standards[3]. PySPEDAS is platform independent, supporting the Windows, macOS, and Linux operating systems, and is released under a permissive free software license (MIT).

Each function in PySPEDAS is documented in the docstring of that function. The HTML documentation is built automatically from these docstrings using the Sphinx documentation generator[4]. The HTML documentation can be found online at:

https://pyspedas.readthedocs.io/

PySPEDAS examples are available as Jupyter notebooks in general and mission-specific repositories at the SPEDAS GitHub organization[5]. Tutorials are regularly held at scientific meetings, and webinars are regularly held online and are made available on the SPEDAS YouTube channel[6].

1 http://www.lmsal.com/solarsoft
2 http://www.spedas.org/
3 https://heliopython.org/docs/
4 https://www.sphinx-doc.org
5 https://github.com/spedas
6 https://www.youtube.com/channel/UCZVyhCNPI3II7Mu086KsoLA

TABLE 3 Projects supported as of July 2022.

Advanced Composition Explorer (ACE)
Arase (ERG)
Cluster
Colorado Student Space Weather Experiment (CSSWE)
Deep Space Climate Observatory (DSCOVR)
Equator-S
Fast Auroral Snapshot Explorer (FAST)
Geotail
Geostationary Operational Environmental Satellite (GOES)
Imager for Magnetopause-to-Aurora Global Exploration (IMAGE)
Kyoto Dst Index
Mars Atmosphere and Volatile Evolution (MAVEN)
Magnetic Induction Coil Array (MICA)
Magnetospheric Multiscale (MMS)
OMNIWeb
Polar Orbiting Environmental Satellites (POES)
Polar
Parker Solar Probe (PSP)
Solar Orbiter (SOLO)
Solar Terrestrial Relations Observatory (STEREO)
Spherical Elementary Currents (SECS)/Equivalent Ionospheric Currents (EICS)
Swarm
Time History of Events and Macroscale Interactions during Substorms (THEMIS)
Two Wide-Angle Imaging Neutral-Atom Spectrometers (TWINS)
Ulysses
Van Allen Probes (RBSP)
Wind

PySPEDAS is tested using standard Python-based unit tests that are automatically ran when code is merged with the master branch. These tests typically exist in the tests directory of each module. While these tests exercise the functionality, they do not typically check the validity of the results. To validate the results, unit tests are implemented in IDL SPEDAS which compare the data products from PySPEDAS to the data products in IDL SPEDAS. These validation tests are released in IDL SPEDAS, as well as in a separate repository at the SPEDAS GitHub organization.

In the next section, we give an overview of the "tplot" model, describe the load routine design, and finally describe several analysis tools currently available.

2 Materials and methods

Internally, PySPEDAS uses the tplot data model to store, manipulate and visualize time series data; in Python, this model is implemented by the PyTplot[7] project. The tplot data model is an extremely powerful model for working with time series data, with heritage from IDL, that allows users to reference complex data sets, along with their metadata, using simple string identifiers.

2.1 Tplot model

Because of its importance, here we repeat the principles of the tplot data model. In the tplot data model, time series data are stored, alongside metadata, in global objects called tplot variables. The tplot data model is shown in Table 1. The time values are stored as Unix times (number of seconds since 00:00:00 UTC on 1 January 1970, excluding leap seconds) in the x component of the variable. These values should be monotonically increasing. The data values are stored in the y component, error bars are stored in the optional dy component, and additional dependencies (e.g., energy and angular ranges) are stored in the optional v, v1, v2, and v3 components of the variable.

PyTplot provides importers for a variety of file formats, including CDF files, netCDF files, STS files, as well as IDL sav files. These importers use other open-source libraries (e.g., cdflib[8], netCDF4[9], scipy[10], astropy[11], etc.) to load the data and time values into numpy arrays, then convert the time values to a common epoch (Unix time), and then store the tplot variables containing the time values, data, and metadata, using the name of the variable as a reference.

A suite of PyTplot routines provides an environment for working with tplot variables, e.g., creating variables (pytplot.store_data), plotting those variables (pytplot.tplot), returning the data in numpy arrays or xarray objects (pytplot.get_data), setting figure options (pytplot.tplot_options), setting panel options (pytplot.options), as well as doing basic calculations such as data cropping (pytplot.crop) and data averaging (pytplot.avg_res_data). A list of the tplot variables in the current session can be found using the pytplot.tplot_names function, and the metadata for a variable can be accessed by setting the metadata option when calling (pytplot.get_data). A full list of the routines and their functionality can be found in the PyTplot documentation.

The plotting features provided by PyTplot are extensive, supporting multiple possible backends (including Bokeh, Qt, and matplotlib), each supporting stacked time series plots containing any combination of line and spectrogram panels. In addition to the standard features supported by the Bokeh and Qt backends, the current backend used by PySPEDAS (matplotlib) supports error bars, annotations, highlighting

7 https://github.com/MAVENSDC/PyTplot

8 https://github.com/MAVENSDC/cdflib

9 https://github.com/Unidata/netcdf4-python

10 https://scipy.org/

11 https://www.astropy.org/

TABLE 4 The particle data structure.

Project_name	Mission name
Spacecraft	Spacecraft name
Data_name	General description of the data
Units_name	Units name
Units_procedure	Name of procedure for unit conversions
Species	Species name
Charge	Particle charge (+1 for ions, −1 for electrons)
Mass	Particle mass [in units of eV/(km/s)^2]
start_time	Start of the accumulation interval (unix time)
end_time	End of the accumulation interval (unix time)
n_energy	Number of energy bins
n_theta	Number of theta bins
n_phi	Number of phi bins
bins	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing 1 (active bin) or 0 (inactive bin)
data	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the data values
theta	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the theta (elevation) values; values should be from [−90, +90]
phi	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the phi (azimuth) values; values should be from [0, 360)
Energy	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the energy values (eV units)
dtheta	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the widths of the theta bins
dphi	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the widths of the phi bins
denergy	numpy.ndarray of shape: [n_energy, n_phi, n_theta]; containing the widths of the energy bins

time intervals, automatic spectrogram interpolation, overplotting lines on top of spectrograms, creating figures in Jupyter notebooks, as well as saving publication quality figures.

2.2 Load routines

The instrument load routines in PySPEDAS follow the form: pyspedas.mission.instrument ()

And implement a common set of keywords for access to the data (as shown in Table 2). Some missions [e.g., Magnetospheric Multiscale (MMS; Burch et al., 2016)] implement additional keywords to provide additional options when loading the data. Each mission implements at least one core load routine, with instrument-specific wrappers implemented as separate functions. The core load routines typically download the data files, load the data files into PyTplot variables, perform common post-processing (e.g., time clipping of the data), then return a list of the variables that were loaded. The instrument-specific functions can then call these core routines to load the data. This model, adapted from IDL SPEDAS, allows for instrument-level pre- and post-processing in the wrapper routines, while most instrument-independent loading is done in a common core routine shared by the instruments. Multiple core routines can be developed for access to different data servers (e.g., one for a mission's official data server and one for the NASA archive).

PySPEDAS contains several support routines for downloading data files from remote repositories, including:

pyspedas.dailynames: generates a list of file names from a time range using the strftime format

pyspedas.download: general download routine that uses the open-source requests[12] library internally, supports authentication, allows parsing and searching HTML directory index pages generated by Apache, and provides local caching and file version numbers.

pyspedas.hapi.hapi.hapi: supports loading data from Heliophysics Data Application Programmer's Interface (HAPI) [13] servers; uses hapiclient[14] package to load the parameters into numpy arrays, then creates tplot variables.

By default, the data files are stored in a subdirectory of the current working directory; this directory can be changed using the SPEDAS_DATA_DIR environment variable, as well as mission-specific environment variables (e.g., MMS_DATA_DIR for MMS, THM_DATA_DIR for THEMIS). The mission-specific environment variables override the global SPEDAS data directory set in SPEDAS_DATA_DIR. The local data directories can also be changed in the Python interpreter by setting the 'local_data_dir' key in the mission's CONFIG dictionary; e.g., pyspedas.themis.config.CONFIG ['local_data_dir'] = '/path/to/data/' for THEMIS.

12 https://requests.readthedocs.io/

13 https://hapi-server.github.io/

14 https://github.com/hapi-server/client-python

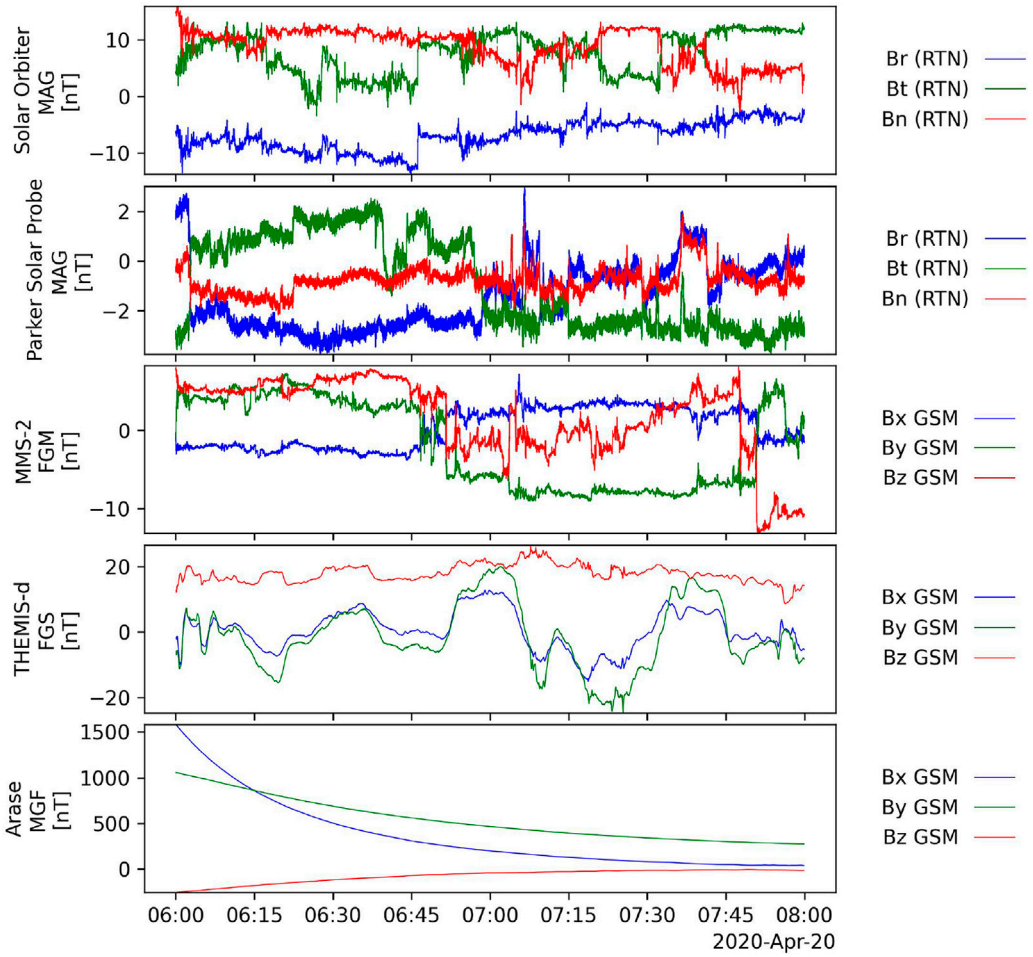


FIGURE 1
Magnetic field data from Solar Orbiter, Parker Solar Probe, Magnetic Multiscale Mission, THEMIS, and Arase missions for 2 h on 20 April 2020.

Once the data files are saved locally, the core load routines typically use importers inside PyTplot, described in (Section 2.1), to save the parameters as tplot variables. The wrapper routines then take these loaded variables and do any instrument-specific post-processing (e.g., creating additional data products, applying calibrations and corrections, setting plot metadata, etc.).

Table 3 shows a list of the current projects supported by PySPEDAS as of July 2022.

2.3 Analysis tools

2.3.1 Coordinate transformations

Coordinate transformations are implemented in the pyspedas.cotrans module. The cotrans function, located in the pyspedas.cotrans module, accepts any tplot variable containing vector data in Cartesian coordinates and supports transformations to and from several coordinate systems (Hapgood 1992): Geocentric Equatorial Inertial (GEI), Geocentric Solar Ecliptic (GSE), Geocentric Solar Magnetospheric (GSM), Solar Magnetic (SM), Geographic (GEO), Geomagnetic (MAG) and Geocentric Equatorial Inertial for epoch J2000.0 (J2000).

Internally, these transformations are direct translations of the IDL SPEDAS coordinate transformation routines, which were originally based on the ROCOTLIB[15] library. In addition to the calculations to perform the various transformations, the pyspedas.cotrans module utilizes the tplot metadata where possible; i.e., if the input coordinate system is stored in the variable's metadata, the user does not need to specify it manually. The output variable's metadata are updated to the new coordinate system (including in any plot annotations), minimizing the amount of user effort to produce transformed data with proper annotations.

15 http://cdpp.irap.omp.eu/index.php/services/scientific-librairies/rocotlib

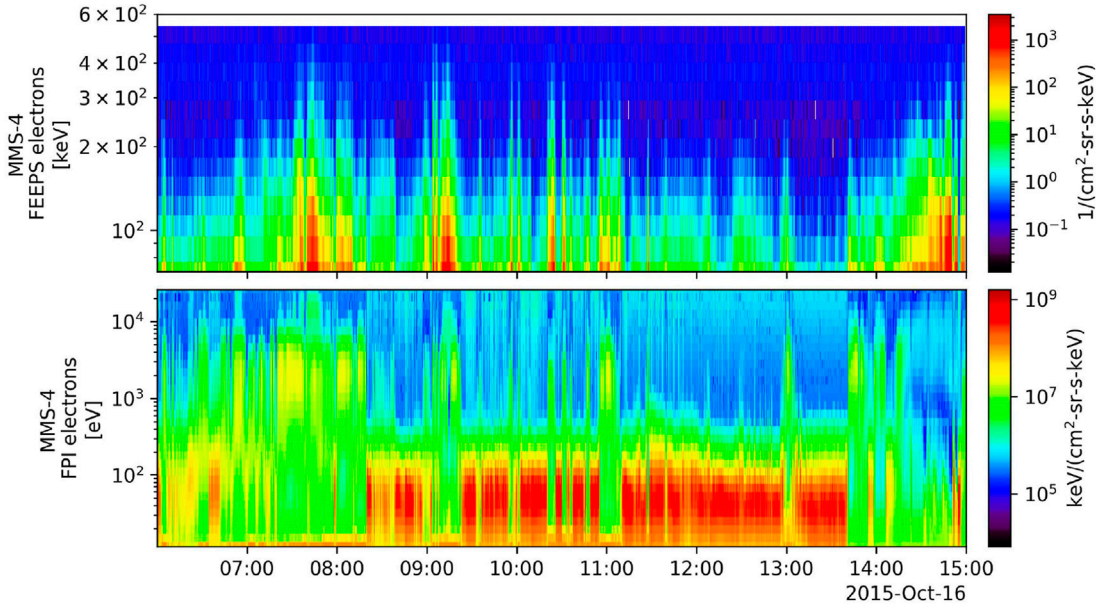


FIGURE 2
Electron energy spectra observed by the **(A)** Fly's Eye Energetic Particle Sensor (FEEPS) and **(B)** Fast Plasma Investigation (FPI) instruments on 16 October 2015.

The pyspedas.cotrans module also provides tools for transforming vector data into various field-aligned coordinate systems (for which the Z axis corresponds to the direction of a given field vector, X and Y axes defining the plane perpendicular to it); pyspedas.cotrans.fac_matrix_make generates field-aligned coordinate transformation matrices, and pyspedas.cotrans.tvector_rotate rotates vector data using these matrices.

In addition to tplot variables, the pyspedas.cotrans module supports data and time inputs as simple numpy arrays. If numpy arrays are provided instead of tplot variables, the pyspedas.cotrans module will create a tplot variable internally prior to performing the transformation.

2.3.2 Magnetic field models

Routines for working with magnetic field models are implemented in the pyspedas.geopack module. Functions are available for generating the Tsyganenko 89, 96, and 2001 models, as well as the Tsyganenko-Sitnov 2004 model at arbitrary points in space (as a function of time) (Tsyganenko, 2013). Internally, these functions extract the input position data from tplot variables, then use the pure-Python implementation of the Geopack library[16] to generate the various field models at each point, then store the results in tplot variables.

The Tsyganenko 96, 2001, and Tsyganenko-Sitnov 2005 models require additional solar wind input. The pyspedas.geopack module contains routines (get_tsy_params, get_w_params) for generating the required input parameters to this model using the OMNI solar wind data loaded using the pyspedas.omni.data function.

2.3.3 Curlometer technique

The curlometer technique is implemented in the pyspedas.analysis.lingradest routine. This function takes position and magnetic field data obtained at four spacecraft (e.g., MMS or Cluster) and applies the linear gradient/curl estimator technique (Dunlop et al., 2021) to calculate the magnetic field gradients, divergence, curl, and field line curvature. This routine is a direct translation of the IDL SPEDAS version originally developed for Cluster (Runov et al., 2003) and most recently used by MMS.

The core lingradest function is called by an MMS-specific wrapper mms_lingradest in the pyspedas.mms.fgm module; this wrapper interpolates the spacecraft position and magnetic field data to the first spacecraft timestamps, then calls the lingradest function to perform the calculations and saves the output in tplot variables.

2.3.4 Dynamic power spectrum

The dynamic power spectrum of a tplot variable can be calculated using the pyspedas.tdpwrspc function. This function extracts the data from the input tplot variables, and then a Hanning window is applied to the input data (its power is

16 https://github.com/tsssss/geopack

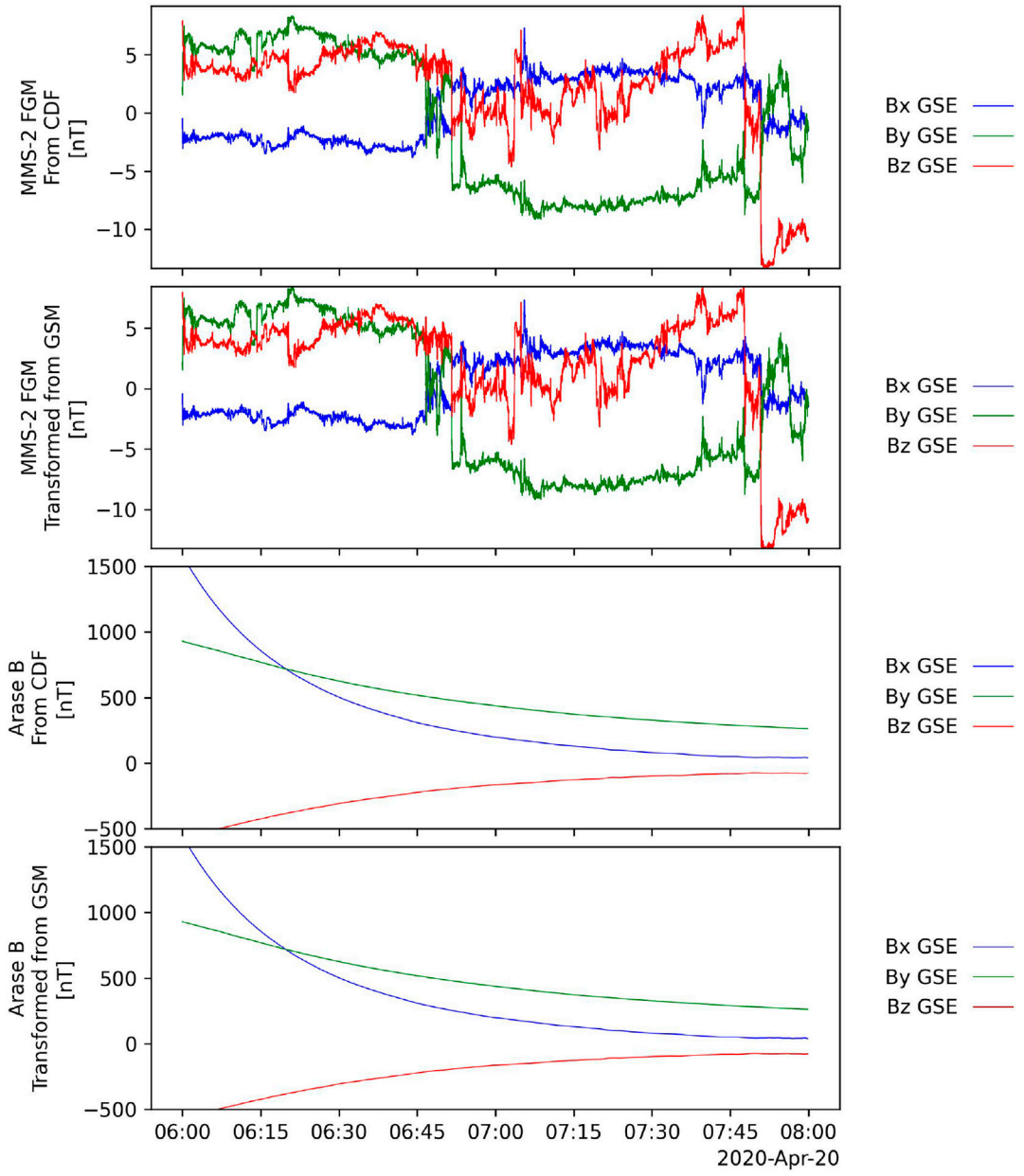


FIGURE 3
From top to bottom, **(A)** FGM data observed by MMS-2 in GSE coordinates (from the CDF files), **(B)** MMS-2 FGM data in GSE coordinates transformed from GSM coordinates, **(C)** Arase MGF data in GSE coordinates (from the CDF files), and **(D)** Arase MGF data in GSE coordinates transformed from GSM coordinates.

divided out of the returned spectrum). A straight line is subtracted from the data to reduce spurious power due to the sawtooth behavior of a background. The output, the mean squared amplitude of the signal at each specific frequency, is then stored in separate tplot variables for each component. Keyword options are available for controlling the number of points to use for the Hanning window, the number of points to shift for each spectrum, and the output frequency bin size. Options are also available for disabling the Hanning window and straight line subtraction.

2.3.5 Wave polarization tools

The pyspedas.twavpol function allows for performing wave polarization analysis of three orthogonal component time series data in tplot variables. This function extracts the data from the input tplot variable, which usually has been moved into a magnetic

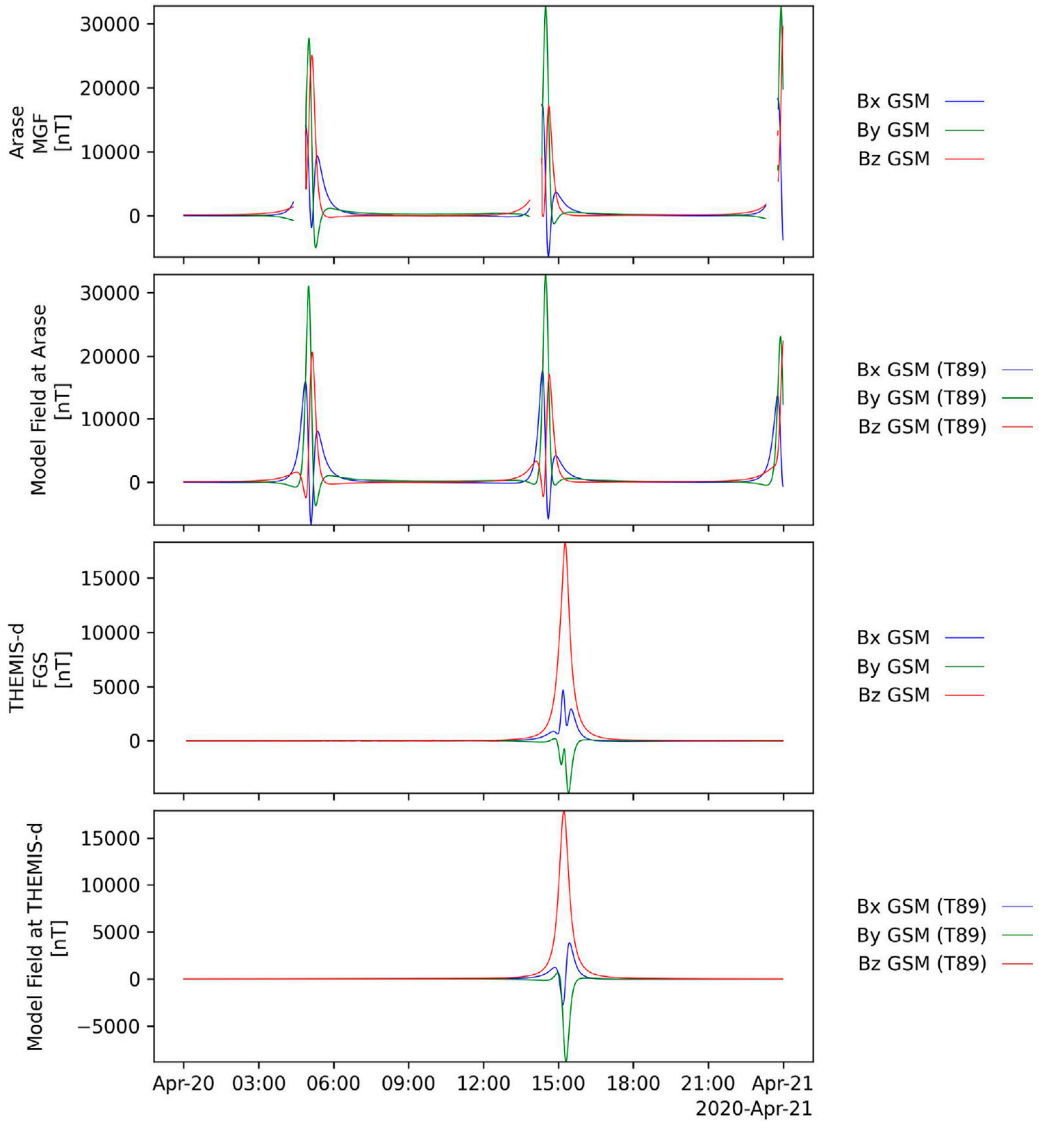


FIGURE 4
From top to bottom, **(A)** Arase MGF data (GSM coordinates), **(B)** T89 model field at the Arase position (GSM coordinates), **(C)** THEMIS-d FGS data (GSM coordinates), and **(D)** T89 model field at the THEMIS-d position (GSM coordinates).

field-aligned coordinate system such that wave angle and ellipticity refer to the magnetic field direction. It then passes the input data to a core wavpol routine (Samson and Olson, 1980), which calculates the degree of polarization, wave normal angle, ellipticity, and helicity, then saves the output in tplot variables. The wave polarization tools in PySPEDAS are direct translations of the IDL SPEDAS routines for performing the same calculations.

2.3.6 Particle tools

General (mission independent) particle tools are implemented in the pyspedas.particles module, with mission-specific particle tools implemented in the particles submodule of a mission's module (e.g., MMS particle tools are implemented in pyspedas.mms.particles). Tools exist for taking particle data structures and calculating the energy, theta, phi, pitch angle, and gyrophase spectrograms, as well as plasma moments of velocity distribution functions. In addition, tools exist for calculating and plotting 2D slices of the velocity distribution functions.

The PySPEDAS particle data structure, which is based on the IDL data structure, is shown in Table 4. The units_name attribute of the particle data structure must be counts, rate, eflux, flux, df, df_cm, df_km, e2flux, or e3flux. For the theta and phi values, the PySPEDAS particle tools use presumed particle trajectories (not look direction of the instrument).

The user-facing functions for doing particle calculations can be found in the particles submodule of the mission's module, e.g.,

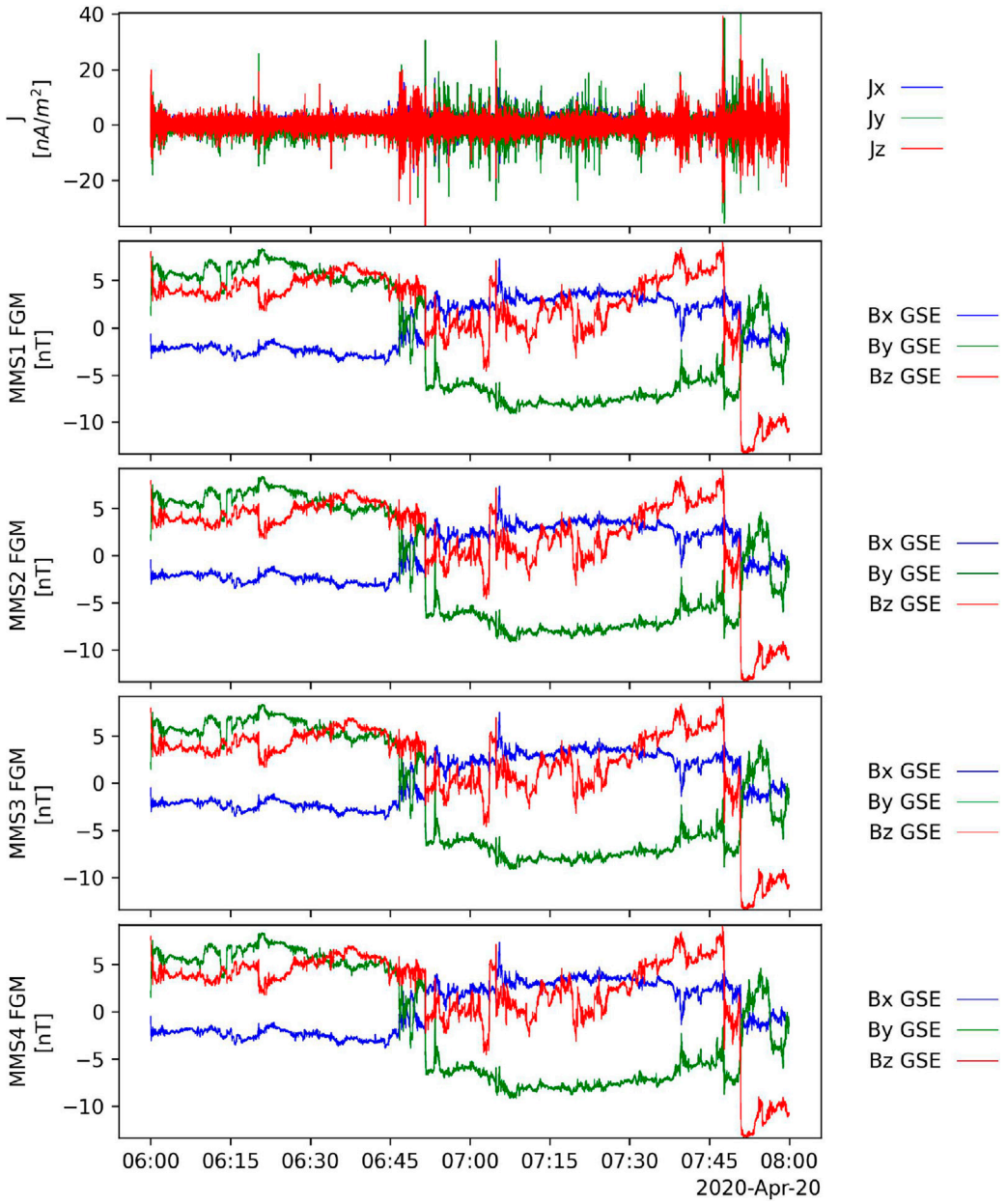


FIGURE 5
From top to bottom, **(A)** the current density from the curlometer technique, **(B)** MMS-1 FGM data **(C)** MMS-2 FGM data **(D)** MMS-3 FGM data **(E)** MMS-4 FGM data (all GSE coordinates).

spectrograms and moments for MMS can be calculated using the mms_part_getspec function found in the pyspedas.mms.particles module. This function loads particle distribution function data (as well as any required support data), re-forms the data into the standard PySPEDAS particle data structure described above, then uses various generic functions to perform the calculations, and stores the output as tplot variables. Options exist for changing the species, limiting the energy and angular ranges, changing the output units, disabling internal and external electron photoelectron corrections [for the Fast Plasma Instrument (FPI; Pollock et al., 2016)], and more.

Slices of MMS distribution function data can be plotted using the mms_part_slice2d function, also found in the pyspedas.mms.particles module. Just as above, this function loads the data and any required support data, creates the PySPEDAS particle data structures, then uses generic tools to calculate and plot the slice. Options exist for rotating the slice into a variety of coordinate systems, limiting the energy range, changing the interpolation type, smoothing, and more.

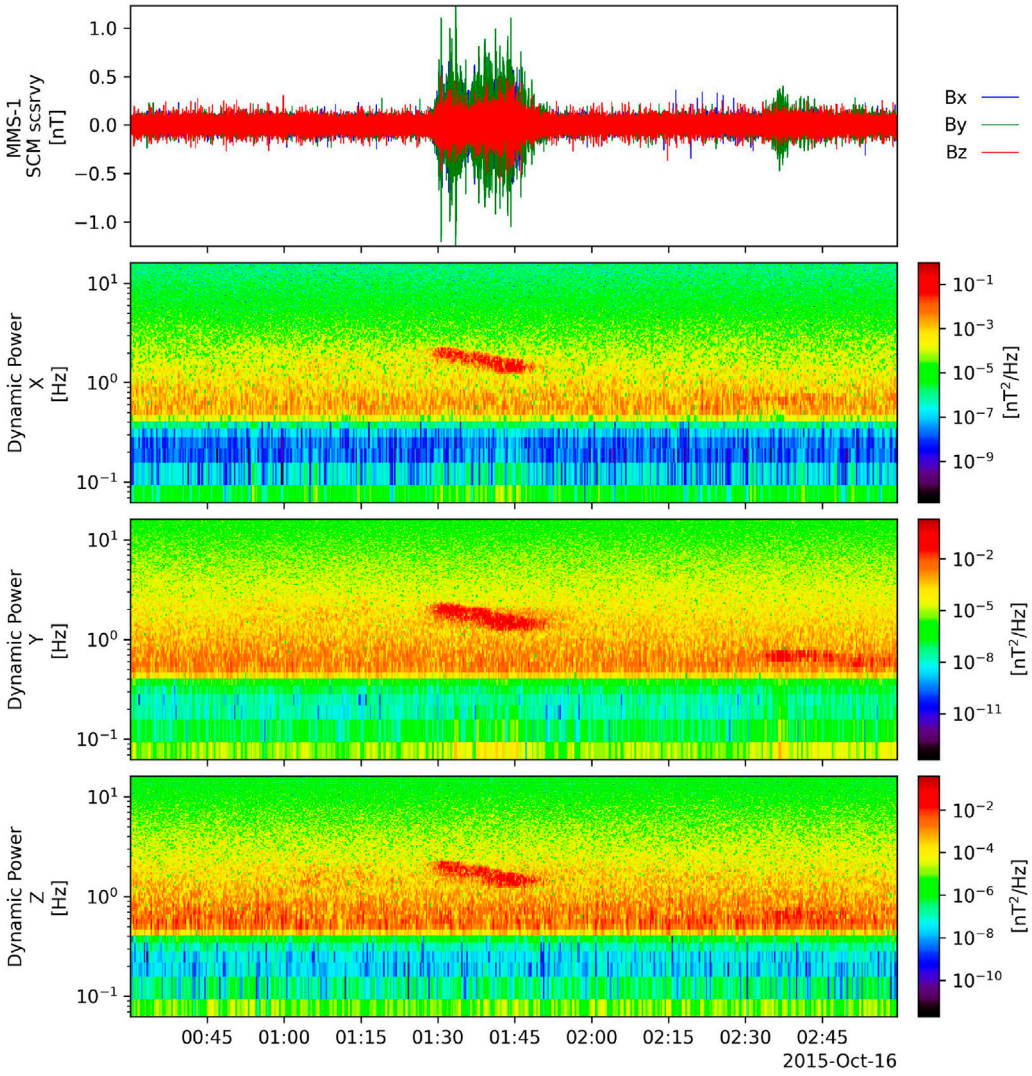


FIGURE 6
From top to bottom, **(A)** MMS-1 survey mode SCM data (GSE coordinates), dynamic power spectra of the **(B)** X component, **(C)** Y component, **(D)** Z component.

2.3.7 Spherical elementary currents/equivalent ionospheric currents plots

The pyspedas.secs module allows for downloading and plotting vector and contour maps of Spherical Elementary Currents and Equivalent Ionospheric Currents data (Weygand et al., 2011). The data are downloaded from the Virtual Magnetospheric Observatory[17] using pyspedas.secs.data, and the data are plotted using the make_plots function in the pyspedas.secs.makeplots module. Internally, this function uses matplotlib to create the figures.

3 Results

3.1 Basic example

Figure 1 shows magnetic field data measured by five spacecraft, including Solar Orbiter, Parker Solar Probe,

17 http://vmo.igpp.ucla.edu/

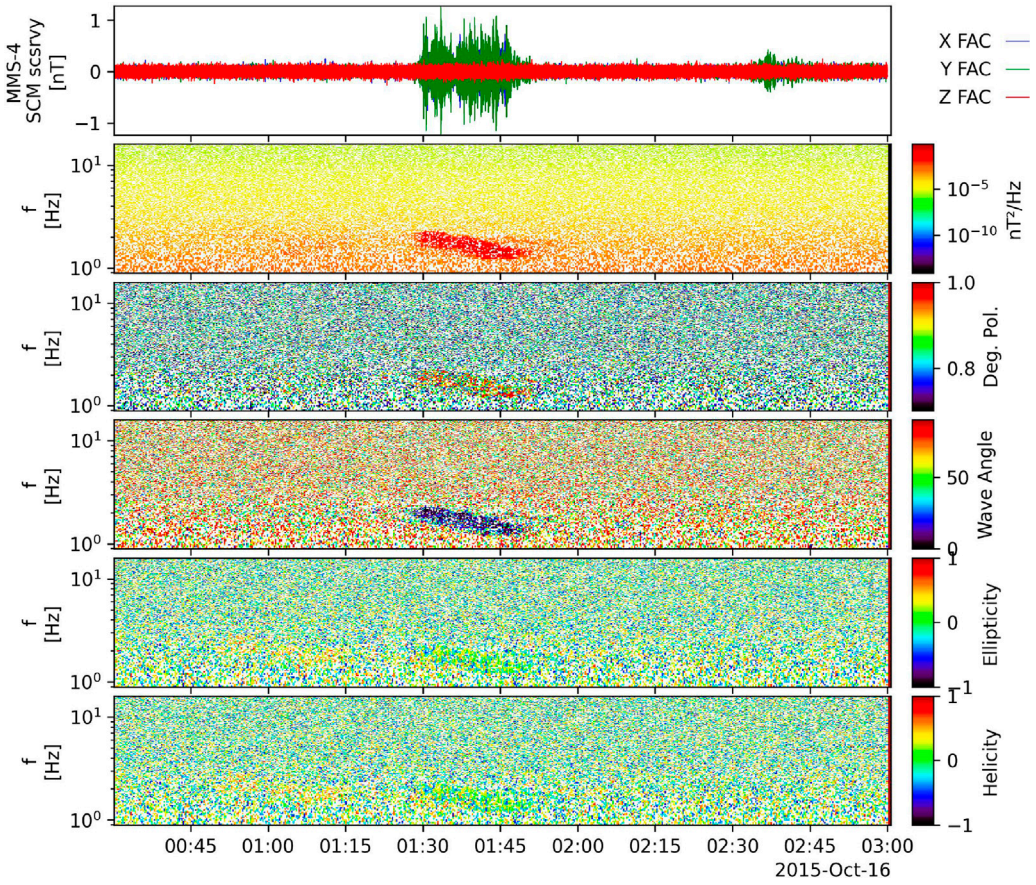


FIGURE 7
From top to bottom, **(A)** MMS-1 survey mode SCM data (field-aligned coordinates, **(B)** wave power, **(C)** degree of polarization, **(D)** wave angle, **(E)** ellipticity, **(F)** helicity.

MMS, THEMIS, and Arase, for 2 h on 20 April 2020. Using PySPEDAS, the data for this figure can be loaded with a total of two imports, five function calls, one for each instrument, and the figure can be plotted with a sixth function call, e.g., Figure 1 can be re-created using the following:

```
import pyspedas
from pytplot import tplot

time_range = ['2020-04-20/06:00', '2020-04-20/08:00']

pyspedas.solo.mag(trange=time_range, time_clip=True)
pyspedas.psp.fields(trange=time_range, time_clip=True)
pyspedas.mms.fgm(trange=time_range, time_clip=True, probe=2)
pyspedas.themis.fgm(trange=time_range, time_clip=True, probe='d')
pyspedas.erg.mgf(trange=time_range, time_clip=True)

tplot(['B_RTN',
       'psp_fld_l2_mag_RTN',
       'mms2_fgm_b_gsm_srvy_l2_bvec',
       'thd_fgs_gsm',
       'erg_mgf_l2_mag_8sec_gsm'])
```

Each load routine call returns a list of the variable names that were loaded, and users are encouraged to review the mission team's documentation for details on the loaded data products.

The tools (load and analysis routines) shown in these examples have additional options that can be found in the docstrings (available in the interpreter by calling the help function with the PySPEDAS function as an argument), as well as in our online documentation.

In this example, as well as several of those that follow, we made minor adjustments to some of the figure annotations prior to saving (using pytplot.options) for consistency; the full code for generating all figures can be found in the Supplementary Material.

3.2 Post-processing example

Figure 2 shows fast survey electron data from the Fast Plasma Investigation (FPI) and Fly's Eye Energetic Particle Sensor

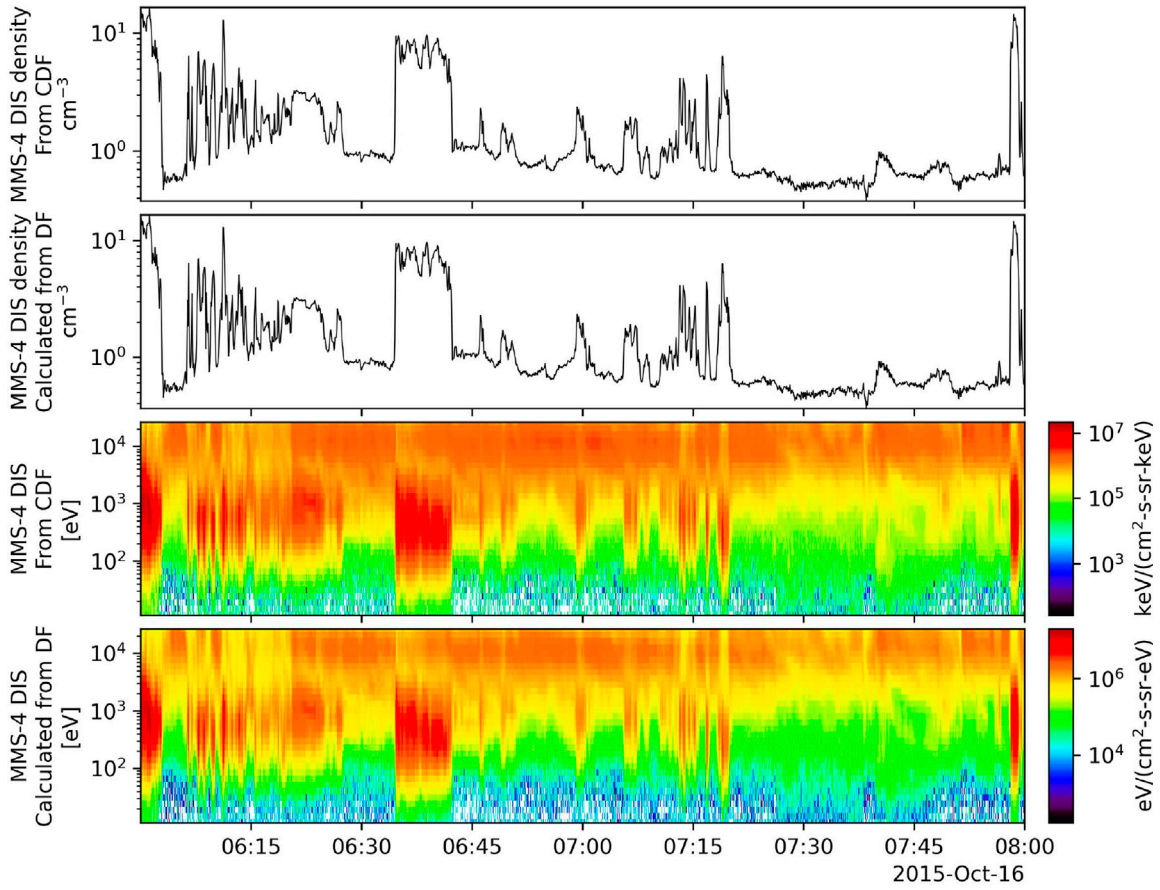


FIGURE 8
From top to bottom, **(A)** MMS-4 DIS number density (from CDF files), **(B)** MMS-4 DIS number density [calculated from the distribution function (DF) data], **(C)** MMS-4 DIS energy spectra (from CDF files), **(D)** MMS-4 DIS energy spectra (calculated from the DF data).

(FEEPS; Blake et al., 2016) instruments onboard MMS-4 on 16 October 2015.

Each measurement is taken over a certain time interval, called the accumulation interval, when data were acquired for that measurement. The FPI measurements are stored at the beginning of the accumulation interval, while other instruments (e.g., FEEPS) are stored at the middle of the accumulation interval; in order to correct this, the user must center the FPI measurements using the center_measurement option in the FPI load routine.

The FEEPS spin-averaged omni-directional data shown in Figure 2 are calculated after the individual telescope data are loaded from the CDF files. These data have numerous corrections applied in post-processing prior to calculating the omni-directional data products, including flat field corrections (for ions), energy table corrections, bad and inactive telescope removal, and sunlight contamination removal.

Figure 2 can be re-created using the following:

```
import pyspedas
from pytplot import tplot

time_range = ['2015-10-16/06:00', '2015-10-16/15:00']

pyspedas.mms.feeps(trange=time_range,
                   probe=4,
                   time_clip=True)
pyspedas.mms.fpi(trange=time_range,
                 probe=4,
                 datatype='des-moms',
                 center_measurement=True,
                 time_clip=True)

tplot(['mms4_epd_feeps_srvy_l2_electron_intensity_omni_spin',
       'mms4_des_energyspectr_omni_fast'])
```

3.3 Coordinate transformation example

Figure 3 shows 2 h of MMS FGM and Arase MGF (Matsuoka et al., 2018) data on 20 April 2020. The first

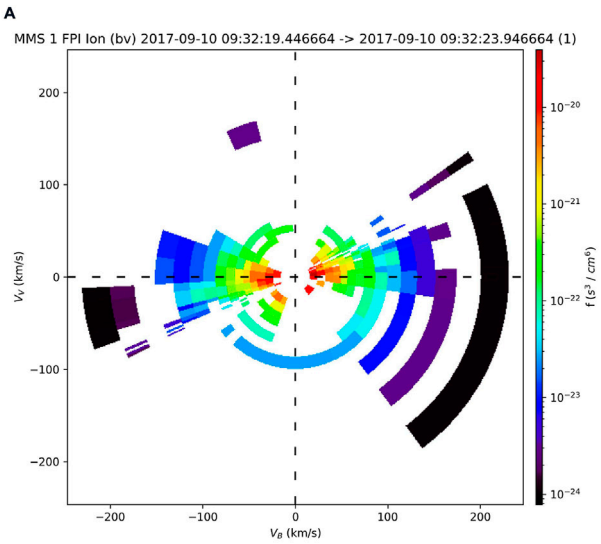


FIGURE 9
2D slices showing a bi-directional beam of 0–300 eV ions observed by MMS-1 **(A)** FPI DIS, rotated such that x-axis is parallel to the magnetic field and the bulk velocity defines the x-y plane. **(B)** HPCA H+, rotated such that the B x V (bulk) vector defines the x-y plane.

panel shows the MMS-2 FGM data in GSE coordinates, loaded from the CDF file, and the second panel shows the MMS-2 data transformed from GSM coordinates. The third and fourth panels of Figure 3 show the Arase MGF data in GSE coordinates, loaded from the CDF file, and the Arase MGF data, in GSE coordinates, transformed from GSM. In both cases, the transformed data matches the GSE data loaded from the CDF files.

Figure 3 can be re-created using the following:

```
from pyspedas import cotrans
from pytplot import tplot

cotrans(name_in='erg_mgf_l2_mag_8sec_gsm',
        name_out='erg_mgf_l2_mag_8sec_gsm_2gse',
        coord_in='gsm',
        coord_out='gse')

cotrans(name_in='mms2_fgm_b_gsm_srvy_l2_bvec',
        name_out='mms2_fgm_b_gsm_srvy_l2_bvec_2gse',
        coord_out='gse')

tplot(['mms2_fgm_b_gse_srvy_l2_bvec',
       'mms2_fgm_b_gsm_srvy_l2_bvec_2gse',
       'erg_mgf_l2_mag_8sec_gse',
       'erg_mgf_l2_mag_8sec_gsm_2gse'])
```

3.4 Magnetic field model example

Figure 4 shows a comparison of the measured magnetic field at the Arase and THEMIS spacecraft and the magnetic field model produced by the Tsyganenko 89 model at each spacecraft location on 20 April 2020. The spacecraft position and magnetic field data can be loaded in two function calls per spacecraft, and the Arase position data can be converted to kilometers from Earth radii using pyspedas.tkm2re in a single function call. The T89 model can be calculated at each position data with another function call per spacecraft, and the results can then be plotted with one final function call.

Figure 4 can be re-created using the following:

```
import pyspedas
from pytplot import tplot

time_range = ['2020-04-20', '2020-04-21']

pyspedas.erg.orb(trange=time_range)
pyspedas.erg.mgf(trange=time_range)
pyspedas.themis.state(trange=time_range, probe='d')
pyspedas.themis.fgm(trange=time_range, probe='d')

# convert the Arase position data from Earth radii to kilometers.
from pyspedas import tkm2re
tkm2re('erg_orb_l2_pos_gsm', km=True)

# calculate the field using the T89 model
from pyspedas.geopack import tt89
tt89('erg_orb_l2_pos_gsm_km')
tt89('thd_pos_gsm')

tplot(['erg_mgf_l2_mag_8sec_gsm',
       'erg_orb_l2_pos_gsm_km_bt89',
       'thd_fgs_gsm',
       'thd_pos_gsm_bt89'])
```

3.5 Curlometer example

Figure 5 shows the magnetic field in GSE coordinates measured by all four MMS spacecraft on 20 April 2020 (bottom four panels) and the total current calculated using the linear gradient descent curlometer technique using the field and spacecraft position (MMS Ephemeris/Coordinates; MEC) data for each probe (top panel). The data can be loaded in two function

calls, the curlometer calculations require only another function call, and the results can be plotted using a final function call.

Figure 5 can be re-created using the following:

```
import pyspedas
from pytplot import tplot

time_range = ['2020-04-20/06:00', '2020-04-20/08:00']

pyspedas.mms.fgm(probe=[1, 2, 3, 4],
                 trange=time_range,
                 time_clip=True)
pyspedas.mms.mec(probe=[1, 2, 3, 4],
                 trange=time_range,
                 time_clip=True)

from pyspedas.mms import lingradest

lingradest(fields=['mms1_fgm_b_gse_srvy_l2_bvec',
                   'mms2_fgm_b_gse_srvy_l2_bvec',
                   'mms3_fgm_b_gse_srvy_l2_bvec',
                   'mms4_fgm_b_gse_srvy_l2_bvec'],
           positions=['mms1_mec_r_gse',
                      'mms2_mec_r_gse',
                      'mms3_mec_r_gse',
                      'mms4_mec_r_gse'])

tplot(['jtotal',
       'mms1_fgm_b_gse_srvy_l2_bvec',
       'mms2_fgm_b_gse_srvy_l2_bvec',
       'mms3_fgm_b_gse_srvy_l2_bvec',
       'mms4_fgm_b_gse_srvy_l2_bvec'])
```

3.6 Dynamic power spectrum example

The top panel of Figure 6 shows the fast survey AC magnetic field [(0.5, 16 Hz) frequency range] measured by the Search Coil Magnetometer (SCM) instrument onboard MMS for 3 h on 16 October 2015. The bottom three panels show the X, Y, and Z components of the dynamic power spectra of the SCM data in the top panel. The dynamic power spectra were calculated using a Hanning window of 512 points and 512 points to shift for each spectrum. The data can be loaded with a single function call, the dynamic power spectra can be calculated using another function call, and the results can be plotted with a final function call.

Figure 6 can be re-created using the following:

```
import pyspedas
from pytplot import tplot

pyspedas.mms.scm(trange=['2015-10-16/00:30', '2015-10-16/3:00'],
                 time_clip=True)

from pyspedas import tdpwrspc

tdpwrspc('mms1_scm_acb_gse_scsrvy_srvy_l2',
         nboxpoints=512,
         nshiftpoints=512,
         binsize=1)

tplot(['mms1_scm_acb_gse_scsrvy_srvy_l2',
       'mms1_scm_acb_gse_scsrvy_srvy_l2_x_dpwrspc',
       'mms1_scm_acb_gse_scsrvy_srvy_l2_y_dpwrspc',
       'mms1_scm_acb_gse_scsrvy_srvy_l2_z_dpwrspc'])
```

3.7 Wave polarization example

The top panel of Figure 7 shows the SCM data from Figure 6 transformed into magnetic field-aligned coordinates. The next panels show the wave power, degree of polarization, wave normal angle, ellipticity, and helicity calculated using the SCM data shown in Figure 6. If one component is an order of magnitude greater than the other two, then the polarization results saturate and erroneously indicate high degrees of polarization at all times and frequencies. The script to recreate this example would take multiple pages, so it is provided as a separate file in the Supplementary Material.

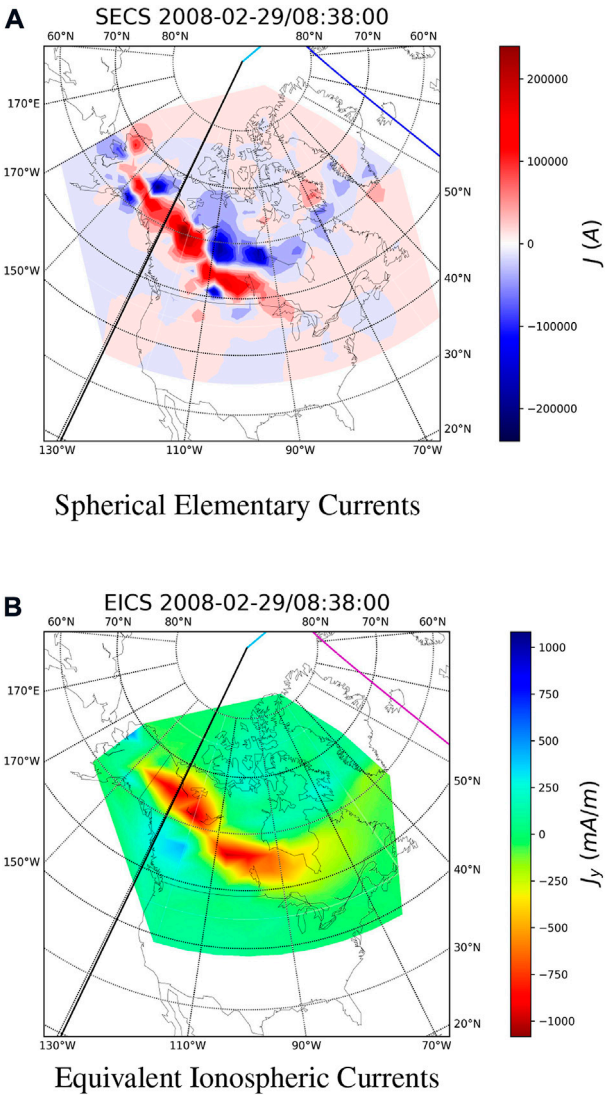


FIGURE 10
Maps showing **(A)** Spherical Elementary Currents and **(B)** Equivalent Ionospheric Currents at 08:38 UT on 29 February 2008. The contour map of ionospheric currents reveals a westward electrojet event, which is likely related to magnetospheric activities such as a substorm.

3.8 Velocity distribution function example

Figure 8 shows a comparison of the ion density and energy spectra calculated using mms_part_getspec with those released by the FPI team in CDFs for 2 h on 16 October 2015.

Figure 8 can be re-created using the following:

```
from pyspedas.mms.particles.mms_part_getspec import mms_part_getspec

time_range = ['2015-10-16/06:00', '2015-10-16/08:00']

# calculate the energy spectra and moments
# from the FPI distribution data
mms_part_getspec(trange=time_range,
                 output=['energy', 'moments'],
                 instrument='fpi',
                 species='i',
                 probe=4)

# load the CDF data for comparison
pyspedas.mms.fpi(trange=time_range,
                 time_clip=True,
                 datatype='dis-moms',
                 probe=4,
                 center_measurement=True)

from pytplot import tplot

tplot(['mms4_dis_numberdensity_fast',
       'mms4_dis_dist_fast_density',
       'mms4_dis_energyspectr_omni_fast',
       'mms4_dis_dist_fast_energy'])
```

3.9 2D Slice of distribution function example

Figure 9A shows a 2D slice of FPI ion distribution data produced with the mms_part_slice2d. The figure shows a field-aligned, bi-directional beam of 0–300 eV ions observed by FPI onboard MMS-1 at 09:32:19 UT on 10 September 2017. The energy range was limited to 0–300 eV, and the data were rotated such that the x-axis is parallel to the magnetic field and the bulk velocity defines the x-y plane. The slice was calculated using "geometric" interpolation, i.e., each point on the plot is given the value of the bin it intersects.

Figure 9B shows a 2D slice of HPCA H+ distribution data produced with the mms_part_slice2d for the same event as above. The energy range was limited to 0–300 eV, but in this case, the data were rotated such that the B x V (bulk) vector defines the x-y plane, and the slice was calculated using 2D interpolation instead of geometric. Using the 2D interpolation method, data points within the specified theta or z-axis range are projected onto the slice plane and linearly interpolated onto a regular 2D grid.

This figure can be re-created using the following:

```
from pyspedas.mms.particles.mms_part_slice2d import mms_part_slice2d

# FPI
mms_part_slice2d(time='2017-09-10/09:32:20',
                 instrument='fpi',
                 species='i',
                 rotation='bv',
                 erange=[0, 300])

# HPCA
mms_part_slice2d(time='2017-09-10/09:32:20',
                 instrument='hpca',
                 species='hplus',
                 rotation='be',
                 interpolation='2d',
                 smooth=5,
                 resolution=250,
                 erange=[0, 300])
```

3.10 Spherical elementary currents/ equivalent ionospheric currents example

Figure 10A shows the Spherical Elementary Currents, and Figure 10B shows the Equivalent Ionospheric Currents (left) at 08:38 UT on 29 February 2008, which can be utilized to investigate the ionospheric currents, and the magnetosphere-ionosphere coupling process, such as the ionospheric response to the magnetospheric substorms.

This figure can be re-created using the following:

```
from pyspedas.secs.makeplots import make_plots

# download the data
pyspedas.secs.data(trange=['2008-02-29', '2008-03-01'],
                   resolution=10,
                   dtype='SECS',
                   no_download=False,
                   downloadonly=False,
                   out_type='df')

# create the figure
make_plots(dtype='SECS',
           dtime='2008-02-29/08:38:00',
           vplot_sized=True,
           contour_den=201,
           s_loc=False,
           quiver_scale=30)

# download the data
pyspedas.secs.data(trange=['2008-02-29', '2008-03-01'],
                   resolution=10,
                   dtype='EICS',
                   no_download=False,
                   downloadonly=False,
                   out_type='df')

# create the figures
make_plots(dtype='EICS',
           dtime='2008-02-29/08:38:00',
           vplot_sized=True,
           contour_den=201,
           s_loc=False,
           quiver_scale=30)
```

4 Discussion

The examples in the previous section show how powerful PySPEDAS is for Heliophysics research; for each example, the data can be downloaded and plotted in less than a page of code.

We plan to add support for additional projects, datasets, and analysis tools, as well as additional post-processing to several of the projects we currently support. As of July 2022, the tools for working with particle distribution function data are limited to the MMS FPI and HPCA instruments. We are planning on extending support to the particle instruments onboard THEMIS and Arase in the near future. We are also currently implementing tools for minimum variance analysis calculations. In addition to these analysis tools, we plan to add a Graphical User Interface (GUI) and a "calc" mini-language for working with tplot variables, much like those that exist in IDL SPEDAS [see Angelopoulos et al., 2019 for more].

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/ Supplementary Material.

Author contributions

BH contributed to the development of cdflib, PyTplot, as well as the MAVEN plug-in. NH and AD contributed to the development of various PySPEDAS routines. JL and VA

provided program leadership. XCa and XCh contributed the SECS/EICS plug-in. TH, SM C-WJ, SN, MK, TS, and YM contributed the Arase plug-in. OL is the Lead Co-I of the MMS SCM instrument, and contributed significantly to the original dynamic power spectra and wave polarization IDL code and examples.

Funding

The core SPEDAS team acknowledges support from NASA contract NNG17PZ01C (for SPEDAS community support) to UCLA, contract NNG04EB99C (as subcontract from SwRI for MMS SPEDAS plug-in development) to UCLA, and contract NAS5-02099 (THEMIS support of TDAS maintenance and SPEDAS infrastructure) to UCB, UCLA, and BU.

Acknowledgments

We would like to acknowledge the developers of the PyTplot, numpy, cdflib, netCDF4, hapiclient, astropy, and requests packages, as well as Sheng Tian for the geopack package.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fspas.2022.1020815/full#supplementary-material

References

Angelopoulos, V., Cruce, P., Drozdov, A., Grimes, E. W., Hatzigeorgiu, N., King, D. A., et al. (2019). The space physics environment data analysis system (spedas). *Space Sci. Rev.* 215, 9. doi:10.1007/s11214-018-0576-4

Blake, J. B., Mauk, B. H., Baker, D. N., Carranza, P., Clemmons, J. H., Craft, J., et al. (2016). The fly's eye energetic particle spectrometer (feeps) sensors for the magnetospheric multiscale (mms) mission. *Space Sci. Rev.* 199, 309–329. doi:10.1007/s11214-015-0163-x

Burch, J. L., Moore, T. E., Torbert, R. B., and Giles, B. L. (2016). Magnetospheric multiscale overview and science objectives. *Space Sci. Rev.* 199, 5–21. doi:10.1007/s11214-015-0164-9

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—An overview of python in heliophysics. *J. Geophys. Res. Space Phys.* 123. doi:10.1029/2018JA025877

Dunlop, M. W., Dong, X.-C., Wang, T.-Y., Eastwood, J. P. S. H., Yang, Y.-Y., Haaland, S., et al. (2021). Curlometer technique and applications. *JGR. Space Phys.* 126. doi:10.1029/2021JA029538

Hapgood, M. (1992). Space physics coordinate transformations: A user guide. *Planet. Space Sci.* 40, 711–717. doi:10.1016/0032-0633(92)90012-D

Matsuoka, A., Teramoto, M., Imajo, S., Kurita, S., Miyoshi, Y., and Shinohara, I. (2018). The mgf instrument level-2 high-resolution magnetic field data of exploration of energization and radiation in geospace (erg) arase satellite. doi:10.34515/DATA.ERG-06000

Pollock, C., Moore, T., Jacques, A., Burch, J., Gliese, U., Saito, Y., et al. (2016). Fast plasma investigation for magnetospheric multiscale. *Space Sci. Rev.* 199, 331–406. doi:10.1007/s11214-016-0245-4

Runov, A., Nakamura, R., Baumjohann, W., Treumann, R. A., Zhang, T. L., Volwerk, M., et al. (2003). Current sheet structure near magnetic x-line observed by cluster. *Geophys. Res. Lett.* 30, 1579. doi:10.1029/2002GL016730

Samson, J. C., and Olson, J. V. (1980). Some comments on the descriptions of the polarization states of waves. *Geophys. J. Int.* 61, 115–129. doi:10.1111/j.1365-246X.1980.tb04308.x

Tsyganenko, N. A. (2013). Data-based modelling of the earth's dynamic magnetosphere: A review. *Ann. Geophys.* 31, 1745–1772. doi:10.5194/angeo-31-1745-2013

Weygand, J., Amm, O., Viljanen, A., Angelopoulos, V., Murr, D., Engebretson, M. J., et al. (2011). Application and validation of the spherical elementary currents systems technique for deriving ionospheric equivalent currents with the north American and Greenland ground magnetometer arrays. *J. Geophys. Res.* 116. doi:10.1029/2010JA016177

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Original Research
PUBLISHED 19 October 2022
DOI 10.3389/fspas.2022.1005672

Check for updates

OPEN ACCESS

EDITED BY
John Coxon,
Northumbria University,
United Kingdom

REVIEWED BY
Denny Oliveira,
University of Maryland, Baltimore County, United States
James Plank,
University of Southampton,
United Kingdom

*CORRESPONDENCE
D. Trotta,
d.trotta@imperial.ac.uk

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 28 July 2022
ACCEPTED 30 September 2022
PUBLISHED 19 October 2022

CITATION
Trotta D, Vuorinen L, Hietala H, Horbury T, Dresing N, Gieseler J, Kouloumvakos A, Price D, Valentini F, Kilpua E and Vainio R (2022), Single-spacecraft techniques for shock parameters estimation: A systematic approach.
Front. Astron. Space Sci. 9:1005672.
doi: 10.3389/fspas.2022.1005672

COPYRIGHT
© 2022 Trotta, Vuorinen, Hietala, Horbury, Dresing, Gieseler, Kouloumvakos, Price, Valentini, Kilpua and Vainio. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Single-spacecraft techniques for shock parameters estimation: A systematic approach

D. Trotta[1]*, L. Vuorinen[2], H. Hietala[1,3], T. Horbury[1], N. Dresing[2], J. Gieseler[2], A. Kouloumvakos[4], D. Price[5], F. Valentini[6], E. Kilpua[5] and R. Vainio[2]

[1]The Blackett Laboratory, Department of Physics, Imperial College London, London, United Kingdom, [2]Department of Physics and Astronomy, University of Turku, Turku, Finland, [3]School of Physics and Astronomy, Queen Mary University of London, London, United Kingdom, [4]Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD, United States, [5]Department of Physics, University of Helsinki, Helsinki, Finland, [6]Dipartimento di Fisica, Universita' della Calabria, Rende, Italy

Spacecraft missions provide the unique opportunity to study the properties of collisionless shocks utilising *in situ* measurements. In the past years, several diagnostics have been developed to address key shock parameters using time series of magnetic field (and plasma) data collected by a single spacecraft crossing a shock front. A critical aspect of such diagnostics is the averaging process involved in the evaluation of upstream/downstream quantities. In this work, we discuss several of these techniques, with a particular focus on the shock obliquity (defined as the angle between the upstream magnetic field and the shock normal vector) estimation. We introduce a systematic variation of the upstream/downstream averaging windows, yielding to an ensemble of shock parameters, which is a useful tool to address the robustness of their estimation. This approach is first tested with a synthetic shock dataset compliant with the Rankine-Hugoniot jump conditions for a shock, including the presence of noise and disturbances. We then employ self-consistent, hybrid kinetic shock simulations to apply the diagnostics to virtual spacecraft crossing the shock front at various stages of its evolution, highlighting the role of shock-induced fluctuations in the parameters' estimation. This approach has the strong advantage of retaining some important properties of collisionless shock (such as, for example, the shock front microstructure) while being able to set a known, nominal set of shock parameters. Finally, two recent observations of interplanetary shocks from the Solar Orbiter spacecraft are presented, to demonstrate the use of this systematic approach to real events of shock crossings. The approach is also tested on an interplanetary shock measured by the four spacecraft of the Magnetospheric Multiscale (MMS) mission. All the Python software developed and used for the diagnostics (SerPyShock) is made available for the public, including an example of parameter estimation for a shock wave recently observed *in-situ* by the Solar Orbiter spacecraft.

KEYWORDS

space physics, shock waves, collisionless shocks, spacecraft data, heliosphere, python

1 Introduction

The understanding of the processes controlling our Universe has always been intimately related to our capability to observe the large ensemble of phenomena taking place in astrophysical systems. Since the last century, our knowledge of the Universe has grown tremendously, thanks to the exceptional technological advances which made us able to observe it with great levels of detail. Such observational advances, together with important theoretical/modelling breakthroughs, reveal that some phenomena are universal, i.e., common to many astrophysical environments.

Among these universal phenomena, collisionless shock waves, i.e., abrupt transitions from super-magnetosonic (upstream) and sub-magnetosonic (downstream) flows, are particularly important to address various aspect of energy conversion in many astrophysical systems, ranging from solar flares (e.g., Benz, 2008) to the largest (Mpc) scales of galaxy cluster merging (see Brunetti and Jones, 2014, for a review).

In the late 1950s, thanks to the possibility of spacecraft flight in interplanetary space, the first collisionless shocks were observed in the heliosphere (see Kivelson and Russell, 1995, for a comprehensive introduction). The possibility to carry out *in-situ* shock observations has then become fundamental to address many of their properties and behaviour.

Shocks that are routinely observed in the heliosphere can be divided in several categories, namely interplanetary (IP) shocks, generated as a consequence of solar activity phenomena, such as Coronal Mass Ejections (CME) and Stream Interaction Regions (SIR) (Dessler and Fejer, 1963; Gosling et al., 1974), and planetary bow shocks, resulting from the interaction between the supersonic solar wind and planetary bodies behaving as obstacles for the solar wind flow (e.g., Dungey, 1979; Hoppe and Russell, 1982; Lepping, 1984). Other examples of shocks observed *in-situ* include the *Voyager* observations of the heliospheric termination shock at the interface between the heliosphere and the interstellar environment (e.g., McComas et al., 2019) and cometary bow shocks (Thomsen et al., 1986; Coates et al., 1997; Naeem et al., 2020). Despite the important differences between these different types of shocks, they share their fundamental, underlying physics.

The shock structure and behaviour is regulated by several parameters, the most important being the angle between the shock normal direction and the upstream magnetic field, θ_{Bn}. When θ_{Bn} approaches 90°, the shock is quasi-perpendicular, i.e., the upstream magnetic field is along the shock front. On the other hand, for θ_{Bn} values close to 0° (corresponding to an upstream magnetic field almost normal to the shock surface), the shock is quasi-parallel. Particle reflection and propagation far upstream is favoured at quasi-parallel shocks (Kennel et al., 1985), introducing the possibility for reflected particles to interact with the upstream plasma over long distances, creating unstable distributions and a collection of disturbances in the plasma properties. Other important parameters for the shock behaviour are the shock Alfvénic and fast magnetosonic Mach numbers, i.e., the ratio between the shock speed in the upstream flow frame and the upstream Alfvén and fast magnetosonic speed, respectively ($M_A \equiv v_{sh}/v_A$ and $M_{fms} \equiv v_{sh}/v_{fms}$) (see Burgess and Scholer, 2015, for an extensive review).

Since the early *Pioneer* evidences of the Earth's bow shock existence (Dungey, 1979) to modern missions crossing various kind of heliospheric shocks (e.g., Masters et al., 2008; Blanco-Cano et al., 2016), various techniques to extract shock parameters as the ones described above from single spacecraft observations have been developed, starting from various theoretical formulations to describe the behaviour of the plasma properties across shock transitions (Lepping and Argentiero, 1971; Abraham-Shrauner, 1972; Viñas and Scudder, 1986). Further advancements on such investigations have been made using multiple satellite missions, such as *Cluster* and the Magnetospheric MultiScale (MMS) (Escoubet et al., 2001; Burch et al., 2016). Using multi-spacecraft measurements, important properties of shock transitions have been discovered (e.g., Johlander et al., 2016; Kajdič et al., 2019), and new parameter estimation techniques relying on multiple crossings have been developed (e.g., Russell et al., 1983a,b). However, many modern spacecraft missions, for example the recently launched Solar Orbiter mission, rely on a single spacecraft (Muller et al., 2020), making it important to understand the limits and advantages of shock parameters estimation techniques relying on the study of a single time series collected by the instrumentation on board of spacecraft during a shock crossing.

The understanding of collisionless plasmas in several environments, ranging from controlled fusion devices to astrophysical systems, has always been intimately related to the use of computer simulations, due to the complexity of the problems studied (e.g., Birdsall and Langdon, 1991). In fact, another crucial source of knowledge around collisionless shocks comes from simulation studies. Collisionless shocks have been modelled using several different methods, ranging from local and global magnetohydrodynamic (MHD) simulations studying the large scale properties of shock transitions (e.g., Mignone et al., 2007; Mejnertsen et al., 2018), to local simulations including kinetic effects, addressing the shock structuring and behaviour at small scales (e.g., Caprioli and Spitkovsky, 2014; Ha et al., 2022). Combining numerical and observational efforts, has often proved to be very effective for the understanding of shock behaviour (e.g., Sundberg et al., 2016).

As we shall see in the below discussion, most shock parameter estimation techniques from spacecraft data involve an operation of averaging plasma quantities upstream/downstream of the shock crossing, making the results particularly sensitive to the choice of averaging windows. The idea to adopt an ensemble-based approach for such choices of averaging windows was first suggested by Balogh et al. (1995),

looking at magnetic field Ulysses observations of interplanetary shocks.

In this work, we review some of these techniques, proposing a systematic way of varying the averaging windows used for such parameter estimations. The importance and implications of this approach are discussed, and the approach is then tested on a model shock. We then propose the important, novel test of these parameter estimation techniques on a shock simulated using state-of-the-art, self-consistent plasma simulations, in which we have the opportunity to investigate the shock geometry at different stages of its evolution. Finally, two different real world observations are presented, obtained in late 2021 by the Solar Orbiter spacecraft, and two applications of the systematic shock parameters estimation are shown.

Recently, the importance to share tools for data analysis is object of discussion across several scientific communities. A virtuous example of such an effort in the space physics community is the publicly available software for spacecraft data analysis irfu-matlab (https://github.com/irfu/irfu-matlab). We make the Python software developed in this work and used for the systematic approach to *in-situ* shock analyses (SerPyShock) publicly available at https://github.com/trottadom/SerPyShock. This code release includes the routines for the analysis, an interactive test case for the Rankine-Hugoniot compliant shocks shown below, and an example of analysis on a Solar Orbiter dataset, for which Python data loaders are also provided. The software is written using the PEP8 standard, in Python 3.9 and requires a minimal number of standard dependencies, e.g., numpy and scipy, listed in the repository. Future updates to the software will include further toold of data analysis, such as for example an algorithm for particle/wave foreshock identification.

The paper is organised as follows: a brief summary of the theoretical framework is presented in Section 2; in Section 3, the main results of this work are shown, discussing the analytical, simulation and observational cases in Sections 3.1–3.3, respectively; the conclusions are in Section 4. In this work, particular emphasis is given to addressing the shock obliquity and in particular the computation of the θ_{Bn} angle. Software to compute other parameters is also provided and briefly discussed in the Appendix.

2 Theoretical framework

Below, we briefly review the theoretical framework underlying the parameter estimation techniques object of this work. To do so, we closely follow Chapter 10 of Paschmann and Schwartz (2000), Burgess and Scholer (2015) and Hietala (2012).

This discussion starts from a widely used approach to model the plasma properties across a shock, namely the Rankine-Hugoniot jump conditions (Rankine, 1870; Hugoniot, 1887). An MHD model is built for the shock, treated as an abrupt transition between the convected upstream plasma (supersonic) to downstream (subsonic). Here, the term "abrupt" may seem slightly vague, but is inspired by the consideration that the thickness of the transition must be related to a typical dissipation scale, that is much smaller than the typical convection scale in high temperature plasmas.

Let us consider the shock as a discontinuity and move into the shock rest frame (i.e., a reference frame in which the shock transition has speed equal to zero). In this frame, two fundamental assumptions are made, namely the fact that the shock front is spatially uniform, and time stationary. If we imagine the transition from supersonic to subsonic flow happening along the x direction of a cartesian system, the above assumptions imply that $\partial/\partial t = \partial/\partial y = \partial/\partial z = 0$. Furthermore, due to the above, the shock thickness is considered to be infinitesimal in this model. At this point, the MHD equation for mass, momentum and energy conservation, and the divergence-free condition for the magnetic field are considered and integrated, yielding to the following relations:

$$[\rho_m V_n] = 0 \tag{1}$$

$$\left[\rho_m V_n^2 + P + \frac{B_t^2}{2\mu_0}\right] = 0 \tag{2}$$

$$\left[\rho_m \mathbf{V}_t V_n + P + \frac{\mathbf{B}_t B_n}{\mu_0}\right] = 0 \tag{3}$$

$$\left[\frac{1}{2}\rho_m V^2 V_n + \frac{\gamma}{\gamma-1} P V_n + \frac{B_t^2}{\mu_0} V_n - \frac{\mathbf{V}_n \cdot \mathbf{B}_t}{\mu_0} B_n\right] = 0 \tag{4}$$

$$[B_n] = 0 \tag{5}$$

$$[\mathbf{V}_t B_n - \mathbf{B}_t V_n] = 0. \tag{6}$$

Where ρ_m is the mass density, $\mathbf{V}$ is the bulk flow speed, P is the scalar thermal pressure of the plasma, $\mathbf{B}$ the magnetic field. γ and μ_0 are the polytropic index and the vacuum permeability, respectively. Here, the subscripts n and t indicate that the quantity is considered normal to the shock and transverse to it, respectively. In the equations above, the symbols $[\ldots]$ indicate the difference between downstream and upstream quantities, e.g., $[F] = F_2 - F_1$ for a generic field F, where the subscript 2 and 1 indicate that the field is evaluated downstream and upstream, respectively. The Rankine-Hugoniot relations 1-6 are of crucial importance to address shock properties in a variety of systems.

We now consider the shock geometry to be oblique (i.e., $\theta_{Bn} \neq 90^\circ, 0^\circ$), and move in de Hoffmann-Teller frame, i.e., a frame aligned to the shock normal that moves at a speed such that the upstream convective electric field ($\mathbf{E}_1 = -\mathbf{V}_1 \times \mathbf{B}_1$) vanishes (de Hoffmann and Teller, 1950). Such a frame is extremely important for shock physics, especially to understand various properties of accelerated particles across shock transitions (e.g., Leroy and Mangeney, 1984).

Unrolling the [. . .] operation in Eqs 1–6 for an oblique shock in the de Hoffmann-Teller frame, the Rankine-Hugoniot jump conditions become:

$$\frac{\rho_{m2}}{\rho_{m1}} = \mathrm{r_{gas}} \tag{7}$$

$$\frac{V_{n2}}{V_{n1}} = \frac{1}{\mathrm{r_{gas}}} \tag{8}$$

$$\frac{V_{t2}}{V_{t1}} = \frac{M_{A1}^2 - 1}{M_{A1}^2 - \mathrm{r_{gas}}} \tag{9}$$

$$\frac{B_{n2}}{B_{n1}} = 1 \tag{10}$$

$$\frac{B_{t2}}{B_{t1}} = r\frac{M_{A1}^2 - 1}{M_{A1}^2 - \mathrm{r_{gas}}} \tag{11}$$

$$\frac{P_2}{P_1} = \mathrm{r_{gas}} + \frac{(\gamma - 1)\mathrm{r_{gas}} V_1^2}{2V_S 1}\left(1 - \frac{V_2^2}{V_1^2}\right). \tag{12}$$

Here, two important shock parameters arise: $\mathrm{r_{gas}}$, i.e., the shock gas compression ratio (notation has been chosen to distinguish it from the shock magnetic compression ratio $\mathrm{r_B} \equiv \frac{B_2}{B_1}$) and $M_{A1} \equiv V_{n1}\sqrt{\mu_0 \rho_{m1}}/B_{n1}$. V_{S1} is the upstream sound speed. As it is known, several sound speeds are admitted in the MHD formulation for wave propagation, and indeed the MHD system admits three different types of shocks, namely slow, fast and intermediate. We focus here on fast shocks, and refer the reader to Burgess and Scholer (2015) for a detailed discussion.

At this point, the Rankine-Hugoniot conditions Eqs 7–12 can be used to address the shock geometry. This is done in conjunction with the use of the coplanarity theorem, stating that the upstream, downstream magnetic field and shock normal vectors lie all in the same plane (see Eqs 10–11). It is with such considerations in mind that we introduce the first diagnostic for computing the shock normal, namely the magnetic coplanarity (MC) method (Colburn and Sonett, 1966):

$$\hat{n}_{\mathrm{MC}} = \pm \frac{(\mathbf{B}_2 \times \mathbf{B}_1) \times \Delta\mathbf{B}}{|(\mathbf{B}_2 \times \mathbf{B}_1) \times \Delta\mathbf{B}|}. \tag{13}$$

As it can be seen, this diagnostic relies on upstream/downstream magnetic field estimation only, with $\Delta\mathbf{B} = \mathbf{B}_2 - \mathbf{B}_1$. The shock normal sign obtained with such a method is arbitrary, and it is adjusted by convention such that $\hat{n}$ points upstream, hence the ± sign. Through a corollary of the coplanarity theorem, it is possible to see that the vector $\Delta\mathbf{V}$, i.e., the difference between downstream and upstream velocity lies also in the same plane as $\mathbf{B}_2$, $\mathbf{B}_1$ and $\hat{n}$ (see Paschmann and Schwartz, 2000). Using this corollary, it is possible to introduce the following additional methods for the shock normal estimation, namely the three *mixed mode* (MX1, MX2, MX3) methods. These combine magnetic field and bulk flow speed measurements across the shock transition:

$$\hat{n}_{\mathrm{MX1}} = \pm \frac{(\mathbf{B}_1 \times \Delta\mathbf{V}^{\mathrm{arb}}) \times \Delta\mathbf{B}}{|(\mathbf{B}_1 \times \Delta\mathbf{V}^{\mathrm{arb}}) \times \Delta\mathbf{B}|} \tag{14}$$

$$\hat{n}_{\mathrm{MX2}} = \pm \frac{(\mathbf{B}_2 \times \Delta\mathbf{V}^{\mathrm{arb}}) \times \Delta\mathbf{B}}{|(\mathbf{B}_2 \times \Delta\mathbf{V}^{\mathrm{arb}}) \times \Delta\mathbf{B}|} \tag{15}$$

$$\hat{n}_{\mathrm{MX3}} = \pm \frac{(\Delta\mathbf{B} \times \Delta\mathbf{V}^{\mathrm{arb}}) \times \Delta\mathbf{B}}{|(\Delta\mathbf{B} \times \Delta\mathbf{V}^{\mathrm{arb}}) \times \Delta\mathbf{B}|}, \tag{16}$$

Where $\Delta\mathbf{V}^{\mathrm{arb}} \equiv \mathbf{V}_2 - \mathbf{V}_1$, and the arb superscript indicates that the velocity difference is calculated in an arbitrary reference frame. Mixed mode methods have been extremely popular in addressing shock geometries in previous literature (e.g., Volkmer and Neubauer, 1985; Balogh et al., 1995; Kilpua et al., 2015), including a recent effort looking at geomagnetic activity triggered by shocks with different inclinations (Oliveira and Samsonov, 2018). There are also other methods, which are not discussed in this work, to estimate the shock normal. These are, for example, the velocity coplanarity method (Abraham-Shrauner, 1972) and the minimum variance analysis (Sonnerup et al., 1967). Another class of diagnostics not discussed here are the ones dealing with multi-spacecraft observations (Paschmann and Schwartz, 2000).

Even though we focus mostly on techniques estimating shock geometry, an algorithm for shock speed estimation is here reviewed (and included in the SerPyShock software released within this work), namely the mass flux algorithm, defined as:

$$V_{\mathrm{sh}}^{\mathrm{arb}} = \frac{\Delta(\rho \mathbf{V}^{arb})}{\Delta\rho} \cdot \hat{n}. \tag{17}$$

The definition above, obtained applying the mass flux conservation equation across the shock discontinuity, yields to an estimation of the shock speed along the shock normal. As above, the Δ symbol indicates that a difference between the downstream and the upstream quantity is performed. An important limitation of the mass flux algorithm is the fact that is reliant on plasma measurements only. An example of shock speed estimation using the mass flux algorithm, systematically varying the averaging windows, as discussed in detail in this work in the context of shock geometry estimation, is shown in the Appendix of this manuscript. Finally, we note that an interesting approach to parameter estimation, using the so-called modified Rankine-Hugoniot shock fitting technique, was proposed by Viñas and Scudder (1986) and since applied successfully to spacecraft data (Koval and Szabo, 2008), yielding to a simultaneous estimation of shock normal and speed.

3 Methodology and results

Spacecraft measured quantities consist of time-series data streams. As discussed above, all the methodologies discussed so far rely on a crucial choice, namely the extent of upstream/downstream averaging windows for the quantities involved in the diagnostics. Clearly, the parameters deducted from the observations will depend on the averaging windows sizes, and

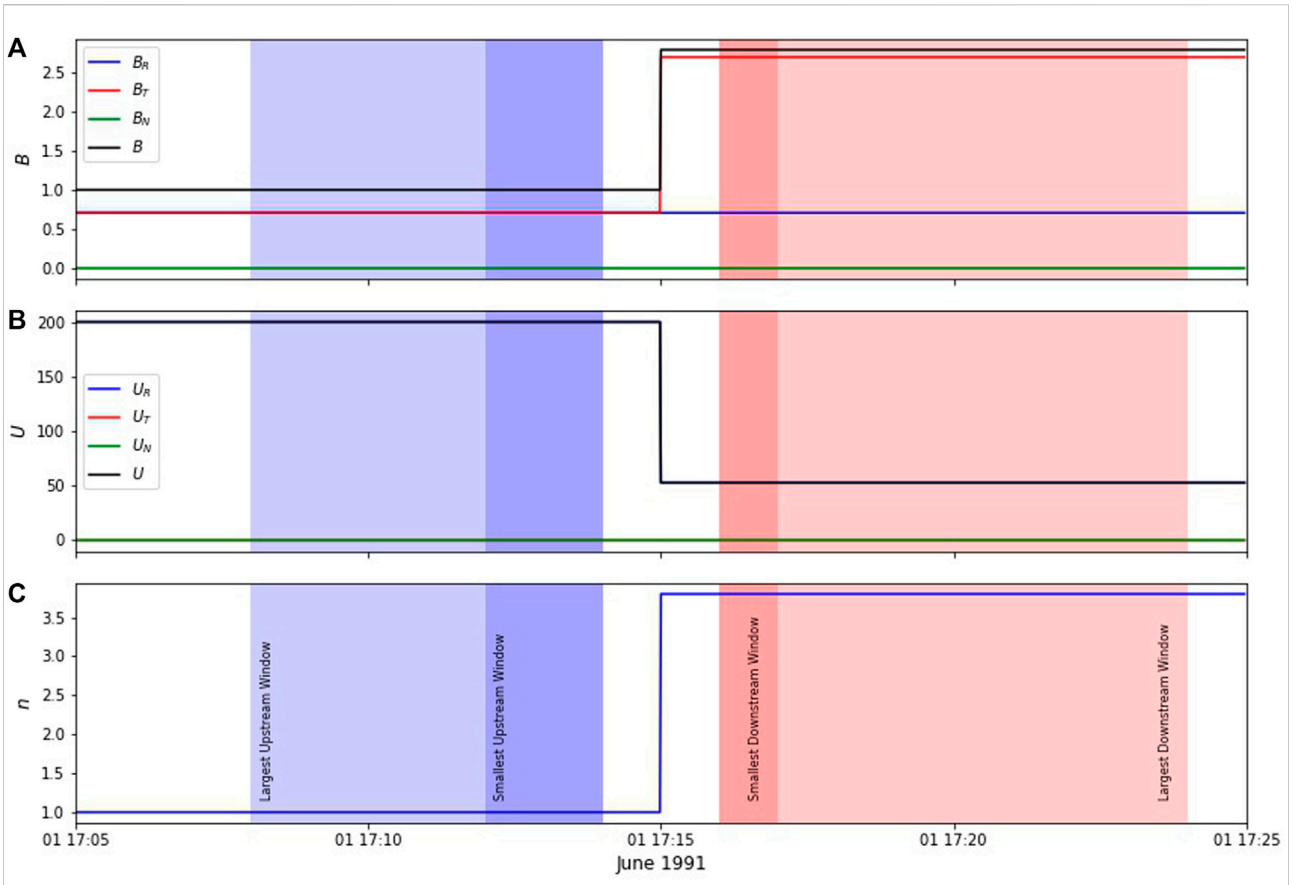


FIGURE 1
Synthetic measurements for a Rankine-Hugoniot compliant shock with θ_{Bn} = 45° and gas compression ratio r_{gas} = 3.8. Panel show magnetic field magnitude and components **(A)**, plasma bulk flow speed **(B)** and density **(C)**. The blue (red) shaded areas highlight the windows chosen for upstream (downstream) averaging.

the parameter estimation is robust when the results do not change for different window choices. Such approach is here formalised and carried out systematically, then tested on different theoretical and observational examples.

3.1 Rankine-Hugoniot compliant synthetic field

For the introduction and first testing of our approach, a synthetic spacecraft measurement was generated using the Rankine-Hugoniot jump conditions discussed in Section 2.

Results are shown in Figure 1, where such synthetic measurements for a Rankine-Hugoniot compliant shock with $\theta_{Bn} = 45^{\circ}$ and gas compression ratio $r_{gas} = 3.8$ is represented. As it can be seen from magnetic field, bulk flow speed and density time-series measurements (top, middle and bottom of Figure 1, respectively), the shock transition is such that the upstream is on the left-hand side of the Figure, i.e. before 17:15 of 1 June 1991, and the downstream is represented by the timeseries after 17:15 (obviously, time here has no physical meaning).

The parameter estimation is performed as follows: a smallest possible averaging window is chosen for upstream and downstream. Given the discussion about the Rankine-Hugoniot relations assuming the shock transition as infinitesimal, the idea is to choose such windows as close as possible to the shock front, without including it in the averaging process. Care must be taken in excluding also the shock foot in the very close upstream, as well as the downstream overshoot. When applying these diagnostics, as it will be discussed later concerning real observations, the choice for smallest/largest upstream window will depend on many different factors, e.g., the amount of disturbances that may be present upstream/downstream of the shock and the resolution available for the measurements. Another advice is that it is preferable to choose the smallest (largest) averaging windows to be of comparable scale sizes, thus avoiding systematic mixing of very different scales.

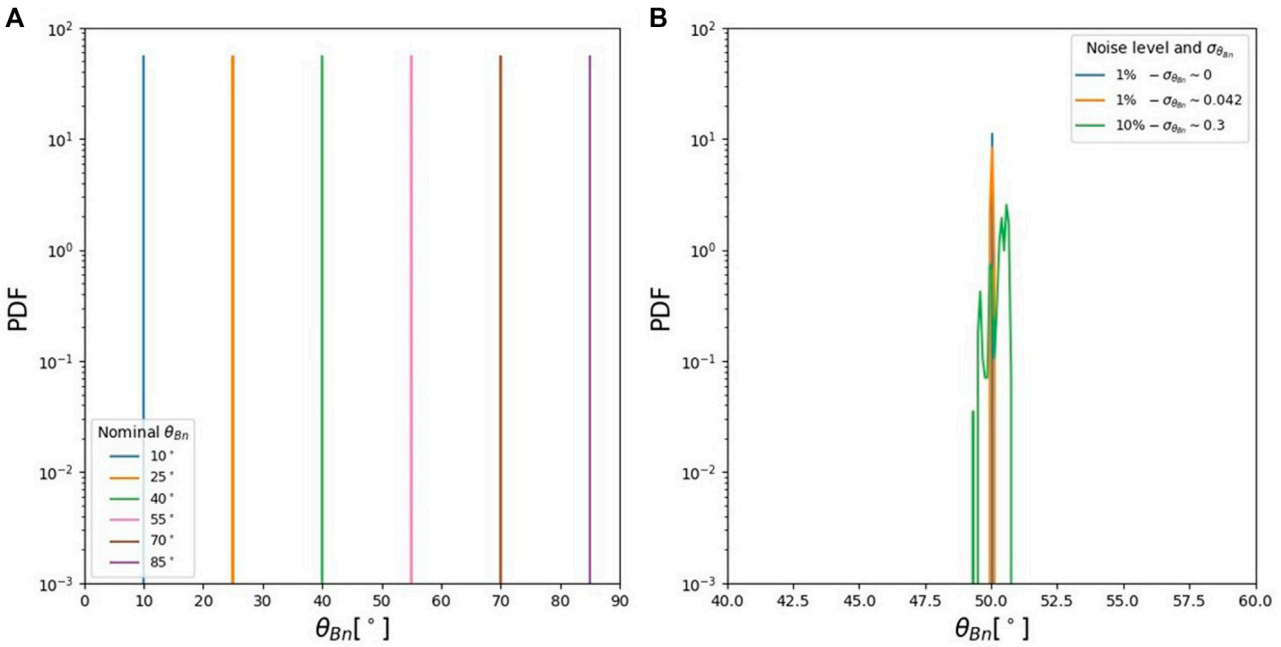


FIGURE 2
(A) PDF distributions for θ_{Bn} computed using magnetic coplanarity and mixed mode techniques, for six different synthetic, Rankine-Hugoniot compliant shocks. **(B)** PDF distributions of θ_{Bn} for three shocks with the same nominal geometry and different levels of added noise.

In Figure 1, the smallest upstream (downstream) averaging windows are shown with the dark shaded blue (red) panels. Then, after choosing an appropriate cadence, the smallest averaging windows are extended to the largest, and overlapped with the smallest windows. For each couple of upstream/downstream windows, a shock normal (and subsequently θ_{Bn} value) is evaluated. In this way, an ensemble of shock parameters is computed, and it is possible to address the robustness of the parameters estimation by looking at how is the ensemble distributed, as we shall see below. Note that, since we add duration to the smallest upstream/downstream windows, the measurements close to the shock are counted multiple times, ensuring that the calculation is carried out as close as possible to the shock front at all times.

A first example of such a systematic approach is shown below. We considered 6 different Rankine-Hugoniot compliant shocks with different θ_{Bn} (10, 25, 40, 55, 70 and 85°, respectively), with the same approach described above. A collection of about 1,000 different windows have then been used to compute the shock geometry.

The results of this experiment have been reported in Figure 2A, where the probability density functions (PDFs) for the computed θ_{Bn} values are shown. Such PDFs, as expected, are peaked (within machine precision) at the nominal θ_{Bn} value for each shock. This test is useful for purposes of routine testing, as well as for a sanity check of the code that is then used for real-world cases. Furthermore, this case and the code used to generate the Ranking-Hugoniot compliant shocks is released as an interactive test for the SerPyShock user.

At this stage, the robustness of the approach has been tested on slightly more complicated case. Three synthetic shocks have been generated, all of them with the same $\theta_{Bn} = 50^\circ$, considering the baseline case of Figure 2A with other two cases where white noise of 1% and 10% level was added, respectively (Figure 2B). In this case, it can be seen that when noise is added, the PDF distribution of θ_{Bn} values broadens, as it is particularly evident for the 10% noise level case (green line in Figure 2B). Another interesting feature of this method is that it is possible to characterise the spread of this PDFs computing, together with the average value $\langle\theta_{Bn}\rangle$, its standard deviation $\sigma_{\theta_{Bn}}$, which becomes a measure for the sensitivity of the parameter estimation to the choice of upstream/downstream window and, in other words, an uncertainty for the parameter estimation. It is possible to note that the broadening observed here is rather small (with a maximum discrepancy of about 1° from the nominal shock geometry), due to the fact that the averaging process averages out the white noise. Furthermore, it is possible to chack if the broadening of the distribution follows a gaussian profile by computing the skewness of the PDFs of shock parameters. In the added noise case of these synthetic shocks, a very small values of skewness is always found, with the Fisher-Pearson coefficients of skewness (see Kokoska and Zwillinger,

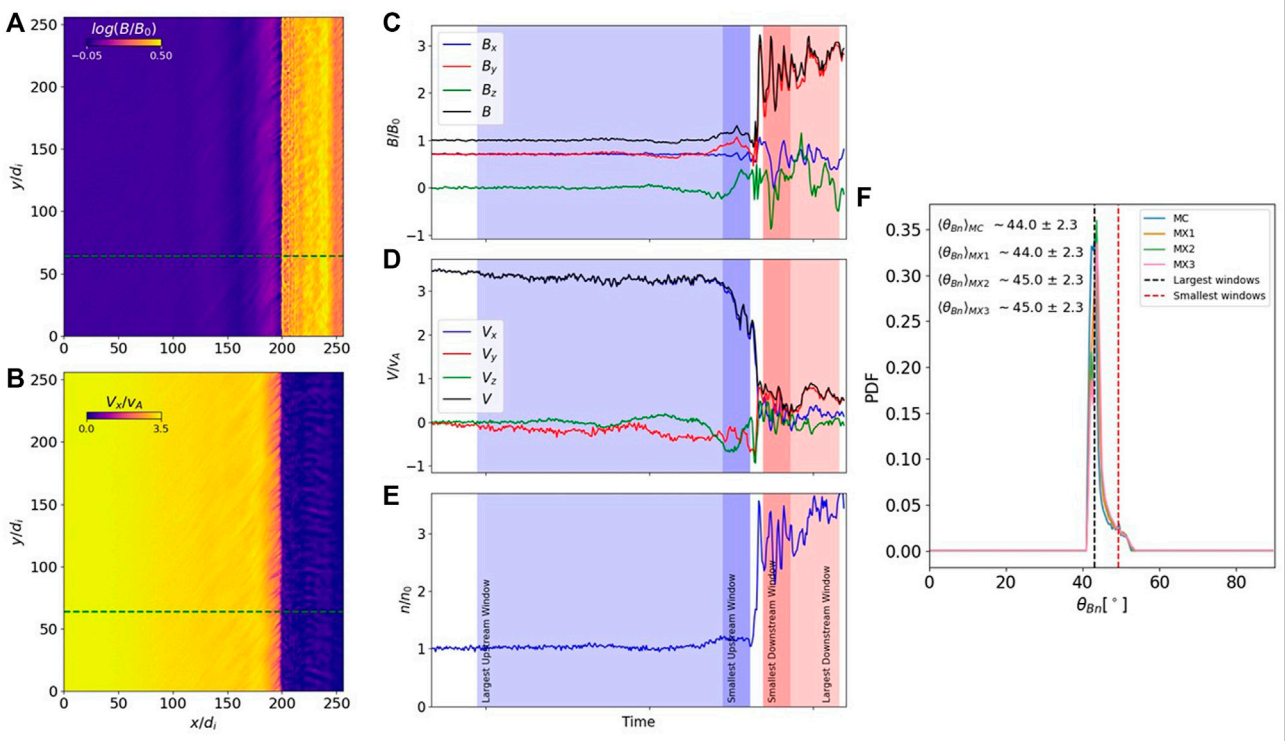


FIGURE 3
Snapshot of Hybrid PIC simulation of 45° degrees shock captured at simulation time $T\Omega_{ci} = 40$. Panels **(A)** and **(B)** show magnetic field magnitude and proton bulk flow speed along the shock nominal shock normal direction, respectively. In the middle panels, are shown one-dimensional signals of magnetic field and its components **(C)**, proton bulk flow speed magnitude and its components **(D)** and proton density **(E)** as seen by a virtual spacecraft crossing the shock along the green-shaded line in panels **(A,B)**. In these panels, the smallest/largest averaging windows used for the upstream/downstream regions are highlighted (red/blue shaded panels). Finally, PDF distributions of θ_{Bn} obtained with MC, MX1, MX2 and MX3 methods are shown in **(F)**.

1999) being $g_1 = 0.0, 2 \times 10^{-3}$ and 3×10^{-2} for the 0%, 1% and 10% noise cases, respectively.

3.2 Hybrid PIC simulations

Here, we test the methods to compute the shock obliquity with the systematic window variation approach on a self-consistent, kinetic simulation of a supercritical shock. For this experiment, we use the HYPSI code, successfully used in the past for several studies addressing collisionless shocks (e.g., Trotta and Burgess, 2019; Preisser et al., 2020a; Trotta et al., 2021).

In the simulation, protons are modelled as macroparticles and advanced using the standard PIC method. The electrons, on the other hand, are modelled as a massless, charge-neutralizing fluid with an adiabatic equation of state. The HYPSI code is based on the Current Advance Method and Cyclic Leapfrog (CAM-CL) algorithm (Matthews, 1994). The shock is initiated by the injection method (Quest, 1985), in which the plasma flows in the x-direction with a defined (super-Alfvénic) velocity $V_{\rm in}$. The right-hand boundary of the simulation domain acts as a reflecting wall, and at the left-hand boundary plasma is continuously injected. The simulation is periodic in the y direction. A shock is created as a consequence of reflection at the wall, and it propagates in the negative x-direction. In the simulation frame, the (mean) upstream flow is along the shock normal.

In the hybrid simulations, distances are normalised to the ion inertial length $d_i \equiv c/\omega_{pi}$, times to the inverse cyclotron frequency ${\Omega_{ci}}^{-1}$, velocity to the Alfvén speed v_A (all referred to the upstream state), and the magnetic field and density to their upstream values, B_0 and n_0, respectively. For the upstream flow velocity, a value of $V_{\rm in} = 3.5 v_A$ has been chosen, and the resulting Alfvénic Mach number of the shock is approximately $M_A \sim 5$. The upstream ion distribution function is an isotropic Maxwellian and the ion β_i is 1. The simulation $x - y$ domain is $256 \times 256\ d_i$, and the simulation is therefore 2.5-dimensional. The spatial resolution used is $\Delta x = \Delta y = 0.5\ d_i$. The system is evolved for 70 Ω_{ci}^{-1}, with a time step for particle (ion) advance of $\Delta t = 0.01\ \Omega_{ci}^{-1}$. Substepping is used for the magnetic field advance, with an effective time step of $\Delta t_B = \Delta t/10$. A small, nonzero resistivity is introduced in the magnetic induction equation, with a value

that is chosen such that there are not excessive fluctuations at the grid scale. The number of particles per cell used is always greater than 500 (upstream), in order to keep the statistical noise characteristic of PIC simulations to a reasonable level. For the simulation studied here, the upstream magnetic field is in the x-y plane, resulting in a nominal θ_{Bn} of the shock of 45°. As we shall see below, this is an excellent test case for our method, but analogous results can be obtained for different nominal shock geometries.

Figure 3 shows an overview of the shock simulation, at time $T\Omega_{ci} = 40$. Panels (a) and (b) highlight the main features of the shock transition, showing magnetic field magnitude and proton bulk flow speed along the $\hat{x}$-direction, i.e., the shock normal nominal direction. Here, it can be seen that the shock front appears marginally stable, with small scale rippling due to its supercritical nature (e.g., Johlander et al., 2016). The presence of reflected particles streaming away from the shock front in the upstream region induces the upstream fluctuations in the vicinity of the shock. The shock transition is then shown through the one-dimensional plots in panels (c)-(e), that represent horizontal slices through the simulation domain along the green lines in panels (a) and (b). Such one-dimensional signals are the ones that a (virtual) spacecraft would observe while crossing the shock transition, provided that the spatial information can be translated into time information through the so-called Taylor hypothesis (Taylor, 1938), widely used in the context of space plasma observations (see Perri et al., 2017, for a detailed study of its validity).

At this point, the shock obliquity is calculated with the method described above, for the one-dimensional simulated spacecraft trajectory. The smallest and largest windows used for the upstream/downstream averaging are displayed using the shaded areas of Figure 3C–E. The result of the analysis is displayed in Figure 3F, where the PDF distributions of the θ_{Bn} values obtained with MC and MX1-2-3 techniques are shown. The errors shown together with the $\langle\theta_{Bn}\rangle$ in Figure 3F correspond to the standard deviation $\sigma_{\theta_{Bn}}$ for each estimation. Here, in contrast with the analytical examples shown above, a wider distribution of θ_{Bn} values is observed. Importantly, the PDF distributions yield to an average θ_{Bn} value of 45°, as set up in the simulation initialisation. In this case, the MC and MX methods agree very well. The spread of the distribution extending to higher values of θ_{Bn} is then due to the field oscillations present in the upstream/downstream shock regions. We would like to underline the importance of this example for the problem addressed in this work, from two points of view. First of all, it is demonstrated in numerical simulations, that the shock environment is often much more complicated with respect to the assumptions that go into the use of shock geometry estimation. Secondly, it is shown that to use the ensemble approach for the averaging windows choice is an optimal choice, as the result converge to the nominal shock geometry. Contrarily to this systematic approach, choosing two windows would only get one value for the θ_{Bn} measurement, and this may depart significantly to the average shock geometry. Such departures, due to shock front instabilities and various field-particle interaction, are interesting in their own respect, studied in recent literature (e.g., Kajdič et al., 2019; Preisser et al., 2020b).

From Figure 3, it is intuitive to link the spread of obtained θ_{Bn} values to the level of upstream/downstream fluctuations and disturbances. It is then natural to investigate how results change throughout different stages of the shock evolution. This is done in Figure 4, where the parameter estimation has been carried out at different simulation times. As the shock travels in the negative x-direction of the simulation domain, it is possible to see several microinstabilities being excited along the shock front. The presence of reflected particles, generating unstable upstream particle distributions, induce upstream fluctuations, more and more evident for later times, that are then convected downstream, generating a complex scenario for the shock transition.

When shock obliquity is addressed at the different shock evolution stages, the PDF distribution of the observed θ_{Bn} value broadens significantly. At $T\Omega_{ci} = 20$, with a quasi-laminar shock transition (Figure 4D), the PDF is strongly peaked at the nominal value of $\theta_{Bn} = 45°$, with a tail departing from it probably due to microinstabilities very close to the shock front. The scenario changes dramatically for the well-developed shock transition case $T\Omega_{ci} = 70$, where the PDF is spread across a very large range of values. The skewness coefficients for the three crossings are $g_1 = -1.5, -1.3, -0.43$, indicating that the asymmetry due to small scale instabilties is stronger at early times and decreases at $T\Omega_{ci} = 70$, where a larger spread of values is observed. Very importantly, the PDF distributions average very close to the nominal θ_{Bn} value, while the $\sigma_{\theta_{Bn}}$ increases for increasingly disturbed scenarios. This feature highlights the parameter estimation capabilities provided by the systematic variation of averaging windows.

To further investigate the dependence of the parameter distributions on the shock crossing performed by the virtual spacecraft, the inset of Figure 4E shows the total PDF of θ_{Bn} (black line) for the virtual spacecraft crossing in Figure 4B, together with the θ_{Bn} PDF obtained using only very small windows (max $\sim 3d_i$, red line). It is therefore clear that the bump in the distribution is due to the virtual spacecraft crossing the shock along a ripple that is rotated towards the upstream field, i.e., a quasi-parallel portion of the shock front. This information is then lost when averaging over larger scales. This analysis shows that the systematic variation of averaging windows may be leveraged to obtained further information about shock structuring. Future upgrades of our software will include the possibility to cross-correlate estimated parameters and upstream/downstream window length. However, it is worth underlining that the spacecraft observations provide a more complex scenario, in which the multi-dimensional picture of the shock is not available, and the presence of pre-existing

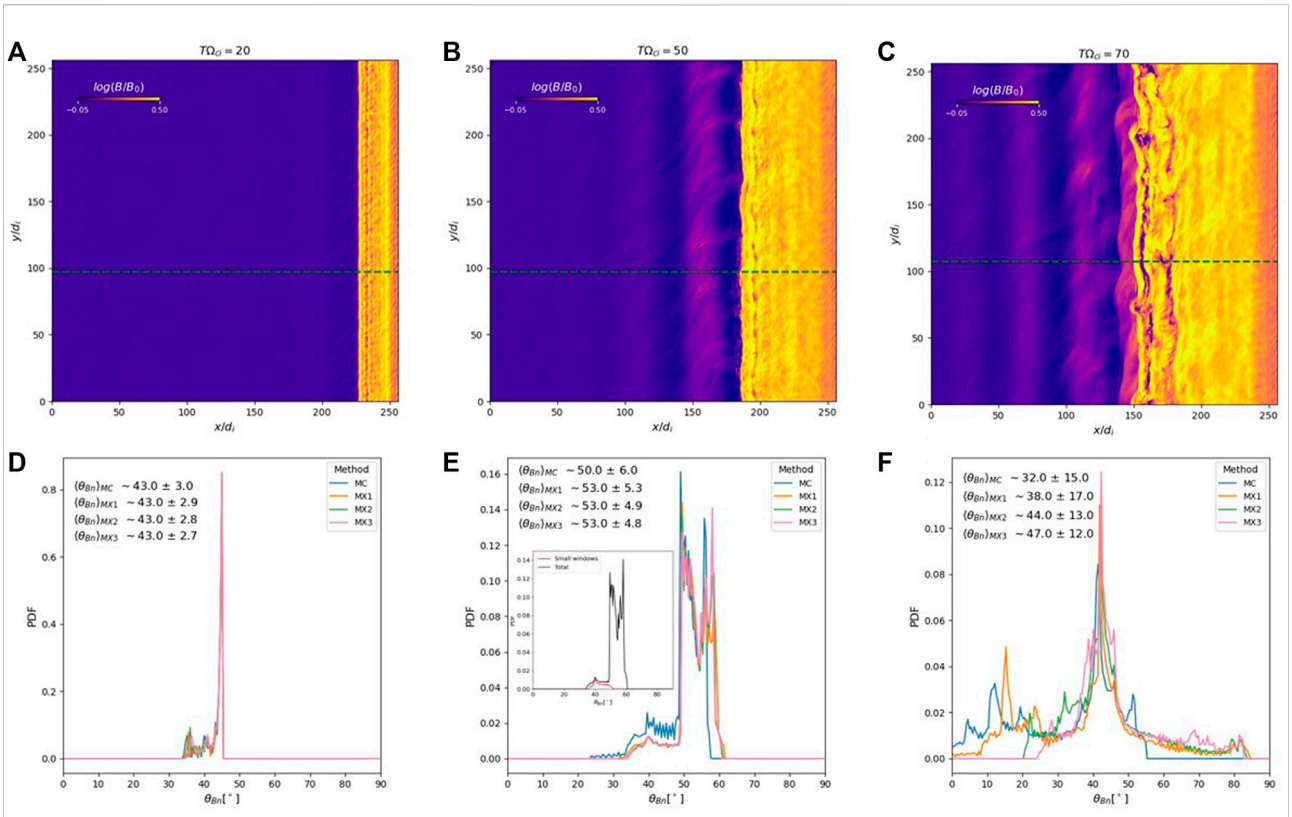


FIGURE 4
Magnetic field magnitude at different stages of shock evolution **(A–C)** and PDF distributions of θ_{Bn} values **(D–F)** evaluated on virtual spacecraft signals (with trajectories along green dashed lines) using systematic averaging window variations. The inset in panel **(E)** shows the full distribution of θ_{Bn} values (black) together with the distribution of θ_{Bn} obtained with small windows (max 2 d_i) only.

fluctuations can be hard to disentangle from shock-produced features of the upstream/downstream medium. Thus, care must be taken when interpreting such distributions of parameters, as it will be discussed in Sections 3.3, 3.4.

3.3 Solar orbiter shock observations

In the previous section, the diagnostics have been tested on a virtual spacecraft signal obtained using a self consistent simulation, which locally reconstructs the properties of a supercritical shock transition. Even though such simulations have been successful at improving our understanding of many observational features of shock transitions, they still represent an idealisation of very small spatial and temporal behaviour for the plasma. In this section, we finally apply our systematic method for shock parameter estimation on two events recently observed by the Solar Orbiter spacecraft.

Two different instruments on-board of Solar Orbiter have been used. The magnetic field, measured with a resolution of 128 vectors/s in burst mode by the flux-gate magnetometer MAG (Horbury et al., 2020), and the ground computed plasma moments, namely ion bulk flow, density and temperature, measured by the Solar Wind Analyser (SWA) suite (Owen et al., 2020), with 4 s resolution.

The first shock analysed here is a Coronal Mass Ejection (CME)-driven shock that crossed Solar Orbiter, while it was at 0.69 AU from the Sun at 07:32 UT of 11 October 2021. This shock is characterised by rather low Alfvén and fast magnetosonic Mach numbers ($M_{fms} \sim 2.04$, $M_A \sim 2.5$), low gas compression ratio $r_{gas} \sim 1.74$ and a low level of upstream magnetic fluctuations $\langle \delta B/B_0 \rangle_{rms} \sim 0.08$, where B_0 indicates the magnetic field averaged upstream. Inspired by such a parameter set, indicating a subcritical shock transition, together with the absence of strong pre-existing Solar Wind structures surrounding the shock, this event was named the "quiet" event.

Figure 5 shows an overview for the quiet event, with ~ 30 min of spacecraft collected data. The shock transition, highlighted by the vertical magenta line in the Figure, appears well-behaved, without strong upstream/downstream structuring. In particular, low levels of upstream/downstream fluctuations make this a good observational case for the systematic shock parameters estimation method.

The parameter estimation reveals that the shock geometry is quasi-perpendicular, with $\langle \theta_{Bn} \rangle \sim 60^\circ$. This has been obtained using

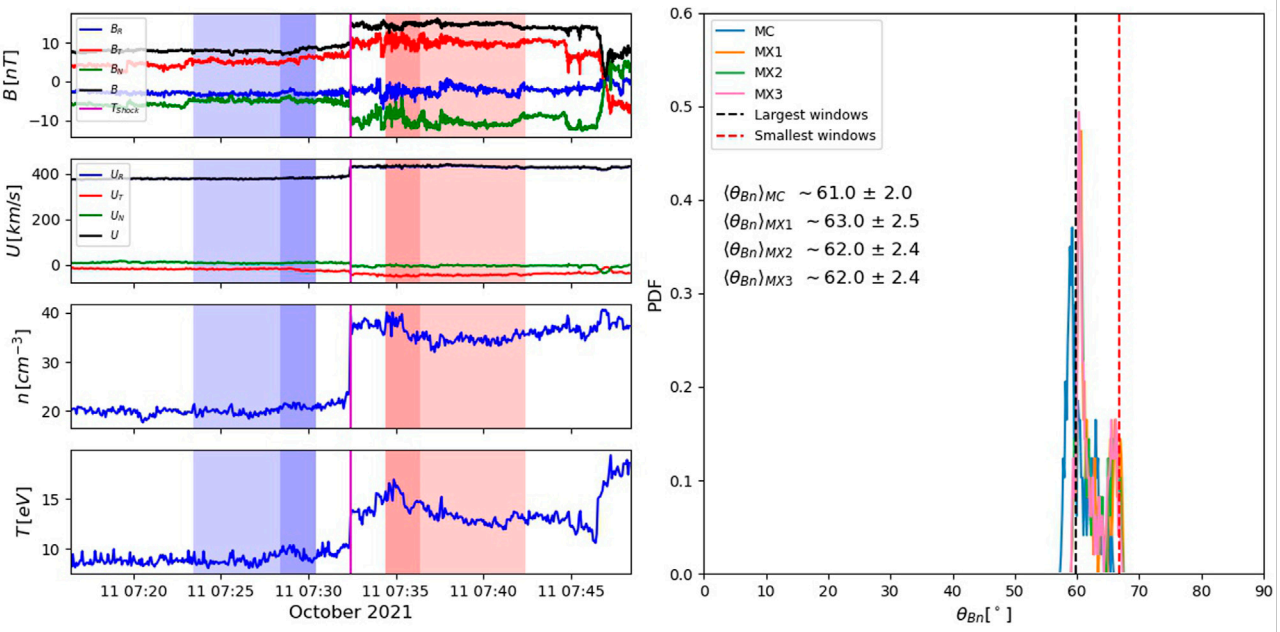


FIGURE 5
The quiet event of 11 October 2021 seen by Solar Orbiter. The left-hand panels show magnetic field *B*, ion bulk flow *U*, ion density *n* and temperature *T* measured by the MAG and SWA instruments. Vector quantities are shown in the RTN coordinate system. The blue/red shaded panels show the smallest (dark) to largest (lighter) averaging windows in the upstream/downstream regions. The vertical purple line shows the shock crossing time. The right-hand side panel shows the PDF distribution of θ_{Bn} values obtained for this event, with the vertical red (blue) dashed line showing the parameter esimation using the smallest (largest) window choice. The average values of θ_{Bn} obtained with each technique are also shown.

smallest averaging windows of about 2 min upstream and downstream, and largest windows of about 10 min. This choice is such that the time window over which the average is taken is always larger than kinetic timescales $(\Omega_{ci}{}^{-1}$ is of order ~ 10 s), making sure that the MHD description on which the data analyis technique rely is appropriate. Other studies looking at IP shock statistics and catalogues have analogous choices for upstream/downstream windows (see, for example, http://www.ipshocks.fi). The upstream/downstream averaging windows have been broadened with timesteps of 16 s, larger than the resolution of the Solar Orbiter plasma instrument. As it can be seen from the right-hand side panel of Figure 5, the PDF distribution of θ_{Bn} values is strongly peaked, with a small value of $\sigma_{\theta_{Bn}}$ ~ 2. The skewness of the distribution is about 0.7, due to the secondary peak observable in the distribution at for θ_{Bn} around 68°. Furthermore, we find strong agreement between the results obtained using different techniques MC and MX1-2-3 for the shock normal evaluation. These results indicate that the parameter estimation for this shock transition is particularly robust, i.e., it has weak dependence on the choice of upstream/downstream averaging windows.

A different situation is observed for the second shock crossing event presented here. This is another CME-driven shock, that crossed Solar Orbiter a few days later than the quiet event, at 22:02 on 30 October 2021, while Solar Orbiter was at 0.82 AU from the Sun. This shock is stronger than the previous one, with M_{fms} ~ 4.66 and M_A ~ 6.92. For the gas compression ratio, we found r_{gas} ~ 2.68, and the level of upstream magnetic fluctuations is moderate $\langle \delta B/B_0 \rangle_{rms}$ ~ 0.2.

The ~30 min' event overview, shown in Figure 6, shows a much more structured shock crossing, characterised by high levels of fluctuations downstream, as well as a rather disturbed upstream region. This behaviour has a strong impact on the assessment of the shock geometry. In fact, looking at the PDF distribution of θ_{Bn} values obtained using the same averaging parameters described for the quiet event, we find a much larger spread with a more gaussian behaviour, with smaller values of skewness $(\sim 10^{-2})$ with respect to the quiet event of October 11. The average θ_{Bn} computed with the mixed mode method is consistent, and between 42 and 45°. These results are different than the ones found using magnetic coplanarity, confirming the strong variability of the parameter estimation for this event. However, it is possible to notice the fact that the systematic approach to upstream/downstream averages yields to an accurate estimation of the shock geometry while addressing the sensitivity to the averaging window choice, and therefore accuracy through the $\sigma_{\theta_{Bn}}$ parameter.

The two above examples represent two very recent shock observations, and their study is extremely interesting in their own respect, and will be part of a separate work. Furthermore,

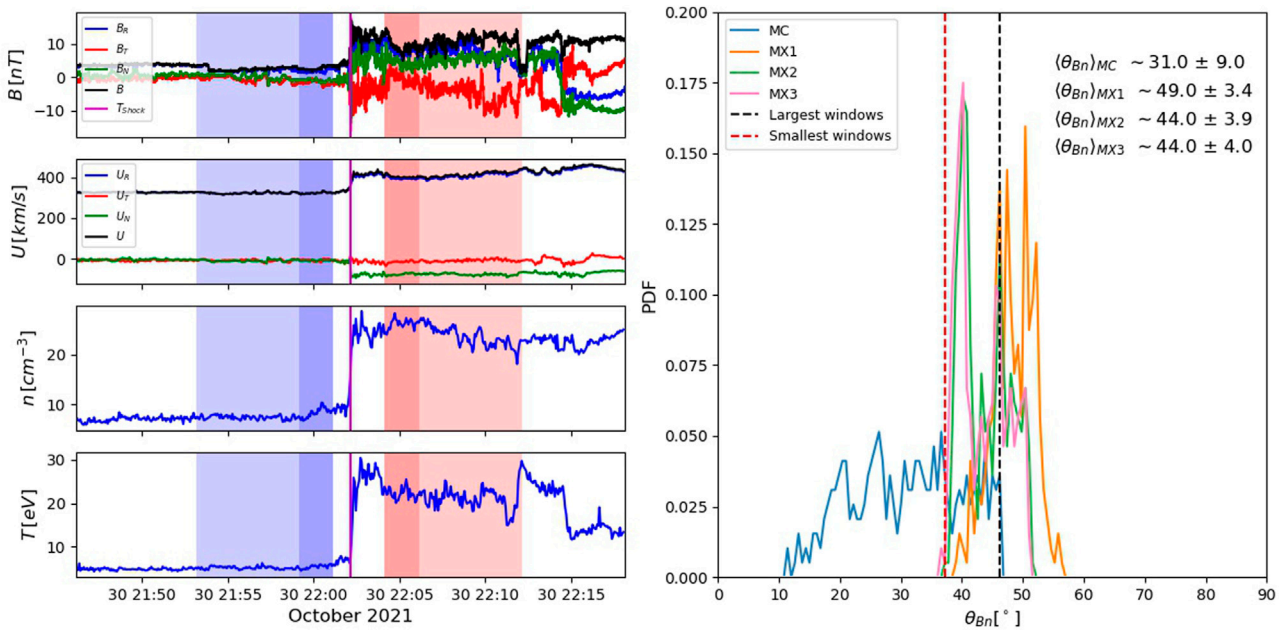


FIGURE 6
The structured event of 30 October 2021 seen by Solar Orbiter. The Figure is organised in the same fashion as Figure 5.

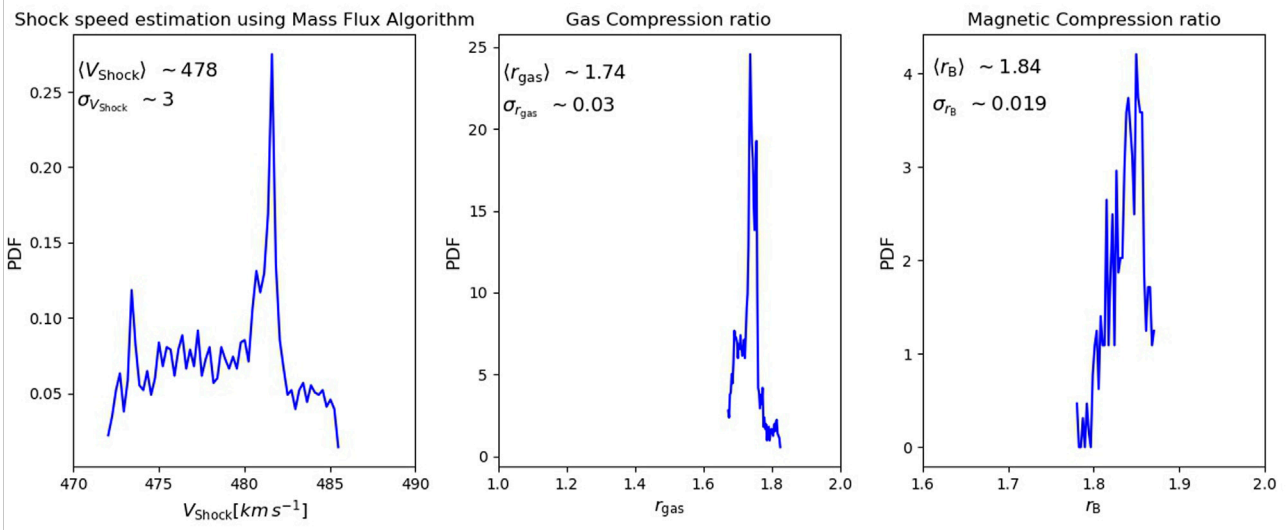


FIGURE 7
A test for shock speed, gas compression and magnetic compression ratio calculations using the systematic variation of averaging window technique for the quiet event of 11 October 2021 (see Figure 5).

for other shock parameters estimated, such as Mach numbers and compression ratio, a systematic approach has been followed as well. The results of such approach are reported in the Appendix of this work (see Figure 7), and the software used to compute them is included in the Python package SerPyShock.

3.4 MMS shock observations

The magnetospheric multiscale mission (MMS) provided the community with unprecedented multi-spacecraft high-resolution measurements of the magnetospheric environment

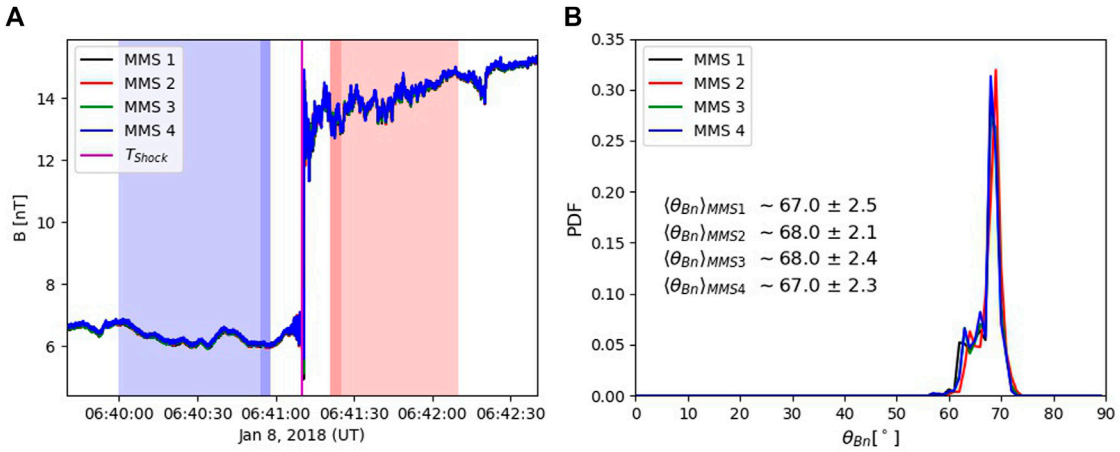


FIGURE 8
(A): Magnetic field magnitude observed by the four MMS spacecraft of 8 January 2018. The purple line represents the time of the shock crossing.
(B): PDF distributions of θ_{Bn} values obtained with the magnetic coplanarity method for each spacecraft.

(Burch et al., 2016). On 8 January 2018, while in the pristine solar wind, MMS observed a supercritical interplanetary shock. This interesting event was studied in detail by Cohen et al. (2019), reporting for the first time direct high temporal resolution near specularly reflected ions at an IP shock. During the shock crossing, the MMS spacecraft had small separation (~ 20 km), and it therefore represents a good opportunity to test our diagnostics on multi-spacecraft crossing of the same shock, not possible with Solar Orbiter. Figure 8 shows magnetic field measurement for the four different MMS spacecraft around the shock crossing time, and the analysis for the shock obliquity is carried out using magnetic coplanarity in the right hand-panel. Here, the range of upstream/downstream time windows has been set to be close to the averaging windows used by Cohen et al. (2019). We find strong agreement between the shock obliquity estimated at each shock crossing (Figure 8B), with a θ_{Bn} value consistent with the one reported in Table 1 of Cohen et al. (2019), that is of 67°.

4 Conclusion

In this work, we revisited shock parameter estimation using single spacecraft signals. Starting from the early seminal works of Balogh et al. (1995), a method involving a systematic variation of upstream/downstream averaging windows has been implemented, yielding to an ensemble of shock parameters estimations for a shock crossing, as opposed to a single value that corresponds to a particular choice of upstream/downstream windows. With such a statistic of shock parameters, it is possible to address their mean value as a more accurate parameter estimation, and the standard deviation as a measure of uncertainty/sensitivity to the choice of averaging windows. We discussed the implication of adopting such an approach.

We started by introducing the main shock parameters estimation techniques tested throughout this work (Section 2), with particular emphasis on shock normal estimation methods, reviewing the theoretical framework for Magnetic Coplanarity (MC) and Mixed Mode (MX1-2-3) methods, closely following previous, more extensive documentations (Paschmann and Schwartz, 2000; Hietala, 2012; Burgess and Scholer, 2015). Throughout the work, we focus on shock geometry estimation (shock normals and consequently θ_{Bn} angles).

The ensemble technique to compute shock parameters is then introduced on the simplest possible test cases, namely synthetic timeseries of Rankine-Hugoniot compliant shock transitions (Section 3.1). Such synthetic shocks have been analysed both in the purely analytic case, and also adding white noise, to mimic some level of uncertainty in the fields across the shock transition. The technique is tested, and the method followed is highlighted. It is important to note that, due to the nature of the shock parameters estimation involved, that suppose an upstream/downstream averaging region as close as possible to the shock with an exclusion zone containing the shock itself, our systematic averaging variation is done in such a way that, after starting from the smallest possible upstream (downstream) windows, these are enlarged at every iteration by some extent. In this way, points close to the shock transition are included multiple times in the parameters statistics. Using random window generation upstream/downstream, it may be possible to accidentally choose windows that are very far from the shock transition, violating the premise of the parameter estimation techniques analysed here (Hietala, 2012).

An hybrid kinetic, self-consistent simulation of an oblique, supercritical shock is then presented as a further test for shock geometry evaluation. These simulations are very successful at reproducing many features of collisionless shock transitions, including shock front instabilities and the presence of self-consistently generated upstream/downstream fluctuations. The advantage is that a nominal θ_{Bn} is chosen for the shock as an initial condition for the simulation. Using a virtual satellite crossing the shock transition, we applied the single-spacecraft techniques mentioned above, a numerical experiment that has not been reported in previous literature, to the best of our knowledge. We found that at the early stage of the shock simulation, where upstream/downstream wave activity is low and the shock front is quasi-laminar, the PDF distribution of shock normal angles peaks very well around the nominal θ_{Bn}. For later simulation stages, where the shock front is unstable and the upstream/downstream regions filled with fluctuations related to unstable particle distributions generated at the shock, the distribution of θ_{Bn} angles widens remarkably. Interestingly though, even with a single (virtual) spacecraft crossing of the shock front, the averaged θ_{Bn} values obtained from the distribution for MC and MX1-2-3 techniques are rather close to the nominal one posed as an initial condition. Such an experiment highlights, on one hand, the capabilities of the ensemble approach to shock parameter estimation, and on the other hand the fact that single spacecraft crossings, even though limited, contain extremely valuable information about shock features. This numerical experiment will be extended in future works, including crossings at different angles with respect to the shock normal, as well as employing simulations using full three-dimensional geometry.

In Section 3.3, we applied the systematic shock parameter estimation technique to two recent fast forward, CME-driven shocks observed by Solar Orbiter. These events are interesting in their own respect and will be object of detailed investigation addressing the particle behaviour around the shock transitions. Shock parameters were estimated for the two different shocks, one characterised by rather low Mach numbers and a quiet upstream/downstream environment, named here the October 11 "quiet event", and another one characterised by rather high Mach numbers, that is propagating though a structured upstream and downstream medium. The technique performed well, revealing sharper PDF distributions for θ_{Bn} for the quiet event with respect to the broader PDFs observed for the structured event. These examples show that this systematic way of estimating shock parameters is consistent with the expected sensitivity to the averaging window choice, and represent a step forward with respect to choosing one single combination of upstream/downstream windows. Future works will compare the results obtained with this approach with multi-spacecraft techniques to evaluate parameters such as, for example, the shock speed.

We have developed an open-source Python software package, SerPyShock, that can be used to perform an analysis of shock wave properties similar to the one presented in this study. This package provides the code we developed for such shock analyses, an example script, and the data loaders needed to work with Solar Orbiter *in situ* datasets. The source code of the package is provided in a publicly available GitHub repository https://github.com/trottadom/SerPyShock and it is licensed under the GNU General Public License version 3. We plan to extend the capabilities of this software package by including other useful techniques for shock physics investigation, such as, for example, a set of routines looking at the properties of energetic particles.

Data availability statement

The original contributions presented in the study are publicly available. The software package SerPyShock used to make all the Figures and analyses is available at https://github.com/trottadom/SerPyShock. The hybrid simulation datasets used in Section 3.2 and Figures 3, 4 are available to download here: https://doi.org/10.5281/zenodo.6913596. The Solar Orbiter data used are available for download at https://soar.esac.esa.int/soar/. The MMS data used in Figure 7 can be downloaded at https://cdaweb.gsfc.nasa.gov.

Author contributions

DT and HH discussed and laid the foundations for the technique presented here. DT, HH, and TH decided the structure of this work and the analysis plan. DT performed the analyses and produced the figures. DT performed the hybrid kinetic simulation, looking at virtual spacecraft signals in collaboration with FV. EK and DT tested and expanded the diagnostics proposed in our work. The SerPyShock Python code has been developed by DT. LV and JG developed the Solar Orbiter data loaders included in the software. JG, AK, and DP provided support for the Python development. ND, FV, and RV helped to discuss the relevance of this work for the field. All authors contributed to the global interpretation of the results, as well as to their discussion and presentation in the manuscript. All authors revised the manuscript before submission.

Funding

This work has received funding from the European Unions Horizon 2020 research and innovation pro-gramme under grant agreement No. 101004159 (SERPENTINE, www.serpentine-h2020.eu). Part of this work was performed using the DiRAC Data Intensive service at Leicester, operated by the University of

Leicester IT Services, which forms part of the STFC DiRAC HPC Facility (www.dirac.ac.uk), under the project "dp031 Turbulence, Shocks and Dissipation in Space Plasmas". AK acknowledges financial support from NASA NNN06AA01C (SO-SIS Phase-E) contract.

Acknowledgments

DT thanks Dr. A. Dimmock for the fruitful discussions around the capabilities of the toolbox. DT would like to thank Prof. D. Burgess for the support in performing the hybrid kinetic simulation. ND is grateful for support by the Turku Collegium for Science, Medicine and Technology of the University of Turku, Finland.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Abraham-Shrauner, B. (1972). Determination of magnetohydrodynamic shock normals. *J. Geophys. Res.* 77, 736–739. doi:10.1029/JA077i004p00736

Balogh, A., Gonzalez-Esparza, J. A., Forsyth, R. J., Burton, M. E., Goldstein, B. E., Smith, E. J., et al. (1995). Interplanetary shock waves: ULYSSES observations in and out of the ecliptic plane. *Space Sci. Rev.* 72, 171–180. doi:10.1007/BF00768774

Benz, A. O. (2008). Flare observations. *Living Rev. Sol. Phys.* 5, 1. doi:10.12942/lrsp-2008-1

Birdsall, C. K., and Langdon, A. B. (1991). *Plasma physics via computer simulation*. boca raton: CRC Press.

Blanco-Cano, X., Kajdič, P., Aguilar-Rodríguez, E., Russell, C. T., Jian, L. K., and Luhmann, J. G. (2016). Interplanetary shocks and foreshocks observed by stereo during 2007–2010. *J. Geophys. Res. Space Phys.* 121, 992–1008. doi:10.1002/2015JA021645

Brunetti, G., and Jones, T. W. (2014). Cosmic rays in galaxy clusters and their nonthermal emission. *Int. J. Mod. Phys. D.* 23, 1430007–1430098. doi:10.1142/S0218271814300079

Burch, J. L., Moore, T. E., Torbert, R. B., Giles, B. L., and Burch, B. J. L. (2016). Magnetospheric multiscale overview and science objectives. *Space Sci. Rev.* 199, 5–21. doi:10.1007/s11214-015-0164-9

Burgess, D., and Scholer, M. (2015). *Collisionless shocks in space plasmas*. United Kingdom: Cambridge University Press. doi:10.1017/CBO9781139044097

Caprioli, D., and Spitkovsky, A. (2014). Simulations of ion acceleration at non-relativistic shocks. I. Acceleration efficiency. *Astrophys. J.* 783, 91. doi:10.1088/0004-637X/783/2/91

Coates, A. J., Mazelle, C., and Neubauer, F. M. (1997). Bow shock analysis at comets Halley and grigg-skjellerup. *J. Geophys. Res.* 102, 7105–7113. doi:10.1029/96JA04002

Cohen, I. J., Schwartz, S. J., Goodrich, K. A., Ahmadi, N., Ergun, R. E., Fuselier, S. A., et al. (2019). High-resolution measurements of the cross-shock potential, ion reflection, and electron heating at an interplanetary shock by mms. *J. Geophys. Res. Space Phys.* 124, 3961–3978. doi:10.1029/2018JA026197

Colburn, D. S., and Sonett, C. P. (1966). Discontinuities in the solar wind. *Space Sci. Rev.* 5, 439–506. doi:10.1007/BF00240575

de Hoffmann, F., and Teller, E. (1950). Magneto-hydrodynamic shocks. *Phys. Rev.* 80, 692–703. doi:10.1103/PhysRev.80.692

Dessler, A. J., and Fejer, J. A. (1963). Interpretation of kp index and m-region geomagnetic storms. *Planet. Space Sci.* 11, 505–511. doi:10.1016/0032-0633(63)90074-6

Dungey, J. W. (1979). First evidence and early studies of the Earth's bow shock. *Il Nuovo Cimento C* 2, 655–660. doi:10.1007/BF02558123

Escoubet, C. P., Fehringer, M., and Goldstein, M. (2001). <i>Introduction</i>The Cluster mission. *Ann. Geophys.* 19, 1197–1200. doi:10.5194/angeo-19-1197-2001

Gosling, J. T., Hildner, E., MacQueen, R. M., Munro, R. H., Poland, A. I., and Ross, C. L. (1974). Mass ejections from the sun: A view from skylab. *J. Geophys. Res.* 79, 4581–4587. doi:10.1029/JA079I031P04581

Ha, J.-H., Ryu, D., Kang, H., and Kim, S. (2022). Electron preacceleration at weak quasi-perpendicular intracluster shocks: Effects of preexisting nonthermal electrons. *Astrophys. J.* 925, 88. doi:10.3847/1538-4357/ac3bc0

Hietala, H. (2012). *Multi-spacecraft studies on space plasma shocks*. Finland: University of Helsinki. Ph.D. thesis.

Hoppe, M. M., and Russell, C. T. (1982). *J. Proc. Natn. Acad. Sci. U. S. A.* 295, 236–247.

Horbury, T. S., O'Brien, H., Carrasco Blazquez, I., Bendyk, M., Brown, P., Hudson, R., et al. (2020). The solar orbiter magnetometer. *Astron. Astrophys.* 642, A9. doi:10.1051/0004-6361/201937257

Hugoniot, H. (1887). Memoir on the propagation of movements in bodies, especially perfect gases (first part). *l'Ecole Polytech.* 57, 3–97.

Johlander, A., Schwartz, S. J., Vaivads, A., Khotyaintsev, Y. V., Gingell, I., Peng, I. B., et al. (2016). Rippled quasiperpendicular shock observed by the magnetospheric multiscale spacecraft. *Phys. Rev. Lett.* 117, 165101. doi:10.1103/PhysRevLett.117.165101

Kajdič, P., Preisser, L., Blanco-Cano, X., Burgess, D., and Trotta, D. (2019). First observations of irregular surface of interplanetary shocks at ion scales by cluster. *Astrophys. J.* 874, L13. doi:10.3847/2041-8213/ab0e84

Kennel, C. F., Edmiston, J. P., and Hada, T. (1985). *A quarter century of collisionless shock research*. Washington, D.C.: American Geophysical Union AGU. doi:10.1029/GM034p0001

Kilpua, E. K., Lumme, E., Andreeova, K., Isavnin, A., and Koskinen, H. E. (2015). Properties and drivers of fast interplanetary shocks near the orbit of the Earth (1995-2013). *J. Geophys. Res. Space Phys.* 120, 4112–4125. doi:10.1002/2015JA021138

Kivelson, M., and Russell, C. (1995). *Introduction to space physics*. United Kingdom: Cambridge University Press. doi:10.1017/9781139878296

Kokoska, S., and Zwillinger, D. (1999). *Crc standard probability and statistics tables and formulae*. Boca Raton: CRC Press. student edition.

Koval, A., and Szabo, A. (2008). Modified "rankine-hugoniot" shock fitting technique: Simultaneous solution for shock normal and speed. *J. Geophys. Res.* 113 (A10). doi:10.1029/2008JA013337

Lepping, R. P. (1984). *A comparative review of bow shocks and magnetopauses*. Washington, DC: National Aeronautics and Space Administration.

Lepping, R. P., and Argentiero, P. D. (1971). Single spacecraft method of estimating shock normals. *J. Geophys. Res.* 76, 4349–4359. doi:10.1029/JA076i019p04349

Leroy, M. M., and Mangeney, A. (1984). A theory of energization of solar wind electrons by the Earth's bow shock. *Ann. Geophys.* 2, 449–456.

Masters, A., Achilleos, N., Dougherty, M. K., Slavin, J. A., Hospodarsky, G. B., Arridge, C. S., et al. (2008). An empirical model of saturn's bow shock: Cassini

observations of shock location and shape. *J. Geophys. Res.* 113 (A10). doi:10.1029/2008JA013276

Matthews, A. P. (1994). Current advance method and cyclic leapfrog for 2d multispecies hybrid plasma simulations. *J. Comput. Phys.* 112, 102–116. doi:10.1006/jcph.1994.1084

McComas, D. J., Rankin, J. S., Schwadron, N. A., and Swaczyna, P. (2019). Termination shock measured by voyagers and IBEX. *Astrophys. J.* 884, 145. doi:10.3847/1538-4357/ab441a

Mejnertsen, L., Eastwood, J. P., Hietala, H., Schwartz, S. J., and Chittenden, J. P. (2018). Global mhd simulations of the Earth's bow shock shape and motion under variable solar wind conditions. *JGR. Space Phys.* 123, 259–271. doi:10.1002/2017JA024690

Mignone, A., Bodo, G., Massaglia, S., Matsakos, T., Tesileanu, O., Zanni, C., et al. (2007). Pluto: A numerical code for computational astrophysics. *Astrophys. J. Suppl. Ser.* 170, 228–242. doi:10.1086/513316

Muller, D., Cyr, O. C., Zouganelis, I., Gilbert, H. R., Marsden, R., Nieves-Chinchilla, T., et al. (2020). The solar orbiter mission - science overview. *Astron. Astrophys.* 642, A1. doi:10.1051/0004-6361/202038467

Naeem, I., Ehsan, Z., Mirza, A. M., and Murtaza, G. (2020). Shocklets in the comet Halley plasma. *Phys. Plasmas* 27, 043703. doi:10.1063/5.0002521

Oliveira, D., and Samsonov, A. (2018). Geoeffectiveness of interplanetary shocks controlled by impact angles: A review. *Adv. Space Res.* 61, 1–44. doi:10.1016/j.asr.2017.10.006

Owen, C. J., Bruno, R., Livi, S., Louarn, P., Janabi, K. A., Allegrini, F., et al. (2020). The solar orbiter solar wind analyser (swa) suite. *Astron. Astrophys.* 642, A16. doi:10.1051/0004-6361/201937259

Paschmann, G., and Schwartz, S. J. (2000). "ISSI book on analysis methods for multi-spacecraft data," in *Cluster-II workshop multiscale / multipoint plasma measurements*. Editor R. A. Harris (Bern, Switzerland: The International Space Science Institute), 449, 99.

Perri, S., Servidio, S., Vaivads, A., and Valentini, F. (2017). Numerical study on the validity of the Taylor hypothesis in space plasmas. *Astrophys. J. Suppl. Ser.* 231, 4. doi:10.3847/1538-4365/aa755a

Preisser, L., Blanco-Cano, X., Kajdič, P., Burgess, D., and Trotta, D. (2020a). Magnetosheath jets and plasmoids: Characteristics and formation mechanisms from hybrid simulations. *Astrophys. J. Lett.* 900, L6. doi:10.3847/2041-8213/abad2b

Preisser, L., Blanco-Cano, X., Trotta, D., Burgess, D., and Kajdič, P. (2020b). Influence of he++ and shock geometry on interplanetary shocks in the solar wind: 2d hybrid simulations. *J. Geophys. Res. Space Phys.* 125, e2019JA027442. doi:10.1029/2019ja027442

Quest, K. B. (1985). Simulations of high-Mach-number collisionless perpendicular shocks in astrophysical plasmas. *Phys. Rev. Lett.* 54, 1872–1874. doi:10.1103/PhysRevLett.54.1872

Rankine, W. J. M. (1870). Xv. on the thermodynamic theory of waves of finite longitudinal disturbance. *Philosophical Trans. R. Soc. Lond.* 160, 277–288. doi:10.1098/rstl.1870.0015

Russell, C. T., Gosling, J. T., Zwickl, R. D., and Smith, E. J. (1983a). Multiple spacecraft observations of interplanetary shocks: ISEE three-dimensional plasma measurements. *J. Geophys. Res.* 88, 9941–9948. doi:10.1029/JA088iA12p09941

Russell, C. T., Mellott, M. M., Smith, E. J., and King, J. H. (1983b). Multiple spacecraft observations of interplanetary shocks: Four spacecraft determination of shock normals. *J. Geophys. Res.* 88, 4739–4748. doi:10.1029/JA088iA06p04739

Sonnerup, B. U. O., and Cahill, J. L. J. (1967). Magnetopause structure and attitude from explorer 12 observations. *J. Geophys. Res.* 72, 171. doi:10.1029/JZ072i001p00171

Sundberg, T., Haynes, C. T., Burgess, D., and Mazelle, C. X. (2016). Ion acceleration at the quasi-parallel bow shock: Decoding the signature of injection. *Astrophys. J.* 820, 21. doi:10.3847/0004-637X/820/1/21

Taylor, G. I. (1938). The spectrum of turbulence. *Proc. R. Soc. Lond. A* 164, 476–490. doi:10.1098/rspa.1938.0032

Thomsen, M. F., Bame, S. J., Feldman, W. C., Gosling, J. T., McComas, D. J., and Young, D. T. (1986). The comet/solar wind transition region at giacobini-zinner. *Geophys. Res. Lett.* 13, 393–396. doi:10.1029/gl013i004p00393

Trotta, D., and Burgess, D. (2019). Electron acceleration at quasi-perpendicular shocks in sub- and supercritical regimes: 2D and 3D simulations. *Mon. Not. R. Astron. Soc.* 482, 1154–1162. doi:10.1093/mnras/sty2756

Trotta, D., Valentini, F., Burgess, D., and Servidio, S. (2021). Phase space transport in the interaction between shocks and plasma turbulence. *Proc. Natl. Acad. Sci. U. S. A.* 118, e2026764118. doi:10.1073/pnas.2026764118

Viñas, A. F., and Scudder, J. D. (1986). Fast and optimal solution to the "rankine-hugoniot problem". *J. Geophys. Res.* 91, 39–58. doi:10.1029/ja091ia01p00039

Volkmer, P. M., and Neubauer, F. M. (1985). Statistical properties of fast magnetoacoustic shock waves in the solar wind between 0.3 AU and 1 AU: Helios-1, 2 observations. *Ann. Geophys.* 3, 1–12.

10.3389/fspas.2022.1005672

Appendix: Shock speed estimation and other parameters

This work demonstrates the use and implications of having a systematic variation of upstream/downstream averaging windows when addressing shock parameters using single spacecraft crossings. In the discussion, our main focus has been around the estimation of shock normal vectors and θ_{Bn} angles to address the shock geometry, crucial for many features of collisionless shocks. However, other important parameters can be estimated using the same ensemble approach for the averaging operation. In the SerPyShock software released together with this work, we also provide routines for the systematic computation of shock speed using the mass flux algorithm V_{shock}, and the shock gas and magnetic compression ratios r_{gas} and r_{B}. The definitions of these parameters are given in Section 2. An example of usage of such routines is shown in Figure 8, where they have been applied to the shock observed by Solar Orbiter on 11 October 2021 (i.e., the "quiet event" discussed in Section 3.3).

frontiers | Frontiers in Physics

TYPE Methods
PUBLISHED 20 October 2022
DOI 10.3389/fphy.2022.1005621

Check for updates

OPEN ACCESS

EDITED BY
Sophie A Murray,
Dublin Institute for Advanced Studies (DIAS), Ireland

REVIEWED BY
Stefano Markidis,
KTH Royal Institute of Technology, Sweden
Camilla Scolini,
University of New Hampshire, United States
Ravindra Desai,
Imperial College London, United Kingdom

*CORRESPONDENCE
Luke Barnard,
l.a.barnard@reading.ac.uk
Mathew Owens,
m.j.owens@reading.ac.uk

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Physics

RECEIVED 28 July 2022
ACCEPTED 05 October 2022
PUBLISHED 20 October 2022

CITATION
Barnard L and Owens M (2022), HUXt—An open source, computationally efficient reduced-physics solar wind model, written in Python.
Front. Phys. 10:1005621.
doi: 10.3389/fphy.2022.1005621

COPYRIGHT
© 2022 Barnard and Owens. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

HUXt—An open source, computationally efficient reduced-physics solar wind model, written in Python

Luke Barnard* and Mathew Owens*

Department of Meteorology, University of Reading, Reading, United Kingdom

HUXt is an open source numerical model of the solar wind written in Python. It is based on the solution of the 1D inviscid Burger's equation. This reduced-physics approach produces solar wind flow simulations that closely emulate the flow produced by 3-D magnetohydrodynamic (MHD) solar wind models at a small fraction of the computational expense. While not intended as a replacement for 3-D MHD, the simplicity and computational efficiency of HUXt offers several key advantages that enable experiments and the use of techniques that would otherwise be cost prohibitive. For example, large ensembles of 10^2–10^5 members can easily be run with modest computing resources, which are useful for exploring and quantifying the uncertainty in space weather predictions, as well as for the application of some data assimilation methods. In this article we present the developments in the latest version of HUXt, v4.0, and discuss our plans for future developments and applications of the model. The three key developments in v4.0 are: 1) a restructuring of the models solver to enable fully time-dependent boundary conditions, such that HUXt can in principle be initialised with *in-situ* observations from any of the fleet of heliospheric monitors; 2) new functionality to trace streaklines through the HUXt flow solutions, which can be used to track features such as the Heliospheric Current Sheet; 3) introduction of a small test-suite so that we can better ensure the reliability and reproducibility of HUXt simulations for all users across future versions. Other more minor developments are discussed in the article. Future applications of HUXt are discussed, including the development of both sequential and variational data assimilation schemes for assimilation of both remote sensing and *in-situ* plasma measures. Finally, we briefly discuss the progress of transitioning HUXt into an operational model at the UK's Met Office Space Weather Operations Center as part of the UK governments SWIMMR programme.

KEYWORDS

space weather, heliophysics, simulations, Python, open source

1 Introduction

Variability in the near-Earth space environment gives rise to space weather, which can have adverse effects on space- and ground-based technologies [1]. The largest disturbances to the terrestrial magnetospheric system are the result of coronal mass ejections (CMEs) arriving in near-Earth space [2]. Thus, advanced forecasting of the arrival time and properties of CMEs at Earth is highly desirable [3]. While near-Sun CME properties—notably speed and direction – can be estimated from coronagraph observations [4], forecasting at Earth also requires accurate knowledge of the variable solar wind conditions through which CMEs propagate [5].

The dynamic interaction between a CME and the ambient solar wind is typically modelled using time-dependent magnetohydrodynamic (MHD) simulations of the solar wind (e.g., [6–9]). These models capture the large-scale MHD fluid behaviour which governs much of the physics of CME propagation and evolution to Earth, and are a vital tool that enable both heliophysics research and space weather forecasting (e.g., [3,10,11]). More recently, it has been demonstrated that similar forecasting results, at least to first order, can be obtained with greatly simplified models [12,13]. Such models are not intended to replace the more sophisticated simulations, but open up complementary capabilities *via* greatly reduced computational overhead.

This paper is intended to briefly review the reduced-physics model, HUXt (heliospheric upwind extrapolation with time-dependence), and then highlight some of the functionality of the HUXt Python implementation.

HUXt has already proved useful in a number of contexts. Barnard et al. [14] demonstrated how an ensemble of HUXt runs could be weighted by comparison with STEREO Heliospheric Imager (HI) time-elongation profiles of CME fronts to return improved ensemble hindcasts of CME arrival times. Chi et al. [15] used HUXt simulations to examine the evolving structure of two interacting CMEs, finding that the HUXt simulations were consistent with the STEREO HI observations of these CMEs.

Barnard and Owens [16] used HUXt simulations of Cone CMEs to examine the validity of the assumptions of CME geometric models and the ability of these models to reconstruct a CMEs kinematic profile from time-elongation profiles of a CMEs flank, such as those derived from HI observations. Similarly, Hinterreiter et al. [17] used HUXt simulations as part of their work to introduce time-dependent structure to their ElEvoHI geometric model. These works both highlighted the importance of including time-dependent structure and solar-wind CME interactions for increasing the real-world representivity of CME modelling.

Additionally, separate from these CME focused studies, Macneil et al. [18] demonstrated how HUXt may be used to backmap *in-situ* solar wind observation to the source regions in the low corona, which is an important technique for estimating the origins of solar wind plasma structures. Furthermore, Bunting and Morgan [19] used HUXt simulations in their calibration and long-term validation experiments into using coronal tomography to generate inner boundary conditions for solar wind numerical models.

We now proceed to review both the background and the current status of our Python implementation of HUXt. Section 2 describes the theoretical background to HUXt, whilst Section 3 discusses the numerical scheme used to solve the discretised model equations. Section 3.1 presents some new results of testing the performance of the numerical scheme. Section 5 describes some of they key functionality included in our Python implementation of HUXt. Finally, some developing applications of HUXt are described in Section 6.

2 The HUXt reduced-physics solar wind model

HUXt takes a reduced-physics approach to modelling the solar wind flow, employing approximations to greatly reduce the complexity of the MHD momentum equation. A full derivation is provided in Owens et al. [13] and so we provide only a summary here. Magnetic, gravitational, and pressure gradient forces are neglected, as in the solar wind these terms are typically small compared to the flow momentum [12]. Additionally, the solar wind is assumed to be purely radial, and so non radial terms are neglected. In this limit, the solar wind is modelled by the inviscid Burger's equation.

$$\frac{\partial V_r}{\partial t} + V_r \frac{\partial V_r}{\partial r} = 0, \tag{1}$$

Where V_r is the radial solar wind speed. Pizzo [20] and Riley and Lionello [12] outlined this simplified description of the solar wind and made the further assumption of time-stationary flows to give the Heliospheric Upwind eXtrapolation (HUX) model (see also [21]). HUXt, however, maintains explicit time dependence, allowing structures such as coronal mass ejections (CMEs) and time-dependent ambient solar wind flow to be modelled.

The solar wind continues to accelerate throughout the inner heliosphere and in HUXt this residual acceleration, $\Delta V(r)$, is represented by an empirical parameterisation, which for a uniform solar wind is represented as

$$V_r(r) = V_0 + \Delta V(r), \tag{2}$$

Where V_0 is the speed at a reference height and

$$\Delta V(r) = \alpha V_0 \left[1 - e^{\left(\frac{(r_0 - r)}{r_h}\right)}\right], \tag{3}$$

Where α is the acceleration factor, set to be 0.15, and r_h is the scale height over which it applies, set to be 50 $r_\odot$, whilst r_0 is the

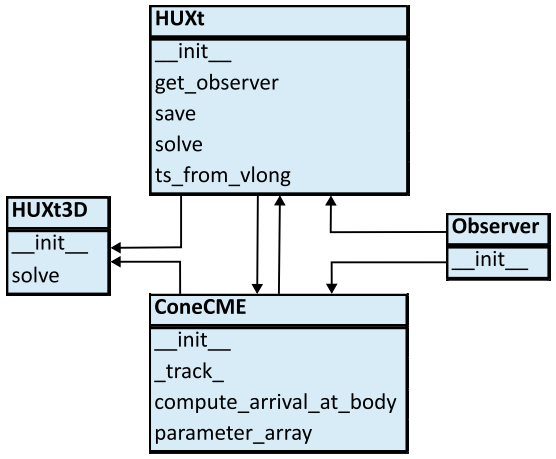


FIGURE 1
This diagram highlights the classes used within HUXt, and their associated methods, and the relationships between them.

radial distance at the reference height corresponding to V_0, taken to be 30 $r_\odot$. This form is designed to mimic that arising from the energy equations in MHD solar wind models [12].

Owens et al. [13] analysed the performance of HUXt relative to the HelioMAS MHD model of the solar wind, comparing each models representation of phenomena such as CIRs and Cone CMEs. Figure 1 of Owens et al. [13] shows that HUXt and HelioMAS both return very similar CIR structures. However, there are small but systematic discrepancies. For example, in HUXt, CIRs tend to have sharper gradients and higher maximum speeds relative to HelioMAS. Two factors likely affect this; firstly, this could be due to non-radial dynamics in HelioMAS allowing flow deflections that HUXt does not model; secondly, the upwind numerical scheme used in HUXt is known to permit sharper gradients than the numerical schemes employed by MHD models. For a more complete review of the scientific justification of HUXt, and its comparison to MHD simulations, see Owens et al. [13].

3 Numerical scheme

The HUXt model equations are solved numerically using a first order upwind scheme [22], and the discretised equations are given in Owens et al. [13].

The default radial grid of HUXt has an inner boundary at 30 $r_\odot$, outer boundary at 240 $r_\odot$, with a radial grid step of 1.5 $r_\odot$. However, HUXt has built-in functionality to work with different radial grids and boundary heights, as discussed in Sections 5.2.3 and Section 5.2.1.

Fundamentally, HUXt is a 1-D radial model of the solar wind. Single 1D solutions are useful in their own right, due to the rapid computation time (much less than a second on a modest CPU for a 5-day simulation out to Earth orbit). Thus, one use-case is to run HUXt in a synodic frame of reference and simulate the Earth-Sun line. This can be done in large ensembles of O (10^3–10^5), enabling case-specific estimates of forecast uncertainty (e.g., [23]). Examples of idealised and data-driven 1-D solutions are provided in the *HUXt_examples.ipynb* notebook, discussed in Section 4. Here, the examples shown are for the more generalised case of 2-D and 3-D solutions that incorporate a range of longitudes and/or latitudes by assembling a set of the 1-D radial HUXt solutions. Such 2-D and 3-D solutions would generally be performed in the sidereal reference frame.

The default longitudinal grid spans 0 to 2π in 128 bins. The default latitudinal grid has 45 equally spaced cells in sine latitude. On initialisation, a single longitude or range of longitudes must be specified, whilst, unless otherwise specified, the latitude is assumed to be 0°. Finer or coarser grids can be used as required. The model grid sets zero longitude to that of Earth at the time of the start of the run. In the synodic reference frame, this is maintained throughout the run. For the sidereal reference frame, Earth increases in longitude by around 1° per day. Note that because the 2-D and 3-D solutions are collections of independent 1-D radial solutions, any subset of the longitude and latitude grid can be considered, without any impact on the solution due to longitude or latitude boundary conditions. (Figure 2 shows an example of this functionality for a 3-D solution, discussed in Section 5.2.4.).

The model time-step, Δt is set by the Courant–Friedrichs–Lewy (CFL) condition, $\Delta t \leq \frac{\Delta r}{v_{max}}$, with the default value of $v_{\max} = 2000\ km\ s^{-1}$ resulting in $\Delta t = 8.70$ min. The default $v_{\max}$ is chosen as a compromise that is suitable for all plausible ambient solar winds and the large majority of plausible Cone CME scenarios, whilst also not being set so high to be an inefficient use of computing resources.

To initialise HUXt, a uniform solar wind speed of 400 $km\ s^{-1}$ is set at all model coordinates. The model is then spun up for a time period that depends on the minimum solar wind speed in the time-dependent boundary conditions. The spin up time is set to 1.5 times the travel time it would take the slowest speed to traverse the model domain. The results of the spin up are discarded, meaning the user is presented with 'usable' solar wind conditions from the start of the requested simulation time.

3.1 Testing the numerical scheme

HUXt was primarily developed as a surrogate for 3-D MHD simulations for situations where 3-D MHD simulations would be too computationally expensive or complex. Consequently, Owens et al. [13] presented a thorough comparison between HUXt and HelioMAS simulations. Over a 40^+ year period of 578 Carrington rotations, the mean absolute error in the ambient solar wind solutions of HUXt and HelioMAS was 25.6 $km\ s^{-1}$ or

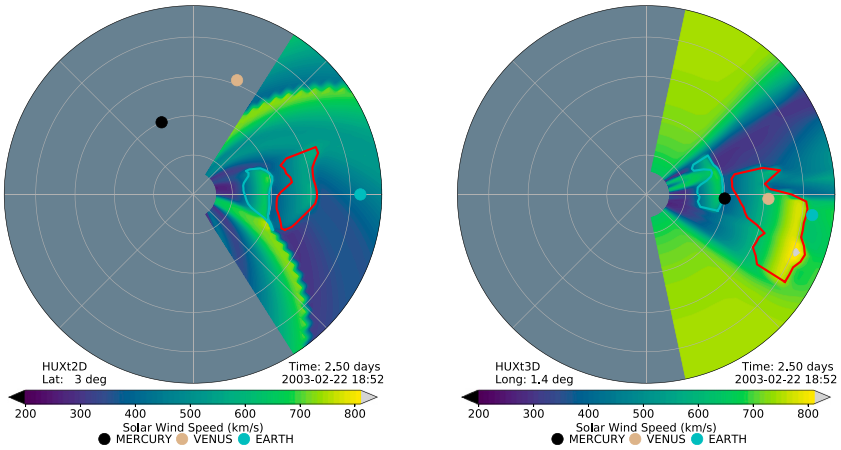


FIGURE 2
A snapshot of a HUXt3D simulation from February 2003. (left) A radius-longitude cut at Earth latitude and (right) a radius-latitude cut at Earth longitude. The boundaries of two cone CMEs are shown in red and cyan.

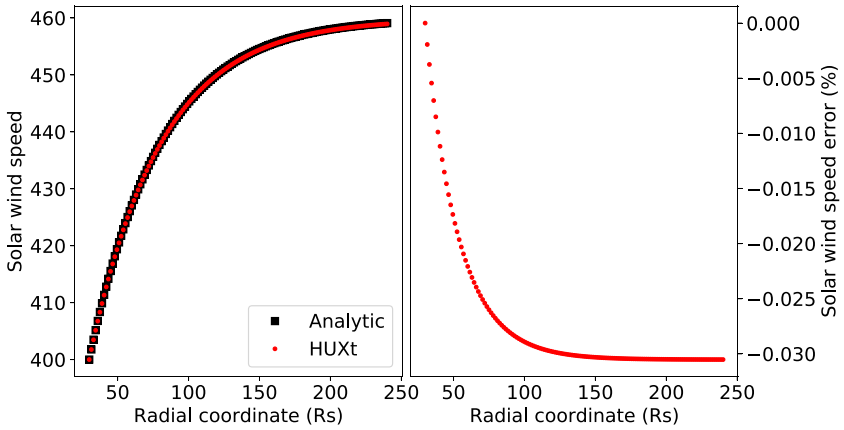


FIGURE 3
Comparison of the analytic and numerical solutions to the HUXt equations for a simplified scenario of uniform and stationary inner boundary conditions. (Left) The solar wind speed as a function of radius for the analytic (black) and HUXt solutions (red). (Right) The percentage error of the HUXt solution relative to the analytical solution as a function of radius.

6.4%. In this sense, the HUXt numerical scheme and default parameters were considered fit-for-purpose in serving as an surrogate for 3-D MHD solar wind simulations. Here we present new analysis that examines the consistency and convergence of the discretised HUXt numerical scheme in approximating the continuous solutions to the model equations.

To assess the consistency of HUXt with a solution to the continuous model equations, we first compare the numerical solution with the analytical solution for the simple scenario of a uniform and stationary inner boundary condition. Figure 3 compares the solutions for a constant inner boundary condition of 400 $km\ s^{-1}$. There is excellent agreement between the HUXt and analytical solution, with a very small negative bias that approaches an asymptotic value of approximately −0.03%. Further experiments show that the magnitude of this error is a function of the radial grid step and so we conclude that this error is related to the discretisation of the HUXt residual acceleration equation. However, the magnitude of this error is insignificant compared to the uncertainties relating to observationally derived boundary conditions (e.g., from coronal models) and from the simplifying physical assumptions used to derive the HUXt equations.

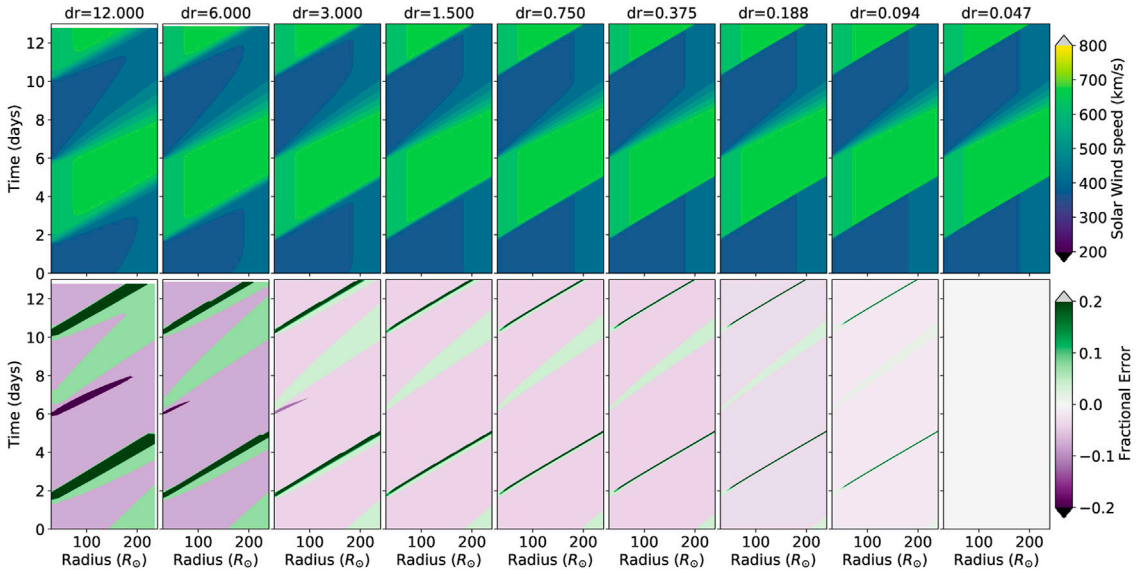


FIGURE 4
(Top) Hovmöller (distance-time) diagrams of the HUXt solar wind speed solutions for the set of different radial grid steps, with radial grid step decreasing from right to left. (Bottom) Hovmöller diagrams of the fractional error in the HUXt solution at a given radial grid step relative the HUXt solution at the finest radial grid step of dr = 0.047 $R_\odot$.

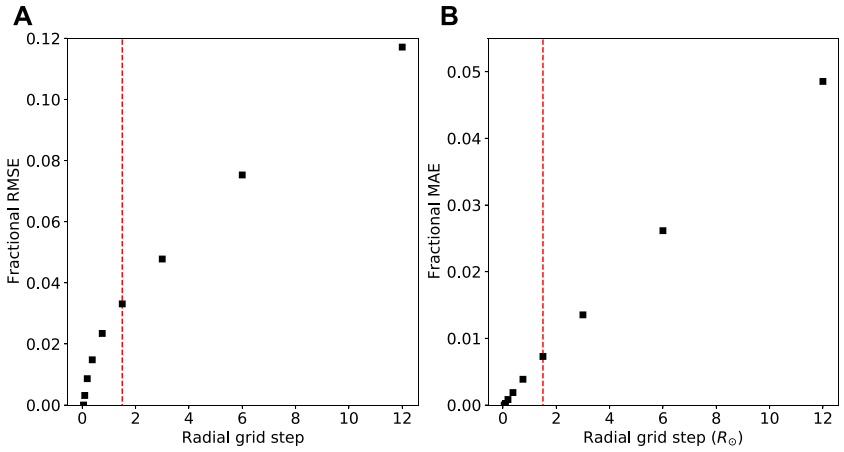


FIGURE 5
(A) The fractional root mean square error (RMSE) of the convergence tests in Figure 4 as a function of radial grid step. **(B)** The corresponding fractional mean absolute error (MAE) as a function of radial grid step. The red dashed lines mark the default HUXt radial grid step of 1.5 $R_\odot$.

The second test is convergence testing, which seeks to assess if and how the numerical solutions are converging towards an exact continuous solution as the discretised grid steps are reduced in size. We ran HUXt for a time dependent scenario of a bimodal solar wind, with slow and fast streams of 350 $km\ s^{-1}$ and 600 $km\ s^{-1}$ at the inner boundary. The model ran for 13.5 days, along a single longitude. The simulations of this scenario were generated using a range of decreasing radial grid steps, with dr being in the set {12, 6, 3, 1.5, 0.75, 0.375, 0.188, 0.094, 0.047}, in units of $R_\odot$. The finest radial grid step, dr = $0.047R_\odot$, was used as a reference standard to compare the other grid steps against.

Figure 4 presents the results of these simulations. For the largest values of dr, there are large regions of significant disagreement with the reference standard. But these errors rapidly decrease with decreasing grid step, which indicates

that the model is converging. As the grid step decreases the errors become more localised to the shock regions between fast and slow streams.

Assessment of the domain over which the model can be said to have converged is subjective, and depends on the acceptable tolerance of error for a particular application. For each of these radial grid steps, we computed both the mean absolute error (MAE) and root mean square error (RMSE). Figure 5 presents the fractional MAE and RMSE data as a function of radial grid step. This summarises what could be inferred from Figure 4, that the errors do decrease with grid step. The red dashed line marks the 1.5 $R_\odot$ grid step, which is the default configuration for HUXt. At this radial grid step, the errors are modest, with MAE being 0.7% and RMSE being 3.3%. Therefore, we consider 1.5 $R_\odot$ to be a sensible upper limit on the radial grid step, which was chosen as the default *dr* to balance the requirement that HUXt run efficiently against the discretisation errors associated with larger radial grid steps. It is simple to specify other radial grid steps in HUXt by modifying the **huxt_constants** function in *huxt.py*.

4 Python implementation

A Jupyter notebook of examples, *HUXt_examples.ipynb*, is provided with HUXt, which shows in detail how HUXt can be used in different scenarios. Here we provide a brief overview of this Python implementation of HUXt. The core codes of HUXt are contained in *huxt.py*, and use of HUXt depends on three classes; **HUXt**, **ConeCME**, and **Observer**. Figure 1 presents a diagram of these classes and their methods. In this diagram, arrows indicate the passing of information from instances of one class to another.

Instantiating an instance of the **HUXt** class configures and initialises the model. There is a minimum required input of specifying the inner boundary condition with an array of solar wind speeds. Different examples of how these can be derived are presented below. Three main methods are attached to the **HUXt** class, **HUXt.solve**, **HUXt.save**, and **HUXt.get_observer**. The **HUXt.solve** method sets the model running with optional inputs such as CMEs and streakline tracing, after which the results will be stored as **HUXt** attributes. In this way it is possible and practical to work with HUXt both programatically and interactively. The **HUXt.save** method writes all the data stored in attributes to an HDF5 file. The **HUXt.ts_from_vlong** method is used to convert a Carrington longitude profile of solar wind speeds into an time series of solar wind speeds at the HUXt inner boundary, under the assumption of synodic or sidereal rotation of the inner boundary. Whilst the **HUXt.get_observer** links to the **Observer** class to provide ephemeris data on a range of solar system bodies interpolated onto the model time grid.

The **Observer** class provides access to ephemeris data for mercury, Venus, Earth, Mars, Jupiter, and Saturn as well as STEREO-A and STEREO-B, for the period spanning 1963-01-01 to 2029-01-01 at 4 h resolution. The **Observer** class requires as input the name of the body and an array of times to output the ephemeris data. The ephemeris are linearly interpolated onto the output times from the 4 h resolution data, and positions are provided in the Heliocentric Earth Equatorial (HEEQ), Heliocentric Aries Ecliptic (HAE), and Carrington coordinate systems.

Cone CMEs are represented by their own class, **ConeCME**. This class requires as input the 6 Cone CME parameters of source longitude, source latitude, full angular width, CME initial speed, CME radial thickness, and launch time relative to the model initiation time. Including Cone CMEs in a HUXt simulation is performed by passing a list of **ConeCME** objects to the **HUXt.solve** method. Attributes of the **ConeCME** class describe all of the CME's properties and its coordinates throughout the HUXt solution. There is also a method attached to the **ConeCME** class to compute the CME arrival time at any of the solar system bodies included in the HUXt ephemeris data. The **ConeCME.parameter_array** method returns a numpy array of the Cone CME parameters, which is primarily intended to pass the CME parameters to the Numba optimised numerical core. This is required as the **HUXt** and **ConeCME** attributes relating to physical parameters are stored as Astropy quantities with the associated units, which are not interoperable with Numba optimised functions. More details on the updated implementation and use of Cone CMEs are discussed Section 5.1.2.

Basic support for 3-D simulations is provided as part of the **HUXt3d** class. In essence, this class is a wrapper around a collection of **HUXt** classes; one for each latitude simulated. Cone CMEs can be included using the same syntax as with the **HUXt** class. At present time-dependent boundary conditions are not fully supported for the **HUXt3d** container class, but all standard data and methods can be accessed for the individual **HUXt** inner classes in the standard manner.

4.1 Test suite

In the latest release of HUXt, v4.0.0, a small test suite has been included in *test_huxt.py*. The aim of the test suite is to help ensure consistency and robustness of HUXt simulations for different users, as well as across future versions of HUXt. The test suite uses the *pytest* testing framework and, at present, includes four tests. The first test is based on the comparison of the HUXt numerical solution and the analytical solution for the simple scenario of a constant inner boundary condition, as discussed in 3.1 and Figure 3. Therefore, this test helps ensure that the numerical scheme is working as expected.

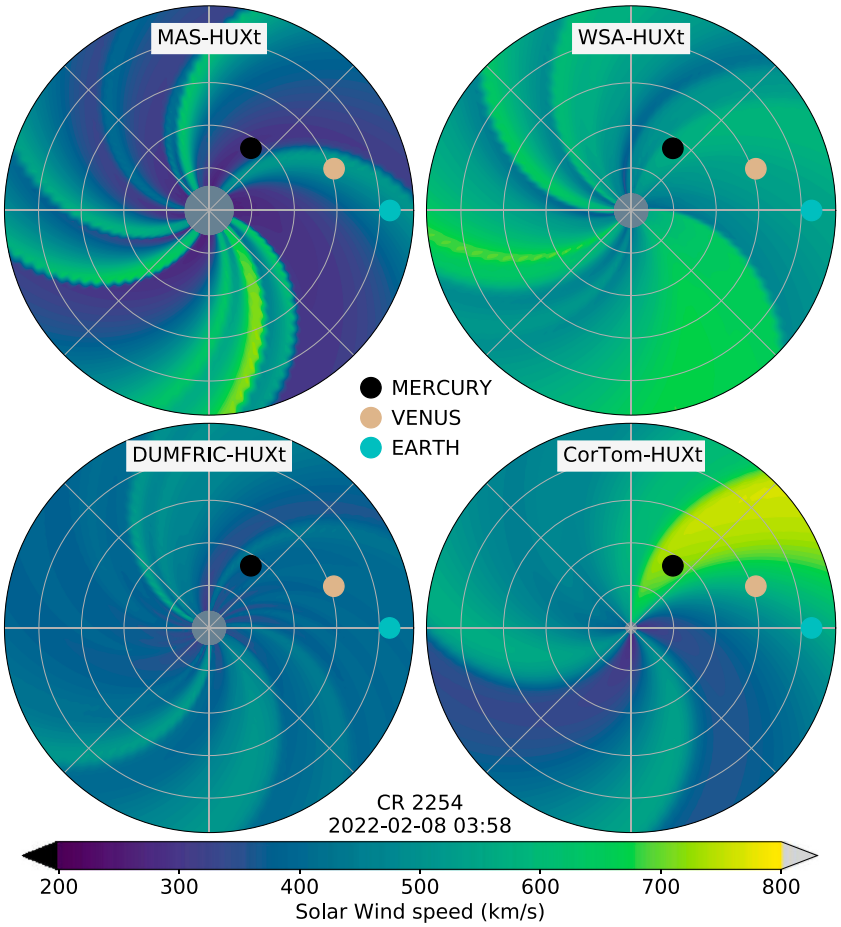


FIGURE 6
HUXt solutions for Carrington rotation 2254, using inner boundary conditions taken from the MAS, WSA, DUMFRIC and CorTom models at the heliographic equator.

The second test compares a HUXt simulation of a Cone CME erupting into structured solar wind with a reference simulation of the same scenario, where these reference data are included as part of HUXt. The solar wind solution, as well as the Cone CME properties and arrival time at Earth are checked for consistency. This test helps to ensure that the core functionality of HUXt is producing consistent results for different users, on different systems, and across different future versions.

The third test similarly compares output with reference data, but to test the reproduction of streakline tracing, described in Section 5.2.2.

Finally, the fourth test compares solutions using an inner boundary at 30 $R_\odot$ with the same boundary conditions backmapped to 10 $R_\odot$, as described in Section 5.2.3. While solutions are not expected to be identical, we define the acceptable tolerance for resulting conditions at 1 AU.

Whilst the current test suite goes some way to ensuring the core functionality of HUXt is performing reliably, at present not all of the HUXt functionality is supported by tests. Therefore it is a development priority to expand the test suite in future versions of HUXt.

5 Functionality

5.1 Updated functionality

5.1.1 Heliospheric extrapolations of coronal model output

The primary function of HUXt is to provide a computationally efficient heliospheric extrapolation of the radial solar wind speed estimated by coronal models, such as, for example, Wang-Sheeley-Arge (WSA) [24], Magnetohydrodyanmics-About-A-Sphere (MAS) Riley et al. [7] and the Durham Magnetofrictional (DUMFRIC) model [25]. HUXt is agnostic concerning the input data series; it is not tuned or intended to be used with

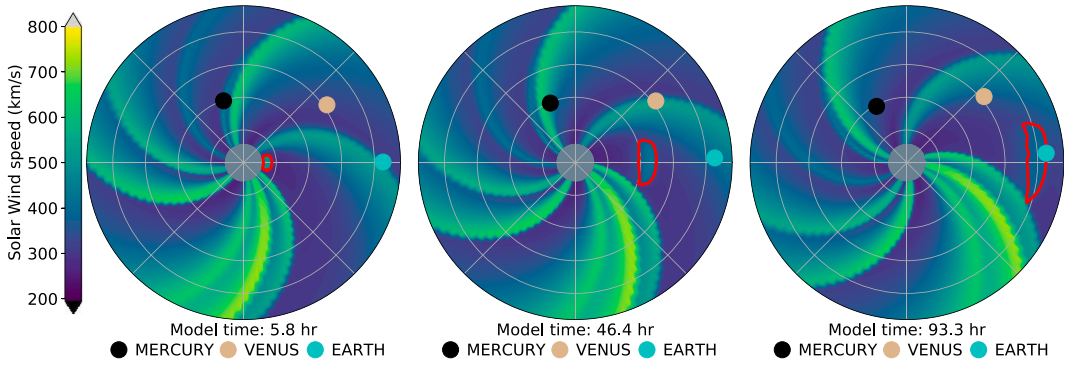


FIGURE 7
Snapshots a HUXt solution including a cone CME. The ambient solar wind boundary condition was taken from MAS for Carrington rotation 2254, as in Figure 6, while the Cone CME parameters represent a climatologically average CME. From left to right, the panels show the Cone CME (in red) shortly after initiation, halfway through its transit to Earth, an on arrival at Earth.

any particular source of boundary conditions. Figure 6 shows the HUXt solutions for Carrington rotation 2254 (beginning 2022-02-08) using boundary conditions taken from the MAS, WSA, and DUMFRIC coronal models, as well as boundary conditions derived from Coronal Tomography (CorTom)[26,27]. MAS provides conditions at 30 $R_\odot$, WSA and DUMFRIC produce conditions at 21.5 $R_\odot$, while CorTom output is at 8 $R_\odot$. HUXt is used with inner boundaries at these heights, without need to map the speeds to other radial distances. Whilst there is some agreement between the stream structure in these solutions, the absolute values and fine scale structure are significantly different. This serves to highlight the impact different model assumptions and architectures can have on the resulting estimate of the state of the corona, and consequently the state of the heliosphere. A detailed analysis of which factors determine the differences between these boundary conditions is outside the scope of this article. But we note that these models each make different approximations regarding the physics governing the structure of the coronal magnetic field; WSA approximates the coronal magnetic field as a potential magnetic field; DUMFRIC approximates the coronal magnetic field as a non-potential field; MAS approximates the coronal state using MHD. Furthermore, for WSA, DUMFRIC, and MHD, differences can arise due to the source and processing of the required magnetogram data [28]. CorTom is more fundamentally distinct from MAS, WSA and DUMFRIC, being based on a tomographic reconstruction of the coronal mass density derived from coronagraph data. Gonzi et al. [28] examined the sensitivity of ENLIL simulations to WSA and DUMFRIC inner boundary conditions. We think a future study that examines the differences between WSA, DUMFRIC, MAS and CorTom derived boundary conditions and their impact on heliospheric simulations would be a valuable addition to the literature.

We also recognise the possibility of using HUXt – which can investigate a large parameter space rapidly—to calibrate the coronal model solar wind speeds [29] in a manner which accounts for heliospheric acceleration and stream interactions. For example, Bunting and Morgan [19] used HUXt in the calibrating an empirical relationship that converts the CorTom coronal densities into solar wind speeds.

Regarding the ambient solar wind boundary conditions, several convenience functions are provided in *huxt_inputs.py* for loading and processing solar wind speed data from the MAS, WSA, PFSS, and DUMFRIC models and CorTom outputs.

5.1.2 Cone coronal mass ejections

CMEs are incorporated in HUXt *via* the Cone CME parameterisation, wherein CMEs are treated as a velocity perturbation at the model inner boundary. Cone CMEs are purely hydrodyanmic structures, and have no magnetic field structure. In Cartesian space the shape of Cone CME is formed by two hemispheres connected by a cylindrical section, akin to a short sausage, with a limiting case of a sphere. A detailed description of the Cone CME geometry is presented in Na et al. [30]. This structure is directed radially away from the Sun and advects through the model inner boundary at the CME speed. Any location on the model inner boundary that intersects the Cone CME volume is assigned the CME speed. A Cone CME is fully specified by 6 parameters; longitude and latitude in HEEQ coordinates, full angular width, speed, thickness (radial length of the cylindrical section), and the launch time relative to the model initialisation time. Conversion of CME coordinates for use with synodic and sidereal frames is handled automatically. Functionality exists for importing a standard 'Cone CME' input file, such as produced by the UK Met Office CAT tool.

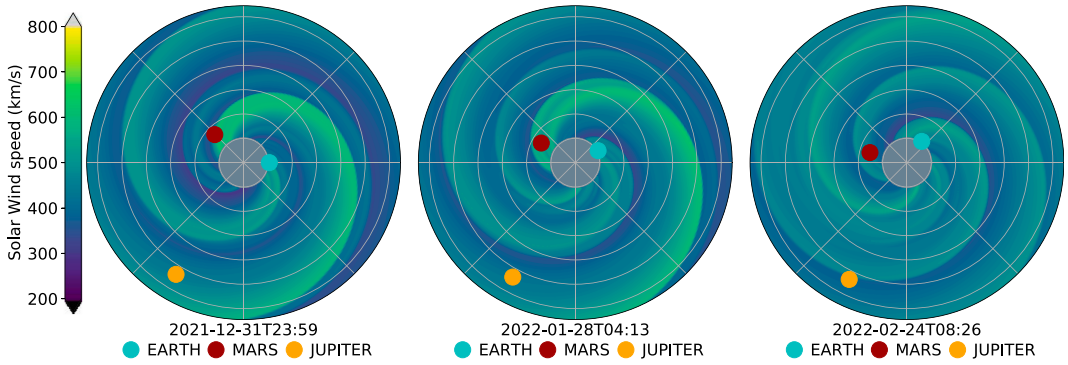


FIGURE 8
Snapshots from a HUXt simulation of a radial domain spanning 1 AU to 6 AU, initialised with the OMNI observations of *in-situ* solar wind conditions near-Earth. The snapshots are taken approximately 27 days apart.

In HUXt v1.0, the position of a Cone CME was tracked by comparing simulations with and without the Cone CME and extracting the boundaries of regions where the simulations differed by more than 20 $km\ s^{-1}$. This approach was generally successful but could struggle to identify the boundary of Cone CMEs with speeds similar to the ambient solar wind speed. To improve upon this, we experimented with tracking the Cone CMEs using a discretised tracer field, but this was found to be too diffusive in practice and consequently required arbitrary thresholds to determine the CME boundaries. Therefore, from v2.0 onwards, Cone CMEs are tracked through the HUXt solution using individual tracer particles that follow the leading and trailing edge of the CME. These tracer particles are inserted into the flow onto the CME boundary as it advects through the model inner boundary and the tracer particles then passively advect through the flow solution. The tracer particles are injected onto every longitudinal and latitudinal coordinate that intersects the ConeCME. An outline of the tracer particle advection algorithm is given in Algorithm 1 The CME boundary is computed automatically at each time-step from the locations of the tracer particles. This method of tracking the CME boundary performs well for all CME speeds, and so is a significant improvement over the tracker function in v1.0.

Algorithm 1 Tracer particle advection

```
for each model time, with time step Δt do
    Update the model solution.
    for each test particle do
        Interpolate the flow speed at the test particle's current location, V_int
        Advance the test particle's radial coordinate by V_int Δt.
        Update array of time history of test particle coordinates.
    end for
end for
```

Figure 7 shows snapshots from an example HUXt simulation including a Cone CME, with the red line marking the boundary of the CME. The Cone CME parameters were set to represent an Earth directed climatological average CME, with initial speed of 495 $km\ s^{-1}$ and full width of 37.4°, where these values were the median speed and width values from the KINCAT catalogue of CME parameters in the HELCATS database [31]. Using the **ConeCME.compute_arrival_at_body()** method, we calculated that the CME in the simulation in Figure 7 arrived at Earth after approximately 93.9 h, with an arrival speed of 360 $km\ s^{-1}$.

5.1.3 Analysis, figures, and animations

A range of basic analysis and plotting capabilities are provided in *huxt_analysis.py*. These capabilities are demonstrated throughout the figures in this paper. Support is provided for:

- Plotting time vs. speed at fixed spatial coordinate.
- Plotting radius vs. speed at fixed longitude and time.
- Polar plot of speed as function of radius and longitude at fixed time and latitude.
- Polar plot of speed as function of radius and latitude at fixed time and longitude.
- Extracting time series of model parameters at bodies included in the **Observer** class.
- Comparison of HUXt simulations with NASA's OMNI data.

Furthermore, both the latitudinal (e.g. Figure 6) and longitudinal (right-hand panel of Figure 2) cuts through the model can be animated over the model run duration using functions provided in *huxt_analysis.py*. Comparison of HUXt simulations with the NASA's OMNI data [32] is facilitated *via* the on-demand download of OMNI data using SunPy's FIDO functionality [33].

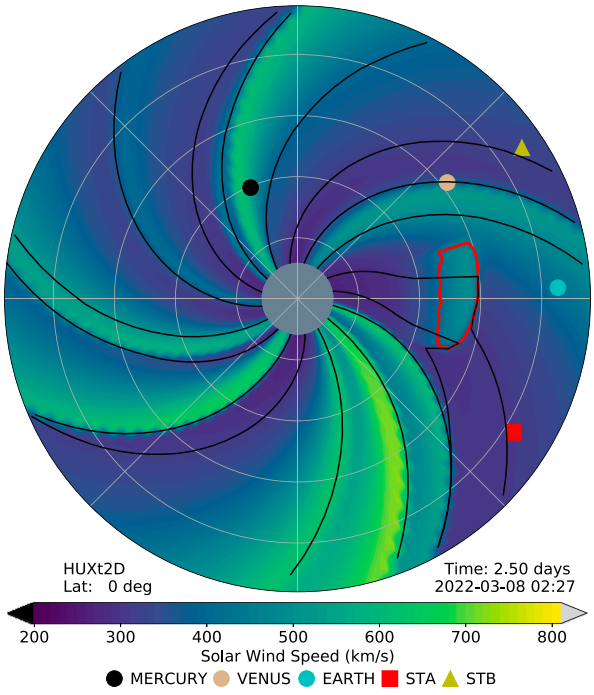


FIGURE 9
A snapshot of the HUXt simulation of Carrington rotation 2254 with streaklines (black lines) plotted for every 22.5° of Carrington longitude. The boundary of a cone CME is shown in red.

5.2 New functionality

5.2.1 *In situ* boundary conditions

HUXt accepts fully time-dependent boundary conditions by specifying solar wind speed (and magnetic field polarity, see Section 5.2.2) as a function of Carrington longitude and time at the inner boundary. In principle, this allows time-dependent coronal model output to be utilised, though this has yet to be fully tested. The second use case is initialisation of HUXt with *in-situ* solar wind measurements that are gridded into a Carrington longitude map. Functions for downloading the necessary OMNI data on-demand and generating the inner boundary condition, as well as setting up the HUXt model, are provided in **huxt_inputs.py**. Simple corotation smoothed back and forward in time (see [34], for more detail) is used to construct the boundary conditions. HUXt will accept inputs from more advanced methods, such as dynamic time warping [34], though such processing is not included as part of the HUXt codebase and can instead be accessed at https://github.com/University-of-Reading-Space-Science/SolarWindInputs_DTW.

Figure 8 shows an example of a HUXt simulation of the region from 1 AU to 6 AU, with the inner boundary condition derived from 4 months of near-Earth solar wind speed observations in the OMNI database [32]. These three snapshots are each approximately one solar rotation (27 days) apart, meaning for time-independent boundary conditions, they would be identical. Here, the time evolution of the ambient solar wind structure can clearly be seen, with the fast solar wind streams decreasing in strength from late 2021 into early 2022. Cone CMEs can be included in such time-dependent boundary condition runs, in the same manner as for 'standard' HUXt runs.

5.2.2 Streakline tracing and sector structure mapping

Functionality exists for tracing streaklines through HUXt flow fields. A streakline is the locus formed by connecting the locations of fluid parcels that originated or passed through a particular location, for example, the curve formed by smoke flowing from a chimney [35]. In HUXt, streaklines are computed by advecting test particles through the flow field from a fixed Carrington longitude. The algorithm for tracking a streakline from a fixed Carrington longitude is similar to tracking the Cone CME boundaries and hence depends primarily on Algorithm 1. The difference between the streakline tracing and the CME boundary tracing is in computing when and where the tracer particles are initialised. For the streaklines, this is done by computing the set of model time and longitude coordinates corresponding to when a specified Carrington longitude rotates into a model longitude bin. Because of the frozen-in-flux theorem, and under the assumption that the magnetic field is passive, the computed streakline will approximate a Parker spiral magnetic field line.

Figure 9 shows a snapshot from a HUXt simulation of CR2254 with streaklines plotted that originate from every 22.5° of Carrington longitude. As expected, the streaklines follow the expected Parker spiral pattern, with 'field lines' in faster flow regions being less tightly wound than in slower flow (c.f. Figure 4 of [36]). An Earth-directed cone CME has been inserted to show the disruption of the Parker spiral, with draping of the 'field' across the CME front.

Previous versions of HUXt used the radial magnetic field polarity at the inner boundary to track magnetic sector structure as a discretised passive tracer field. Unfortunately, this approach proved too diffusive, with narrow sectors being eroded away, particularly for long-duration, outer heliosphere simulations.

The streakline functionality can be used to more effectively track the position of Carrington longitudes of interest through the solar wind, such as the heliospheric current sheet (HCS). Changes in polarity of the radial magnetic field from a coronal model, such as MAS or WSA, can be traced out through the HUXt flow field. This is shown in the left-hand panel of Figure 10. The HCS locations which mark the transition from positive to negative radial field (with increasing radial distance) are shown by white lines, while the converse are shown by black lines. Then, the polarity map is found by associating regions of

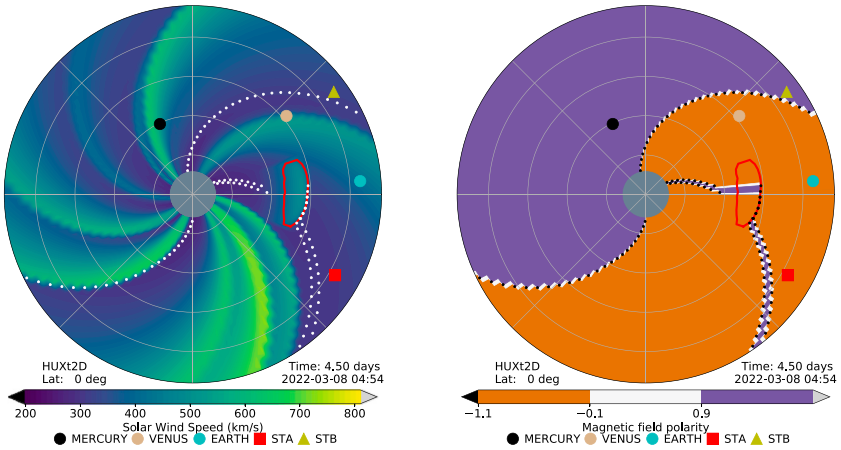


FIGURE 10
Snapshots of the HUXt solar wind speed (left) and magnetic field polarity (right) for Carrington rotation 2254. The boundary of a cone CME is shown in red. Black and white lines show heliospheric current sheet positions of positive to negative radial field transitions (with increasing radial distance) and negative to positive, respectively.

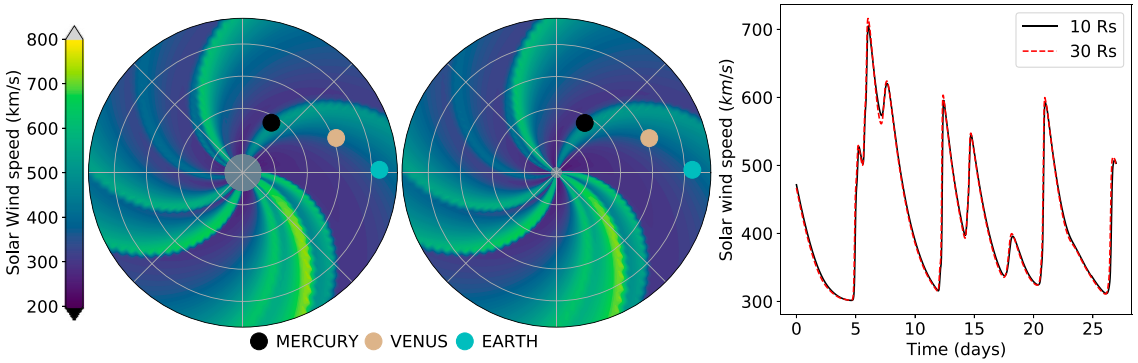


FIGURE 11
(Left) A snapshot of the HUXt simulation of Carrington rotation 2254 initialised at 30 $R_{\odot}$. (Middle) A snapshot of a HUXt simulation of Carrington rotation 2254 where the inner boundary condition has been backmapped to 10 $R_{\odot}$. (Right) Time series of the solar wind speed at Earth from the HUXt simulations initialised at 30 $R_{\odot}$ (red line) and 10 $R_{\odot}$ (black line).

the model domain between the streaklines of the HCS with the appropriate polarity, shown in the right-hand panel of Figure 10. While this period, CR2254, shows a predominantly two-sector magnetic structure, there is a very short pair of HCS crossings around Earth longitude. Despite their proximity, these are preserved by the streakline method. It can also be seen how the addition of a cone CME disrupts the normal Parker spiral pattern of the HCS.

5.2.3 Backmapping

Although the default inner boundary of HUXt is 30 $r_{\odot}$, it is easily configured to use different inner boundary heights. This facilitates easier comparisons with a range of observables, as well as models initialised at other radial distances—for example, the commonly used 21.5 $r_{\odot}$ inner boundary. In these circumstances, it can be necessary to map inner boundary conditions at one altitude to another. For example, mapping the input solar wind speed boundary condition from 30 $r_{\odot}$ down to 15 $r_{\odot}$.

This is a non-trivial calculation, because it is necessary to account for the expected solar wind acceleration between the original and desired altitudes. HUXt provides a function to compute this mapping. For a solar wind parcel on the initial boundary height, this function computes both the expected speed and source longitude of this parcel at the desired altitude, in a manner consistent with the HUXt model dynamics. The derivation of the equations used to compute this mapping are

provided in Appendix A. Note that stream interactions are ignored for backmapping, though this effect is expected to be very small close to the Sun.

Figure 11 shows an example of the results of the backmapping procedure. The left panel shows a HUXt simulation initialised at $30\,R_\odot$ with data from a MAS simulation of CR2254. The middle panel shows a HUXt simulation where the inner boundary was backmapped to $10\,R_\odot$. The right panel shows the resulting solar wind speed time series at Earth for 27 days. The time series are very similar, but the gradients and magnitudes are slightly smaller in the solution initialised at $10\,R_\odot$. This is an artefact of the backmapping procedure, which cannot distinguish between solar wind parcels of different speeds that have distinct source longitudes at a larger radii, but are estimated to have the same source longitude at a smaller radii. These overlapping parcels must necessarily be averaged and interpolated onto the regular HUXt longitude grid, which serves to smooth the inner boundary condition. It is important to be aware of this impact when backmapping an inner boundary condition to a different height.

Note that Cone CME parameters are defined relative to the model inner boundary. Thus moving the inner boundary closer to the Sun will require an earlier Cone CME insertion time and will likely need an increased initial speed, to counteract the addition deceleration of the increased propagation path. As the duration of the Cone CME speed perturbation at the inner boundary is set by the radius of the spherical disturbance, it may also be necessary to increase the Cone CME thickness in order to obtain the same speed perturbation far from the Sun.

5.2.4 3-D solutions

Figure 2 shows snapshots from a **HUXt3D** simulation of Carrington rotation 2000, from February 2003. The right-hand panel of Figure 2 shows a snapshot of the radial-latitudinal plane containing Earth. The left-hand panel of Figure 2 shows the ecliptic plane (i.e. the radial-longitude plane at a constant latitude closest to Earth's latitude). This simulation shows a somewhat typical declining/minimum solar cycle phase structure of fast wind at the poles and slower wind at mid-to-low latitudes. Two cone CMEs have been inserted, with their boundaries shown by the cyan and red lines.

6 Summary and future work

HUXt is an open source reduced physics numerical model of the solar wind, built in Python. The primary purpose of HUXt is to provide a solar wind model that is very computationally cheap and is of minimal complexity, so that it can serve as a surrogate for 3d-MHD solar wind models in context where such simulations are currently impractical, for example, large ensemble simulations and data asssimilative applications.

Version 4.0 includes two key improvements to the models functionality. Firstly, HUXt now accepts fully time-dependant boundary conditions, allowing HUXt to be run by time-series of solar wind speed observations from *in-situ* monitors, such as those provided by the OMNI dataset. Secondly, methods have been included to compute flow streaklines of the HUXt simulations. With these streaklines it is then possible to estimate the location of the Heliospheric Current Sheet, and produce maps radial heliospheric magnetic field polarity.

Additionally, this version is the first to incorporate a small test-suite, which tests the HUXt numerical scheme against a simple analytical solution, and checks a HUXt simulation for consistency with some reference simulation data. This is a first step to improving the reliability and reproducibility of HUXt. Expanding this test suite to cover all of HUXt's functionality is a development priority for future versions.

6.1 Data assimilation

Data assimilation (DA) is the process of combining observations and models, accounting for the uncertainty in both, to achieve an optimal estimate of the state of a system [37]. It is widely used in meteorology to dramatically improve forecasting skill [38], as well as in a number of areas of space physics [39–43]. There are two broad categories of DA methods, variational methods and sequential methods. Variational methodologies, such as 4DVar, aim to find the most probable system state by processing all the observations simultaneously; whereas sequential methodologies, such as the particle filters, aim to find the minimal-variance state by processing observations one-by-one, in a sequential fashion [43]. HUXt is of value to solar wind DA for two reasons.

Firstly, as it is computationally cheap, HUXt is amenable to ensemble DA methods, which require running 10s–1000s of instances of a model for each analysis, be this for research purposes or in a forecasting context [43]. For example, sequential DA methods, such as particle filters [37], use an ensemble of simulations to return an estimate of the state of a dynamical system. DA systems such as this are simple to implement and applicable in a broader range of contexts than some other DA methods (e.g., non-linear systems), but require an ensemble of particles that increases with the dimension of the state space of the model. Consequently, this can become very expensive to compute and becomes quickly impractical for models with large domains and/or numbers of parameters. HUXt is well suited to use with these methods, and we are developing a particle filter DA scheme for assimilating time-elongation profiles of CMEs observed by white-light imagers such as coronagraphs and heliospheric imagers.

Variational DA methods, such as 4DVar, often require the formulation of an 'adjoint' model, with which a models sensitivities to perturbations can be assessed. The relatively simplicity of the HUXt equations and code base means that the 'adjoint' model can be constructed with (relative) ease. This enables powerful variational DA methods to be employed Lang and Owens [44], which are proving very promising for assimilating *in situ* solar wind observations [45].

6.2 Space weather instrumentation, measurement, modelling and risk

The UK government is funding the improvement of its national space weather forecasting capability, particularly through the targeted Space Weather Instrumentation, Measurement, Modelling and Risk (SWIMMR) programme. Within SWIMMR, HUXt is being developed for operational use at the UK Met Office Space Weather Operations Center, as part of the Space Weather Empirical Ensembles Package (SWEEP). The motivation for SWEEP is that the dominant source of uncertainty in numerical model predictions and forecasts of heliosperhic solar wind speed are the uncertainties at the inner boundary of these models.

SWEEP aims to better quantify these uncertainties by producing a large ensemble of forecasts driven with boundary conditions derived from independent data sources (e.g., magnetograms and white light coronal observations) and different physical assumptions, e.g., potential and non-potential coronal magnetic fields. In doing so, SWEEP will produce ensemble forecasts with improved real-world representivity. This work depends critically on having a computationally cheap numerical model like HUXt, as such large ensembles across multiple inputs can not yet be practically generated with operationally used 3D MHD models. SWEEP is expected to be operational by 2024.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/supplementary material.

Author contributions

LB and MO contributed to the conception of this article. MO conceived the HUXt model, whilst both MO and LB co-lead the development of HUXt. Both authors contributed to the writing and revision of the submitted article.

Funding

This work was part-funded by Science and Technology Facilities Council (STFC) grant numbers ST/R000921/1 and ST/V000497/1, and NERC grant number NE/S010033/1.

Acknowledgments

This research made use of Astropy (http://www.astropy.org), a community-developed core Python package for Astronomy [46,47]. This research used version 4.0.0 [48] of the SunPy open source software package [33]. Figures for this article were made with version 3.3.4 of Matplotlib [49,50]. We have benefited greatly from the availability of the large archive of Predictive Science Inc's MAS coronal model solutions (https://www.predsci.com/mhdweb/home.php) for much of the HUXt testing and development and thank Pete Riley for useful discussions on solar wind modelling.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

1. Cannon P, Angling M, Barclay L, Curry C, Dyer C, Edwards R, et al. *Extreme space weather : Impacts on engineered Systems and infrastructure*. London: Royal Academy of Engineering (2013). Tech. rep.

2. Gosling JT. The solar flare myth. *J Geophys Res* (1993) 98:18937–49. doi:10.1029/93JA01896

3. Riley P, Mays ML, Andries J, Amerstorfer T, Biesecker D, Delouille V, et al. Forecasting the arrival time of coronal mass ejections: Analysis of the CCMC CME scoreboard. *Space Weather* (2018) 16:1245–60. doi:10.1029/2018SW001962

4. Gopalswamy N, Yashiro S, Michalek G, Stenborg G, Vourlidas A, Freeland S, et al. The SOHO/LASCO CME catalog. *Earth Moon Planets* (2009) 104:295–313. doi:10.1007/s11038-008-9282-7

5. Case AW, Spence HE, Owens MJ, Riley P, Odstrcil D. Ambient solar wind's effect on ICME transit times. *Geophys Res Lett* (2008) 35:L15105. doi:10.1029/2008GL034493

6. Merkin VG, Lyon JG, Lario D, Arge CN, Henney CJ. Time-dependent magnetohydrodynamic simulations of the inner heliosphere. *JGR Space Phys* (2016) 121:2866–90. doi:10.1002/2015JA022200

7. Riley P, Linker JA, Mikić Z. An empirically-driven global MHD model of the solar corona and inner heliosphere. *J Geophys Res* (2001) 106:15889–901. doi:10.1029/2000JA000121

8. Odstrcil D. Modeling 3-D solar wind structure. *Adv Space Res* (2003) 32: 497–506. doi:10.1016/S0273-1177(03)00332-6

9. Narechania NM, Nikolić L, Freret L, Sterck HD, Groth CPT. An integrated data-driven solar wind – CME numerical framework for space weather forecasting. *J Space Weather Space Clim* (2021) 11:8. doi:10.1051/swsc/2020068

10. Mays ML, Taktakishvili A, Pulkkinen A, MacNeice PJ, Rastätter L, Odstrcil D, et al. Ensemble modeling of CMEs using the WSA–ENLIL+Cone model. *Sol Phys* (2015) 290:1775–814. doi:10.1007/s11207-015-0692-1

11. Odstrcil D, Mays ML, Hess P, Jones SI, Henney CJ, Arge CN. Operational modeling of heliospheric space weather for the parker solar probe. *Astrophys J Suppl Ser* (2020) 246:73. doi:10.3847/1538-4365/ab77cb

12. Riley P, Lionello R. Mapping solar wind streams from the Sun to 1 AU: A comparison of techniques. *Sol Phys* (2011) 270:575–92. doi:10.1007/s11207-011-9766-x

13. Owens MJ, Lang M, Barnard L, Riley P, Ben-Nun M, Scott CJ, et al. A computationally efficient, time-dependent model of the solar wind for use as a surrogate to three-dimensional numerical magnetohydrodynamic simulations. *Sol Phys* (2020) 295:43. doi:10.1007/s11207-020-01605-3

14. Barnard L, Owens MJ, Scott CJ, de Koning CA. Ensemble CME modeling constrained by heliospheric imager observations. *AGU Adv* (2020) 1: e2020AV000214. doi:10.1029/2020AV000214

15. Chi Y, Scott C, Shen C, Barnard L, Owens M, Xu M, et al. Modeling the observed distortion of multiple (ghost) CME fronts in STEREO heliospheric imagers. *Astrophys J Lett* (2021) 917:L16. doi:10.3847/2041-8213/ac1203

16. Barnard L, Owens M. University-of-Reading-Space-Science/HUXt. *Huxt Zenodo*. (2021). doi:10.5281/zenodo.4889327

17. Hinterreiter J, Amerstorfer T, Temmer M, Reiss MA, Weiss AJ, Möstl C, et al. Drag-based CME modeling with heliospheric images incorporating frontal deformation: ELEvoHI 2.0. *Space Weather* (2021) 19:e2021SW002836. doi:10.1029/2021SW002836

18. Macneil AR, Owens MJ, Finley AJ, Matt SP. A statistical evaluation of ballistic backmapping for the slow solar wind: The interplay of solar wind acceleration and corotation. *Mon Not R Astron Soc* (2022) 509:2390–403. doi:10.1093/mnras/stab2965

19. Bunting KA, Morgan H. An inner boundary condition for solar wind models based on coronal density. *J Space Weather Space Clim* (2022) 12:30. doi:10.1051/swsc/2022026

20. Pizzo V. A three-dimensional model of corotating streams in the solar wind, 1. Theoretical foundations. *J Geophys Res* (1978) 83:5563–72. doi:10.1029/JA083iA12p05563

21. Reiss MA, MacNeice PJ, Muglach K, Arge CN, Möstl C, Riley P, et al. Forecasting the ambient solar wind with numerical models. II. An adaptive prediction system for specifying solar wind speed near the Sun. *Astrophys J* (2020) 891:165. doi:10.3847/1538-4357/ab78a0

22. Press W, Teukolsky S, Vetterling W, Flannery B. *Numerical recipes - the art of scientific computing*. 3rd ed. Cambridge University Press (2007).

23. Owens MJ, Riley P. Probabilistic solar wind forecasting using large ensembles of near-sun conditions with a simple one-dimensional "upwind" scheme. *Space Weather* (2017) 15:1461–74. doi:10.1002/2017SW001679

24. Arge CN, Pizzo VJ. Improvement in the prediction of solar wind conditions using near-real time solar magnetic field updates. *J Geophys Res* (2000) 105: 10465–79. doi:10.1029/1999JA000262

25. Yeates AR, Mackay DH, van Ballegooijen AA, Constable JA. A nonpotential model for the Sun's open magnetic flux. *J Geophys Res* (2010) 115. doi:10.1029/2010JA015611

26. Morgan H. An atlas of coronal electron density at $5R_{\odot}$. II. A spherical harmonic method for density reconstruction. *Astrophys J Suppl Ser* (2019) 242:3. doi:10.3847/1538-4365/ab125d

27. Morgan H, Cook AC. The width, density, and outflow of solar coronal streamers. *Astrophys J* (2020) 893:57. doi:10.3847/1538-4357/ab7e32

28. Gonzi S, Weinzierl M, Bocquet F-X, Bisi MM, Odstrcil D, Jackson BV, et al. Impact of inner heliospheric boundary conditions on solar wind predictions at Earth. *Space Weather* (2020) 19:1–40. doi:10.1029/2020SW002499

29. McGregor SL, Hughes WJ, Arge CN, Owens MJ, Odstrcil D. The distribution of solar wind speeds during solar minimum: Calibration for numerical solar wind modeling constraints on the source of the slow solar wind. *J Geophys Res* (2011) 116. doi:10.1029/2010JA015881

30. Na H, Moon Y-J, Lee H. Development of a full ice-cream cone model for halo coronal mass ejections. *Astrophys J* (2017) 839:82. doi:10.3847/1538-4357/aa697c

31. Barnes D, Davies JA, Harrison RA, Byrne JP, Perry CH, Bothmer V, et al. CMEs in the heliosphere: III. A statistical analysis of the kinematic properties derived from stereoscopic geometrical modelling techniques applied to CMEs detected in the heliosphere from 2008 to 2014 by STEREO/HI-1. *Sol Phys* (2020) 295:150. doi:10.1007/s11207-020-01717-w

32. King JH, Papitashvili NE. Solar wind spatial scales in and comparisons of hourly Wind and ACE plasma and magnetic field data. *J Geophys Res* (2005) 110: A02104. doi:10.1029/2004JA010649

33. Barnes WT, Bobra MG, Christe SD, Freij N, Hayes LA, Ireland J, et al. The SunPy project: Open source development and status of the version 1.0 core package. *Astrophys J* (2020) 890:68. doi:10.3847/1538-4357/ab4f7a

34. Owens MJ, Nichols JD. Using *in situ* solar-wind observations to generate inner-boundary conditions to outer-heliosphere simulations – I. Dynamic time warping applied to synthetic observations. *Mon Not R Astron Soc* (2021) 508: 2575–82. doi:10.1093/mnras/stab2512

35. Batchelor GK. *An introduction to fluid dynamics*. Cambridge: Cambridge Mathematical LibraryCambridge University Press (2000). doi:10.1017/CBO9780511800955

36. Owens MJ, Forsyth RJ. The heliospheric magnetic field. *Living Rev Sol Phys* (2013) 10. doi:10.12942/lrsp-2013-5

37. Van Leeuwen PJ. Particle filtering in geophysical systems. *Mon Weather Rev* (2009) 137:4089–114. doi:10.1175/2009MWR2835.1

38. Kalnay E. *Atmospheric modeling, data assimilation and predictability — atmospheric science and meteorology*. Cambridge University Press (2005). doi:10.1017/CBO9780511802270

39. Brun AS. Towards using modern data assimilation and weather forecasting methods in solar physics. *Astron Nachr* (2007) 328:329–38. doi:10.1002/asna.200610739

40. Hickmann KS, Godinez HC, Henney CJ, Arge CN. Data assimilation in the ADAPT photospheric flux transport model. *Sol Phys* (2015) 290:1105–18. doi:10.1007/s11207-015-0666-3

41. Murray SA, Henley EM, Jackson DR, Bruinsma SL. Assessing the performance of thermospheric modeling with data assimilation throughout solar cycles 23 and 24. *Space Weather* (2015) 13(4): 220–232. doi:10.1002/2015SW001163

42. Chartier AT, Matsuo T, Anderson JL, Collins N, Hoar TJ, Lu G, et al. Ionospheric data assimilation and forecasting during storms. *J Geophys Res Space Phys* (2016) 121:764–78. doi:10.1002/2014JA020799

43. Lang M, Browne P, van Leeuwen PJ, Owens MJ. Data assimilation in the solar wind: Challenges and first results. *Space Weather* (2017) 15:1490–510. doi:10.1002/2017SW001681

44. Lang M, Owens MJ. A variational approach to data assimilation in the solar wind. *Space Weather* (2019) 17:59–83. doi:10.1029/2018SW001857

45. Lang M, Witherington J, Turner H, Owens MJ, Riley P. Improving solar wind forecasting using data assimilation. *Space Weather* (2021) 19:e2020SW002698. doi:10.1029/2020SW002698

46. Robitaille TP, Tollerud EJ, Greenfield P, Droettboom M, Bray E, Aldcroft T, et al. Astropy: A community Python package for astronomy. *Astron Astrophys* (2013) 558:A33. doi:10.1051/0004-6361/201322068

47. Price-Whelan AM, Sipőcz BM, Günther HM, Lim PL, Crawford SM, Conseil S, et al. The Astropy project: Building an open-science project and status of the v2.0 core package. *Astron J* (2018) 156:123. doi:10.3847/1538-3881/aabc4f

48. Mumford SJ, Freij N, Stansby D, Christe S, Ireland J, Mayer F, et al. SunPy. *Zenodo* (2022). doi:10.5281/zenodo.6524764

49. Caswell TA, Droettboom M, Lee A, de Andrade ES, Hunter J, Firing E, et al. Matplotlib/matplotlib: Rel: V3.3.4. *Zenodo* (2021). doi:10.5281/zenodo.4475376

50. Hunter JD. Matplotlib: A 2D graphics environment. *Comput Sci Eng* (2007) 9: 90–5. doi:10.1109/MCSE.2007.55

Appendix A: Backmapping equations

Substituting Eq. 3 into Eq. 2, we obtain

$$V = V_0 + \alpha V_0 - \alpha V_0 e^{\left(\frac{r_0 - r}{r_h}\right)} \tag{4}$$

The travel time for a solar wind parcel to propagate between and inner radius, r_{in}, to an outer radius, r_{out}, is

$$T = \int_{r_{in}}^{r_{out}} \frac{1}{V} dr \tag{5}$$

Defining $A = V_0 + \alpha V_0$, this integral becomes

$$T = \int_{r_{in}}^{r_{out}} \frac{1}{A - \alpha V_0 e^{\left(\frac{r_0 - r}{r_h}\right)}} dr \tag{6}$$

For which the solution is

$$T = \left[\frac{r_h \, log\left(Ae^{\left(\frac{r}{r_h}\right)} - \alpha V_0 e^{\left(\frac{r_0}{r_h}\right)} \right)}{A} \right]_{r_{in}}^{r_{out}} \tag{7}$$

This travel time can be used in conjunction with the synodic solar rotation rate to estimate the source longitude of a solar wind parcel at an inner radius relative to its source longitude at the outer radius.

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Original Research
PUBLISHED 20 October 2022
DOI 10.3389/fspas.2022.1023612

Check for updates

OPEN ACCESS

EDITED BY
John Coxon,
Northumbria University,
United Kingdom

REVIEWED BY
Jingnan Guo,
University of Science and Technology of China, China
Sam Mangham,
University of Southampton,
United Kingdom

*CORRESPONDENCE
Jonathan T. Niehof,
jonathan.niehof@unh.edu

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 20 August 2022
ACCEPTED 03 October 2022
PUBLISHED 20 October 2022

CITATION
Niehof JT, Morley SK, Welling DT and Larsen BA (2022), The SpacePy space science package at 12 years.
Front. Astron. Space Sci. 9:1023612.
doi: 10.3389/fspas.2022.1023612

COPYRIGHT
© 2022 Niehof, Morley, Welling and Larsen. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

The SpacePy space science package at 12 years

Jonathan T. Niehof[1]*, Steven K. Morley[2], Daniel T. Welling[3,4] and Brian A. Larsen[2]

[1]Space Science Center, University of New Hampshire, Durham, NH, United States, [2]Space Science and Applications (ISR-1), Los Alamos National Laboratory, Los Alamos, NM, United States, [3]Department of Physics, University of Texas at Arlington, Arlington, TX, United States, [4]Department of Climate and Space, University of Michigan, Ann Arbor, MI, United States

For over a decade, the SpacePy project has contributed open-source solutions for the production and analysis of heliophysics data and simulation results. Here we introduce SpacePy's functionality for the scientific user and present relevant design principles. We examine recent advances and the future of SpacePy in the broader scientific Python ecosystem, concluding with some of the work that has used SpacePy.

KEYWORDS

python, space physics, space plasmas, magnetosphere, heliosphere, software, open source

1 Introduction

The roots of the SpacePy project (Morley et al., 2022) date back to 2009, although the first public presentation of SpacePy was to the 9th Python in Science conference in 2010 (Morley et al., 2011). The mission statement of the open source library, given in both the conference proceedings and on the original web page, was "to promote accurate and open research standards by providing an open environment for code development. In the space physics community there has long been a significant reliance on proprietary languages that restrict free transfer of data and reproducibility of results. By providing a comprehensive library of widely-used analysis and visualization tools in a free, modern and intuitive language, we hope that this reliance will be diminished for non-commercial users"(Morley et al., 2011).

Now, 12 years after the presentation of SpacePy and eleven after its initial open source release, we present a summary of the present state of the library, a retrospective view of SpacePy's development, and a look to the future of SpacePy and its place in the heliophysics scientific software ecosystem (e.g., Burrell et al., 2018). For scientists in the field, we introduce the research-enabling functionality of SpacePy and the scientific Python ecosystem, including examples of previous studies. For research software engineers (e.g., Crouch et al., 2013), we discuss how SpacePy is designed to interoperate with the greater technical and social ecosystem of heliophysics software.

The domain of SpacePy is space physics broadly speaking, i.e., heliospheric and magnetospheric physics, including magnetosphere-ionosphere coupling. Strict solar physics and isolated ionospheric physics are outside of the usual scope, although SpacePy functionality may be useful in those fields.

2 Design goals

SpacePy is designed as a library; that is, its primary access is *via* the public application programming interface (API), rather than a user-facing application. Target users are scientists and engineers writing custom code for specialized analysis and visualization of data or model results, producing archival-quality data sets, or creating a more user-facing interactive application. The developers consider SpacePy a success when it is used to provide functionality to higher-level codes and uses lower-level libraries to provide the "building blocks" of such functionality.

As with other scientific libraries, the reliability and fidelity of results is essential. A thorough testing suite ensures reproducibility of results, and a test-first approach to fixing bugs prevents regressions. Absolute accuracy of results up to the numerical precision of the computer is considered less important than documenting the expected precision, the regimes in which results are reliable, and the source of the algorithm, including citation of the literature where appropriate (e.g., in empirical models).

SpacePy development is user-driven: functionality is developed to meet a specific scientific or mission operations goal. Developers are scientists in the field. This ensures applicability of the implementation; the distinction between SpacePy and project-specific code is the conversion to maintained, tested, and widely applicable functionality.

On the computational side, the API aims to be carefully designed, "Pythonic" in nature (Alexandru et al., 2018) and in accordance with software engineering good practices such as abstraction. One illustration of the success of this approach is the pycdf interface (Section 3.1.1). Although independently developed, the resulting interface is very similar to the h5py HDF5 library (Collette, 2013); the SpacePy datamodel was developed along similar lines at the same time. This minimizes the cognitive load required to access similar data from differing container formats, allowing the user to focus on problem solving rather than interface peculiarities.

SpacePy has been available to the general public under an open source license since 2011. The SpacePy license is essentially that of Python itself, with the only change being replacing references to Python with SpacePy, and to the Python Software Foundation with Triad National Security, LLC, as the initial licensor. This license, often called "the PSF license", is a BSD-style permissive license approved by the Open Source Initiative, the Free Software Foundation, and the Debian Free Software Guidelines. Before arrangements were made for this public release, SpacePy was briefly provided under a restrictive non-commercial license upon request (Morley et al., 2011).

The SpacePy install and update process supports a range of deployment and update strategies. Although the most common means of installation is automatic management *via* pip, manual download and installation from source remain options for those users who wish to install into a shared location on a multi-user system, do not have full Internet access on their deployed system, or have other particular needs. Similarly, SpacePy supports a wide range of versions of its dependencies and changes these requirements at specific version numbers only (where the subminor version is 0, e.g., 0.2.0, 0.3.0). API changes are also made at predetermined version numbers, with deprecation warnings providing a graceful migration path. The documentation clearly states versions of API changes, even for versions in the past, to support users updating their code. The SpacePy team recognizes that users have a range of needs, may have limited control of their operating environment, and need to interoperate with other packages which may have stricter requirements; thus SpacePy is designed to be as flexible as practicable on these issues.

SpacePy takes a balanced approach to using other packages as dependencies: maximizing the use of mature, robust dependencies decreases the maintenance load of SpacePy itself, as well as enhancing interoperability, but may place additional burden on users (even those who do not use the functionality for which a dependency is required). The approach is to bring in a dependency where it provides significant (rather than incidental) functionality, ideally supporting multiple components of SpacePy. The specifics are left intentionally vague. More importantly, the functionality provided by each dependency is explicitly documented, and SpacePy will install without most dependencies. Section 6 describes the future direction of dependency handling.

Over the past few years (Section 4), compliance with Python in Heliophysics Community (PyHC) standards (Annex et al., 2018) has been a major design consideration.

3 Capability and architecture

All capabilities described in this section are available in the current release, SpacePy 0.4.0, available at https://pypi.org/project/spacepy/. Capabilities are also summarized in the SpacePy documentation at https://spacepy.github.io/capabilities.html. A graphical overview of key namespaces (i.e., modules) in SpacePy is shown in Figure 1.

3.1 Datamodel

On of the core capabilities of SpacePy is its data model representation, which was introduced shortly after Morley et al. (2011). SpacePy uses a description, based on that used by HDF5, which uses three key concepts: groups, datasets, and attributes. Groups are analogous to file system directories, and can contain both groups and datasets. Datasets are n-dimensional arrays of data. Attributes are metadata that is carried with either a group or a dataset. SpacePy's spacepy.datamodel.SpaceData class

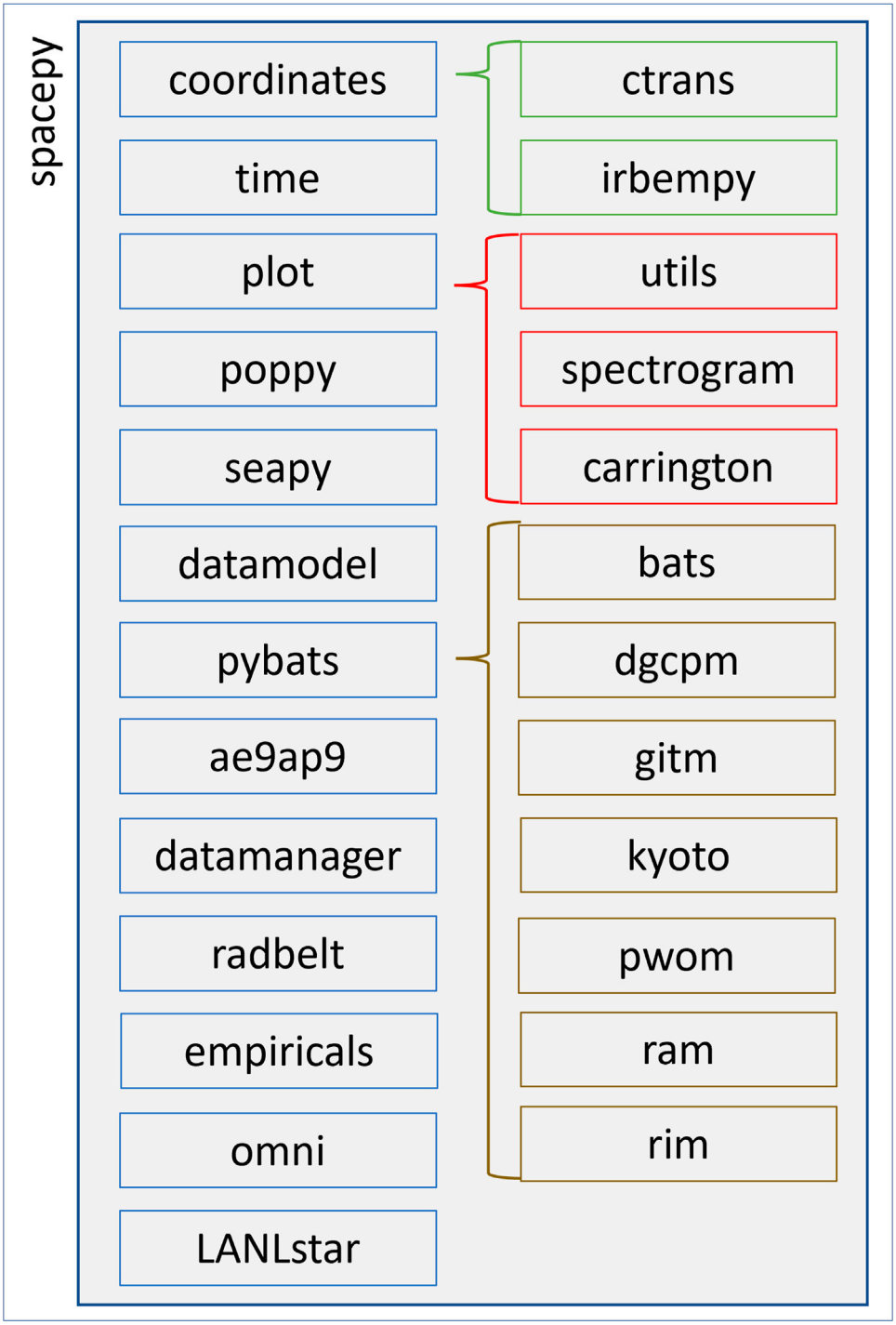


FIGURE 1
Core capabilities organized by namespace in the SpacePy package.

implements the group by subclassing dict, while the spacepy.datamodel.dmarray implements the dataset as a subclass of numpy.ndarray. Each of these classes carries metadata in a Python dictionary accessed *via* the attrs attribute. The structure of the object can be displayed using the spacepy.datamodel.SpaceData.tree method.

```
>>> import spacepy.datamodel as dm
>>> sdata = dm.SpaceData(attrs={"Descriptor": "Test file"})
>>> sdata["Variable0"] = dm.dmarray([0,1,2],
                                    attrs={"CATDESC": "An increasing value"})
>>> sdata["Variable1"] = dm.dmarray([2.7]*3,
                                    attrs={"CATDESC": "The value 2.7 repeated"})
>>> sdata.tree(attrs=True)
+
:|____Descriptor
|____Variable0
    :|____CATDESC
|____Variable1
    :|____CATDESC
```

The datamodel thus provides a file-format agnostic representation of data that preserves metadata. The data model objects can be constructed and used without requiring either input or output, however, read and write support is provided. Supported file formats include:

- NASA CDF: NASA's Common Data Format.
- HDF5: Hierarchical Data Format 5.
- NetCDF: Unidata's Network Common Data Form.
- JSON-headed ASCII.

Spacepy's datamodel readers are currently all greedy by default, in that they load files all-at-once. While this is convenient for many users, for very large data files or for systems with read/write speed limitations this can be sub-optimal.

The datamodel is normally agnostic to the interpretation of metadata, so it can be used for a wide range of metadata standards. This may be a simple human-readable informal representation. Additional functions are provided for the case where metadata are ISTP/SPDF compliant.

3.1.1 NASA's common data format

SpacePy has provided first class support for NASA CDF, including full read and write, since September 2010 through the spacepy.pycdf module. pycdf provides a pythonic interface to the NASA CDF library, and requires that the user obtain that library from NASA. This approach is taken to reduce duplication of functionality and maintain a clear separation of responsibility: NASA develops and maintains CDF, while SpacePy develops and maintains the Python interface. pycdf reads files "on demand", with the ability to read a single variable or even fraction thereof. spacepy.datamodel.fromCDF provides an at-once read into the spacePy datamodel.

3.1.2 Hierarchical data format 5 and derivatives

Several other formats and packages derive from HDF5 and can, unless non-standard features are added, be read directly with spacepy.datamodel.fromHDF5. For example, since MATLAB® release R2006b, mat files can be (and are most likely to be) stored as HDF5 files. Also, NetCDF4 provides an alternative API to build and read data files using the HDF5 library. NetCDF4 files can thus be read using spacepy.datamodel.fromHDF5. Note that NetCDF3 is not compatible with HDF5, even though NetCDF4 provides access to legacy NetCDF3 files. spacepy.datamodel.fromNC3 provides NetCDF-to-datamodel reader functionality by building on the scipy.io.netcdf reader.

To write the contents of a spacepy.datamodel.SpaceData to an HDF5 file, simply call the appropriate write method:

```
>>> sdata.toHDF5("output_filename.h5")
```

3.1.3 Javascript object notation-headed ASCII

This is a text-based data format that uses a header, written in JavaScript Object Notation (JSON) and intended to be both human- and machine-readable, to describe the file layout and to store metadata. While not in broad use, this provides specific support for the magnetic ephemeris ("magephem") files for the Van Allen Probes Energetic particle, Composition, and Thermal plasma (RBSP-ECT) Suite, as well as the energetic particle data from the Global Positioning System (Morley et al., 2017). This format is also supported by Autoplot (Faden et al., 2010).

3.2 Time systems

Handling time and coordinate systems is fundamental to much of space physics. While these capabilities were present in the original release of SpacePy, there have been significant advances over the years.

SpacePy supports multiple time systems: Coordinated Universal Time (both as native Python datetime objects, and expressed as ISO8601 time strings); International Atomic Time (TAI), in seconds since 1958-01-01T00:00:00 UTC; Global Positioning System (GPS) time, in seconds since 1980-01-06T00:00:00 UTC; Julian Day and Modified Julian Day (expressed on the UTC scale); Unix time; Rata Die time, in days since 0001-01-01T00:00:00 UTC; and CDF time (corresponding to the legacy CDF_EPOCH types in NASA's CDF library). Figure 2 shows relationships between these time systems; internal processing is primarily in TAI.

3.2.1 Handling leap seconds

Some time systems ignore leap seconds (e.g., Unix time). Similarly, many standard library time packages do not handle leap seconds, including Python's datetime module (even as used for UTC). On the other hand, there is a need in heliophysics to represent leap seconds and to convert between continuous time representations and those that ignore leap seconds. These conversions are well-defined from the introduction of leap seconds to UTC in 1972 to the present. For systems that cannot represent leap seconds, the leap second moment is considered not to exist. For example, from 23:59:59 on 31/12/2008 to 00:00:00 on 01/01/2009 is 2 s, but only represents a 1-s increment in Unix time. SpacePy uses an user-updatable leap second table referenced to the latest US Naval Observatory data.

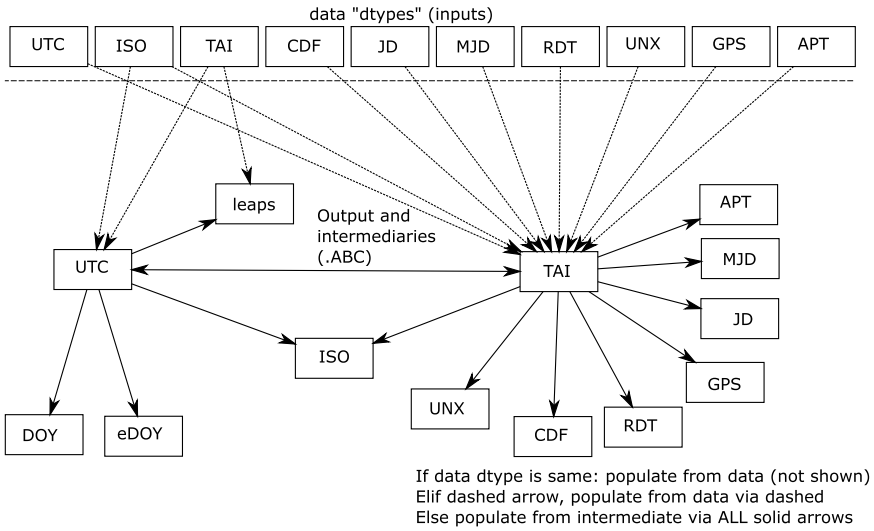


FIGURE 2
Workflow for supported input and output types in SpacePy.time.

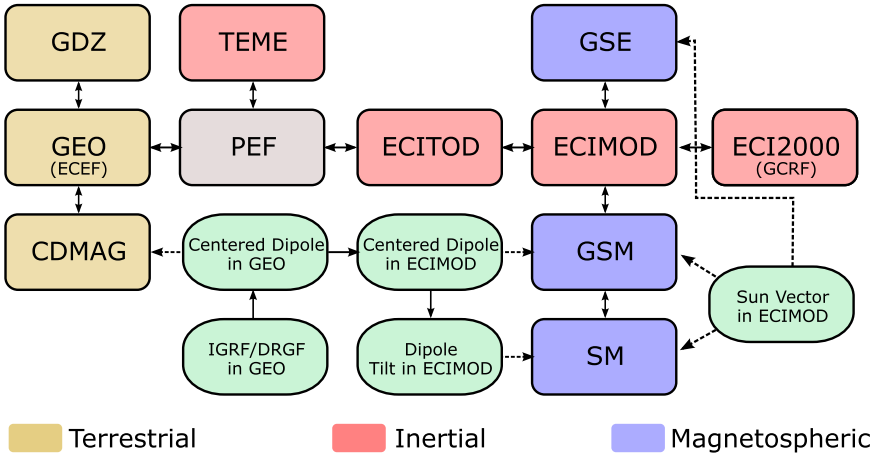


FIGURE 3
Relationships between coordinate systems as implemented in SpacePy.ctrans.

3.3 Coordinates

Since its first release SpacePy has provided a pythonic interface to the IRBEM library (Boscher et al., 2022). This includes access to magnetic field models, field line tracing, and coordinate transformations. In release 0.3 SpacePy introduced a new backend for handling coordinate system transformations, while simultaneously preserving the familiar spacepy.coordinates.Coords interface. This new backend maintained existing functionality, requiring no changes to existing code, while removing the need for Fortran support (for the IRBEM library) to perform coordinate transformations. Both backends are available to the user.

Coordinate systems supported by this module broadly fall into two categories: those that can be defined strictly using astronomical parameters only, and those that require a representation of Earth's geomagnetic field. SpacePy uses transformations that build from the IAU 1976/FK5 system for astrophysical reduction (Lederle, 1980; Fricke, 1982; Seago and Vallado, 2000). Taking the origin of our coordinate systems as

the center of the Earth instead of the solar barycenter gives us an Earth-centered inertial (ECI) system as our starting point. The relationships between the supported coordinate systems are described below, and graphically summarized in Figure 3. In contrast with many other space or heliophysics packages we do not follow the approach given by Russell (1971) or Hapgood (1992) of using first order approximations to the reduction theory. SpacePy uses the full third order relationships in its implementation.

3.3.1 Earth-centered inertial systems

Our fundamental reference system is ECI 2000, sometimes simply referred to as the J2000 frame, though we avoid this to prevent confusion with the J2000 epoch (01 January 2000, 11:58: 55.816 UTC) This system can be considered equivalent to the Geocentric Celestial Reference Frame, to within tens of milliarcseconds. The *z*-axis is perpendicular to the mean celestial equator at the J2000 epoch. The *x*-axis is aligned with the mean equinox at the J2000 epoch. The *y*-axis completes and lies in the celestial equatorial plane.

Correcting the orientations of the equator and equinox for precession yields the mean equinox and mean equator of date, and updating the definition gives us ECIMOD (ECI Mean Of Date). Finally, we account for the nutation (the short-period perturbations on the precession) to obtain the true equator and true equinox of date. Using these corrected axes to define our ECI system gives ECITOD (ECI True Of Date).

3.3.2 Terrestrial systems: Geographic, geodetic, and geomagnetic

SpacePy implements an Earth-Centered Earth-Fixed coordinate system using the name GEO (Geocentric Geographic). The coordinates of a point fixed on (or relative to) the surface of the Earth do not change as the Earth rotates. The *x*-axis lies in the Earth's equatorial plane (zero latitude) and intersects the Prime Meridian (zero longitude; Greenwich, United Kingdom). The *z*-axis points to True North (which is roughly aligned with the instantaneous rotation axis).

While all of the coordinate systems thus far are generally defined as Cartesian systems, geodetic (GDZ) coordinates cannot be properly represented as Cartesian. GDZ is defined in terms of altitude above a reference ellipsoid, the geodetic latitude, and geodetic longitude. Geodetic longitude is identical to geographic longitude, while both the altitude and latitude depend on the ellipsoid used. SpacePy's default is the WGS84 reference ellipsoid and the GEO-GDZ conversion uses Heikkinen's exact algorithm (see Zhu, 1994).

Finally, geomagnetic coordinates can be considered a magnetic analog of GEO. The *z*-axis is aligned with the centered dipole axis of date (defined using the first 3 coefficients of the IGRF/DGRF). The *y*-axis is perpendicular to both the dipole axis and True North and the *x*-axis completes the system.

3.3.3 Magnetospheric systems

Magnetospheric coordinate systems are non-inertial and Earth-centered. We begin with GSE (Geocentric Solar Ecliptic). The *x*-axis points from the center of Earth to the Solar System barycenter, while the *y*-axis is defined to lie in the mean ecliptic plane of date (pointing in the anti-orbit direction) and the *z*-axis is perpendicular to the mean ecliptic plane.

To move to GSM (Geocentric Solar Magnetospheric) we require that the centered dipole axis lies in the x-z plane. The *y*-axis is thus perpendicular to both the Sun-Earth line and the centered dipole axis. GSM is therefore a rotation about the *x*-axis from the GSE system. Finally, we move to SM (Solar Magnetic) where the *z*-axis is aligned with the centered dipole axis of date (positive northward), and the *y*-axis is perpendicular to both the Sun-Earth line and the dipole axis. As with GSE and GSM, y is positive in the anti-orbit direction. The *x*-axis therefore is not aligned with the Sun-Earth line and SM is a rotation about the *y*-axis from the GSM system.

We note that these definitions differ slightly from those used by, e.g., Hapgood (1992) as the mean ecliptic (correcting for precession) is used instead of the true ecliptic (correcting for precession and nutation), with the Earth-Sun vector also defined in ECIMOD. However, they have been adopted for consistency with recent flagship missions following the implementations used for Van Allen Probes and Magnetospheric Multiscale (e.g., Morley, 2015).

3.4 Pybats

The Pybats module of Spacepy provides tools for handling output from the Space Weather Modeling Framework (Tóth et al., 2005, 2012; Gombosi et al., 2021). The SWMF is a framework that executes, synchronizes, and couples together many physics-based domain models of the complex heliosphere system, from solar corona to planetary atmospheres (e.g., Powell et al., 1999; Welling et al., 2015; Mukhopadhyay et al., 2020; Sachdeva et al., 2021). It is widely used in heliophysics, including long-standing availability at NASA's Community Coordinated Modeling Center (CCMC) and real-time operational use at NOAA's Space Weather Prediction Center (SWPC) since 2016. Its wide adoption has necessitated a tool box for reading and handling its complex and heterogeneous output—a need met by the Pybats module.

The fundamental goal of Pybats is to allow users to access SWMF model output within Python environments. It achieves this by subclassing spacepy.datamodel.SpaceData to include file read methods called upon instantiation. This allows for easy exploration of values and attributes as outlined above. In the base spacepy.pybats module, classes are provided for data formats defined at the SWMF control level or common across many SWMF sub-models. Sub-modules provide model-specific

functionality and customization of base classes. For example, the BATS-R-US global MHD model (Powell et al., 1999; De Zeeuw et al., 2000; Groth et al., 2000) produces basic ASCII log files that follow a standard SWMF-defined format and are readable *via* spacepy.pybats.LogFile objects. However, the spacepy.pybats.bats submodule provides model-specific classes and capabilities. When opening BATS-R-US log files, the spacepy.pybats.bats.BatsLog subclass includes additional methods for visualizing values inherent to the BATS-R-US output data, such as Dst index. Conversely, the output files from the Ridley Ionosphere Model [RIM, Ridley et al. (2001)] are proprietary formats, so the base classes for handling RIM output are located in the spacepy.pybats.rim sub-module.

The most fundamental file type handled by Pybats is the SWMF IDL format, which has suffix .out. These files are of a format proprietary to the SWMF, may be either ASCII or binary, and can hold 1, 2, or 3D data sets. These files can be concatenated together to hold multiple epochs of simulation of data in a single file (a .outs file), allowing users to reduce the total number of files produced by a single simulation. SWMF IDL files are used by many different models, including BATS-R-US, PWOM, DGCPM, and others. The base class spacepy.pybats.IdlFile automatically detects file format (ASCII *versus* binary) upon instantiation, reads the file into a spacepy.datamodel.SpaceData-like object, and provides tools to navigate the different frames, or single-epoch sets, stored within the file.

An animation of SWMF output using SpacePy is available in the Supplemental Material and at https://www.youtube.com/watch?v=8bgkgQITFO8. This animation shows the magnetospheric response as the interplanetary magnetic field switches from a purely northward to purely southward direction. The simulation was performed using the SWMF, coupling the BATS-R-US global MHD model with the Rice Convection Model and the RIM. For this simulation, the physics-based Adaptive Mesh Refinement (AMR) capability of BATS-R-US was used to automatically increase spatial resolution to a minimum of 1/8 Earth Radii (R_E). The grid was refined in any block where the current density surpassed $10^{-5}\,\mu A/m^2$ and coarsened if the current dropped below $5 \times 10^{-7}\,\mu A/m^2$. Visualization of the model output was performed entirely with SpacePy's pybats module and submodules. Current density contours in the equatorial plane were plotted using the spacepy.pybats.Bats2d.add_contour method. The colored squares show the BATS-R-US block tree structure; the color of each square shows the grid resolution of the block with brighter colors indicating the regions of finest grid spacing. The spacepy.pybats.Bats2d.add_grid_plot method was used to add the grid information to screen. The animation demonstrates how BATS-R-US AMR can be used to simulate fine structure within the magnetosphere, including Kelvin-Helmholtz instabilities, flux transfer events, and fast flow channels in the tail.

3.5 Interoperability

To maximize flexibility for the researcher and minimize duplication of effort, SpacePy emphasizes interoperability with other packages. SpacePy's reliance on the widely-used NumPy (Harris et al., 2020) array package provides a baseline of low-level interoperability, and the datamodel (Section 3.1) was designed to make the minimum changes necessary to the NumPy array interface.

SpacePy's Ticktock time object supports conversion to and from Astropy's (Astropy Collaboration et al., 2013; Astropy Collaboration et al., 2018) astropy.time.Time representation; similarly, SpacePy Coords can be converted to and from the Astropy astropy.coordinates.SkyCoord. Both conversions are *via* simple to/from methods of the SpacePy objects.

SkyCoord conversion is performed *via* the Earth-centered Earth-fixed frame (GEO in SpacePy, ITRS in astropy). Time conversion uses SpacePy's TAI format and Astropy's TAI scale with GPS format, both being continuously-running counts of seconds since a defined epoch.

Transformation of data structures to and from additional packages is in preparation (Section 4).

3.6 Empirical models

Via the Pythonic interface to the IRBEM library (irbempy), SpacePy supports a wide range of magnetic field models and operations on them, including field line and drift shell tracing. The LANL* neural net based model (Yu et al., 2012; Yu et al., 2014) provides faster calculation of the third adiabatic invariant and the last closed drift shell. This model has recently been migrated from the Fortran-based ffnet library to a new implementation based on numpy linear algebra routines, while maintaining the neural network structure, weights, and results.

Other empirical models include plasmapause models, the magnetopause model of Shue et al. (1997), and access to the output of the AE9/AP9 radiation belt model (Ginet et al., 2013). As inputs to these and other models, SpacePy provides the omni module, simplifying access to the upstream solar wind data set of King and Papitashvili (2005) using the interpolation techniques of Qin et al. (2007).

4 Recent activities

In the summer of 2018, SpacePy transitioned from an open source release model to a fully open development model. All development is done in a "live" github repository at https://github.com/spacepy/spacepy, issues and enhancements are processed with full public visibility, and developer commits go through the same review and workflow as outside contributors. The result has been not only feature requests and issues from the

user community, but also new and improved functionality. AstroPy coordinate support and the new LANL* processing are two examples where the core functionality came from the community and was integrated into SpacePy with developer support.

The transition away from Python 2 is concluding. Although SpacePy has fully supported Python 3 since version 0.1.5 (December 2014), Python 2 support was retained. This has been slowly phased out over several releases, providing time for users to update. Soon Python 2 code will be removed, simplifying the codebase and facilitating further transitions, such as the move away from distutils, which were not possible while supporting Python 2. We successfully supported a dual-version codebase with very little version-specific code for over 7 years.

An ongoing project will connect the datamodel of SpacePy with the HAPI streaming Heliophysics data protocol (Weigel et al., 2021) and the data structures of the SunPy library (The SunPy Community et al., 2020). This work supports the use of functionality in a range of libraries without forcing users into a single data representation or a single library ecosystem. We intend continual interoperability with other packages within the broader scientific Python community.

Part of the datamodel conversions project is extending the ability of SpacePy to interpret ISTP/SPDF metadata (Kovalick, 2022) regardless of its container. Ultimately this will allow the easy manipulation of data using the ISTP metadata standard regardless of its container (SpaceData, HDF5, or CDF). This will not change the fundamental nature of the SpacePy datamodel, which is agnostic to the form of metadata, only allow additional functionality where the metadata are ISTP-compliant.

SpacePy developers have been regularly engaging with the PyHC, including participation in the 2022 summer School (https://heliopython.org/summer-school).

5 Applying SpacePy

SpacePy has been used in many scientific studies as well as in support of mission data processing; only a few examples are provided here.

Recent uses of SpacePy in scientific publications range from probabilistic predictions of geomagnetic storms (Chakraborty and Morley, 2020), visualization and verification of an improved inner magnetosphere model (Engel et al., 2019), through calculation of L-shells on Cubesats (Gieseler et al., 2020) to modeling of geomagnetic response to a "perfect storm" ICME (Welling et al., 2021).

In missions, SpacePy supported the data processing for the Radiation Belt Storm Probes Energetic particle, Composition, and Thermal plasma suite (RBSP-ECT) (Spence et al., 2013; Manweiler et al., 2022), including the ECT combined electron product (Boyd et al., 2019). SpacePy supports data management within the Magnetospheric Multiscale mission (MMS) magnetic ephemeris processing chain (Morley, 2015). Data from the Integrated Science Investigation of the Sun suite (McComas et al., 2016) on Parker Solar Probe are processed with SpacePy.

Functionality used in earlier studies remains fully maintained and available for other studies, such as superposed epoch analysis (Morley et al., 2010; Rogers, 2022) and association of point processes (Niehof et al., 2012).

The SpacePy team maintains a list of publications at https://spacepy.github.io/publications.html and welcomes submissions.

The reference of record for SpacePy code is Morley et al. (2022). SpacePy users are also encouraged to cite the present work in studies which make use of SpacePy. Code releases are available via the PyPI at https://pypi.org/project/spacepy/, development is hosted at https://github.com/spacepy/spacepy/, and documentation at https://spacepy.github.io/.

6 Future directions and challenges

It is clear that the Heliophysics community move away from IDL is well underway, so the SpacePy goal of reducing "reliance on proprietary languages" is at least partially accomplished, through the efforts of many in the community. Proprietary languages are likely to retain some importance but the place of Python as a tool is well established. One significant question then is what the nature of the Python in Heliophysics ecosystem will be.

Since Python is an easy language to write, and modern environments such as github and the Python Package Index (PyPI) make sharing easy, Heliophysics-related Python packages have proliferated. This has resulted in potential issues of duplication of effort and interoperability between packages. The PyHC project has done an excellent job of making packages aware of each other so that they can voluntarily evaluate existing functionality, avoid duplication, and work on interoperability. Given diversity of workflows, facilitating this work is more likely to be successful than any attempt to force the community into a single approved package for each function. Interoperability does raise the possibility of circular dependencies, but this need not be a problem. As long as packages do not depend on each other for installation, modern package managers will successfully install both. Careful interface design can then avoid circular imports; this has been the case for the datamodel interoperability project (which will also produce a set of recommendations for facilitating interoperability).

As the scientific Python ecosystem grows, SpacePy's dependency strategy is constantly evolving. One solution may be for some generic SpacePy functionality to migrate "up the stack" into e.g., scipy; another (not exclusive) may be to use the optional specifications of PEP508 (Collins, 2015) to only install dependencies for SpacePy functionality that a user specifically requests.

One major shift over the life of SpacePy has been the transition from source-based distribution to binary-based (e.g., operating-system specific binary wheels). This places additional demands on package developers, not only in producing these binaries but in supporting newer build systems. The result can be a substantial improvement in ease of installation for the end user, and SpacePy is transitioning away from the assumption that a user will have a working compiler, even on Unix-based systems. Maintaining flexibility of deployment remains a priority. Supporting this installer transition requires significant computer engineering work which is largely separate from the domain expertise.

To date, SpacePy development has been supported primarily *via* the missions that benefit from it. Short-term independent support has been secured to support engineers in addressing these computer engineering based tasks more efficiently than using physics domain experts. We hope similar support will continue across the Python ecosystem, as it is essential to high-quality software.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary Material, further inquiries can be directed to the corresponding author.

Author contributions

SM, JN, and DW wrote the initial draft and made substantial revisions and additions to all portions of the manuscript. All authors contributed to the manuscript concept, read, and approved the submitted version, and have routinely contributed code, documentation, and reviews to SpacePy over the past decade.

Funding

SpacePy development has been supported by several missions, including the Van Allen Probes Radiation Belt Storm Probe, Energetic particle, Composition, and Thermal plasma suite, JHU/APL contract 967399 under NASA prime contract NAS5-01072; and the Parker Solar Probe Integrated Science Investigation of the Sun, JHU/APL contract 136435 under NASA prime contract NNN06AA01C. Further support was provided by United States Government contract 89233218CNA000001 for Los Alamos National Laboratory (LANL), which is operated by Triad National Security, LLC for the United States Department of Energy; and NASA Grant 80NSSC21K0304 "Enhancing Heliophysics Python Library Interoperability by Adapting Common Data Models".

Acknowledgments

The authors would like to thank everybody who has contributed to SpacePy *via* pull requests, discussions, and well-formed issues.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fspas.2022.1023612/full#supplementary-material

References

Alexandru, C. V., Merchante, J. J., Panichella, S., Proksch, S., Gall, H. C., and Robles, G. (2018). "On the usage of pythonic idioms," in *Proceedings of the 2018 ACM SIGPLAN international symposium on new ideas, new paradigms, and reflections on programming and software* (New York, NY, USA: Association for Computing Machinery), 1–11. Onward!. doi:10.1145/3276954.3276960

Annex, A., Alterman, B. L., Azari, A., Barnes, W., Bobra, M., Cecconi, B., et al. (2018). *Python in heliophysics community (PyHC) standards*. Zenodo. doi:10.5281/zenodo.2529131

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., et al. (2018). The astropy project: Building an open-science project and status of the v2.0 core package. *Astron. J.* 156, 123. doi:10.3847/1538-3881/aabc4f

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., et al. (2013). Astropy: A community Python package for astronomy. *Astron. Astrophys.* 558, A33. doi:10.1051/0004-6361/201322068

Boscher, D., Bourdarie, S., O'Brien, T. P., Guild, T., Heynderickx, D., Morley, S., et al. (2022). *PRBEM/IRBEM: v5.0.0 (IRBEM-5.0.0)*. Zenodo. doi:10.5281/zenodo.6867768

Boyd, A. J., Reeves, G. D., Spence, H. E., Funsten, H. O., Larsen, B. A., Skoug, R. M., et al. (2019). RBSP-ECT combined spin-averaged electron flux data product. *JGR. Space Phys.* 124, 9124–9136. doi:10.1029/2019JA026733

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—an overview of Python in heliophysics. *J. Geophys. Res. Space Phys.* 123, 10384–10402. doi:10.1029/2018JA025877

Chakraborty, S., and Morley, S. K. (2020). Probabilistic prediction of geomagnetic storms and the K_p index. *J. Space Weather Space Clim.* 10, 36. doi:10.1051/swsc/2020037

Collette, A. (2013). *Python and HDF5*. California, United States: O'Reilly.

Collins, R. (2015). Dependency specification for Python software packages. PEP 508.

Crouch, S., Hong, N. C., Hettrick, S., Jackson, M., Pawlik, A., Sufi, S., et al. (2013). The software sustainability institute: Changing research software attitudes and practices. *Comput. Sci. Eng.* 15, 74–80. doi:10.1109/MCSE.2013.133

De Zeeuw, D., Gombosi, T., Groth, C., Powell, K., and Stout, Q. (2000). An adaptive MHD method for global space weather simulations. *IEEE Trans. Plasma Sci. IEEE Nucl. Plasma Sci. Soc.* 28, 1956–1965. doi:10.1109/27.902224

Engel, M. A., Morley, S. K., Henderson, M. G., Jordanova, V. K., Woodroffe, J. R., and Mahfuz, R. (2019). Improved simulations of the inner magnetosphere during high geomagnetic activity with the ram-scb model. *J. Geophys. Res. Space Phys.* 124, 4233–4248. doi:10.1029/2018JA026260

Faden, J. B., Weigel, R. S., Merka, J., and Friedel, R. H. W. (2010). Autoplot: A browser for scientific data on the web. *Earth Sci. Inf.* 3, 41–49. doi:10.1007/s12145-010-0049-0

Fricke, W. (1982). Determination of the equinox and equator of the FK5. *Astronomy Astrophysics* 107, L13–L16.

Gieseler, J., Oleynik, P., Hietala, H., Vainio, R., Hedman, H.-P., Peltonen, J., et al. (2020). Radiation monitor RADMON aboard aalto-1 CubeSat: First results. *Adv. Space Res.* 66, 52–65. doi:10.1016/j.asr.2019.11.023

Ginet, G. P., O'Brien, T. P., Huston, S. L., Johnston, W. R., Guild, T. B., Friedel, R., et al. (2013). AE9, AP9 and SPM: New models for specifying the trapped energetic particle and space plasma environment. *Space Sci. Rev.* 179, 579–615. doi:10.1007/s11214-013-9964-y

Gombosi, T. I., Chen, Y., Glocer, A., Huang, Z., Jia, X., Liemohn, M. W., et al. (2021). What sustained multi-disciplinary research can achieve: The space weather modeling framework. *J. Space Weather Space Clim.* 11, 42. doi:10.1051/swsc/2021020

Groth, C. P. T., De Zeeuw, D. L., Gombosi, T. I., and Powell, K. G. (2000). Global three-dimensional MHD simulation of a space weather event: CME formation, interplanetary propagation, and interaction with the magnetosphere. *J. Geophys. Res.* 105, 25053–25078. doi:10.1029/2000JA900093

Hapgood, M. (1992). Space physics coordinate transformations: A user guide. *Planet. Space Sci.* 40, 711–717. doi:10.1016/0032-0633(92)90012-D

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

King, J. H., and Papitashvili, N. E. (2005). Solar wind spatial scales in and comparisons of hourly wind and ace plasma and magnetic field data. *J. Geophys. Res.* 110, A02104. doi:10.1029/2004JA010649

Kovalick, T. (2022). Space physics guidelines for CDF use. Tech. Rep., NASA SPDF. Available at: https://spdf.gsfc.nasa.gov/sp_use_of_cdf.html.

Lederle, T. (1980). The IAU (1976) system of astronomical constants. *Mittl. Astron. Ges. Hambg.* 48, 59–65.

Manweiler, J. W., Breneman, A., Niehof, J., Larsen, B., Romeo, G., and Stephens, G. (2022). Science of the Van Allen Probes science operations centers. *Space Sci. Rev.* 179. in press.

McComas, D. J., Alexander, N., Angold, N., Bale, S., Beebe, C., Birdwell, B., et al. (2016). Integrated science investigation of the Sun (ISIS): Design of the energetic particle investigation. *Space Sci. Rev.* 204, 187–256. doi:10.1007/s11214-014-0059-1

Morley, S. K., Friedel, R. H. W., Spanswick, E. L., Reeves, G. D., Steinberg, J. T., Koller, J., et al. (2010). Dropouts of the outer electron radiation belt in response to solar wind stream interfaces: Global positioning system observations. *Proc. R. Soc. A* 466, 3329–3350. doi:10.1098/rspa.2010.0078

Morley, S. K., Koller, J., Welling, D. T., Larsen, B. A., Henderson, M. G., and Niehof, J. T. (2011). "Spacepy - a python-based library of tools for the space sciences," in Proceedings of the 9th Python in science conference (SciPy 2010), Austin, TX, June 28 - July 3 2010.

Morley, S. K. (2015). Magnetic ephemeris and coordinates: Level 2 ephemeris product update. *Zenodo*. doi:10.5281/zenodo.2594027

Morley, S. K., Niehof, J. T., Welling, D. T., Larsen, B. A., Haiducek, J., Brunet, A., et al. (2022). Spacepy. *Zenodo*. doi:10.5281/zenodo.3252523

Morley, S. K., Sullivan, J. P., Carver, M. R., Kippen, R. M., Friedel, R. H. W., Reeves, G. D., et al. (2017). Energetic particle data from the global positioning system constellation. *Space weather.* 15, 283–289. doi:10.1002/2017SW001604

Mukhopadhyay, A., Welling, D. T., Liemohn, M. W., Ridley, A. J., Chakraborty, S., and Anderson, B. J. (2020). Conductance model for extreme events: Impact of auroral conductance on space weather forecasts. *Space weather.* 18, e2020SW002551. doi:10.1029/2020SW002551

Niehof, J. T., Morley, S. K., and Friedel, R. H. W. (2012). Association of cusp energetic ions with geomagnetic storms and substorms. *Ann. Geophys.* 30, 1633–1643. doi:10.5194/angeo-30-1633-2012

Powell, K., Roe, P., Linde, T., Gombosi, T. I., and De Zeeuw, D. L. (1999). A solution-adaptive upwind scheme for ideal magnetohydrodynamics. *J. Comput. Phys.* 154, 284–309. doi:10.1006/jcph.1999.6299

Qin, Z., Denton, R. E., Tsyganenko, N. A., and Wolf, S. (2007). Solar wind parameters for magnetospheric magnetic field modeling. *Space weather.* 5. doi:10.1029/2006SW000296

Ridley, A. J., De Zeeuw, D. L., Gombosi, T. I., and Powell, K. G. (2001). Using steady state MHD results to predict the global state of the magnetosphere-ionosphere system. *J. Geophys. Res.* 106, 30067–30076. doi:10.1029/2000JA002233

Rogers, A. J. (2022). The active tail in the MMS era: An ion perspective. Ph.D. thesis. (Durham: University of New Hampshire).

Russell, C. T. (1971). Geophysical coordinate transformations. *Cosm. Electrodyn.* 2, 184.

Sachdeva, N., Tóth, G., Manchester, W. B., van der Holst, B., Huang, Z., Sokolov, I. V., et al. (2021). Simulating solar maximum conditions using the alfvén wave solar atmosphere model (AWSoM). *Astrophys. J.* 923, 176. doi:10.3847/1538-4357/ac307c

Seago, J., and Vallado, D. (2000). Coordinate frames of the U.S. Space object catalogs, in Astrodynamics Specialist Conference. 14 August 2000 - 17 August 2000 Denver,CO,U.S.A doi:10.2514/6.2000-4025

Shue, J. H., Chao, J. K., Fu, H. C., Russell, C. T., Song, P., Khurana, K. K., et al. (1997). A new functional form to study the solar wind control of the magnetopause size and shape. *J. Geophys. Res.* 102, 9497–9511. doi:10.1029/97JA00196

Spence, H. E., Reeves, G. D., Baker, D. N., Blake, J. B., Bolton, M., Bourdarie, S., et al. (2013). Science goals and overview of the radiation belt storm Probes (RBSP) energetic particle, composition, and thermal plasma (ECT) suite on NASA's van allen Probes mission. *Space Sci. Rev.* 179, 311–336. doi:10.1007/s11214-013-0007-5

Tóth, G., Sokolov, I. V., Gombosi, T. I., Chesney, D. R., Clauer, C. R., De Zeeuw, D. L., et al. (2005). Space weather modeling framework: A new tool for the space science community. *J. Geophys. Res.* 110, A12226. doi:10.1029/2005JA011126

Tóth, G., van der Holst, B., Sokolov, I. V., De Zeeuw, D. L., Gombosi, T. I., Fang, F., et al. (2012). Adaptive numerical algorithms in space weather modeling. *J. Comput. Phys.* 231, 870–903. doi:10.1016/j.jcp.2011.02.006

The SunPy Community, Barnes, W. T., Bobra, M. G., Christe, S. D., Freij, N., Hayes, L. A., et al. (2020). The sunpy project: Open source development and status of the version 1.0 core package. *Astrophys. J.* 890, 68. doi:10.3847/1538-4357/ab4f7a

Weigel, R. S., Vandegriff, J., Faden, J., King, T., Roberts, D. A., Harris, B., et al. (2021). Hapi: An API standard for accessing heliophysics time series data. *JGR. Space Phys.* 126, e29534. doi:10.1029/2021JA029534

Welling, D. T., Jordanova, V. K., Glocer, A., Toth, G., Liemohn, M. W., and Weimer, D. R. (2015). The two-way relationship between ionospheric outflow and the ring current. *JGR. Space Phys.* 120, 4338–4353. doi:10.1002/2015JA021231

Welling, D. T., Love, J. J., Rigler, E. J., Oliveira, D. M., Komar, C. M., and Morley, S. K. (2021). Numerical simulations of the geospace response to the arrival of an idealized perfect interplanetary coronal mass ejection. *Space weather.* 19, e02489. doi:10.1029/2020SW002489

Yu, Y., Koller, J., Jordanova, V. K., Zaharia, S. G., Friedel, R. W., Morley, S. K., et al. (2014). Application and testing of the L* neural network with the self-consistent magnetic field model of RAM-SCB. *J. Geophys. Res. Space Phys.* 119, 1683–1692. doi:10.1002/2013JA019350

Yu, Y., Koller, J., Zaharia, S., and Jordanova, V. (2012). L* neural networks from different magnetic field models and their applicability. *Space weather.* 10, 02014. doi:10.1029/2011SW000743

Zhu, J. (1994). Conversion of earth-centered earth-fixed coordinates to geodetic coordinates. *IEEE Trans. Aerosp. Electron. Syst.* 30, 957–961. doi:10.1109/7.303772

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 21 October 2022
DOI 10.3389/fspas.2022.1006839

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Lei Cai,
University of Oulu, Finland
Luke Barnard,
University of Reading, United Kingdom

*CORRESPONDENCE
Alex K. Antunes,
sandy.antunes@jhuapl.edu

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 29 July 2022
ACCEPTED 26 September 2022
PUBLISHED 21 October 2022

CITATION
Antunes AK, Winter E, Vandegriff JD, Thomas BA and Bradford JW (2022), Profiling heliophysics data in the pythonic cloud.
Front. Astron. Space Sci. 9:1006839.
doi: 10.3389/fspas.2022.1006839

COPYRIGHT
© 2022 Antunes, Winter, Vandegriff, Thomas and Bradford. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Profiling heliophysics data in the pythonic cloud

Alex K. Antunes[1]*, Eric Winter[1], Jon Duane Vandegriff[1], Brian A. Thomas[2] and Jeffrey W. Bradford[2]

[1]JHU Applied Physics Laboratory, Laurel, MD, United States, [2]NASA Goddard Space Flight Center, Greenbelt, MD, United States

Analysis of long timespan heliophysics and space physics data or application of machine learning algorithms can require access to petabyte-scale and larger data sets and sufficient computational capacity to process such "big data". We provide a summary of *Python* support and performance statistics for the major scientific data formats under consideration for access to heliophysics data in cloud computing environments. The Heliophysics Data Portal lists 21 different formats used in heliophysics and space physics; our study focuses on *Python* support for the most-used formats of CDF, FITS, and NetCDF4/HDF. In terms of package support, there is no single *Python* package that supports all of the common heliophysics file types, while NetCDF/HDF5 is the most supported file type. In terms of technical implementation within a cloud environment, we profile file performance in Amazon Web Services (AWS). Effective use of AWS cloud-based storage requires *Python* libraries designed to read their S3 storage format. In *Python*, S3-aware libraries exist for CDF, FITS, and NetCDF4/HDF. The existing libraries use different approaches to handling cloud-based data, each with tradeoffs. With these caveats, *Python* pairs well with AWS's cloud storage within the current *Python* ecosystem for existing heliophysics data, and cloud performance in *Python* is continually improving. We recommend anyone considering cloud use or optimization of data formats for cloud use specifically profile their given data set, as instrument-specific data characteristics have a strong effect on which approach is best for cloud use.

KEYWORDS

python (programming language), AWS (amazon web service) cloud services, NetCDF file format, HelioCloud, machine learning, cloud computing

1 Introduction

We summarize our survey and *Python* performance evaluation of major scientific data formats that are under consideration for access to heliophysics data in cloud computing environments, which provide scaleable high performance computing (HPC) capability alongside very massive data sets. The ability to engage in science for an entire data sets includes the ability to find and categorize thousands of events, to analyze irradiances and field value variations over long time baselines, create forecasting and predictive models, and to explore new science problems that span large dimensionality and long timespans, among others. In addition, the existence of data sets from multiple satellites stored within

cloud systems allows for cross-instrument science in multiple domains. We define the terms 'big data' to characterize all categories of analysis using data sets larger than can easily be managed on a single desktop machine, and 'machine learning' (ML) as a subset of big data analysis that uses machine learning algorithms for engaging in prediction, event finding, or modeling using entire data sets. This paper both captures the current state of the *Python* landscape with respect to cloud usage, and provides evaluation criteria and a methodology cloud users should consider when deciding how best to make cloud use of their data within *Python*.

With data sizes in heliophysics ranging from smaller terabyte (TB, 1 TB = 1,000 gigabytes) sets such as with the SuperMAG archive up through petabyte-sized (PB, 1 PB = 1000 TB) sets such as with the Solar Dynamics Observatory, science analysis looking at entire solar cycles and machine learning approaches that require full data access increasing are using cloud-based computing environments to store, access, and analyze big data sets. Our analysis of *Python* package support will apply for any cloud or HPC environment. In addition, we provide performance statistics for file access specifically within the Amazon Web Services (AWS) cloud environment.

The challenge with science problems that look at entire data sets is the data sets are too large to fit onto a standard laptop or desktop; transferring the data from an archive or virtual observatory to one's home machine takes prohibitively long; the processing time needed to analyze big data on a single machine is infeasible. Cloud systems are set up to handle big data problems by 1) storing entire instrument data sets in an easy-to-access fashion, 2) moving the compute capability to the data, rather than requiring users to download data to their machine, and 3) allocating multiple CPU processors for analysis tasks so that algorithms can be run on TB or PB of data in finite time. The AWS cloud environment uses AWS servers in what they call their Elastic Cloud Computing (EC2) service and allows storage of yottabytes (YB = 1000 PB) of data on their inexpensive high-capacity Simple Storage Service (S3) object-oriented disk storage. User access can be from user-developed code (such as Jupyter notebooks, *Python* programs, other programming development tools) and commercial software (such as IDL) running on AWS servers using their Elastic Cloud Computing (EC2) service.

Heliophysics data are available from many archives in a wide variety of formats. For example, the Heliophysics Data Portal[1] provides access to mission data in 21 different formats. Fortunately, the bulk of data is primarily encapsulated in only three common formats: CDF, FITS, and NetCDF/HDF5. We also indicate support for Comma-Separate Variable (CSV) files when applicable. There is currently no single dominant format, as each instrument team chooses a format based on their development needs. As a result, a given satellite or instrument community tends to write their analysis tools to work with the subset of data file types that particular team expects; developing generic tools that work with all file types requires extra development time that would take away from science analysis time.

Python is widely used as a language for data analysis in heliophysics and space physics (Hassell et al., 2017; Burrell et al., 2018). There is a movement to create tools that are filetype-independent *via* the community-created PyHC packages such as SunPy, SpacePy, HAPI, and others, and we support such efforts, but cannot mandate that individual scientists support multiple file types in their personal codes. Likewise, requiring data archives to provide all their data in all three formats would result in 3x storage costs with no proven advantage. We accept that the pragmatic state of the landscape is any cloud analysis project can expect to have to deal with either CDF, FITS, or NetCDF/HDF5. We therefore provide performance statistics comparing these three file types when in use in the AWS S3 environment, and likewise document which file types are supported in current *Python* packages.

We examined the most prominent and widely-used formats for this report. Our assumptions include that scientists tend to be trained in (and therefore prefer) their existing workflows and may be new to cloud methods. Barriers for cloud use include the need to adopt new data formats, and a potential shift from traditional instrument-provided raw data to derived data sets which have undergone post-processing, potentially including re-binning or resampling. Adoption of cloud requires scientists to potentially switch from their standard files (CDF, FITS, or NetCDF/HDF5) to cloud-optimized file formats, of which the foremost candidate is the Zarr format. The NASA grant proposal 'ROSES' opportunities include a current call for preparing ML-ready data sets that are post-processed and downsampled to generate new big data products, therefore scientists will be expected to adapt to cloud-specific data needs over time.

One core issue with cloud data is tool use, and whether the big data sets created in the cloud are accessible only from within the cloud, or if they are also accessible to the scientists' traditional work computer workflow. To encourage use of big data sets, cloud architects should maintain the ability for scientists to work with the data using tools they are already familiar with, and we argue that user interface development for cloud environments is as important as raw power.

In Earth Observation, similar considerations have been reported. Lynnes et al. (2020) used the criteria of usability, tools, standards, and cost and found 'no one-size-fits-all' with long-term archives remaining in their existing stable self-described file formats, and derived multidimensional imagery and array data sets using Cloud-optimized GeoTIFF and Zarr (respectively). We agree, in general, that archives should retain their existing formats while derived ML data sets are free to explore more heliophysics-optimal formats, particularly NetCDF and HDF.

1 https://heliophysicsdata.gsfc.nasa.gov/.

For our specific cloud environment, HelioCloud (NASA/GSFC 2022) is an AWS-based cloud environment and big data analysis cache created by NASA/GSFC Heliophysics to facilitate easy access to cloud capabilities specifically for heliophysics and space physics research while minimizing the learning curve for scientists (who should not need to be cloud experts to use cloud capabilities). We used it for our profiling of file formats in the AWS cloud, and likewise our code is available *via* the associate HelioCloud github code repository for easy replicability.

In providing a short encyclopedia of primary heliophysics and space physics *Python* packages, we hope to inform scientists as to which data file formats are accessible for both traditional "home desktop" analysis and big data analysis, as well as hoping that developers will use this information to create better cross-package compatibility. Combined with the file performance statistics in the cloud environment, this paper provides current performance expectations for heliophysics traditional and cloud use in *Python* so that scientists can choose the appropriate packages and file types for engaging in traditional analysis, big data analysis, and ML science analysis.

For our specific performance statistics for file access speeds in a cloud environment, cloud bulk storage such as AWS S3 is slower within the cloud-based EC2 compute environment compared to the faster (but more expensive) AWS conventional disk storage, called Elastic Block Storage (EBS). Likewise AWS S3 is slower than using your local disk on your local machine as with a traditional workflow. A check of basic filesystem performance found that AWS S3 was 30x slower in accessing files than what a user experiences on a reference machine of a local MacBook Pro, and 95x slower in writing and delete speed *versus* the same local system. *Accepting S3 usage is predicated on understanding that it will be slower than conventional disk access, but allow for larger data sets to be stored at a lower cost.* With this compromise in mind, we can look at specific file format performance comparisons.

2 Materials and methods

The materials of cloud file storage are the different file types and the *Python* packages plus underlying libraries that support those file types. We focus on the community drivers of the need for access to big data sets, and the use of large contiguous data sets for machine-learning (ML) research[2]. The performance for a presumed ML-ready data set is the base use case for the context of our analysis, as ML problems typically involve codes that must run on the entirely of a given data set. The community has two approaches towards creating and using data sets for ML. The first is to create tools to feed existing instrument data archives into ML pipelines to perform the analysis, while the second is to create a new derived data set by processing, calibrating, curating and collating massive data sets and apply feature extraction, tagging, or other analytic data reduction.

In the tools category, examples include Bobra and Mason's (2018) published Jupyter Notebooks that use a variety of data sets as input. Schneider et al. (2021) created a *Python* toolset for generating ML-ready data set from SOHO and SDO into either local FITS files or an HDF5 datacube. In the curated data sets category, Galvez et al. (2019) prepared an ML-ready data set for all three of SDO's instruments: AIA, HMI, and EVE. Their 6.5 TB curated and collated set is made available *via* the Stanford Digital Repository as sets of 2–6 GB tar bundles containing compressed Numpy-readable files. Antunes et al. were awarded a 2021 NASA ROSES Tools and Methods proposal to create an ML-ready set for STEREO + SOHO, and a NASA grant proposal program under their ROSES opportunities includes a 2022 call specifically for creating ML-ready data sets.

In terms of technical implementation, AWS S3 storage "supports" all file types, storing them as objects rather than as files within a conventional disk system. Access can be *via* "copy then use", where files are copied from AWS S3 to local disk then read by the program, or using libraries that can directly read them from AWS S3 in object format, which we call "S3-aware libraries". In *Python*, S3-aware libraries exist for FITS (AstroPy), NetCDF/HDF (netCDF4), HDF5, CDF (MAVENSDC[3] libcdf or AstroPy), and Zarr.

Likewise, the three file types are similarly well supported in the *Python* package environment: both Pandas (Zaitsev et al., 2019) and Xarray libraries have good support for NetCDF and HDF5; Xarray supports Zarr; AstroPy supports FITS and HDF5; SpacePy supports CDF and HDF5; SunPy[4] supports FITS, CDF, and NetCDF. We note that NetCDF and HDF5 are often considered identical file formats, and it is not within the scope of this paper to evaluate the compatibility between NetCDF and HDF5 in terms of importing one from the other's library.

Both the CDF and NetCDF projects provide stand-alone file conversion tools to go between the three file formats. In the CDF ecosystem, there are supported tools for going to or from FITS, NetCDF, and HDF5. For NetCDFs, there are tools to convert to and from the older NetCDF3 and the current NetCDF4/HDF5 format. Converting to HDF5 and Zarr from FITS, CDF and NetCDF[5] are problematic. Not all files can be converted, and converting from the richer data types (e.g., HDF5, NetCDF) to the simpler file types (FITS, CDF) is lossy. We did not look deeply

2 https://towardsdatascience.com/guide-to-file-formats-for-machine-learning-columnar-training-inferencing-and-the-feature-store-2e0c3d18d4f9.

3 https://github.com/MAVENSDC/cdflib.

4 https://docs.sunpy.org/en/stable/code_ref/io.html.

5 https://discourse.pangeo.io/t/netcdf-to-zarr-best-practices/1119.

at the accuracy or fidelity of metadata for these transformations, as that was outside the scope of our performance evaluations.

In terms of package support, there is no single *Python* package that supports all the common heliophysics file types, while NetCDF (with cross-support for HDF5) is the most-supported file type. This necessitates that scientists be aware that a given analysis package may or may not support the data file type they require, which presents a barrier to both analysis and code sharing. Assuming scientists choose a package that can read the data format, we assert that *Python* currently has good support for data access directly from AWS S3 for cloud usage. One concern is performance, as there are three library approaches to handling S3 data:

1) Copying from AWS S3 to local storage, then reading the file locally for analysis: fast, easy, but inefficiently duplicates disk storage
2) Reading from AWS S3 and storing the full data set in memory, then carrying out the analysis: fast but can overwhelm RAM for large/many files
3) Reading from AWS S3 directly in pieces and analyzing the data in smaller chunks: efficient but slow due to S3 limitations and requires your workflow be designed to operate with chunks

Which approach is "best" is very dependent on the specific task at hand. A pipeline processing of small data files, for example, can easily live within memory and gain that speed performance boost, while a machine learning algorithm that must hold vast sets of training data simultaneously will have to do a memory analysis dependent on the project size as well as the capacity of the analysis hardware. Going deeper into these tradeoffs is not within the scope of this paper.

2.1 File types

We converted between file formats using the CDF toolset to explore file size metrics. The tools are provided by NASA and have the straightforward names of netCDF-to-cdf, cdf-to-netCDF, cdf-to-fits and fits-to-cdf (cdf.gsfc.nasa.gov/html/dttools.html). This was the most robust tool set, but as it uses CDF as its intermediate transfer function (for example, doing FITS to NetCDF required fits-to-cdf then cdf-to-netCDF); this may have introduced a bias towards CDF files having the typically smaller file size for identical data.

A key concern in conversion is the preservation and accessibility of metadata. For example, the cdflib library returns this information for an MMS file:

CDF': PosixPath("mms1. cdf"), "Version": "3.6.0", 'Encoding": 6, "Majority": "Column_major", "rVariables" [], "zVariables": ["Epoch", "mms1_fgm_b_gse_brst_l2", "mms1_fgm_b_gsm_brst_l2" [etc].

While the same data converted to FITS yields these header fields:

TFIELDS = 12/number of fields in each row.
TTYPE1 = 'Epoch '/label for field one.
TFORM1 = '30A30 '/data format of field: ASCII Character.
TUNIT1 = 'ns '/physical unit of field.
TTYPE2 = 'mms1_fgm_b_gse_brst_l2'/label for field two.
TFORM2 = '4E '/data format of field: 4-byte REAL.
TUNIT2 = 'nT '/physical unit of field.
[etc]

The core elements are there (variable names, associated metadata) but interpretation and validation are not seamless. Programs to validate metadata across the conversion would be needed. In terms of accessibility, the difference in file formats require either conversion programs within the *Python* ingest routine, or a known mapping so that users of one approach (CDF) and library (SunPy) can access the other approach (FITS) and library (AstroPy). Similarly, engaging in analysis within one *Python* library creates internal data representations (such as SunPy maps) that are not immediately useable within a different *Python* library (such as with AstroPy and its NDData representations). There is a funded effort already underway to unify several of the core *Python* heliophysics libraries so that they can share datatypes. Table 1 lists the current cross-file conversion capabilities available for immediate validated bulk conversion between the file formats we discuss.

As a cautionary note, chaining to convert a file (e.g., going from CDF to HDF5 to Zarr) is usually one-way, and not reversible. Conversion from CDF to Zarr is possible *via* chaining, but not from Zarr to CDF[6,7]. In general, the multidimensional or more complex data formats (NetCDF, HDF, Zarr) cannot down-convert to the tabular data formats (FITS, some CDF). We also consider that Zarr is not mature enough yet to warrant bulk-converting data archives; if we convert now to Zarr, we might have to re-convert later as the specification evolves.

To dig deeper, in Table 2 we benchmark three different mission data sets: the Magnetic Multiscale Mission (MMS) FGM measurements (CDF-native format), Parker Solar Probe WISPR image data (FITS-native format), and Global Ultraviolet Imager (GUVI) Spectral data (NetCDF-native format). We converted each data set to all three types (FITS, CDF, NetCDF) and measured the resulting file sizes. File sizes upon conversion were somewhat inconsistent, and we do not recommend converting existing archives at this time.

6 https://www.unidata.ucar.edu/mailing_lists/archives/netcdf-hdf/1992/msg00017.html.

7 https://cdf.gsfc.nasa.gov/html/dttools.html.

TABLE 1 File conversions using native tools.

To	*FITS*	CDF	NetCDF3	NetCDF4	HDF4	HDF5[19]	Zarr
From	.						
CDF	✓	.	✓	✓	✓	✓	x
NetCDF3	x	✓	.	✓	x	x	x
NetCDF4	x	?	?	.	✓	✓	✓
FITS	.	✓	x	x	x	✓	x
HDF4	x	✓	x	x	.	✓	x
HDF5	x	x	x	x	x	.	✓
Zarr	x	x	x	✓	x	x	.
(NCZarr)	x	x	x	✓	x	x	x

✓indicates compatible conversion, x indicates no stable conversion available.

TABLE 2 File Sizes when converting files.

File size	FITS (M)	CDF (M)	NetCDF3	NetCDF4	HDF5
WISPR	***7.5***	2.6	2.7 M	7.5 M	7.5 M
MMS FGM	2.1	***1.2***	—	9.1 M	—
MMS FEEPS	11	***11***	—	(conversion errors)	—
GUVI	0.36	0.32	0.28 M	***0.50 M***	—

bold = native format for that instrument - = conversion incomplete and not testable

2.2 Support examples in AWS

We provide short code examples for accessing FITS, CDF and NetCDF data within the AWS S3 environment. In all cases we are accessing a presumed AWS S3 object that it in a bucket (S3 directory-equivalent) named *"mybucket"* with a file name in subdirectories path *"mypath"* named in some form of *"example.*"*. These three examples show the similarities and differences in accessing S3 data.

2.2.1 FITS

Access to FITS data is *via* the astropy and boto3 libraries.

```
import astropy. io.fits
s3c = boto3. client ('s3')
fobj = s3c.get_object (Bucket = 'mybucket',Key = 'mypath/example.fts')
rawdata = fobj ['Body']. read ()
bdata = io. BytesIO(rawdata).
Valid_data = astropy. io.fits.open (bdata).
Valid_header = valid_data [0]. header.
```

2.2.2 CDF

Access to CDF data is *via* the MAVENSDC cdflib and boto3 libraries.

```
import cdflib
Valid_data = cdflib. CDF ('s3://mybucket/mypath/example.cdf).
```

2.2.3 NetCDF/HDF5

Access to NetCDF and HDF5 data is *via* the *Python* xarray library[8,9].

```
import xarray
filename = "https://s3. yourregion.amazonaws.com/bucketname/mypath/example.h5".
fgrab = fs. open (s3name).
Valid_data = xarray. open_data set (fgrab).
```

3 Results

We considered the following issues when examining each file format:

1) Compatibility with existing user code and tools.
2) Documented support by software package and programming lang.
3) Current usage of data format in heliophysics community.
4) Ease of conversion between formats.
5) Data file size.
6) Performance in AWS.

8 http://opendap.ccst.inpe.br/Observations/ARGO/tmp/netCDF4-0.9.8/docs/netCDF4.dataset-class.html.

9 https://howto.eurec4a.eu/netcdf_datatypes.html.

7) Support in multiple languages and support in *Python*.

3.1 File types

The most common file types in heliophysics and solar physics are currently CDF, FITS, and NetCDF/HDF5. CDF (Common Data Format[10]) has large file support and allows for compression, and has a large body of existing tools. Language bindings include *Python*, C, FORTRAN, Java, Perl, C#/Visual Basic, IDL, and Matlab. Many space physics data sets are in CDF format. However, not all *Python* implementations of CDF are cloud-aware, and it often requires an external C or Fortran library for rapid read speeds.

The FITS (Flexible Image Transport System[11]) standard is heavily used in solar physics and astronomy and has a large body of support tools. Conversely, as an older format, it is not designed for modern data. Its language bindings include C, FORTRAN, *Python*, C++, C#,.Net, Pascal, IDL, Java, JavaScript, Perl, Tcl, Matlab, LabVIEW, Mathematica, IGOR Pro, R, Photoshop (plugin), golang, and Swift. In *Python*, most implementations use the AstroPy fitsio library as a FITS handling, making the read/write usage identical across use cases.

As an evolution of FITS, ASDF (Advanced Scientific Data Format[12]) was proposed in 2015 (Greenfield et al., 2015) as an evolutionary replacement for the FITS format, which is widely-used in the astronomical community. It has language bindings in *Python*, C++ (incomplete), and Julia (incomplete). Its advantages include use of JSON, ability to stream data, and ability to be embedded in FITS. However, it is designed as an interchange format rather than an archival format, is not cloud-aware, and the standard is still young and in flux. FITS has also been implemented on top of HDF5 (Price et al., 2015).

NetCDF (Network Common Data Format[13]) is built on top of the HDF5 format and includes S3 support. HDF (Hierarchical Data Format[14]) is very wildly used, has a large existing body of tools, and is highly scalable. Language bindings exist for nearly every language. S3 support is built into NetCDF, and available for HDF5 *via* a virtual file systems layer and read-only. NetCDF is relatively stable while HDF5 is an evolving standard.

Zarr[15] is a newer Python-supported cloud-native standard. It is based on NumPy, so a large body of existing code can use it. The Zarr standard is still evolving and is not natively used for existing helio/space physics data sets. Its most common use is currently as an alternative storage format specifically for cloud data.

3.2 File performance in python

Performance of file types on AWS was very dependent on the data type (time series, spectra, image) and instrument specifics. Performance also varies as individual *Python* libraries are improved and updated over time. For three sample cases (MMS FGM data in NetCDF4, GUVI spectral data in CDF, and WISPR images in FITS), we converted all three into FITS, CDF and NetCDF. We found file sizes varied by factors of 1-5x and S3 access times varied by 1-3x. Library support for chunking (not needing to load an entire file when only parts of it are needed) leads to implementation-specific speed advantages as well, and further library development is recommended.

Within the HelioCloud setup, we compare EC2 access to S3 access for the file types of FITS, CDF, and NetCDF using the Python-native interpreters of *AstroPy*, *netCDF*, and *cdflib*. The *AstroPy* FITS reader and the *cdflib* reader used the AWS-supported *boto3* S3 library, while NetCDF requires use of the external *fs. s3* file system library and the *Xarray Python* package. We ran this test twice over a 6 month period, during which the *Python* libraries had improved and produced more consistent results. We present the current state as of the September 2022 version of the libraries. Results are generally valid to within +/-10%, that is, for any given set of 100 runs, the timings will vary across runs but fall within 10% of our results here.

For a given file size, FITS reads were fastest, and CDF reads were slowest. Both FITS and CDF reads were longer as file sizes increased. The NetCDF4 read for the smallest file size took longer than for the two larger file sizes; as AWS S3 requires some overhead to initially access any file, that this indicates NetCDF4 read speeds are fast and the bottleneck is initially accessing S3. Averaged, typical FITS reads were on order of 20 ± 17 MB/s while CDF reads were on the order of 2.2 ± 1.6 MB/s and NetCDF at 24 ± 23 MB/sec. These rates are also on order of 2-3x faster than results from 6 months ago (not pictured), again an indication of library improvements. The wide variance of read times within each file type reinforces that all data conversions and transfers are very instrument-specific and thus broad conclusions over a 'best' file format cannot be made without specifically profiling the desired data set.

With the latest version of the libraries, we saw no strong performance preference for accessing a given instrument in its original native format, *versus* performance from the same data transformed into the other two file formats. As *Python* library support matures, this suggests that converting datasets to a file format for performance increases is not infeasible. The limiting factor in deciding on such conversions is whether the conversion itself is feasible. For our chosen sets, for example, the MMS FGM instrument was able to convert from CDF to NetCDF4, but the MMS FEEPS instrument was not able to generate a valid

10 https://cdf.gsfc.nasa.gov/.

11 https://fits.gsfc.nasa.gov/.

12 https://asdf-standard.readthedocs.io/.

13 https://www.unidata.ucar.edu/software/netcdf/.

14 https://www.hdfgroup.org/.

15 https://zarr.readthedocs.io/en/stable/.

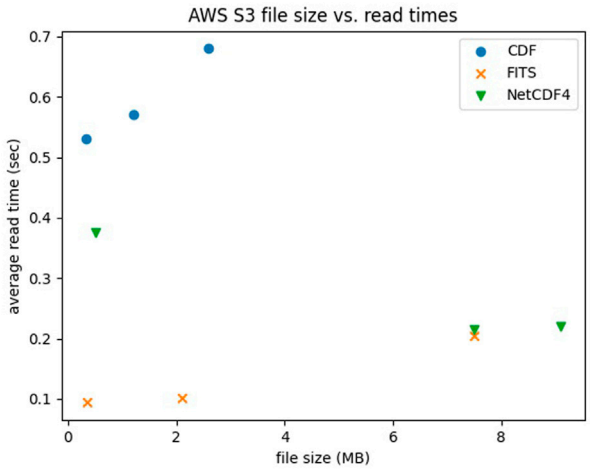


FIGURE 1
Average read speeds (in sec) for three different datasets (MMS, WISPR and GUVI) in each of three file formats (CDF, FITS and NetCDF4).

NetCDF4. Conversion between file formats requires that the new format support the multidimensionality and internal data organization and therefore we do not advise projects convert to a new format without first performing data validation.

We plot size (in MB) *versus* scaled read speeds (using average read time from 100 reads) [Figure 1] for three different instruments, each provided for the three different file types (of differing sizes). While the units are 'MB' and 'seconds', we recommend observing trends rather than documenting specific performance, primarily because in cloud environments the choice of cloud processor will affect read speeds. Therefore absolute values will differ based on hardware, while relative load speeds will remain proportional. In terms of file performance, one would expect a linear trend that, as file sizes increase, file read times also increase; this trend was consistent for FITS and CDF files but NetCDF4 file read times as noted struggled with the small GUVI data file.

FITS was the fastest format, a result that surprised us; possibly it is because it is a flat format with little parsing needed, but we hesitate to ascribe a cause without further study. We recommend that library optimizations for S3-aware reads are low-hanging fruit in terms of boosting performance, preferential to reformatting entire data archives. Based on our analysis, we recommend cloud-ready archives do a quick check on their data to see if conversion yields significant (2x storage or >2x S3 speed) gains because of how the gains or losses are very instrument-specific. However, they should not abandon their native formats because those were chosen to support their existing user communities and toolsets, and need to be retained.

Additionally, ML-ready data sets should aim for accessibility and choose a format that allows analysis both within and outside of the *Python* ecosystem, with HDF5 as the strongest potential future candidate based on the number of languages that support it, support within existing *Python* libraries such as *AstroPy* (HDF native) and *SunPy* (HDF *via* NetCDF4 support).

3.3 Native python support

Defining the ecosystem of *Python* package support is always a work in progress, and we capture the current state of the heliophysics-relevant core *Python* package external file types and internal data types supported. There is a funded effort this year to ensure that SunPy, AstroPy, SciPy and HAPI are able to read each other's formats. This and related efforts will go a long way towards simplifying the data landscape for *Python* users.

1) Pandas: CSV, NetCDF4, HDF5, Feather
2) Xarray: CSV, NetCDF3, NetCDF4, HDF5, Zarr, pandas
3) AstroPy [Pandas DataFrames]: CSV, FITS, HDF5
4) SpacePy [SpaceData, Numpy-compatible]: CDF, HDF5
5) SunPy [Map, TimeSeries]: FITS, pandas, AstroPy, J2000, ASDF, CDF, ANA, GenX
6) HelioPy: CDF, astropy, sunpy
7) SciPy: IDL, Matlab, pandas
8) PySat[16]: NetCDF, xarray, pandas.

We look at Python-specific library support by python data types and python packages. For each package, we summarize the file formats supported and indicate the data structures and basic readers included. In notable cases we include code examples illustrating the data reads. Note that *Python* natively supports CSV for Lists and Dictionaries. The primary internal data objects across *Python* heliophysics packages are:

- in AstroPy: Table
- in Numpy: ndarray
- in Pandas: DataFrame
- in SpacePy: SpaceData
- in SunPy: Maps, TimeSeries, NDCube
- in Xarray: xarray or DataFrame

3.4 *Python* packages

We provide per-package details on the primary Heliophysics *Python* packages, including their internal data object representations and which files they support, as well as additional relevant details. This information is intended to support users seeking out packages to work with specific files and data, and for developers who are interested in investigating package interoperability.

16 https://github.com/pysat/pysatCDF.

3.4.1 AstroPy (astropy.org)

The primary AstroPy data objects are *Table* and *CCDData*, and Pandas *DataTable* and *DataFrame* are supported. Supported file formats are HDF/HDF5, CSV and FITS, as well as JSON and others. The built-in *read*() and *write*() functions determine the file types when reading. *CCDData* can be converted to a Numpy *ndarray* or an *NDData* object. AstroPy is designed to support astronomy as well as encouraging interoperability.

3.4.2 Cdaswd library (cdaweb.gsfc.nasa.gov

The cdaswd library supports the SpacePy data model using the NASA CDF C library (requiring a C compiler or pre-built binary package to install), and also supports cdaweb's netCDF[17].

3.4.3 csv (built into python)

CSV files can be read into *Python* with no additional library needed.

3.4.4 Cdflib (github.com/MAVENSDC/cdflib)

A variant CDF library that does not require the NASA CDF C library, cdflib does require the *Python* Numpy package.

3.4.5 HAPI, hapi-server.org

The primary HAPI data objects are CSV or JSON data formatted to the HAPI specification. Available Java and *Python* HAPI server code bases support ingest and streaming of CDF, HDF, and CSV files. The HAPI *Python* client program uses Numpy arrays as its internal representation. HAPI is a time series download and streaming format specification intended as a common data access API for space science and space weather data.

3.4.6 HDF5 (h5py.org)

The primary HDF5 data objects are *dataset* (which are like Numpy arrays) and *Group* (which are like a *Python* dictionary). The *h5py* function is used to ingest files.

3.4.7 HelioPy (heliopy.org, no longer supported as of 2022)

The primary HelioPy data object is the SunPy *TimeSeries* object. HelioPy uses the *Python* CDFlib and can convert cdf to a Pandas *Dataframe*.

3.4.8 netCDF4 (unidata.github.io/netcdf4-python/)

The primary netCDF54 data objects are *dataset* and *Variable*.

3.4.9 NumPy, numpy.org

The primary NumPy (sometimes written as Numpy) data object is the *ndarray*. Numpy does not yet have a standard way to deal with metadata. Numpy supports CSV files and data. Many *Python* packages use *ndarrays* within their internal data representations.

3.4.10 Pandas (pandas.pydata.org)

The primary Pandas data objects is the *DataFrame*, and for HDF there is also *HDFStore*. Pandas supports HDF and CSV files. Pandas does not yet have a standard way to deal with metadata.

3.4.11 PyTables, pytables.org

The primary PyTables object is the *Table*, and it also supports any NumPy data type. PyTables is built on top of the HDF5 and NumPy libraries.

3.4.12 SciPy: scipy.org

SciPy is a package that can read IDL and Matlab data. It requires the NumPy, SciPy, Matplotlib, IPython, SymPy, and Pandas packages. SciPy provides algorithm and data structure support for a variety of science domains.

3.4.13 SpacePy: spacepy.github.io

The primary SpacePy data object is *SpaceData* and also has the *dmarray* class that is equivalent to NumPy arrays plus attributes. Supported file formats include CDF and HDF. SpacePy is NumPy-compatible. Future versions of SpacePy intend to use CDF as the primary internal datatype instead of *SpaceData*. SpacePy is a data anlysis, modeling and visualization package for space science.

3.4.14 SunPy (docs.sunpy.org)

The primary SunPy data objects are *Map* and *TimeSeries*. Supported file formats include FITS, but they recommend using the AstroPy FITS reader as it is more robust. Internal data is primary a wrapper around the Pandas DataFrame and NumPy ndarray. SunPy is an open source solar data analysis environment.

3.4.15 Xarray: xarray.pydata.org

The primary Xarray data object is the *xarray*, and Pandas *DataFrame* and NumPy *ndarray* are also supported. Supported file formats include CDF, NetCDF, HDF5, CSV and Zarr.

3.4.16 PySat (github.com/pysat)

The primary PySat data object is the *xarray*. Supported file formats include NetCDF3, NetCDF4 and HDF5. PySat is intended to provide an extensible framework for data sources from satellite instruments and constellations.

4 Discussion

Most languages suppx`ort heliophysics core datafile formats, but are not necessarily S3-aware. The 0th-level solution of "copy files from S3 to local storage" is always a fallback option, but we recommend resources be put into adding and improving

17 https://cdaweb.gsfc.nasa.gov/WebServices/REST/py/FAQ.html.

S3 capabilities in libraries as this is generally not difficult (relies on existing external libraries and drivers) and benefits the community. *Python* has strong support for the major datafile formats both in S3 and local storage as a language, but no single package supports all the formats. For the major three formats of FITS, CDF, and NetCDF/HDF5, *Python* does support direct S3 reads, with overall good performance statistics.

For archives looking at reformatting existing data, L1/L2 bulk file convert should be tested, and is not usually recommended because the file size and performance gains are frequently marginal and occasionally detrimental. Analysis, model and ML-ready data set creation (L3/L4 data sets) should look into HDF5 because of its support and cross-compatibility across a variety of languages. As a side note, we notice the ML community L3/L4 data sets are currently a mix of "tools to make ML streams" and "L4 ML-ready data sets".

Python supports S3 for the major file/data types, but there are idiosyncrasies across packages. In addition, *Python* + S3 is a dominant cloud architecture, but we note the community wants the user experience and UI to be more like their native tools. For specific Cloud and ML use cases, the trade-off between S3 access methods of "read into memory" (fast, resource-heavy) and "read chunks as needed from S3" (slow but resource-light) means use of High Performance Computing (HPC) tasks with S3 need to be diligent in tracking their disk and memory architecture for their specific task.

While there is no "one size fits all" approach to using S3 for heliophysics, there are also no impediments to using S3 and Cloud within *Python* in the current state of the art of the *Python* heliophysics ecosphere[18].

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://git.mysmce.com/heliocloud/cloud-tutorials/-/blob/master/S3_all_tests.ipynb.

Author contributions

AA and EW carried out the bulk of the analysis, while JB, BT, and JDV provided infrastructure and analysis support.

Funding

This project was supported under the NASA/Goddard Space Flight Center award AD-79106 "Enhancing Science Analysis with Cloud-Based Tools and Frameworks". This work was enabled by the NASA GSFC HelioCloud environment.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). "Snakes on a spaceship—an overview of Python in heliophysics." *J. Geophys. Res. Space Phys.*, vol. 123, no. 12, Dec. 2018. DOI.org (Crossref), doi:10.1029/2018JA025877

Greenfield, P., Droettboom, M., and Bray, E. (2015). Asdf: A new data format for astronomy. *Astronomy Comput.* 12, 240–251. Astronomy and Computing. doi:10.1016/j.ascom.2015.06.004

Hassell, D., Gregory, J., Blower, J., Bryan, N. L, and Karl, E. T. (2017). "A data model of the climate and forecast metadata conventions (CF-1.6) with a software implementation (Cf-Python v2.1)." *Geosci. Model Dev.*, vol. 10, no. 12, Dec.2017, pp. 4619–4646. DOI.org (Crossref), doi:10.5194/gmd-10-4619-2017

Lynnes, C., Quinn, P., Durbin, C., and Shum, D. (2020). Cloud optimized data formats. *Comm. Earth Observing Satell. Meet. #4.*

NASA/GSFC (2022). The HelioCloud Project [cloud environment] Available at: https://heliocloud.org.

Price, D. C., Barsdell, B., and Greenhill, L. (2015). Hdfits: Porting the FITS data model to HDF5. *Astronomy Comput.* 12, 212–220. arXiv.org. doi:10.1016/j.ascom.2015.05.001

SPDF Filenaming recommendations. Available at: https://spdf.gsfc.nasa.gov/guidelines/filenaming_recommendations.html.

Zaitsev, I. "The best format to save pandas data." *Medium*, 29 Mar. 2019, Available at: https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d

18 http://heliocloud.org/.

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 25 October 2022
DOI 10.3389/fspas.2022.1005598

Check for updates

OPEN ACCESS

EDITED BY
John Coxon,
Northumbria University,
United Kingdom

REVIEWED BY
Kyle Murphy,
Northumbria University,
United Kingdom
Anton Artemyev,
College of Physical Sciences, University of California, Los Angeles, United States

*CORRESPONDENCE
Stelios Tourgaidis,
stourgai@ee.duth.gr

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 28 July 2022
ACCEPTED 22 September 2022
PUBLISHED 25 October 2022

CITATION
Tourgaidis S and Sarris T (2022), Wave-particle interactions toolset: A python-based toolset to model wave-particle interactions in the magnetosphere. *Front. Astron. Space Sci.* 9:1005598. doi: 10.3389/fspas.2022.1005598

COPYRIGHT
© 2022 Tourgaidis and Sarris. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Wave-particle interactions toolset: A python-based toolset to model wave-particle interactions in the magnetosphere

Stelios Tourgaidis* and Theodore Sarris

Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

Wave particle interactions are known to be an efficient yet unquantified driver of the variability of particle populations in Earth's magnetosphere, and their quantification and understanding through modelling has been a subject of ceaseless and extensive research during the last decades. Moreover, there is an increasing interest in techniques for radiation belt remediation, which refers to artificially controlling energetic particle populations in the near-Earth space environment via the scattering of particles from artificially generated electromagnetic waves. Whereas numerous modelling techniques are described in literature, there is a lack of a unified open-source toolset that incorporates the equations and parameterizations used by different wave-particle interaction models in a user-friendly environment. We present WPIT, the Wave-Particle Interactions Toolset, an open source, Python-based set of tools for modelling the interactions between energetic charged particles and VLF waves in the magnetosphere through test particle simulations. WPIT incorporates key routines related to wave-particle interactions in Python modules and also in Jupyter Notebook environment, enabling the traceability of all relevant equations in terms of their derivation and key assumptions, together with the programming environment and integrated graphics that enable users to conduct state-of-the-art wave-particle interaction simulations rapidly and efficiently. WPIT can be used either as a stand-alone simulation tool or as a library of routines that the user can extract and incorporate into an independent simulation. We present an analytic description of the code, the methodology used, and examples based on each of the WPIT modules. WPIT examples include the exact reproduction of simulation results that have been reported in literature, based on the same sets of parameters and assumptions, allowing the user to expand upon state-of-the-art. Finally, using the WPIT toolset, we perform a parametric analysis on the onset of nonlinear interactions between electrons with whistler-mode waves by varying the relevant parameters of the waves (amplitude, wave normal angle and frequency), the particles (pitch angle and energy) and the plasma environment (electron density and ion composition).

KEYWORDS

wave-particle interactions toolset, Python for space physics, magnetosphere, whistler, emic, VLF waves, nonlinear interactions, WPIT

1 Introduction

1.1 Background on wave-particle interaction modeling in the magnetosphere

The observed variability in the radiation belts is the outcome of an imbalance between a variety of source and loss processes. In the collisionless regime of the magnetosphere the changes in the particle populations are mainly controlled by interactions with a plethora of plasma waves, which may lead to the violation of one or more of the adiabatic invariants (Schulz and Lanzerotti, 2012). Very low frequency (VLF) frequency waves can violate the first and second adiabatic invariants, leading to pitch angle scattering, acceleration and potential loss of particles to the upper atmosphere (Horne and Thorne, 1998; Kivelson, 2005; Shprits, 2009). Resonant wave-particle interactions are an efficient scattering mechanism of energetic particles, leading to pitch angle and energy changes of energetic particles (Koskinen and Kilpua, 2022).

There are several different approaches that are commonly used to simulate wave particle interactions. These are generally classified in three categories, namely quasi linear theory (e.g., see Albert (2005) and Summers (2005)), Particle-In-Cell (PIC) methodology (e.g., see Allanson et al., 2019 and Allanson et al., 2020) and test particle simulations. WPIT focuses on the test particle simulation methodology.

There are two main approaches in the test particle simulation methodology: the first approach is to integrate the Lorentz equations of charged particle motion in order to trace particle trajectories under the effect of the waves while monitoring changes in the particles' pitch angle and energy, termed here the full Lorentz approach. The second approach is to use the gyro-averaged equations of motion (e.g. Bell, 1984; Jasna et al., 1992; Bortnik 2004; Bortnik et al., 2015; Li et al., 2015). The gyro-averaged approach has the advantage of reducing the system from a six-dimensional space (i.e. x, y, z, u_x, u_y, u_z, or 3R3V space) to a four-dimensional space (i.e. $u_{\parallel}$, $u_{\perp}$, z, η, or 2R2V space]). WPIT in its current form implements the gyro-averaged test particle simulation approach.

In an early publication on wave-particle interactions, Laval and Pellat (1970), investigated the acceleration of particles due to interactions with electrostatic waves. They assumed a parallel propagating electrostatic wave with fixed frequency. They concluded that the trapping of particles in an electrostatic wave could account for particle acceleration and regarded this as a potential mechanism for the precipitation of low-energy electrons. Later on, Nunn (1974) investigated the generation of VLF triggered emissions through nonlinear cyclotron resonance interactions between electrons and a narrow band whistler mode wave travelling in a magnetospheric duct (i.e. parallel propagation). Their results indicated that the nonlinear interactions of electrons with a ducted whistler mode wave can account for the generation of triggered emissions. Karpman et al. (1975) presented an analytic formulation for the investigation of the effects of nonlinear interactions between particles and monochromatic waves. Starting their theoretical analysis for Langmuir waves, they extended their formulation to parallel propagating whistler-mode waves. Karpman and Shklyar (1977) calculated particle precipitation caused by interactions of electrons with a coherent whistler-mode wave. It is noted that in the above studies, the wave was assumed to propagate parallel to the magnetic field, thus omitting any potential effects of interactions with an obliquely propagating wave.

The first formulation of wave-particle interactions for non-relativistic electrons under the effect of oblique whistler-mode waves using the gyro-averaged approach was given by Inan and Tkalcevic (1982). Subsequently, Bell (1984) applied a similar methodology to model first-order cyclotron resonant interactions. Ginet and Albert (1991) and Bortnik et al. (2003) generalised Bell's formulation to account for interactions with relativistic electrons. Li et al. (2015) found that the formulation of Bortnik et al. (2003) lacked a term that led to differences in the perpendicular motion of electrons.

Many studies where performed in the last decades using the test particle approach. For example, Fu et al. (2020) used full Lorentz approach to investigate cyclotron, Landau and bounce resonances of electrons with hiss. Tao et al. (2012) compared diffusion coefficients with test particle results. Chang et al. (2014) used the full Lorentz approach for investigating interactions of electrons with whistlers artificially generated by ionospheric modification. Li et al. (2015) compared their derived gyro-averaged equations with results from the full Lorentz approach, and found that the two approaches lead to fairly similar results, except for small amplitude fluctuations at gyro-frequency timescales. Huang et al. (2017) proposed the crucial role of the initial gyrophase in the acceleration of electrons by low frequency waves. Inan (1987) investigated the interactions of electrons with field-aligned whistler-mode wave packets. Gao et al. (2014) investigated the interaction of electrons with chorus waves. Su et al. (2013) explored interactions of electrons with EMIC waves. Su et al. (2014) addressed interactions of ring current ions with EMIC waves. Shklyar and Matsumoto (2009), in a review paper, presented analytically the theory of resonant interactions between energetic charged particles and an oblique whistler-mode wave in a non-uniform magnetic field and in the inhomogeneous plasma of the magnetosphere. They used a Hamiltonian approach to derive the basic equations for the wave field and for the particle dynamics. They applied their formulation on two

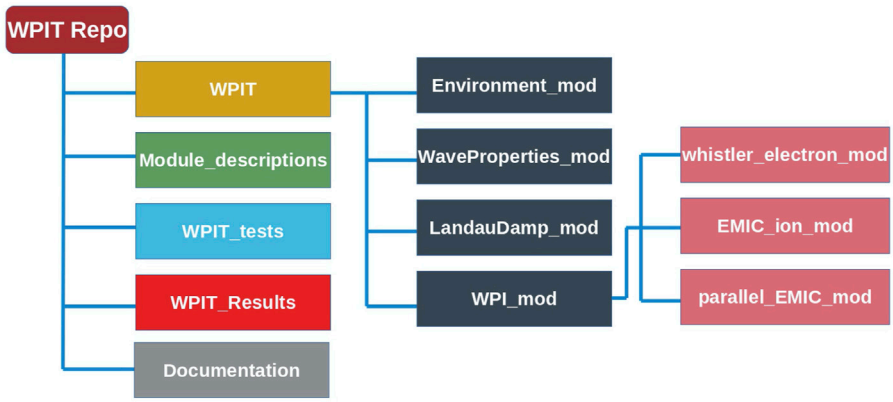


FIGURE 1
A schematic of the organization and contents of the WPIT repository.

applications: they fisrt calculated the damping (or growth) of an oblique whistler wave and subsequently they used their formulation to calculate proton precipitation by ground-based VLF transmitters. In a review paper, Albert et al. (2012) derived gyro-averaged equations both directly from the full Lorentz equations, as well as through a Hamiltonian approach, accounting for relativistic particles and oblique waves in their formulations. One key result from that study is that the Hamiltonian approach manages to reduce the system to a '1 1/2' dimension, allowing the development of analytical treatments of the change in pitch angle due to resonant interactions with whistler-mode waves.

An important aspect of wave-particle interactions is the appearance of nonlinear effects, which may lead to large pitch angle scattering and energy change for a subset of the particle population undergoing wave-particle interactions. Several studies have focused on identifying nonlinear effects by utilising test particle simulations: Nunn and Omura (2015) explored nonlinear effects in VLF wave fields. Liu et al. (2012) investigated phase bunching and phase trapping effects in interactions of electrons with EMIC waves of large amplitude. Artemyev et al. (2013) compared the importance of Landau vs. cyclotron resonances for nonlinear effects in interactions of electrons with oblique whistler waves. Bortnik et al. (2008) investigated the behaviour of electrons interacting with large amplitude chorus waves. Albert and Bortnik (2009) addressed the rapid loss of radiation belt electrons during geomagnetic storms with test particle simulations under the effect of large amplitude EMIC waves. Gan et al. (2020) explored the effects of wave amplitude modulation in nonlinear electron-chorus interactions. Tao and Bortnik (2010) derived a map of probable regions where nonlinear interactions could be expected. Wang et al. (2019) and Wang et al. (2016) explored nonlinear interactions of EMIC waves with electrons, whereas Lee et al. (2020) focused on interactions of EMIC waves with 90° pitch angle electrons. Bell (1986) calculated the minimum threshold in terms of wave amplitude for nonlinear effects to occur. Artemyev et al. (2020) and Vainchtein et al. (2018) used Hamiltonian theory to address nonlinear wave particle interactions and explore the evolution of electron distributions.

The aforementioned studies constitute only a small subset of the available literature on test particle simulations of wave-particle interactions, however it is often the case that the underlying code that implements the various approximations and equations that are used is not readily available. Furthermore, inter-comparisons between the various methods and the sensitivity to a different set of initial conditions can not be easily implemented. WPIT addresses this gap by providing a python-based, user-friendly toolset of routines that simulate wave-particle interactions in the Earth's magnetosphere under various models and assumptions proposed in literature and for a range of user-defined environment, wave and particle conditions.

1.2 Overview of WPIT

WPIT is composed in Python, in four different modules as described in the following section. The source code of each of the four modules is accompanied by corresponding Jupyter Notebook, that combines in a comprehensive way the corresponding code, the underlying equations and theory, the output of computations performed by the code, and visualizations of the code results, along with explanatory text on the usage of the underlying code. An advantage of Jupyter Notebooks is that all the above are provided in a single document, allowing scientists to easily access all elements of the programming process while tracing the methodology that is being implemented to its source in literature.

In the remainder of this paper, the methodology employed by each module is described in Section 2, including a brief description

TABLE 1 Environment module routines.

WPIT.Environment_mod

—	Routine name	Description	Refs
1	Bmag_dipole	Calculates geomagnetic dipole field strength	e.g. Parks (1991)
2	density_equ_carpenter_anderson	Calculates equatorial electron density	Carpenter and Anderson (1992)
3	density_FL_denton	Calculates electron density along a geomagnetic field line	Denton et al. (2002)
4	density_ozhogin	Calculates electron density at the equator and along a magnetic field line	Denton et al. (2002)
5	density_equ_sheeley	Calculates electron density along a geomagnetic field line	Sheeley et al. (2001)
6	omega_cyclotron	Calculates gyrofrequency of a particle	e.g. Parks (1991)
7	omega_plasma	Calculates plasma frequency	e.g. Parks (1991)
8	omega_uhr	Calculates upper hybrid resonance frequency	e.g. Parks (1991)
9	omega_lhr	Calculates lower hybrid resonance frequency	e.g. Parks (1991)
10	aeq2alpha	Translates equatorial pitch angle to local pitch angle	—
11	alpha2aeq	Translates local pitch angle to equatorial pitch angle	—
12	dwc_ds	Calculates the gradient of the cyclotron frequency along a field line	—
13	dB_ds	Calculates the gradient of the magnetic field along a field line	—
14	Lshell	Calculates magnetic L shell at position	e.g. Tao et al. (2012)
15	T_bounce	Calculates the bounce period of a trapped particle	Öztürk, (2012)
16	T_drift	Calculates the drift period of a trapped particle	Öztürk, (2012)
17	R_Larmor	Calculates particles's Larmor radius	e.g. Parks (1991)
18	mu_adiabatic	Calculates the first adiabatic invariant	e.g. Parks (1991)
19	loss_cone	Calculates the bounce loss cone angle	Kivelson et al. (1995)
20	loss_cone_v2	Calculates the bounce loss cone angle	Lauben et al. (2001)
21	debye_length	Calculates the Debye length	e.g. Parks (1991)
22	initial_velocity	Calculates velocity from energy and pitch angle	—
23	Const	Includes the definition of all the needed constants, such as charges, masses, dielectric constant etc.	—

of the code. Examples from the use of WPIT are included in Section 3, including usage limitations. Finally, the scalability of WPIT and potential other uses are described in Section 4, including, but not limited to, its potential application for theoretical studies of wave particle interactions, missions targeting wave-particle interaction experiments and investigations on the efficiency of radiation belt remediation techniques.

1.3 WPIT repository and structure

An overview of the WPIT repository is presented in Figure 1. In this figure, *WPIT Repo* represents the WPIT repository, which is publicly available and can be found at: https://github.com/stourgai/WPIT. Within the repository, the *WPIT* folder, marked in yellow in Figure 1, contains the source code that is built in the form of four main modules. These are: (a) *Environment_mod*, used to setup the environment parameters for each simulation; (b) *WaveProperties_mod*, used to define the properties of the waves; (c) the *LandauDamp_mod*, which calculates the attenuation of waves according to the Landau damping theory; and (d) *WPI_mod*, within which the actual wave-particle interactions are implemented. *WPI_mod* includes three sub-modules, implementing different types of wave-particle interactions, under the effect of oblique whistler mode waves, oblique EMIC waves and parallel EMIC waves, as marked. Each of the modules will be described in further detail in the following sections. The *Module_descriptions* folder contains Jupyter notebooks with analytic theoretical descriptions of the equations used by each of the four modules described above, along with samples for calling each routine. The *WPIT_tests* folder contains a set of WPIT implementations that aim to reproduce results found in literature. These are written in Jupyter notebooks, and act as a verification of the code and as tutorials of the use of WPIT. The *WPIT_Results* folder contains the Jupyter notebooks of the simulations presented in Section 4.

TABLE 2 Wave properties module routines.

WPIT.WaveProperties_mod

—	Routine name	Description	Refs
1	stix_parameters	Calculates the Stix parameters	Stix, (1992)
2	stix_parameters_warm	Calculate Stix parameters with warm plasma corrections	Maxworth and Gołkowski, (2017)
3	refr_index_full	Calculates the refractive index	Kimura, (1966)
4	refr_index_appleton	Calculates the refractive index and the wave numbers	Appleton, (1932)
5	refr_index_parallel_EMIC	Calculates the refractive index for parallel propgating EMIC waves	Summers and Thorne, (2003)
6	refr_index_warm	Calculate the refractive index accounting warm plasma corrections	Kulkarni et al. (2015)
7	cold_dielectric_tensor	Calculate the dielectric tensor based on cold plasma theory	Maxworth et al. (2020)
8	warm_dielectric_tensor	Calculate the dielectric tensor with warm plasma corrections	Aubry et al. (1970)
9	res_angle	Routine to calculate the resonance angle	Stix, (1992)
10	gendrin_angle	Routine to calculate the Gendrin angle	Bortnik et al. (2006)
11	wave_amplitudes_bell	Routine to calculate the wave electric and magnetic fields	Bell, (1984)
12	wave_amplitudes_li	Routine to calculatethe the wave electric and magnetic fields	Li et al. (2015)
13	wave_amplitudes_jasna	Routine to calculate the wave electric and magnetic fields	Jasna et al. (1992)
14	resonant_velocity	Routine to calculate the resonant velocity and the resonant energy	Sousa, (2018)
15	wave_packet_one_sided	Simulate a static, monochromatic and one-sided wave packet	Bortnik et al. (2008)
16	wave_packet_two_sided	Simulate a static, monochromatic and two-sided wave packet	—
17	wave_packet_gauss	Simulate a static, monochromatic gaussian wave packet	—
18	dispersion_R	Dispersion relation of R-mode wave	Swanson, (2012)
19	dispersion_L	Dispersion relation of L-mode wave	Swanson, (2012)
20	dispersion_O	Dispersion relation of O-mode wave	Swanson, (2012)
21	dispersion_X	Dispersion relation of X-mode wave	Swanson, (2012)
22	dispersion_light	Dispersion relation of light	Swanson, (2012)
23	cutoff_R	Cut-off frequency for R-mode waves	Swanson, (2012)
24	cutoff_L	Cut-off frequency for L-mode waves	Swanson, (2012)

Finally, the *Documentation* folder includes the API documentation of the source code in *.html* format.

2 Methods

The code comprising WPIT is formatted in Python modules with specific applications as described below, and is accompanied by Jupyter notebooks, whereby the code is complemented by references to the equations and methods used and where examples can be plotted as direct outputs of the calculations performed. The WPIT source code can be found at: https://github.com/stourgai/WPIT/tree/main/WPIT. The code has been tested on Ubuntu 18.04, Intel Core i7, 2.6 GHz and 16 GB RAM.

As an example of the processing time and the computational resources needed for integrations of particle trajectories under the effect of waves, it is noted that in the machines used for testing of the code as discussed above, the simulation of the resonant interactions of a wave with a 45° pitch angle electron at L = 5 and with energy of 500 over one bounce period requires a real time of computation of approximately 30 s. The wave in the simulation is assumed to be present at every step of the particle trajectory, thus the wave characteristics are calculated at every time-step. A Jupyter notebook of the simulation is available in *WPIT_results* folder of the WPIT repository (*WPIT_Computational_Time.ipynb*). It is noted that this computation time is only indicative, and is dependent on particle parameters, and primarily on the particle's pitch

TABLE 3 Landau damping module routines.

WPIT.LandauDamp_mod

Routine name	Description	Refs
Thermal electron distributions		
distribution_bell	Calculate the suprathermal electron distribution	Bell et al. (2002)
distribution_bortnik	Calculate the suprathermal electron distribution	Bortnik et al. (2007)
golden_distribution	Calculate the suprathermal electron distribution	Golden et al. (2010)
distribution_bimaxwellian	Calculate the suprathermal electron distribution	Maxworth et al. (2020)
Landau Damping		
landau_damping	Calculate the Landau damping of a ray	Brinca, (1972), Sousa, (2018)
RayUtils		
read_input_ray	Routine to read Stanford's 3D Ray tracer output	—
read_appended_ray	Routine to append wave parameters to the ray file and convert to csv	—
read_appended_ray	Routine to read the append files from ray_mod.append_ray	—
resonance_along_raypath	Routine to calculate the resonant velocity and the resonant energy along the ray path	e.g. Sousa (2018)
enhancement_factor	Routine to calculate the magnetospheric cavity enhancement factor	Kulkarni et al. (2006)
ray_plots	Routine to produce several plots of the ray parameters	—

angle; it is also noted that the computation time scales linearly with the total number of bounce periods.

In the following we outline the functionality of each of the modules comprising WPIT.

2.1 Environment characterization module

Module *WPIT. Environment_mod* includes routines for the calculation of environment parameters that are needed for wave particle interaction simulations. The routines of the of module are listed in Table 1. These include routines for the calculation of: the geomagnetic field, the electron density through various models including the models by Carpenter and Anderson (1992), Sheeley et al. (2001), Denton et al. (2002) and Ozhogin et al. (2012), the particle gyro-frequency, the plasma frequency, the upper and lower hybrid frequencies, the gradients of the gyro-frequency and the strength of the magnetic field along a magnetic field line, the L-shell, the bounce and drift periods, equatorial pitch angle mapping, the Larmor radius, the first adiabatic invariant, the bounce loss cone angle, the Debye length, and a routine to calculate particle velocity from a particle's energy and local pitch angle. Moreover, this module includes the *WPIT. Environment_mod.const* routine which sets all the constants that are needed for the simulations. Detailed description of each routine along with example runs can be found in the *Environment_mod_description* Jupyter notebook, located in the *Module_descriptions* folder of the WPIT repository.

To illustrate how the module is run, the following calling routine calculates the electron and proton gyro-frequency, the local plasma frequency, the upper and lower hybrid frequencies, in the region between −60 and 60° magnetic latitude at L = 4:

```
import WPIT.Environment_mod as env

lats_deg = np.arange(-60,60,0.1)   #define latitude range in deg
lats_rad = np.deg2rad(lats_deg)    #convert to rad
L_shell=4 #L shell of interest

Bmag=env.Bmag_dipole(L_shell,lats_rad)    #dipole field strength,T
wce=env.omega_cyclotron(Bmag,const.qe,const.me) #e- cyclotron freq
wcH=env.omega_cyclotron(Bmag,const.qi,const.mH) #H+ cyclotron freq

ne0=env.density_equ_carpenter_anderson(L_shell,
            Kpmax,day,mlt,Rsun)     #equatorial e- density in cm^-3
ne0=ne0*10**6                       #convert to m^-3
ne=env.density_FL_denton(ne0,lats_rad) #e- density along field line
wpe=env.omega_plasma(ne,const.qe,const.me) #electron plasma freq
wpH=env.omega_plasma(ne,const.qi,const.mH) #H+ freq assuming ne=nH
wuhr=env.omega_uhr(wce,wpe,wcH,wpH)    #upper hybrid resonance freq
wlhr=env.omega_lhr(wce,wpe,wcH,wpH)    #lower hybrid resonance freq
```

2.2 Wave properties module

Module *WPIT. WaveProperties_mod* includes routines for the characterization of wave properties in the magnetosphere. The routines of the module are presented in Table 2. These include estimations of the Stix parameters, the refractive index, the refractive

TABLE 4 WPI modlule routines.

WPI_mod					
whistler_electron_mod		**EMIC_ion_mod**		**parallel_EMIC_mod**	
Routine Name	**Eq**	**Routine Name**	**Eq**	**Routine Name**	**Eq**
wpi_params	14	wpi_params	27	dzdt	7
dzdt	7	dzdt	7	dppardt	37
dppardt	11	dppardt	24	dpperdt	38
dpperdt	12	Dpperdt	25	detadt	39
detadt	13	Detadt	26	dlamdadt	6
dlamdadt	6	dlamdadt	6	dalphadt	40
dalphadt	15	dalphadt	28	daeqdt	41
daeqdt	16	Daeqdt	29	dEkdt	42
dEkdt	17	DEkdt	30	dgammadt	43
dgammadt	18	dgammadt	31	nonlinear_S	9
nonlinear_S	9	nonlinear_S	9	nonlinear_H	44
nonlinear_H	19	nonlinear_H	32	nonlinear_theta	45
nonlinear_theta	20	nonlinear_theta	33	—	—
nonlinear_C0	21	nonlinear_C0	34	—	—
nonlinear_C1p	22	nonlinear_C1p	35	—	—
nonlinear_C1m	23	nonlinear_C1m	36	—	—

index based on the Appleton-Hartree approximation, the refractive index of parallel propagating EMIC waves and the dielectric tensor according to cold plasma theory. WPIT includes also routines for the calculation of the above parameters accounting for warm plasma corrections, according to Kulkarni et al. (2015) and Maxworth et al. (2020). Furthermore, this module includes routines for the calculation of the resonance cone angle and the Gendrin angle according to Bortnik et al. (2006), wave electric and magnetic component amplitudes based on the formulations of Bell (1984), Jasna et al. (1992) and Li et al. (2015). WPIT includes a routine to calculate the resonant energy of an electron that interacts with a specific wave and routines to define a wave packet. In the current version of WPIT, a wave packet is defined as the latitudinal profile of the wave amplitude, that can be one sided, two sided or Gaussian in shape with respect to magnetic latitude (see, e.g., Figure 3D). For completeness, WPIT includes also, routines for the calculation of the dispersion relation of different types of waves in plasma, including the R-mode, L-mode, O-mode, X-mode and light, as well as the cut off frequencies of R- and L-mode waves (see, e.g., Swanson (2012)). Detailed description of each routine along with example runs can be found in the *WaveProperties_mod_description* Jupyter notebook, located in the *Module_descriptions* folder of the WPIT repository.

As an illustration of how the module is called, the following routine calculates the Stix parameters, the refractive index and the amplitudes of the electric and magnetic field components of the wave at L = 4 and in the region between −60 and 60° in magnetic latitude:

```
import WPIT.WaveProperties_mod as wave
import WPIT.Environment_mod as env
lats_deg = np.arange(-60,60,0.1) # define latitudinal range in deg
lats_rad = np.deg2rad(lats_deg)  # convert to rad
L_shell=4                        # L shell of interest
f_wave=2000                      # wave frequency in Hertz
w_wave=2*np.pi*f_wave
Bmag=env.Bmag_dipole(L_shell,lats_rad) # dipole field strengh in T
ne0=10**7                        # equatorial electron density
ne_lam=env.density_FL_denton(ne0,lats_rad) # e- density along field line
nH_lam=ne_lam                    #e-H+ species only
theta=np.deg2rad(0)              #wave normal angle in rad
wce=env.omega_cyclotron(Bmag,const.qe,const.me)[0] # e- cyclotron freq
wpe=env.omega_plasma(ne_lam,const.qe,const.me)[0]  # e- plasma frequency

S_,D_,P_,R_,L_=wave.stix_parameters(w_wave,
                ne_lam, nH_lam, 0, 0, Bmag)    #Stix parameters

mu_sq_plus_full,mu_sq_minus_full,ref_ind_full, \
kappa_full,kappa_par_full,kappa_per_full= \
wave.refr_index_full(theta,S_,P_,R_,L_)            #dispersion relation

mu_sq_plus_appl,mu_sq_minus_appl,ref_ind_appl, \
kappa_appl,kappa_par_appl,kappa_per_appl= \
wave.refr_index_appleton(w_wave,wpe,wce,theta) #Appleton-Hartree equation

Byw_=np.empty(len(mu_))
# y component of the wave magnetic field (1nT)
Byw_.fill(10**(-9))
# magnetic field amplitude (B in T,E in V/m)
Bxw_, Byw_, Bzw_, Exw_, Eyw_, Ezw_= \
wave.wave_amplitudes_bell(ref_ind_full,P_,D_,S_,Byw_,theta)
```

2.3 Landau damping module

The propagation of a wave within the magnetosphere and the flow of wave power are well-captured by various ray tracing techniques, which trace the path of a monochromatic wave based

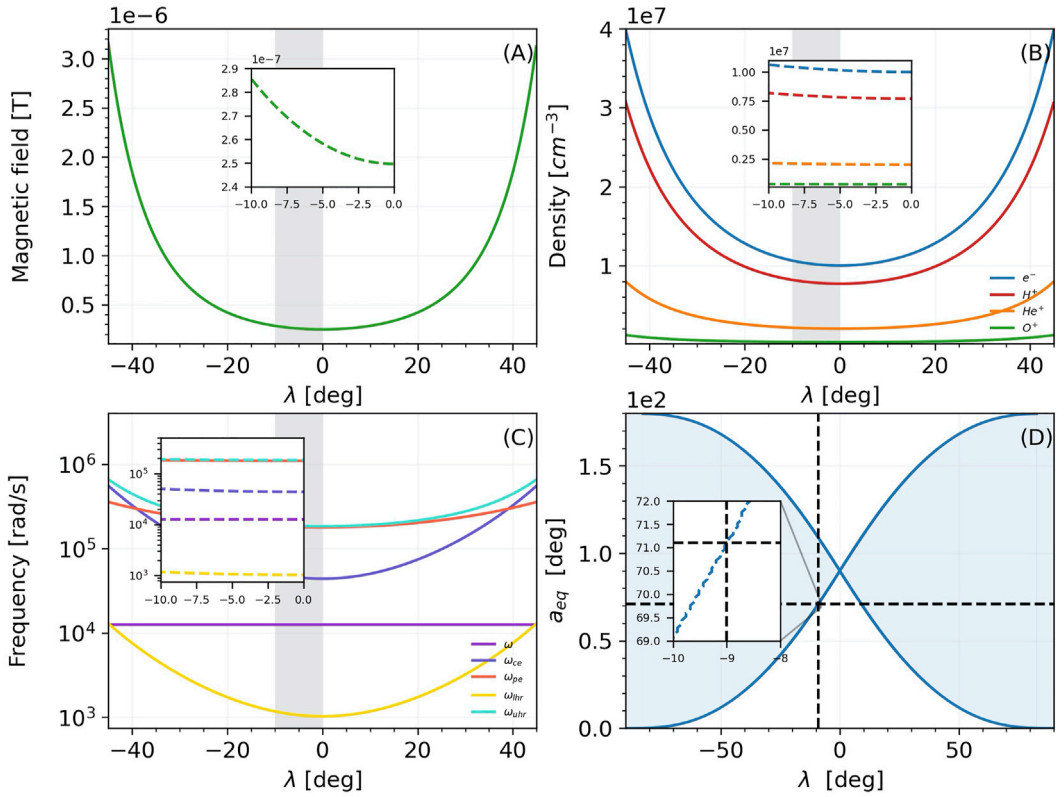


FIGURE 2
Outputs of *WPIT. Environment_mod*. **(A)**: Background dipole magnetic field. **(B)**: Electron and ion densities. **(C)**: Frequencies. **(D)**: Equatorial pitch angle mapping.

on geometric optics. Most commonly, ray tracing models are based on cold plasma theory (see, e.g., Kimura, 1966); thus, whereas they provide important parameters of a wave packet, such as its trajectory or the wave normal angle, they assume no attenuation of the wave energy. In the current version of WPIT, Landau damping is applied to a predefined ray path, as obtained from a ray-tracing model (Stanford 3D in the cases presented herein). This approach has been followed by several past studies, such as, e.g., Bell et al. (2002), Sousa (2018), Bortnik et al. (2006), Bortnik et al. (2007), Kulkarni et al. (2008). It is noted that damping or growth of the waves is expected to affect the ray path, and that a more precise approach involves calculating the damping or growth of the wave in a consistent way during the ray tracing calculation; such an approach is followed, for example, in the HOTRAY code (Horne and Thorne, 1993; Horne, 2015; Chen et al., 2009). Introducing results from self-consistent ray tracing simulations, such as HOTRAY, and in particular evaluating the differences in the calculated ray paths and the effects in the resulting wave fields and wave particle interactions compared to the current approach in WPIT that is also commonly used in literature needs to be investigated in further detail.

Module *WPIT. LandauDamping_mod* enables the estimation of the attenuation of a wave along the ray path according to Landau damping, which refers to the damping of an electromagnetic wave due to its interaction with thermal electrons with velocities that have a component parallel to the ambient geomagnetic field, close to the phase velocity of the wave. The theory behind the calculation of Landau damping is based on the work by Brinca (1972), who expanded upon the work of Kennel (1966).

For the calculation of Landau damping along a ray path the local thermal electron distribution is required, and WPIT enables the selection among four different models, namely:

- *WPIT. LandauDamp_mod.bell_distribution*, which is based on the work by Bell et al. (2002),
- *WPIT. LandauDamp_mod.bortnik_distribution*, which is based on Bortnik et al. (2007),
- *WPIT. LandauDamp_mod.golden_distribution*, which is based on Golden et al. (2010), and
- *WPIT. LandauDamp_mod.bi_maxwellian_distribution*, which is based on Maxworth et al. (2020).

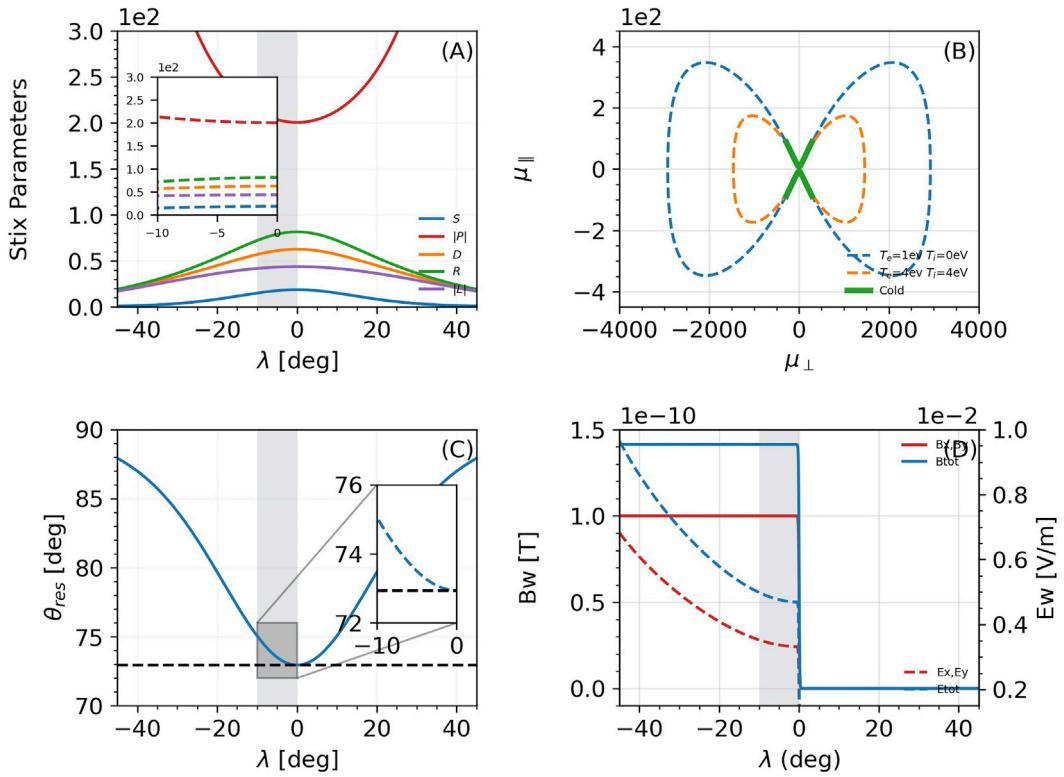


FIGURE 3
Outputs of *WPIT. WaveProperties_mod*. **(A)**: Stix parameters. **(B)**: Refractive index surface. **(C)**: Resonance cone angle. **(D)**: Wave electric and magnetic fields for a wave with B_{yw} =100*pT*.

WPIT also enables introducing any user-defined distribution of the form $f = f(u_\perp, u_\parallel)$.

The local Landau damping of a wave along its propagation path in WPIT is calculated by routine *WPIT. LandauDamp_mod.landau_damping*. Landau damping calculations in WPIT are based on the work of Sousa (2018) and on the corresponding code found at https://github.com/asousa/damping written in Matlab. For WPIT this code has been transcribed into a Python code, while being further enhanced in terms of usability by enabling the selection of different thermal electron distributions and the integration with outputs from ray tracing simulations.

At the time of writing, the ray paths that are introduced in WPIT are pre-calculated using the Stanford 3D ray tracer based on the code publicly available at https://github.com/asousa/Stanford_Raytracer. *WPIT. LandauDamp_mod* includes also the sub-module *WPIT. LandauDamp_mod. RayUtils* which is a collection of routines in relevance with the Ray Tracer output. After a ray tracing simulation is performed with the Stanford 3D Ray Tracer, the ray tracing simulation is exported in . *ray* format, which includes information such as the wave group velocity, phase velocity, refractive indices, magnetic field as a function of time and position along the ray path. Examples of ray tracing outputs are included in the WPIT github, at https://github.com/stourgai/WPIT/tree/main/Module_descriptions/example_rays. These data files are read in WPIT by routine *WPIT. LandauDamp_mod.RayUtils.read_input_ray*. Subsequently, using routine *WPIT. LandauDamp_mod.RayUtils.append_ray*, the required wave properties that are calculated by module *WPIT. WaveProperties_mod* as described above in Section 2.2, are calculated along the ray path and are appended in a new file that is saved in . *csv* format. Moreover, routine *WPIT. LandauDamp_mod.RayUtils.resonance_along_raypath* calculates the velocities and energies of particles that can interact resonantly with the ray-wave for a range of pitch angles. Routine *WPIT. LandauDamp_mod.RayUtils. enhancement_factor* calculates the magnetospheric cavity enhancement factor as defined in Kulkarni et al. (2006). Finally, routine *WPIT. LandauDamp_mod.RayUtils.ray_plots* produces plots of the ray path and wave parameters. The routines of the *WPIT. LandauDamp_mod* module are presented in Table 3.

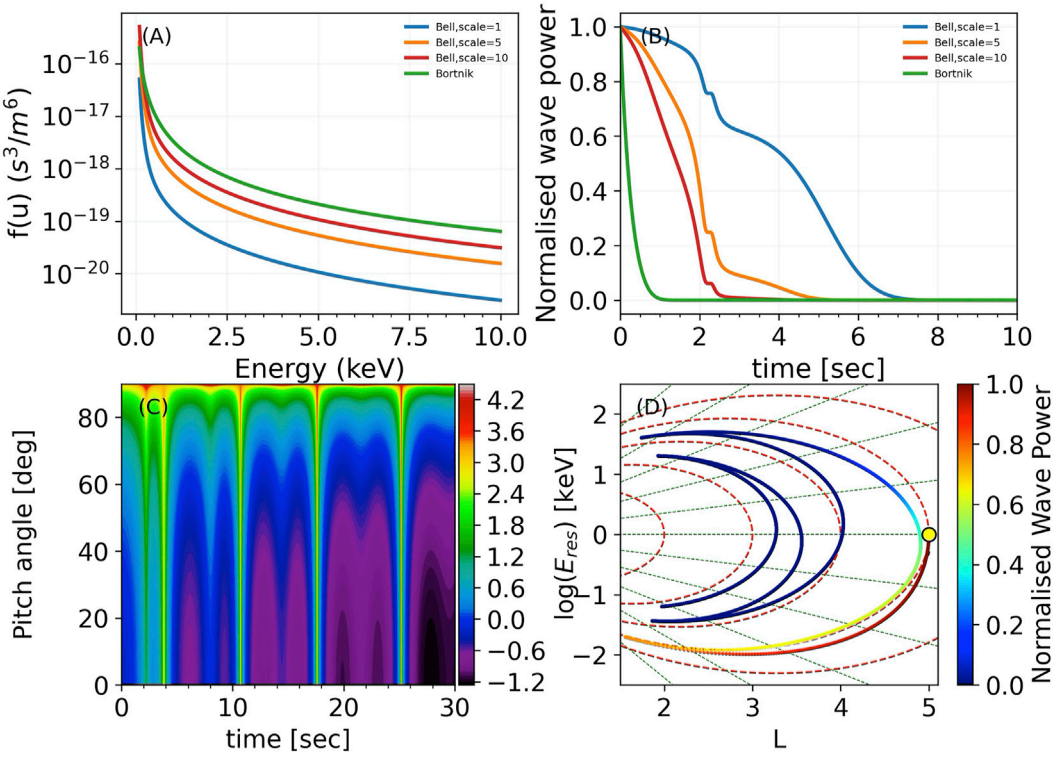


FIGURE 4
Outputs of *WPIT. LandauDamp_mod*. **(A)**: Thermal electron distributions. **(B)**: Landau damping calculations for each distribution. **(C)**: Electron resonant energy along the ray path for a range of pitch angles. **(D)**: Ray path with color coded Landau damping.

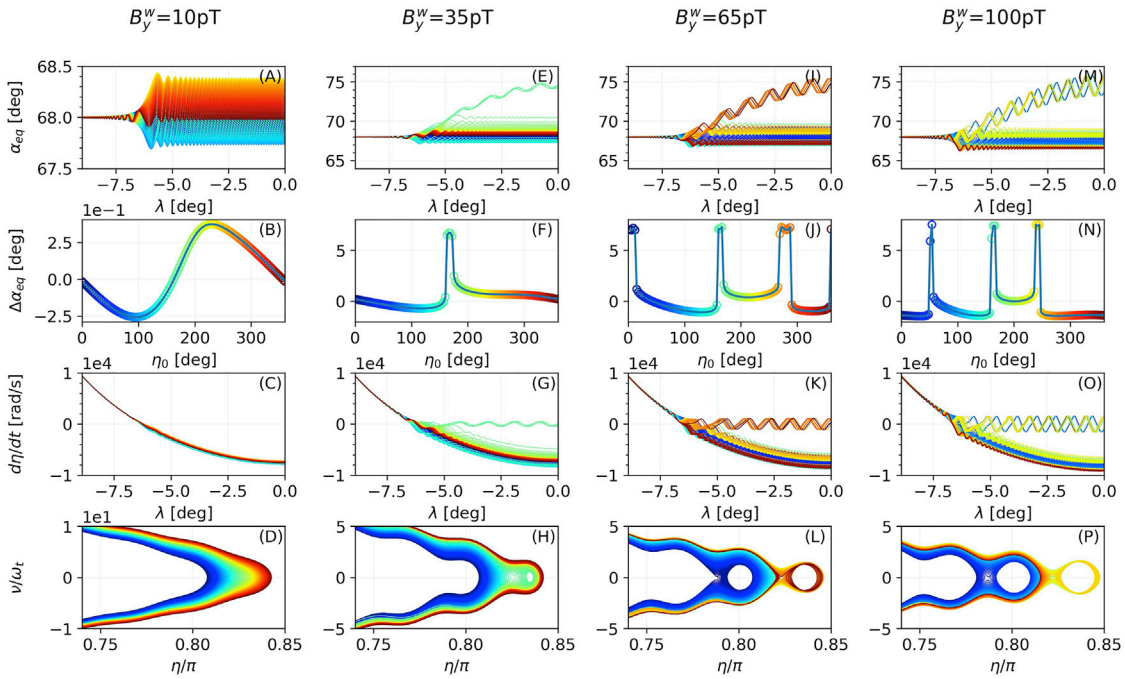


FIGURE 5
Evolution of α_{eq} along λ **(A,E,I,M)**, distribution of $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0 **(B,F,J,N)**, derivative of wave-particle phase η along λ **(C,G,K,O)** and electron trajectories in the ν-η plane **(D,H,L,P)** for different values of the wave magnetic field y-component.

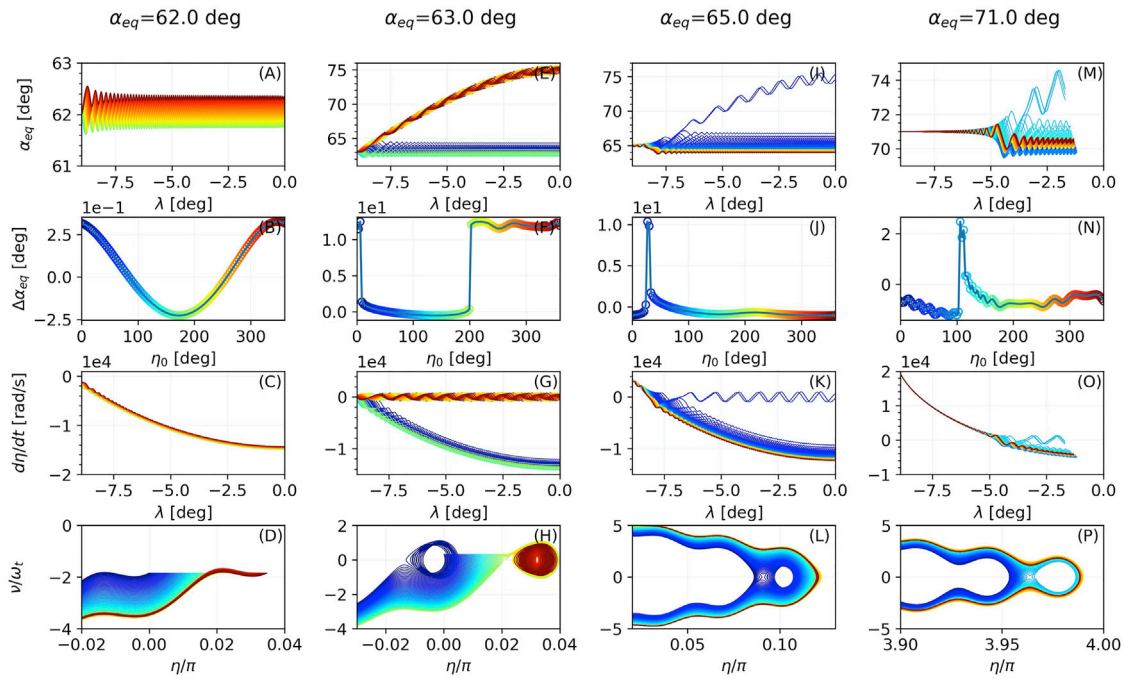


FIGURE 6
Evolution of α_{eq} along λ **(A,E,I,M)**, distribution of $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0 **(B,F,J,N)**, derivative of wave-particle phase η along λ **(C,G,K,O)** and electron trajectories in the ν-η plane **(D,H,L,P)** for different values of equatorial pitch angle.

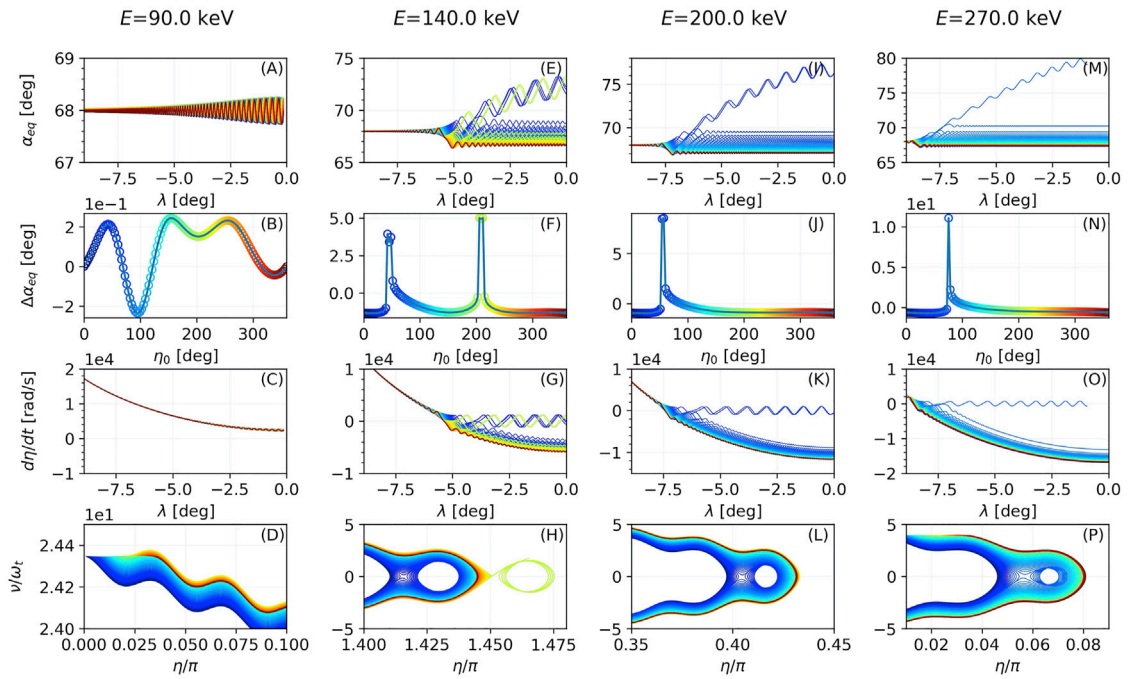


FIGURE 7
Evolution of α_{eq} along λ **(A,E,I,M)**, distribution of $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0 **(B,F,J,N)**, derivative of wave-particle phase η along λ **(C,G,K,O)** and electron trajectories in the ν-η plane **(D,H,L,P)** for different values of the electron energy.

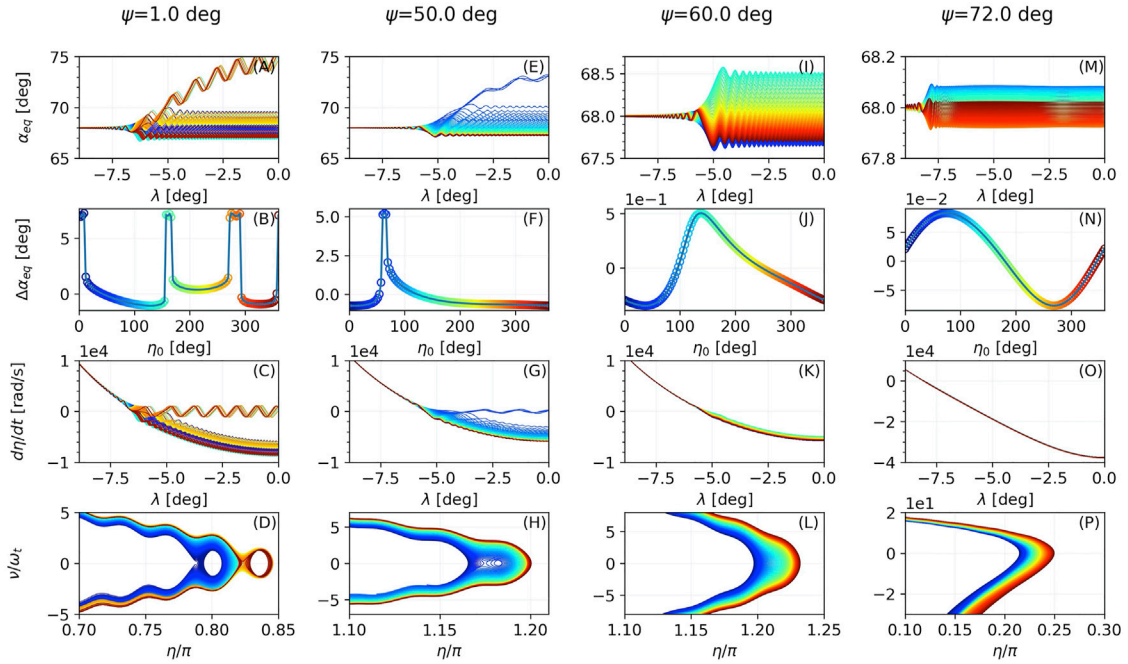


FIGURE 8
Evolution of α_{eq} along λ **(A,E,I,M)**, distribution of $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0 **(B,F,J,N)**, derivative of wave-particle phase η along λ **(C,G,K,O)** and electron trajectories in the ν-η plane **(D,H,L,P)** for different values of the wave normal angle.

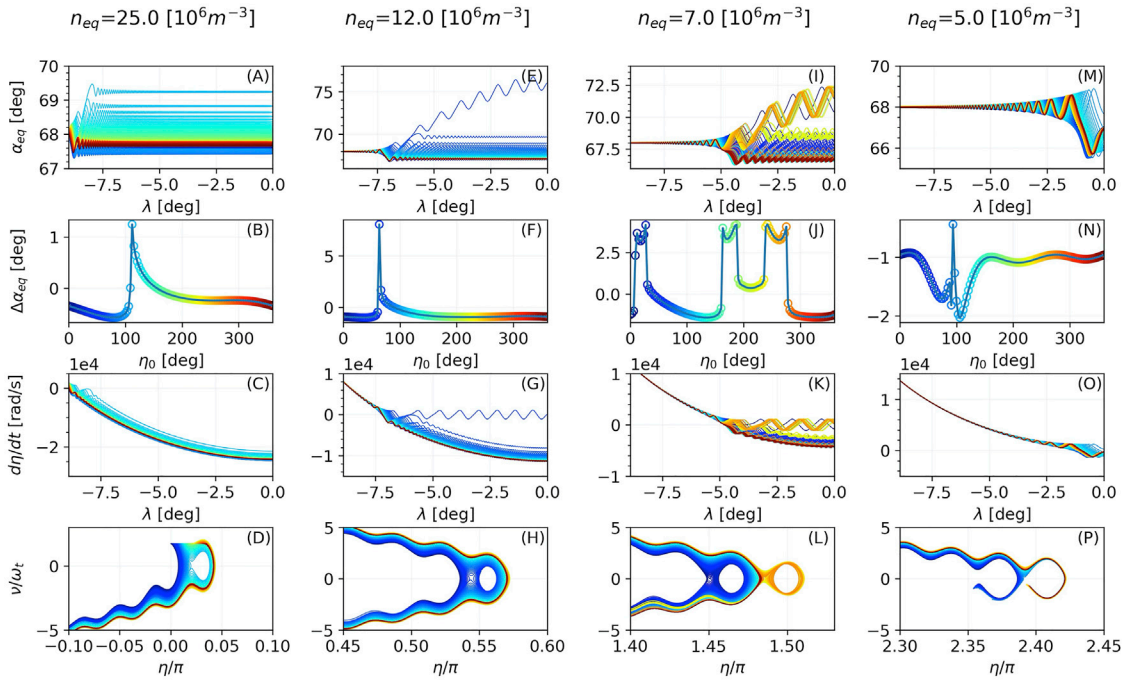


FIGURE 9
Evolution of α_{eq} along λ **(A,E,I,M)**, distribution of $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0 **(B,F,J,N)**, derivative of wave-particle phase η along λ **(C,G,K,O)** and electron trajectories in the ν-η plane **(D,H,L,P)** for different values of the equatorial electron density.

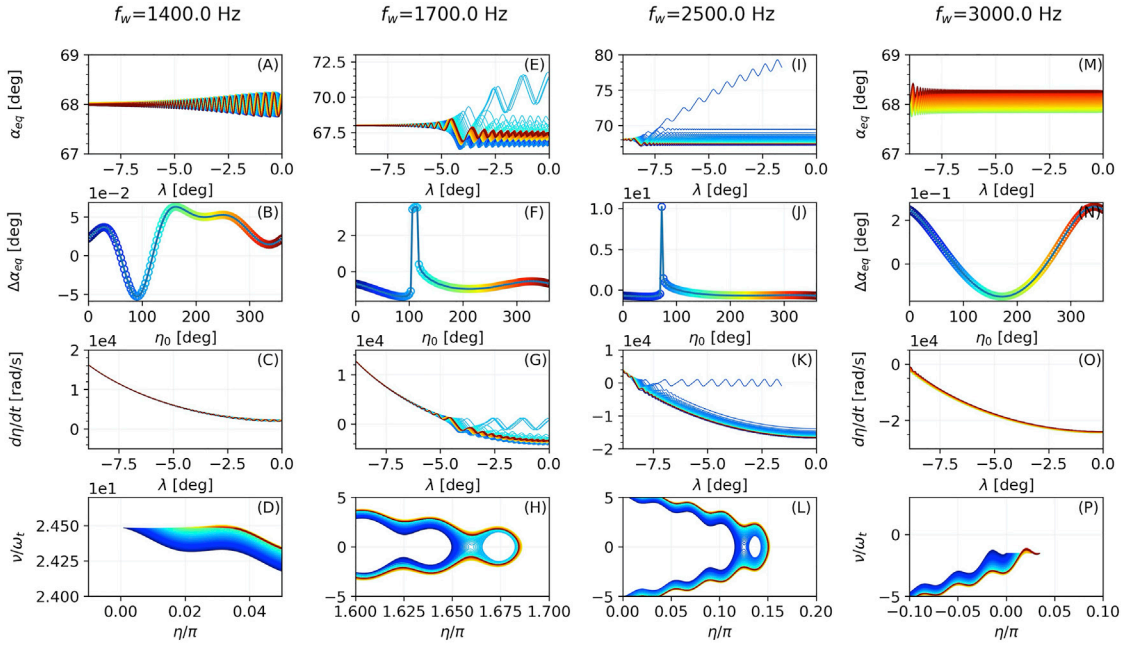


FIGURE 10
Evolution of α_{eq} along λ **(A,E,I,M)**, distribution of $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0 **(B,F,J,N)**, derivative of wave-particle phase η along λ **(C,G,K,O)** and electron trajectories in the v-η plane **(D,H,L,P)** for different wave frequencies.

Detailed description of each routine along with sample runs can be found in the *Module_descriptions* folder of the WPIT repository in the corresponding *LandauDamp_mod_description* Jupyter notebook.

It is noted that WPIT enables incorporating ray path information from ray tracing models other than the Stanford 3D ray tracer, with the proper modification of the routines for importing ray tracer simulation outputs, named *WPIT. LandauDamp_mod.RayUtils.read_input_ray*.

2.4 Wave—particle interactions module

This module includes routines for estimating the gyro-averaged wave-particle interaction of relativistic particles with a monochromatic wave. The environment and wave properties used in this module are derived from modules *WPIT. Enironment_mod* and *WPIT. WaveProperties_mod*, as described above. At the time of writing, *WPIT. WPI_mod* includes three sub-modules in order to simulate different types of waves and particles: (a) sub-module *WPIT. WPI_mod.whistler_electron_mod* that is used to simulate the interactions of electrons with whistler mode or magnetosonic waves (either parallel or oblique), (b) sub-module *WPIT. WPI_mod.EMIC_ion_mod* that is used for the investigation of interactions of ions with EMIC waves (either parallel or oblique) and (c) sub-module *WPIT. WPI_mod.parallel_EMIC_mod* for the investigation of interactions of either electrons or ions with parallel EMIC waves.

Before introducing in further detail the wave-particle interaction simulations, the formulations of several parameters are discussed, which are required by each of the three types of wave-particle interactions that are tackled in WPIT. These include: dz/dt, which refers to the rate of change of the distance traveled by a particle along the field line; $dp_{\parallel}/dt$, the rate of change of the parallel momentum; $dp_{\perp}/dt$, the rate of change of the perpendicular momentum and $d\eta/dt$, the rate of change of the wave-particle phase, which in case of WPIT are taken directly from literature. WPIT also includes routines for the calculations of the rate of change of the local pitch angle $d\alpha/dt$, the rate of change of the equatorial pitch angle $d\alpha_{eq}/dt$, the rate of change of the instantaneous particle kinetic energy dE/dt, the rate of change of the relativistic Lorentz factor dy/dt, and the rate of change of the magnetic latitude, $d\lambda/dt$. In the following the methodology for the derivation of each of the above parameters is presented and in the following subsections these are applied specifically for each of the three sub-modules.

The rate of change of the local pitch angle, is based on the derivation:

$$\frac{d}{dt}\tan\alpha = \frac{1}{\cos^2\alpha}\frac{d\alpha}{dt}$$

$$\frac{d\alpha}{dt} = \cos^2\alpha \frac{d}{dt}\tan\alpha = \cos^2\alpha \frac{d}{dt}\frac{p_\perp}{p_\parallel} = \frac{1}{p^2}\left(p_\parallel \frac{dp_\perp}{dt} - p_\perp \frac{dp_\parallel}{dt}\right) \tag{1}$$

where α is the local pitch angle, p is the magnitude of the momentum, $p_\parallel$ is the component of the momentum parallel to the ambient magnetic field and $p_\perp$ is the component of the momentum perpendicular to the ambient magnetic field.

The rate of change of the equatorial pitch angle, is derived as:

$$\frac{d\alpha_{eq}}{dt} = \frac{\tan\alpha_{eq}}{\tan\alpha}\left(\frac{d\alpha}{dt} - \frac{p_\perp}{2\gamma m_s \omega_{ce}}\frac{\partial\omega_{cj}}{dz}\right) \tag{2}$$

where we made use of the relation between the local and the equatorial pitch angle in a dipole magnetic field. In Eq. 2, α_{eq} is the local pitch angle, m_s is the particle mass, γ is the Lorentz factor, ω_{cj}, with $j = [e, i]$, for electrons or ions respectively, is the particle cyclotron frequency and $\partial\omega_{cj}/\partial z$ is the gradient of the cyclotron frequency along the field line.

The kinetic energy and the Lorentz factor γ are derived based on the following equations:

$$\frac{dE_k}{dt} = \frac{1}{\gamma m_s}\left(p_\parallel \frac{dp_\parallel}{dt} + p_\perp \frac{dp_\perp}{dt}\right) \tag{3}$$

$$\frac{d\gamma}{dt} = \frac{d}{dt}\frac{E_k}{m_s c^2} = \frac{1}{\gamma m_s^2 c^2}\left(p_\parallel \frac{dp_\parallel}{dt} + p_\perp \frac{dp_\perp}{dt}\right) \tag{4}$$

where E_k is the particle's kinetic energy, c is the speed of light and the remaining terms as defined above.

Finally, for the rate of change of the particle's magnetic latitude, we use the following equation, that relates the distance along a field with the latitude for a dipole field:

$$dz = LR_e\left(1 + 3\sin^2\lambda\right)^{1/2}\cos\lambda d\lambda \tag{5}$$

thus

$$\frac{d\lambda}{dt} = \frac{d\lambda}{dz}\frac{dz}{dt} = \frac{1}{LR_e\left(1 + 3\sin^2\lambda\right)^{1/2}\cos\lambda}\frac{p_\parallel}{\gamma m_s} \tag{6}$$

where we used:

$$\frac{dz}{dt} = \frac{p_\parallel}{\gamma m_s} \tag{7}$$

where λ is the latitude, z is the distance along the field line, L is the L shell, R_e is the Earth's radius, m_s is the particle's mass and all the other parameters as defined previously. Equations 6 and 7 will be used in the following Sections 2.4.1, Section 2.4.2 and Section 2.4.3.

For the investigation and quantification of nonlinear effects during wave-particle interactions, we derive the relevant equations for each module based on the reasoning of Su et al. (2014).

We start by calculating the second derivative of the wave-particle phase, and we transform the resulting equation in the form:

$$\frac{d\nu}{dt} = \omega_t^2 sign(\theta)\sin\eta + H = \omega_t^2\left[sign(\theta)\sin\eta + S\right] \tag{8}$$

with

$$S = \frac{H}{\omega_t^2} \tag{9}$$

and

$$\nu = \frac{d\eta}{dt} \tag{10}$$

ω_t is the trapping frequency, and S is a ratio that defines the relative importance of the wave induced motion to the adiabatic motion. When $|S| > 1$ the adiabatic motion dominates, while when $|S| < 1$ the wave induced motion prevails (Su et al., 2014).

The routines of each sub-module are presented in the following sub-sections.

2.4.1 Whistler mode—electron interactions module

The routines of this module can be used for the investigation of the interactions of electrons with whistler or magnetosonic waves. The equations for the calculation of $dp_\parallel/dt$, $dp_\perp/dt$ and $d\eta/dt$ are derived from Bortnik et al. (2015).

The evolution of the component of the electron momentum parallel to the magnetic field, $p_\parallel$, is calculated as:

$$\frac{dp_\parallel}{dt} = \omega_{\tau m}^2 m_e k_z^{-1}\sin\eta - \frac{1}{\gamma m_e}\frac{p_\perp^2}{2\omega_H}\frac{\partial\omega_{ce}}{\partial z} \tag{11}$$

while the corresponding term for the perpendicular component, $p_\perp$, is calculated as:

$$\begin{aligned}\frac{dp_\perp}{dt} = -(-1)^{m-1}\Bigg[&\omega_1\left(\frac{p_z}{\gamma} + m_e R_1\right)J_{m-1}(\beta)\\ &-\omega_2\left(\frac{p_z}{\gamma} - m_e R_2\right)J_{m+1}(\beta)\Bigg]\sin\eta + \frac{1}{\gamma m_e}\frac{p_\perp p_z}{2\omega_H}\frac{\partial\omega_{ce}}{\partial z}\end{aligned} \tag{12}$$

and the evolution of electron-wave phase η is calculated as:

$$\frac{d\eta}{dt} = \frac{m\omega_{ce}}{\gamma} - \omega - k_z\frac{p_z}{\gamma m_e} \tag{13}$$

where

$$\begin{aligned}\beta &= \frac{k_x p_\perp}{m_e\gamma\omega_{ce}}, k_z = k\cos\psi = (\mu\omega/c)\cos\psi, k_x = k\sin\theta\\ \omega_{\tau m}^2 &= (-1)^{m-1}\omega_{\tau 0}^2\left[J_{m-1}(\beta) - \alpha_1 J_{m+1}(\beta) + \gamma\alpha_2 J_m(\beta)\right]\\ \omega_{\tau 0}^2 &= \frac{\omega_1 k_z p_\perp}{\gamma m_e}, \omega_1 = \frac{e}{2m_e}\left(B_x^w + B_y^w\right), \omega_2 = \frac{e}{2m_e}\left(B_x^w - B_y^w\right),\\ \alpha_1 &= \frac{\omega_2}{\omega_1}\\ \alpha_2 &= \frac{eE_z^w}{\omega_1 p_\perp}, R_1 = \frac{E_x^w + E_y^w}{B_x^w + B_y^w}, R_2 = \frac{E_x^w - E_y^w}{B_x^w - B_y^w}\end{aligned} \tag{14}$$

where μ is the refractive index, ψ is the wave normal angle, J_i are Bessel functions of the first kind, of order i and argument β, k_x and k_z are the x and z components of the wave number k, ω_{ce} is

the electron cyclotron frequency and B_j^w and E_j^w, with $j = [x, y, z]$, are the magnetic and electric field components of the wave.

By applying Eqs 1–4, we derive the set of auxiliary equations for the sub-module related to whistler-electron interactions:

$$\frac{d\alpha}{dt} = -\frac{1}{p^2}\left\{(-1)^{m-1}\left[\omega_1\left(\frac{p_\parallel}{\gamma}+m_eR_1\right)J_{m-1}(\beta) -\omega_2\left(\frac{p_\parallel}{\gamma}-m_eR_2\right)J_{m+1}(\beta)\right]p_\parallel +\omega_{\tau m}^2 m_e k_\parallel^{-1} p_\perp\right\}\sin\eta + \frac{1}{m_e\gamma}\frac{p_\parallel p^2}{2\omega_H}\frac{\partial\omega_{ce}}{\partial z} \tag{15}$$

$$\frac{d\alpha_{eq}}{dt} = -\frac{1}{p^2}\frac{\tan\alpha_{eq}}{\tan\alpha}\left\{(-1)^{m-1}\left[\omega_1\left(\frac{p_\parallel}{\gamma}+m_eR_1\right)J_{m-1}(\beta) -\omega_2\left(\frac{p_\parallel}{\gamma}-m_eR_2\right)J_{m+1}(\beta)\right]p_\parallel +\omega_{\tau m}^2 m_e k_z^{-1} p_\perp\right\}\sin\eta \tag{16}$$

$$\frac{dE_k}{dt} = (-1)^{m-1}\frac{e}{\gamma m_e}\left[E_z^w p_\parallel J_m(\beta) - p_\perp E_L J_{m+1}(\beta) - p_\perp E_R J_{m-1}(\beta)\right]\sin\eta \tag{17}$$

$$\frac{d\gamma}{dt} = (-1)^{m-1}\frac{e}{\gamma m_e^2 c^2}\left[E_z^w p_\parallel J_m(\beta) - p_\perp E_L J_{m+1}(\beta) - p_\perp E_R J_{m-1}(\beta)\right]\sin\eta \tag{18}$$

For the investigation of nonlinear effects we derive eight for the case of whistler-electron interactions. The resulting equations are:

$$H = \frac{m}{\gamma}\frac{d\omega_{ce}}{dt} + \frac{p_\parallel}{\gamma m_e}\frac{dk_\parallel}{dt} - \frac{d\omega_{ce}}{dt} - \frac{k_\parallel p_\perp^2}{2\gamma^2 m_e^2\omega_{ce}}\frac{\partial\omega_{ce}}{\partial z} \tag{19}$$

$$\theta = C_0 J_m(\beta) + C_{+1}J_{m+1}(\beta) + C_{-1}J_{m-1}(\beta) \tag{20}$$

$$C_0 = -(-1)^{m-1}\left[\frac{q_e p_\parallel}{\gamma^3 m_e^2 c^2}\left(m\omega_{ce} - \frac{k_\parallel p_\parallel}{m_e}\right) + \frac{k_\parallel q_e}{\gamma m_e}\right]E_z^w \tag{21}$$

$$C_{+1} = (-1)^{m-1}\left[\frac{q_e p_\perp E_L}{\gamma^3 m_e^2 c^2}\left(m\omega_{ce} - \frac{k_\parallel p_\parallel}{m_e}\right) + \frac{p_\perp k_\parallel \omega_2}{m\gamma^2}\right] \tag{22}$$

$$C_{-1} = (-1)^{m-1}\left[\frac{q_e p_\perp E_R}{\gamma^3 m_e^2 c^2}\left(m\omega_{ce} - \frac{k_\parallel p_\parallel}{m_e}\right) - \frac{p_\perp k_\parallel \omega_1}{m\gamma^2}\right] \tag{23}$$

The corresponding routines that incorporate the above equations are presented in Table 4.

2.4.2 EMIC wave—ion interactions module

The routines of this module can be used for the investigation of the interactions of ions with EMIC waves. The equations for the calculation of $dp_\parallel/dt$, $dp_\perp/dt$ and $d\eta/dt$ are derived from (Su et al., 2014).

The evolution of the component of the electron momentum that is parallel to the magnetic field, $p_\parallel$, is calculated as:

$$\frac{dp_\parallel}{dt} = (-1)^{l+1}\left[qE_w^z J_l(\beta) - \omega_R p_\perp J_{l+1}(\beta) + \omega_L p_\perp J_{l-1}(\beta)\right] \times \sin\eta - \frac{p_\perp^2}{2\gamma m\omega_{ci}}\frac{\partial\omega_{ci}}{\partial z} \tag{24}$$

while for the corresponding term for the perpendicular component, $p_\perp$, is calculated as:

$$\frac{dp_\perp}{dt} = (-1)^{l+1}\left[(p_\parallel - p_w^R)\omega_R J_{l+1}(\beta) -(p_\parallel - p_w^L)\omega_L J_{l-1}(\beta)\right]\sin\eta + \frac{p_\perp p_\parallel}{2\gamma m\omega_{ci}}\frac{\partial\omega_{ci}}{\partial z} \tag{25}$$

The evolution of the ion-wave phase η is calculated as:

$$\frac{d\eta}{dt} = \frac{l\omega_{ci}}{\gamma} + \frac{k_\parallel p_\parallel}{\gamma m} - \omega \tag{26}$$

where

$$\beta = -\frac{k_\perp p_\perp}{qB_D},\ p_w^R = \gamma m\frac{E_w^R}{B_w^R},\ p_w^L = \gamma m\frac{E_w^L}{B_w^L},\ B_w^R = \frac{B_w^x + B_w^y}{2}$$
$$B_w^L = \frac{B_w^x - B_w^y}{2},\ E_w^R = \frac{E_w^x + E_w^y}{2},\ E_w^L = \frac{E_w^x - E_w^y}{2},\ \omega_R = \frac{qB_w^R}{\gamma m},\quad \omega_L = \frac{qB_w^L}{\gamma m} \tag{27}$$

where B_D is dipole field strength.

By applying Eqs 1–4 for the case of EMIC waves, we get:

$$\frac{d\alpha}{dt} = (-1)^{l+1}\frac{\sin\eta}{p^2}\left[qE_w^z p_\parallel J_l(\beta) - \omega_R p_w^R p_\perp J_{l+1}(\beta) + \omega_L p_w^L p_\perp J_{l+1}(\beta)\right] \tag{28}$$

$$\frac{d\alpha_{eq}}{dt} = (-1)^{l+1}\frac{p_\parallel}{p_\perp}\frac{\tan\alpha_{eq}\sin\eta}{p^2}\left[-qE_w^z p_\perp J_l(\beta) +\omega_R(p^2 - p_w^R p_\parallel)J_{l+1}(\beta) - \omega_L(p^2 + p_w^L p_\parallel)J_{l-1}(\beta)\right] \tag{29}$$

$$\frac{dE_k}{dt} = (-1)^{l+1}\frac{q}{m\gamma}\left[E_w^z p_\parallel J_l(\beta) - E_w^R p_\perp J_{l+1}(\beta) - E_w^L p_\perp J_{l-1}(\beta)\right]\sin\eta \tag{30}$$

$$\frac{d\gamma}{dt} = (-1)^{l+1}\frac{q}{\gamma m^2 c^2}\left[E_w^z p_\parallel J_l(\beta) - E_w^R p_\perp J_{l+1}(\beta) - E_w^L p_\perp J_{l-1}(\beta)\right]\sin\eta \tag{31}$$

For the investigation of nonlinear effects related to the interaction of EMIC waves with ions, WPIT uses the approximation and equations described in (Su et al., 2014):

$$H = \frac{l}{\gamma}\frac{d\omega_{ci}}{dt} + \frac{p_\parallel}{\gamma m}\frac{dk_\parallel}{dt} - \frac{d\omega}{dt} - \frac{k_\parallel p_\perp^2}{2\gamma^2 m^2\Omega}\frac{\partial\omega_{ci}}{\partial z} \tag{32}$$

$$\theta = C_0 J_l(\beta) + C_{+1}J_{l+1}(\beta) + C_{-1}J_{l-1}(\beta) \tag{33}$$

$$C_0 = (-1)^{l+1}\left[\frac{qk_\parallel}{\gamma m} - \frac{qp_\parallel}{\gamma^3 m^2 c^2}\left(l\omega_{ci} + \frac{k_\parallel p_\parallel}{m}\right)\right]E_w^z \tag{34}$$

$$C_{+1} = (-1)^{l+1}\left[-\frac{\omega_R k_\parallel}{\gamma m} + \frac{qE_w^R}{\gamma^3 m^2 x^2}\left(l\omega_{ci} + \frac{k_\parallel p_\parallel}{m}\right)\right]p_\perp \tag{35}$$

$$C_{-1} = (-1)^{l+1}\left[\frac{\omega_L k_\parallel}{\gamma m} + \frac{qE_w^L}{\gamma^3 m^2 x^2}\left(l\omega_{ci} + \frac{k_\parallel p_\parallel}{m}\right)\right]p_\perp \tag{36}$$

The corresponding routines that incorporate the above equations are presented in Table 4.

2.4.3 Parallel EMIC wave—electron & ion interactions module

The routines of this module can be used for the investigation of the interactions of either electrons or ions with parallel

propagating EMIC waves. The equations for the calculation of $dp_{\parallel}/dt$, $dp_{\perp}/dt$ and $d\eta/dt$ are derived from (Su et al., 2013):

$$\frac{dp_{\parallel}}{dt} = \frac{eB_w}{\gamma m} p_{\perp} \sin\eta - \frac{p_{\perp}^2}{2\gamma m \omega_{cj}} \frac{\partial \omega_{cj}}{\partial z} \tag{37}$$

$$\frac{dp_{\perp}}{dt} = eB_w \left(\frac{\omega}{k} - \frac{p_{\parallel}}{\gamma m}\right) \sin\eta + \frac{p_{\perp} p_{\parallel}}{2\gamma m \omega_{cj}} \frac{\partial \omega_{cj}}{\partial z} \tag{38}$$

$$\frac{d\eta}{dt} = \frac{eB_w}{p_{\perp}} \left(\frac{\omega}{k} - \frac{p_{\parallel}}{\gamma m}\right) \cos\eta + \left(\frac{k p_{\parallel}}{\gamma m} - \omega - \frac{|\omega_{cj}|}{\gamma}\right) \tag{39}$$

By applying Eqs 1–4 for the parallel EMIC case, we get:

$$\frac{d\alpha}{dt} = \frac{eB_w}{p^2} \left[\left(\frac{\omega}{k} - \frac{p}{\gamma m}\right) p_{\parallel} - \frac{p_{\perp}^2}{\gamma m}\right] \sin\eta + \frac{p_{\perp}}{2\gamma m \omega_c} \frac{\partial \omega_{cj}}{\partial z} \tag{40}$$

$$\frac{d\alpha_{eq}}{dt} = \frac{eB_w}{p^2} \frac{\tan\alpha_{eq}}{\tan\alpha} \left[\left(\frac{\omega}{k} - \frac{p_{\parallel}}{\gamma m}\right) p_{\parallel} - \frac{p_{\perp}^2}{\gamma m}\right] \sin\eta \tag{41}$$

$$\frac{dE_k}{dt} = eB_w \frac{\omega}{k} \frac{p_{\perp}}{\gamma m} \sin\eta \tag{42}$$

$$\frac{d\gamma}{dt} = eB_w \frac{\omega}{k} \frac{p_{\perp}}{\gamma m^2 c^2} \sin\eta \tag{43}$$

For the investigation of nonlinear effects we derive eight for the case of parallel EMIC waves. The resulting equations are:

$$H = -\frac{1}{\gamma} \frac{d\omega_{cj}}{dt} + \frac{p_{\parallel}}{\gamma m} \frac{dk_{\parallel}}{dt} - \frac{d\omega}{dt} - \frac{k_{\parallel} p_{\perp}^2}{2\gamma^2 m^2 \omega_{cj}} \frac{\partial \omega_{cj}}{\partial z} \tag{44}$$

$$\theta = eB_w p_{\perp} \left[\frac{k}{\gamma^2 m^2} + \left(\omega_c - \frac{k_{\parallel} p_{\parallel}}{m} \frac{\omega}{k\gamma^3 m c^2}\right)\right] \tag{45}$$

The corresponding routines that incorporate the above equations are presented in Table 4.

2.5 Integration

Based on the equations described above, wave-particle interaction parameters are estimated along a particle's trajectory via the integration of the corresponding differential equations. For the simulations presented herein, we used a 4^{th} order Runge-Kutta integrator, but a user of WPIT could use any convenient integrator to integrate the differential equations of the particle. The set-up of the integrator for each simulation can be found in the corresponding Jupyter notebooks in *WPIT_Results* and *WPIT_tests* folders of WPIT repository.

2.6 WPIT requirements

Apart from the modules described above, WPIT requires the installation of some open source Python packages. These are the *matplotilb*, the *numpy*, the *pandas*, the *scipy*, the *spacepy* and the *notebook* packages. *Matplotlib* is a visualization library in Python and it is used for producing all the output figures of the WPIT repository. *Numpy* is a package for scientific computing in Python. In WPIT it is called in all of the routines for performing mathematical calculations. *Pandas* is a tool for data analysis. It is used for reading and writing data files in WPIT. *Scipy* is a set of algorithms for scientiffic computing. WPIT uses the *scipy. special* module for the calculation of Bessel functions. Finally, the installation of the *notebook* package, enables the use of Jupyter Notebooks. The version of each package used in WPIT testing and an installation file is included in the WPIT repository (https://github.com/stourgai/WPIT/blob/main/requirements.txt).

2.7 Importing WPIT modules

It is noted that in its current version, WPIT does not include a setup file. Thus, the user must add the WPIT source code path to the Python path. This is done as follows: the built-in Python modules *os* and *sys* should be imported first. The *os* module implements functions on pathnames, while the *sys* module contains parameters specific to the system. The *os. path.abspath*() function is first used to define the path of the WPIT folder which contains the source code of the package; subsequently the *sys. path.append*() function is used to add the WPIT source code path to the Python path. Then the WPIT modules can be imported to the code. In the following, we present a code snippet which illustrates this procedure:

```
import os #import os module
import os #import sys module

#define the WPIT source code path
WPITpath=os.path.abspath("/path_to_WPIT_source_code")

#add WPIT path to Python path
sys.path.append(WPITpath)


# import Environment Characterization Module
import WPIT.Environment_mod

# import Wave Properties Module
import WPIT.WaveProperties_mod

# import Landau Damping Module
import WPIT.LandauDamp_mod

# import Ray Utils Sub-module
import WPIT.LandauDamp_mod.RayUtils_mod

# import Whistler-mode-Electron Interactions Module
import WPIT.WPI_mod.whistler_electron_mod

# import EMIC Wave-Ion Interactions Module
import WPIT.WPI_mod.EMIC_ion_mod

# import Parallel EMIC Wave-Electron & Ion Interactions Module
import WPIT.WPI_mod.EMIC_ion_mod.parallel_EMIC_mod
```

2.8 Methodology verification

For the verification of the WPIT code and as a demonstration of its capabilities, the results from a number of key past studies related to wave-particle interactions are reproduced using WPIT. These include results: Bortnik et al. (2008), Albert and Bortnik (2009), Su et al. (2012) and Su et al. (2014). The corresponding parameterizations of WPIT that reproduce the results of the above studies are formatted as Jupyter notebooks and are included in the project's folder *WPIT_tests* located at https://github.com/stourgai/WPIT/tree/main/WPIT_tests. It is noted

that these Jupyter notebooks confirming past results can also be used as tutorials for the use of the WPIT routines. Reproductions of results of these publications are presented in Supplementary Figures S1-S12 as follows: results by Bortnik et al. (2008) are included as Supplementary Figures S1-S3; results by Albert and Bortnik (2009) are included as Supplementary Figures S4-S6; results by Su et al. (2012) are included as Supplementary Figures S7, S8; and results by Su et al. (2014) are included as Supplementary Figures S9-S12.

3 Results

In the following, and in order to present the capabilities of WPIT, simulation outputs are presented for each of the four WPIT modules. As a case study, we expand upon the simulations by Bortnik et al. (2008) through a parametric study that explores the impact of each parameter in the onset of nonlinearity in whistler-electron interactions. More specifically, the dependence of the appearance of nonlinear effects on wave field amplitude, equatorial pitch angle, wave normal angle, electron energy, electron density, wave frequency and ion composition are explored.

We first define a baseline simulation that follows the simulation parameters introduced in Bortnik et al. (2008), with some variations discussed in the following:

Bortnik et al. (2008) studied the interactions of electrons with whistler-mode waves at L = 5, in a dipole geomagnetic field and in a multi-component plasma with equatorial electron density $n_{e,eq} = 10\,cm^{-3}$. They simulated a total of 24 electrons with equatorial pitch angle $\alpha_{eq} = 70^{\circ}$, energy $E = 168.3$ keV and equally spaced in wave-particle phase, in the range 0–360°, thus they had a resolution of 5° in initial wave-particle phase. The electrons started at an initial latitude of $\lambda = -9^{\circ}$ and were followed until they reached the magnetic equator. The wave packet used was static, monochromatic and one-sided, with a frequency of 2 kHz and wave normal angle of $\psi = 0^{\circ}$. Two wave amplitudes of 1pT and 1nT were investigated.

For our baseline simulation, we retain the latitudinal range, the equatorial electron density, the electron energy and the wave packet morphology, frequency and wave normal angle of Bortnik et al. (2008). We simulate the variation of the electron density along the magnetic field line using the (Denton et al., 2002) model. The ion composition is taken as $n_H = 0.77n_e$ for H^+ ions, $n_{He} = 0.20n_e$ for He^+ ions and $n_O = 0.03n_e$ for O^+ ions (Jordanova et al., 2008). As was derived from our simulations (not shown here, but the results are available at *WPIT_results* folder of WPIT repository), there is a need for higher resolution in the initial wave-particle phase, as there are nonlinear effects that can be missed if the wave-particle phase resolution is not high enough. From our simulations, during the evaluation phase of WPIT, we found that a minimum resolution of 3° is required in order to fully resolve non linear interactions. Hence, for our simulations we explore the behaviour of 120 electrons equally spaced in initial wave-particle angle η_0 in the range 0–360° thus achieving the wave-particle phase resolution threshold. The initial equatorial pitch angle is chosen at 68°. Finally, the amplitude of the y-component of the wave magnetic field is chosen at 65pT in the baseline simulation.

We start by presenting outputs of the *Environment_mod* and the *WaveProperties_mod* modules for the baseline simulation, in Section 3.1 and Section 3.2 respectively. We then present a sample run from the *LandauDamp_mod* module corresponding to the above conditions in Section 3.3. Finally, we proceed to exploring the onset of nonlinearity of wave-particle interactions by varying each parameter of the baseline simulation separately, in Section 3.4.

The corresponding Jupyter Notebooks for each simulation can be found in https://github.com/stourgai/WPIT/tree/main/WPIT_results.

3.1 Environment characterization module results

For the calculation of required local environment parameters we use the *WPIT/Environment_mod* module. In Figure 2 we present environmental parameters of the simulation as a function of the magnetic latitude. In Figure 2A, we present the magnetic dipole field strength as calculated by the routine *WPIT.Environment_mod.Bmag_dipole*, in Figure 2B, the electron and ion densities calculated by *WPIT.Environment_mod.density_FL_denton* with an equatorial electron density of $10cm^{-3}$, in Figure 2C relevant frequencies (i.e. wave, cyclotron, plasma, upper hybrid resonance and lower hybrid resonance frequencies). In the baseline simulation the wave frequency lies above the lower hybrid resonance frequency but well below the electron cyclotron frequency. In Figure 2D, we present the equatorial pitch angles corresponding to the maximum latitude that a particle can reach in a dipole magnetic field. Each graph spans magnetic latitudes in the range −45 to 45°, with the inset figures zooming-in in the region of interest for our simulations.

The code for the calculations can be found in *WPIT_results/Environment_Results* notebook.

3.2 Wave properties module results

With respect to the wave field of the simulation we simulate a static monochromatic parallel propagating whistler-mode wave with a frequency of 2 kHz (Bortnik et al., 2008). With the environmental parameters defined, we calculate the required wave properties for the definition of the whistler-mode wave, which include the Stix parameters, the refractive index, the wave number and the resonance cone angle, as described in Section 2.2. As examples,

in Figure 3A we present the Stix parameters along the magnetic latitude calculated by *WPIT. WaveProperties_mod.stix_parameters*. In Figure 3B we plot the refractive index surface of a 2 kHz wave at the equator, at L = 5. We calculate the surface based on both cold plasma theory and also by assigning finite temperatures to the electron and ion populations. The code for calculating the refractive index surface can be found in the relevant Jupyter notebook [*WPIT_results/Environment_And_Wave_Results*]. As also mentioned in Kulkarni et al. (2015), the inclusion of finite temperatures in electron and ion populations closes the refractive index surface, which remains open based on the cold plasma theory. Here, as examples of this behaviour, we calculated the refractive index surface for 1 eV electrons and also for 4 eV electrons 4 eV ions. In Figure 3C the resonance cone angle is plotted (*WPIT.WaveProperties_mod.refr_index_full*) and in Figure 3D the calculated electric and magnetic components of a 2 kHz parallel wave with $B_y^w = 100\,pT$ are shown. Here we have used the *WPIT. WaveProperties_mod.wave_packet_one_sided* for the definition of the wave packet along with *WPIT. WaveProperties_mod.wave_amplitudes_bell* for the calculation of the wave field components, to define a one-sided wave packet similarly to Bortnik et al. (2008). As the wave normal angle for this case is considered zero (i.e. parallel propagation) the x- and y-components of the wave electric and magnetic field are the same while the z-components are zero. This condition will change for oblique waves. Also, the electric field decreases as the wave reaches the equator. This behaviour is explained from the dispersion relation of the wave, which determines that the ratio of the electric wave field to the magnetic wave field decreases as the geomagnetic field strength decreases, hence for our case where the y-component of the wave magnetic field is fixed (Bell, 1984) the electric field component has its minimum at the equator.

3.3 Landau damping module results

We performed ray tracing simulations with the Stanford 3D Raytracer, for a ray of $f = 2$ kHz, injected at the magnetic equator ($\lambda = 0$) of $L = 5$ and with initial wave normal angle $\psi = 180^{\circ}$, i.e. anti-parallel to the ambient magnetic field. The raytracer output can be found in *Module_descriptions/example_rays* folder of WPIT repository. Figure 4 presents example outputs of *LandauDamp_mod* module. In Figure 4**A** we present different thermal electron distributions calculated with *WPIT. LandauDamp_mod.distribution_bell* (Bell et al., 2002) and *WPIT. LandauDamp_mod.distribution_bortnik* (Bortnik et al., 2007) routines. Bortnik et al. (2006) used a distribution of suprathermal electrons of $f_0 = 10 f_0^{Bell}$, where f_0^{Bell} is the Bell et al. (2002) distribution. This scaling was applied in order to account for fluxes outside of the plasmasphere as the Bell distribution was derived from measurements inside the plasmapause. Furthermore (Bortnik et al., 2007), investigated the effects of different scaling factors of the distribution on Landau damping. Following a similar analysis, we calculated Landau damping for three cases: (i) the original Bell distribution ("scale = 1"), (ii) a distribution multiplied by a factor of 5 (termed as "scale = 5" herein) and (iii) a distribution multiplied by a factor of ten (termed "scale = 10"). For each of these distributions we calculate the Landau damping with *WPIT. LandauDamp_mod.landau_damping* routine and the results are presented in Figure 4B. For the original Bell distribution, the ray attenuates at around 7 s. Scaling up the distribution 5 and 10 times leads to an attenuation time of 5 and 2.5 s respectively. While for the Bortnik distribution, the ray survives for only about 1 s. Thus, it becomes obvious that the choice of the thermal distribution is a crucial part of Landau damping calculations. Following on, in Figure 4C we present the resonant energy of electrons of pitch angles, in the range 0–90°, with the wave along the ray path [*WPIT.LandauDamp_mod.resonance_along_raypath*]. The colorscale is in log (E_{res}) with E_{res} the resonant energy in *keV*. It is mentioned that in this plot we calculate the resonant energy along the ray path without taking account the attenuation, thus the time axis ranges from 0 up to 30s, although after 7 s the wave would be damped. Finally, in Figure 4D we present the ray path with color coded the Landau damping [*WPIT.LandauDamp_mod.RayUtils_mod.ray_plots*] for the Bell distribution case. The yellow dot indicates the injection point of the ray and the black arrow indicates the initial wave normal angle.

3.4 Wave—particle interactions module results

Firstly, we explore the dependence of the onset of nonlinearity on the amplitude of the y-component of the wave magnetic field, as wave amplitude is the primary parameter in controlling the nonlinear behaviour of the particles, as also discussed in Bell (1986). There are two kinds of nonlinear effects that can arise: phase trapping and phase bunching. During phase trapping, particles follows closed trajectories in $v - \eta$ plane, around the center of a resonance island. In this case particles stay in resonance with the wave for a significant amount of time, which leads to large changes in pitch angle and energy. On the other hand, during phase bunching, particles follow open trajectories that enclose the resonance island. As the particles move in $v - \eta$ plane, they gradually approach the resonance island. This leads to some particles showing clustered trajectories (termed "bunching" of the trajectories); subsequently the particles cross to the other side of the island and then diverge away from it (see, e.g., Albert et al., 2012). In $\eta - \lambda$ plane, the trajectories of phase trapped particles experience several oscillations, which are confined to a limited range of wave-particle phases. On the other hand, trajectories of phase bunched particles are clustered, and span the entire range

(0–360°) in phase (see, e.g., Su et al., 2014). As mentioned above, we use Bell (1984) for the calculation of the wave components, which requires the amplitude of the y-component of the wave magnetic field to be defined first in order for the rest of the magnetic and electric field components to be determined. Thus in our simulations we consider a constant B_y^w component along the electron path, modulated only by the wave packet definition, as described above in Section 3.2 and shown in Figure 3D). In Figure 5 we present the trajectories of the 120 electrons for B_y^w = 10pT, 35pT, 65pT and 100pT. The line colors correspond to the initial wave-particle phase (η_0) of the electrons, from 0 to 360°. In the first row we plot the evolution of the equatorial pitch angle ($\Delta\alpha_{eq}$) along the magnetic latitude (λ). In the second row we plot the net change of the equatorial pitch angle as a function of the electrons' η_0. In the third row we plot the time derivative of η, $d\eta/dt$, along the magnetic latitude. In the last row we plot the first derivative of η, ν, normalised to the trapping frequency (ω_t), as a function of η.

In all cases, the electrons experience resonance with the waves, which can be seen from the plots on the third row, as the resonance condition is expressed as:

$$\frac{d\eta}{dt} = 0 \tag{46}$$

The resonance location for all four amplitudes is at $\lambda \approx -6.2°$. For the 10pT case the interactions are rather linear with small pitch angle scattering and the change in pitch angle follows an almost sinusoidal dependence on η_0, with $|\Delta\alpha_{eq}|_{\max} \approx 0.5°$. For all amplitudes from 35pT and higher phase trapping effects are observed. For the 35pT case, five out of the 120 electrons, with η_0 in the range 160–173°, experience phase trapping, following closed trajectories in the $\nu - \eta$ plane, and their $d\eta/dt$ oscillates around 0 throughout the simulation. As the amplitude increases to 65pT, the number of phase trapped electrons increases to 16, binned in four regions of η_0, approximately at 0–10, 156–167, 270–290 and 358–360°, whereas the untrapped electrons are almost symmetrically scattered to lower and higher equatorial pitch angles. The amplitude of the oscillation of their $d\eta/dt$ also becomes higher. Finally, at 100pT, the trajectories are similar, with 8 phase trapped electrons, but in this case the majority of the untrapped electrons are scattered to lower pitch angles. Three ranges of η_0 experience phase trapping in this case, approximately at 51–55, 158–168 and 238–245°. It is noted, that as the wave field becomes higher, $d\eta/dt$ oscillates around 0 with higher amplitude, which in turn leads to oscillations of the pitch angles of the trapped electrons around a mean value. Based also on simulations of other wave amplitudes (not shown herein), it is found that the lowest threshold for the appearance of nonlinear effects in wave-particle interactions is 35pT.

In addition to the above figures, examples of phase trapping and phase bunching are shown in Supplementary Figures S13-S15, where the trajectories of the electrons, for each wave amplitude, are plotted in $\nu - \eta$ and $\eta - \lambda$ planes. In Supplementary Figures S13 the trajectories due to interactions with a 1pT wave are presented. The wave amplitude in this case is to low to cause any nonlinear effects. In Supplementary Figures S14, the wave amplitude was raised to 10pT, which causes the onset of a weak phase bunching effect (indicated with black arrows). For the 35pT case, in Supplementary Figures S15, the wave amplitude is high enough for both phase trapping and phase bunching effects to occur.

In the second step of the parametric study, we explore the dependence of the onset of nonlinearity on the initial equatorial pitch angle. As the equatorial pitch angle defines the highest latitude that can be reached by a particle, we calculate the highest equatorial pitch angle that an electron located at λ = -9° can have. For L = 5 and assuming a dipole magnetic field, the latitudinal range per pitch angle is plotted in Figure 2D. For $\lambda = -9$, the maximum pitch angle is found to be around 71.2°, which is set as the upper limit of our investigation. The resonance condition is met where Equation 13 is zero, or

$$\frac{d\eta}{dt} = \frac{m\omega_H}{\gamma} - \omega - k_z\frac{u_\parallel}{m_e} = 0 \tag{47}$$

where $u_\parallel = u\cos\alpha$ with α the local pitch angle which in turn depends on α_{eq}. Thus the equatorial pitch angle defines the parallel velocity of the electron which in turn defines the location where the resonance condition is satisfied. This is evident in Figure 6, where the results for different initial α_{eq} are presented. As the pitch angle gets higher, the resonance condition is satisfied in progressively higher latitudes. In the case of α_{eq} = 62°, the electrons missed the resonance point (located below λ = -9), so the pitch angle change has a sinusoidal form with a maximum amplitude of around 0.25°. The case changes dramatically for α_{eq} = 63°: in this case the electrons start just below the resonance point, and half of the electrons are almost immediately phase trapped by the wave, reaching pitch angles up to 75°. At α_{eq} = 65° both phase trapping and moderate phase bunching effects can be observed, with two out of 120 electrons being phase trapped. We also contrast here the panels (I),(J),(K) and (L) of Figure 5, which is the baseline simulation for α_{eq} = 68°, with 18 electrons being phase trapped. Finally for the case of α_{eq} = 71°, again both phase bunching and phase trapping are present but with fewer electrons being phase trapped. Thus, for electrons with 168.3 keV energy starting at λ = -9° and interacting with a parallel whistler-mode wave of 2 kHz frequency, and with environmental parameters as defined above, the lowest threshold in terms of equatorial pitch angle for nonlinear effects to be observed is 63°.

As part of the third step of the parametric study, in Figure 7 we present wave-particle interaction results for electrons with energy in the range from 90 to 270 keV, while keeping the rest of the simulation parameters as in the baseline simulation presented above. We note that, as the energy becomes higher, the resonance point moves to higher latitudes. For the case of 90 keV, the resonance point is beyond the equator, and as the electrons are

simulated until they reach the equator, no resonance is observed to occur. The resonance point is within the latitudinal range of our simulations for the range around 100–270 keV. Another feature of the simulations is that, as the energy becomes higher, fewer electrons become phase trapped, but with higher pitch angle scattering: thus, for 140 keV electrons the maximum pitch angle change is around 5°, for 200 keV electrons around 8° and for 270 keV electrons around 11°. Also, the oscillation of the pitch angle of trapped electrons becomes smaller with higher energy.

Under the fourth step of the parametric study, we explore how the wave normal angle affects the nonlinear behaviour of the electrons. In all the simulations presented above, the waves have been considered to be parallel (i.e. $\psi = 0$ deg). The wave normal angle affects the resonance condition (Eq 47) through the parallel wave number $k_{\parallel} = k \cos \psi$. The upper limit of the wave normal angle for our simulations is defined by the resonance cone angle. Outside of the resonance cone no wave modes propagate and as ψ approaches the resonance cone angle, the wave number goes to infinity, or equivalently the wavelength goes to zero. By using *WPIT.WaveProperties_mod.res_angle* we calculate the resonance angle θ_{res} along the magnetic latitude and the results are presented in Figure 3C. For our latitudinal range of interest (−9 to 0°), θ_{res} ranges from around 73–75°. In order to ensure that the wave normal angle is inside the resonance cone for all the latitudes of interest, we set the upper limit of the wave normal angle at 72°. In Figure 8 we present the results for $\psi = 1^o, 50^o, 60^o, 72^o$. For the case of $\psi = 1^o$, the results are similar to the baseline case with some electrons experiencing phase trapping and some experiencing weak phase bunching. As the wave becomes more oblique, the nonlinear effects begin to decline. Thus, for $\psi = 50°$ only three electrons are phase trapped and for $\psi = 60°$ only weak phase bunching occurs. As the wave normal angle approaches the resonance cone angle ($\psi = 72^o$) the interactions become linear. Thus for the parameters of our simulation, the interactions become progressively more linear as the wave becomes more oblique.

In the baseline simulation we used an equatorial electron density of $10 \times 10^6\ m^{-3}$ based on Bortnik et al. (2008). In Figure 9 we present results for interactions under different equatorial electron densities. Equatorial electron density affects the location of the resonance point, in the sense that as the electron density is increased, the resonance point moves to lower latitudes. For the case of $n_{eq} = 25 \times 10^6\ m^{-3}$, the electrons start at the resonance point as can be seen in Figure 9C, on the other hand for densities $n_{eq} < 5 \times 10^6\ m^{-3}$ the resonance point is located higher than the equator. In the case of $n_{eq} = 25 \times 10^6\ m^{-3}$ only strong phase bunching has occurred with none of the electrons trapped. If we compare the cases of $n_{eq} = 12 \times 10^6\ m^{-3}$, $n_{eq} = 7 \times 10^6\ m^{-3}$ and the baseline simulation ($n_{eq} = 10 \times 10^6\ m^{-3}$), we conclude that as the electron density becomes smaller more electrons become phase trapped, though with progressively lower maximum pitch angle scattering.

In Figure 10 we present results of the interactions with a wave of frequency in the range 1.4–3 kHz. For the 1.4 kHz case, the resonance point is located beyond the equator so the electrons did not reach it. On the other hand for the 3 kHz case the resonance point is located below $\lambda = -9°$ and again is missed by the electrons. Hence, it is concluded that the higher the frequency, the lower the location of the resonance point. As it can be inferred from our simulations, the wave field that could potentially drive nonlinear scattering, for the parameters defined in our simulation, has a rather wide bandwidth of 1.5 kHz (1.5–3 kHz).

We have also explore the effect that variations in plasma composition have on the nonlinear behaviour of the electrons. The results are not presented here but can be found in *WPIT_Results* folder of the WPIT repository. No large discrepancies between the runs with different ion compositions were found, with the only difference consisting of slight shifts in η_0 of the phase-trapped populations.

4 Discussion

We have presented WPIT, an open source, Python-based toolset for the investigation of interactions of charged particles with very low frequency waves in Earth's magnetosphere. WPIT is the first step, aiming to provide a unified code for the exploration of wave particle interactions inside the magnetosphere. With the ever-increasing interest of the science community on wave particles interactions, WPIT comprises a useful tool for both theoretical analyses and quantitative assessments of wave-particle interaction processes.

In Section 3, we presented results of the use of the code, by simulating the interactions of electrons with whistler-mode waves. We performed a parametric study by adjusting parameters such as the amplitude of the wave magnetic field, the equatorial pitch angle, the electron energy, the wave normal angle, the equatorial electron density, the wave frequency, and the electron and ion composition, and we examined the sensitivity of the onset of nonlinear effects for each case.

In terms of the amplitude of the wave magnetic field, we find that, in the simulations performed, there is a lower threshold in wave amplitude for the appearance of nonlinearity in wave-particle interactions. WPIT enables the identification of this threshold, based also on the entire parameter space that is explored. The pitch angle was also found to greatly affect the resonance location, and also whether electrons will be phase trapped. We note that an expanded parametric study should include a wider range of initial latitudes for the electrons than was presented herein, so that pitch angle effects can be better quantified. WPIT enables such parametric studies to be pursued. In terms of the effect of electron energy, it is found that the resonance location in terms of latitude generally decreases as the electron energy increases. Extended

parametric studies could include such an extended range of initial latitudes for the electrons.

With this study we presented the capabilities of WPIT modules for extensive analysis of wave particle interactions. A potential expansion of the parametric study presented herein would be to conduct extensive simulations for the characterisation of the dependence of nonlinear effects on each wave and particle characteristic. Another important aspect that we conclude from our simulations is the need for higher resolution wave-particle phase distributions for the investigation of nonlinear wave particle interactions. With the simulation code available in Jupyter Notebook format in WPIT repository, a user can be guided through the set up of the relevant simulations for the performance of such more detailed parametric studies.

The goal of WPIT is to provide an open-source toolset for wave-particle interaction simulations to the scientific community, and it is expected and envisioned that other researchers will be contributing extensively to future versions, greatly enhancing the functionalities and capabilities of WPIT. However, we note that, since the implementation of WPIT involves a long sequence of processes as described above, some of which involve complex equations that require extensive testing and proper application of the assumptions used, it is preferred that any potential new functionalities and additions to WPIT by other researchers are implemented after communication with the authors.

Further to the scientific analysis of wave-particle interactions that was presented, and related to the quantification of the effectiveness of waves to resonantly scatter electrons, a practical application of WPIT is related to investigations of optimal schemes for radiation belt remediation. This refers to the active removal of energetic electrons from the radiation belts for the protection of satellite systems: Energetic electrons originating either from the solar wind or from high-altitude nuclear explosions can severely damage satellites, particularly in Low-Earth Orbit (LEO) (see, e.g., Carlsten et al., 2019, and references therein). With the emergence of NewSpace mega-constellations (Zhang et al., 2022) and the shift of space usage to LEO satellites for a range of applications, naturally or artificially produced energetic electrons constitute a serious potential vulnerability. In order to provide means for addressing this vulnerability, various schemes have been proposed that can potentially remove trapped electrons from the radiation belts (e.g. Hoyt and Minor, 2005; Sauvaud et al., 2008; Shao et al., 2009; de Soria-Santacruz and Martinez-Sanchez, 2013). Amongst the most promising schemes is the use of Very Low Frequency (VLF) waves to scatter electrons and lead them to precipitate into the Earth's atmosphere. WPIT provides an optimal simulation tool that enables quantifying the wave characteristics that can lead to particle scattering, as well as evaluating the effectiveness of the scattering mechanism. To this direction, future work on WPIT involves including a module for the calculation of *in situ* transmissions of VLF waves by user-defined antenna characteristics. With the inclusion of a module that calculates the near and far field of antennas in magnetospheric plasmas, and along with calculation of the generated wave field through ray tracing, WPIT can become a useful tool, not only for theoretical studies, but also for space missions targeting wave particle interactions, as well as for VLF transmitters immersed in magnetospheric plasmas and for investigations of the efficiency of radiation belt remediation schemes.

Data availability statement

The datasets presented in this study can be accessed from the WPIT online repository through the link: https://github.com/stourgai/WPIT.

Author contributions

ST wrote the WPIT code and performed the scientific analysis presented in the manuscript. TS contributed to conception and design of the study. ST and TS contributed to the scientific analysis of the simulation results. ST wrote the first draft of the manuscript. ST and TS contributed to manuscript revision, read, and approved the submitted version.

Funding

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the HFRI PhD Fellowship grant (Fellowship Number: 400), and DUTH projects KE82372 and KE82875.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fspas.2022.1005598/full#supplementary-material.

SUPPLEMENTARY FIGURE S1
Evolution of electron's equatorial pitch angle α_{eq} along magnetic latitude λ. Reproduction of Figure 2i of Bortnik et al. (2008) with WPIT's "Nonlinear interaction of energetic electrons with large amplitude chorus.ipynb".

SUPPLEMENTARY FIGURE S2
Evolution of electron energy along magnetic latitude λ. Reproduction of Figure 2j of Bortnik et al. (2008) with WPIT's "Nonlinear interaction of energetic electrons with large amplitude chorus.ipynb".

SUPPLEMENTARY FIGURE S3
Distribution of total pitch angle change $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0. Reproduction of Figure 2k of Bortnik et al. (2008) with "WPIT's Nonlinear interaction of energetic electrons with large amplitude chorus.ipynb".

SUPPLEMENTARY FIGURE S4
Evolution of electron's equatorial pitch angle α_{eq} along magnetic latitude λ. Reproduction of Figure 4a of Albert and Bortnik (2009) with WPIT's "Nonlinear interaction of radiation belt electrons with electromagnetic ion cyclotron waves.ipynb".

SUPPLEMENTARY FIGURE S5
Evolution of electron's energy along magnetic latitude λ. Reproduction of Figure 4b of Albert and Bortnik (2009) with WPIT's "Nonlinear interaction of radiation belt electrons with electromagnetic ion cyclotron waves.ipynb".

SUPPLEMENTARY FIGURE S6
Distribution of total pitch angle change $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0. Reproduction of Figure 4c of Albert and Bortnik (2009) with WPIT's "Nonlinear interaction of radiation belt electrons with electromagnetic ion cyclotron waves.ipynb".

SUPPLEMENTARY FIGURE S7
Evolution of electron's equatorial pitch angle α_{eq} along magnetic latitude λ. Reproduction of Figure 12a of Su et al. (2012) with WPIT's "Bounce-averaged advection and diffusion coefficients formonochromatic electromagnetic ion cyclotron wave Comparison between test-particle and quasi-linear models.ipynb".

SUPPLEMENTARY FIGURE S8
Distribution of total pitch angle change $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0. Reproduction of Figure 12b of Su et al. (2012) with WPIT's "Bounce-averaged advection and diffusion coefficients for monochromatic electromagnetic ion cyclotron wave Comparison between test-particle and quasi-linear models.ipynb".

SUPPLEMENTARY FIGURE S9
Evolution of ion's equatorial pitch angle α_{eq} along magnetic latitude λ. Reproduction of Figure 3f of Su et al. (2014) with WPIT's "Latitudinal dependence of nonlinear interaction between electromagnetic ion cyclotron wave and terrestrial ring current ions.ipynb".

SUPPLEMENTARY FIGURE S10
Evolution of ion energy along magnetic latitude λ. Reproduction of Figure 3g of Su et al. (2014) with WPIT's "Latitudinal dependence of nonlinear interaction between electromagnetic ion cyclotron wave and terrestrial ring current ions.ipynb".

SUPPLEMENTARY FIGURE S11
Nonlinear parameter S as a function of magnetic latitude λ. Reproduction of Figure 3i of Su et al. (2014) with WPIT's "Latitudinal dependence of nonlinear interaction between electromagnetic ion cyclotron wave and terrestrial ring current ions.ipynb".

SUPPLEMENTARY FIGURE S12
Distribution of total pitch angle change $\Delta\alpha_{eq}$ with respect to the initial wave-particle phase η_0. Reproduction of Figure 4c of Su et al. (2014) with WPIT's "Latitudinal dependence of nonlinear interaction between electromagnetic ion cyclotron wave and terrestrial ring current ions.ipynb".

SUPPLEMENTARY FIGURE S13
Electron trajectories in ν-η and η-λ planes for interactions with 1pT wave. The interactions are linear (no phase trapping or phase bunching).

SUPPLEMENTARY FIGURE S14
Electron trajectories in ν-η and η-λ planes for interactions with 10pT wave. Weak phase bunching is present (black arrows).

SUPPLEMENTARY FIGURE S15
Electron trajectories in ν-η and η-λ planes for interactions with 35pT wave. Both phase trapping and phase bunching are present (black arrows).

References

Albert, J., and Bortnik, J. (2009). Nonlinear interaction of radiation belt electrons with electromagnetic ion cyclotron waves. *Geophys. Res. Lett.* 36, L12110. doi:10.1029/2009gl038904

Albert, J. (2005). Evaluation of quasi-linear diffusion coefficients for whistler mode waves in a plasma with arbitrary density ratio. *J. Geophys. Res.* 110, A03218. doi:10.1029/2004ja010844

Albert, J. M., Tao, X., and Bortnik, J. (2012). "Aspects of nonlinear wave-particle interactions," in *Dynamics of the Earth's radiation belts and inner magnetosphere* (Washington: AGU), 199, 255–264.

Allanson, O., Watt, C., Ratcliffe, H., Allison, H. J., Meredith, N., Bentley, S., et al. (2020). Particle-in-cell experiments examine electron diffusion by whistler-mode waves: 2. Quasi-linear and nonlinear dynamics. *J. Geophys. Res. Space Phys.* 125, e2020JA027949. doi:10.1029/2020ja027949

Allanson, O., Watt, C., Ratcliffe, H., Meredith, N., Allison, H. J., Bentley, S., et al. (2019). Particle-in-cell experiments examine electron diffusion by whistler-mode waves: 1. Benchmarking with a cold plasma. *J. Geophys. Res. Space Phys.* 124, 8893–8912. doi:10.1029/2019ja027088

Appleton, E. V. (1932). Wireless studies of the ionosphere. *Proc. Inst. Electr. Eng.* 7, 257–265. doi:10.1049/pws.1932.0027

Artemyev, A., Neishtadt, A., and Vasiliev, A. (2020). Mapping for nonlinear electron interaction with whistler-mode waves. *Phys. Plasmas* 27, 042902. doi:10.1063/1.5144477

Artemyev, A., Vasiliev, A., Mourenas, D., Agapitov, O., and Krasnoselskikh, V. (2013). Nonlinear electron acceleration by oblique whistler waves: Landau resonance vs. cyclotron resonance. *Phys. Plasmas* 20, 122901. doi:10.1063/1.4836595

Aubry, M., Bitoun, J., and Graff, P. (1970). Propagation and group velocity in a warm magnetoplasma. *Radio Sci.* 5, 635–645. doi:10.1029/rs005i003p00635

Bell, T. F. (1984). The nonlinear gyroresonance interaction between energetic electrons and coherent vlf waves propagating at an arbitrary angle with respect to the Earth's magnetic field. *J. Geophys. Res.* 89, 905–918. doi:10.1029/ja089ia02p00905

Bell, T. F. (1986). The wave magnetic field amplitude threshold for nonlinear trapping of energetic gyroresonant and landau resonant electrons by nonducted vlf waves in the magnetosphere. *J. Geophys. Res.* 91, 4365–4379. doi:10.1029/ja091ia04p04365

Bell, T., Inan, U., Bortnik, J., and Scudder, J. (2002). The landau damping of magnetospherically reflected whistlers within the plasmasphere. *Geophys. Res. Lett.* 29, 23-1–23-4. doi:10.1029/2002gl014752

Bortnik, J., Inan, U. S., and Bell, T. F. (2003b). Energy distribution and lifetime of magnetospherically reflecting whistlers in the plasmasphere. *J. Geophys. Res.* 108 (A5), 1199. doi:10.1029/2002ja009316

Bortnik, J., Inan, U. S., and Bell, T. F. (2006). Landau damping and resultant unidirectional propagation of chorus waves. *Geophys. Res. Lett.* 33, L03102. doi:10.1029/2005gl024553

Bortnik, J. (2004). *Precipitation of radiation belt electrons by lightning-generated magnetospherically reflecting whistler waves*. Stanford, California: stanford university.

Bortnik, J., Thorne, R., and Inan, U. S. (2008). Nonlinear interaction of energetic electrons with large amplitude chorus. *Geophys. Res. Lett.* 35, L21102. doi:10.1029/2008gl035500

Bortnik, J., Thorne, R. M., and Meredith, N. (2007). Modeling the propagation characteristics of chorus using crres suprathermal electron fluxes. *J. Geophys. Res.* 112, A08204. doi:10.1029/2006ja012237

Bortnik, J., Thorne, R., Ni, B., and Li, J. (2015). Analytical approximation of transit time scattering due to magnetosonic waves. *Geophys. Res. Lett.* 42, 1318–1325. doi:10.1002/2014gl062710

Brinca, A. (1972). On the stability of obliquely propagating whistlers. *J. Geophys. Res.* 77, 3495–3507. doi:10.1029/ja077i019p03495

Carlsten, B. E., Colestock, P. L., Cunningham, G. S., Delzanno, G. L., Dors, E. E., Holloway, M. A., et al. (2019). Radiation-belt remediation using space-based antennas and electron beams. *IEEE Trans. Plasma Sci. IEEE Nucl. Plasma Sci. Soc.* 47, 2045–2063. doi:10.1109/tps.2019.2910829

Carpenter, D., and Anderson, R. (1992). An isee/whistler model of equatorial electron density in the magnetosphere. *J. Geophys. Res.* 97, 1097–1108. doi:10.1029/91ja01548

Chang, S., Ni, B., Bortnik, J., Zhou, C., Zhao, Z., Li, J., et al. (2014). Resonant scattering of energetic electrons in the plasmasphere by monotonic whistler-mode waves artificially generated by ionospheric modification. *Ann. Geophys.* 32, 507–518. doi:10.5194/angeo-32-507-2014

Chen, L., Bortnik, J., Thorne, R. M., Horne, R. B., and Jordanova, V. K. (2009). Three-dimensional ray tracing of vlf waves in a magnetospheric environment containing a plasmaspheric plume. *Geophys. Res. Lett.* 36, L22101. doi:10.1029/2009gl040451

de Soria-Santacruz, M., and Martinez-Sanchez, M. (2013). Electromagnetic ion cyclotron waves for radiation belt remediation applications. *IEEE Trans. Plasma Sci. IEEE Nucl. Plasma Sci. Soc.* 41, 3329–3337. doi:10.1109/tps.2013.2260181

Denton, R., Goldstein, J., and Menietti, J. (2002). Field line dependence of magnetospheric electron density. *Geophys. Res. Lett.* 29, 58-1–58-4. doi:10.1029/2002gl015963

Fu, S., Yi, J., Ni, B., Zhou, R., Hu, Z., Cao, X., et al. (2020). Combined scattering of radiation belt electrons by low-frequency hiss: Cyclotron, landau, and bounce resonances. *Geophys. Res. Lett.* 47, e2020GL086963. doi:10.1029/2020gl086963

Gan, L., Li, W., Ma, Q., Albert, J., Artemyev, A., and Bortnik, J. (2020). Nonlinear interactions between radiation belt electrons and chorus waves: Dependence on wave amplitude modulation. *Geophys. Res. Lett.* 47, e2019GL085987. doi:10.1029/2019gl085987

Gao, Z., Zhu, H., Zhang, L., Zhou, Q., Yang, C., and Xiao, F. (2014). Test particle simulations of interaction between monochromatic chorus waves and radiation belt relativistic electrons. *Astrophys. Space Sci.* 351, 427–434. doi:10.1007/s10509-014-1859-1

Ginet, G. P., and Albert, J. M. (1991). Test particle motion in the cyclotron resonance regime. *Phys. Fluids B Plasma Phys.* 3, 2994–3012. doi:10.1063/1.859778

Golden, D., Spasojevic, M., Foust, F., Lehtinen, N., Meredith, N. P., and Inan, U. (2010). Role of the plasmapause in dictating the ground accessibility of elf/vlf chorus. *J. Geophys. Res.* 115. doi:10.1029/2010ja015955

Horne, R. B., and Thorne, R. M. (1993). On the preferred source location for the convective amplification of ion cyclotron waves. *J. Geophys. Res.* 98, 9233–9247. doi:10.1029/92ja02972

Horne, R. B., and Thorne, R. M. (1998). Potential waves for relativistic electron scattering and stochastic acceleration during magnetic storms. *Geophys. Res. Lett.* 25, 3011–3014. doi:10.1029/98gl01002

Horne, R. B. (2015). Trapping and acceleration of upflowing ionospheric electrons in the magnetosphere by electrostatic electron cyclotron harmonic waves. *Geophys. Res. Lett.* 42, 975–980. doi:10.1002/2014gl062406

Hoyt, R. P., and Minor, B. M. (2005). "Remediation of radiation belts using electrostatic tether structures," in 2005 IEEE Aerospace Conference, Big Sky, MT, USA, 05-12 March 2005 (Piscataway, NJ: IEEE), 583–594.

Huang, H., Gao, X.-T., Wang, X.-G., and Wang, Z.-B. (2017). Electron acceleration and diffusion in the gyrophase space by low-frequency electromagnetic waves. *IEEE Trans. Plasma Sci. IEEE Nucl. Plasma Sci. Soc.* 46, 225–229. doi:10.1109/tps.2017.2771277

Inan, U. S. (1987). Gyroresonant pitch angle scattering by coherent and incoherent whistler mode waves in the magnetosphere. *J. Geophys. Res.* 92, 127–142. doi:10.1029/ja092ia01p00127

Inan, U. S., and Tkalcevic, S. (1982). Nonlinear equations of motion for landau resonance interactions with a whistler mode wave. *J. Geophys. Res.* 87, 2363–2367. doi:10.1029/ja087ia04p02363

Jasna, D., Inan, U. S., and Bell, T. F. (1992). Precipitation of suprathermal (100 ev) electrons by oblique whistler waves. *Geophys. Res. Lett.* 19, 1639–1642. doi:10.1029/92gl01811

Jordanova, V., Albert, K. J., and Miyoshi, Y. (2008). Relativistic electron precipitation by emic waves from self-consistent global simulations. *J. Geophys. Res.* 113, A00A10. doi:10.1029/2008ja013239

Karpman, V., Istomin, J. N., and Shklyar, D. (1975). Effects of nonlinear interaction of monochromatic waves with resonant particles in the inhomogeneous plasma. *Phys. Scr.* 11, 278–284. doi:10.1088/0031-8949/11/5/008

Karpman, V., and Shklyar, D. (1977). Partcle precipitation caused by a single whistler-mode wave injected into the magnetosphere. *Planet. Space Sci.* 25, 395–403. doi:10.1016/0032-0633(77)90055-1

Kennel, C. (1966). Low-frequency whistler mode. *Phys. Fluids* 9, 2190–2202. doi:10.1063/1.1761588

Kimura, I. (1966). Effects of ions on whistler-mode ray tracing. *Radio Sci.* 1, 269–284. doi:10.1002/rds196613269

Kivelson, M. G., Kivelson, M. G., and Russell, C. T. (1995). *Introduction to space physics*. Cambridge, United Kingdom: Cambridge University Press.

Kivelson, M. (2005). Transport and acceleration of plasma in the magnetospheres of Earth and jupiter and expectations for saturn. *Adv. Space Res.* 36, 2077–2089. doi:10.1016/j.asr.2005.05.104

Koskinen, H. E., and Kilpua, E. K. (2022). "Particle source and loss processes," in *Physics of Earth's radiation belts* (Cham, Switzerland: Springer), 159–211.

Kulkarni, P., Gołkowski, M., Inan, U., and Bell, T. (2015). The effect of electron and ion temperature on the refractive index surface of 1–10 khz whistler mode waves in the inner magnetosphere. *J. Geophys. Res. Space Phys.* 120, 581–591. doi:10.1002/2014ja020669

Kulkarni, P., Inan, U. S., and Bell, T. F. (2008). Energetic electron precipitation induced by space based vlf transmitters. *J. Geophys. Res.* 113. doi:10.1029/2008ja013120

Kulkarni, P., Inan, U. S., and Bell, T. F. (2006). Whistler mode illumination of the plasmaspheric resonant cavity via *in situ* injection of elf/vlf waves. *J. Geophys. Res.* 111, A10215. doi:10.1029/2006ja011654

Lauben, D. S., Inan, U. S., and Bell, T. F. (2001). Precipitation of radiation belt electrons induced by obliquely propagating lightning-generated whistlers. *J. Geophys. Res.* 106, 29745–29770. doi:10.1029/1999ja000155

Laval, G., and Pellat, R. (1970). Particle acceleration by electrostatic waves propagating in an inhomogeneous plasma. *J. Geophys. Res.* 75, 3255–3256. doi:10.1029/ja075i016p03255

Lee, D.-Y., Kim, K.-C., and Choi, C.-R. (2020). Nonlinear scattering of 90 pitch angle electrons in the outer radiation belt by large-amplitude emic waves. *Geophys. Res. Lett.* 47, e2019GL086738. doi:10.1029/2019gl086738

Li, J., Bortnik, J., Xie, L., Pu, Z., Chen, L., Ni, B., et al. (2015). Comparison of formulas for resonant interactions between energetic electrons and oblique whistler-mode waves. *Phys. Plasmas* 22, 052902. doi:10.1063/1.4914852

Liu, K., Winske, D., Gary, S. P., and Reeves, G. D. (2012). Relativistic electron scattering by large amplitude electromagnetic ion cyclotron waves: The role of phase bunching and trapping. *J. Geophys. Res.* 117. doi:10.1029/2011ja017476

Maxworth, A., and Gołkowski, M. (2017). Magnetospheric whistler mode ray tracing in a warm background plasma with finite electron and ion temperature. *J. Geophys. Res. Space Phys.* 122, 7323–7335. doi:10.1002/2016ja023546

Maxworth, A., Gołkowski, M., Malaspina, D., and Jaynes, A. (2020). Raytracing study of source regions of whistler mode wave power distribution relative to the plasmapause. *J. Geophys. Res. Space Phys.* 125, e2019JA027154. doi:10.1029/2019ja027154

Nunn, D. (1974). A self-consistent theory of triggered vlf emissions. *Planet. Space Sci.* 22, 349–378. doi:10.1016/0032-0633(74)90070-1

Nunn, D., and Omura, Y. (2015). A computational and theoretical investigation of nonlinear wave-particle interactions in oblique whistlers. *J. Geophys. Res. Space Phys.* 120, 2890–2911. doi:10.1002/2014JA020898

Ozhogin, P., Tu, J., Song, P., and Reinisch, B. (2012). Field-aligned distribution of the plasmaspheric electron density: An empirical model derived from the image rpi measurements. *J. Geophys. Res.* 117. doi:10.1029/2011ja017330

Öztürk, M. K. (2012). Trajectories of charged particles trapped in Earth's magnetic field. *Am. J. Phys.* 80, 420–428. doi:10.1119/1.3684537

Parks, G. K. (1991). *Physics of space plasmas. An introduction*. Redwood City, CA: Addison-Wesley Publishing Co..

Sauvaud, J.-A., Maggiolo, R., Jacquey, C., Parrot, M., Berthelier, J.-J., Gamble, R., et al. (2008). Radiation belt electron precipitation due to vlf transmitters: Satellite observations. *Geophys. Res. Lett.* 35, L09101. doi:10.1029/2008gl033194

Schulz, M., and Lanzerotti, L. J. (2012). *Particle diffusion in the radiation belts*, 7. Berlin, Germany: Springer Science & Business Media.

Shao, X., Papadopoulos, K., and Sharma, A. (2009). Control of the energetic proton flux in the inner radiation belt by artificial means. *J. Geophys. Res.* 114, A07214. doi:10.1029/2009ja014066

Sheeley, B., Moldwin, M., Rassoul, H., and Anderson, R. (2001). An empirical plasmasphere and trough density model: Crres observations. *J. Geophys. Res.* 106, 25631–25641. doi:10.1029/2000JA000286

Shklyar, D., and Matsumoto, H. (2009). Oblique whistler-mode waves in the inhomogeneous magnetospheric plasma: Resonant interactions with energetic charged particles. *Surv. Geophys.* 30, 55–104. doi:10.1007/s10712-009-9061-7

Shprits, Y. Y. (2009). Potential waves for pitch-angle scattering of near-equatorially mirroring energetic electrons due to the violation of the second adiabatic invariant. *Geophys. Res. Lett.* 36, L12106. doi:10.1029/2009gl038322

Sousa, A. P. (2018). *Global and seasonal effects of lightning-induced electron precipitation*. Stanford, California: Stanford University.

Stix, T. H. (1992). *Waves in plasmas*. New York: Springer Science & Business Media.

Su, Z., Zhu, H., Xiao, F., Zheng, H., Shen, C., Wang, Y., et al. (2012). Bounce-averaged advection and diffusion coefficients for monochromatic electromagnetic ion cyclotron wave: Comparison between test-particle and quasi-linear models. *J. Geophys. Res.* 117, A09222. doi:10.1029/2012ja017917

Su, Z., Zhu, H., Xiao, F., Zheng, H., Shen, C., Wang, Y., et al. (2013). Latitudinal dependence of nonlinear interaction between electromagnetic ion cyclotron wave and radiation belt relativistic electrons. *J. Geophys. Res. Space Phys.* 118, 3188–3202. doi:10.1002/jgra.50289

Su, Z., Zhu, H., Xiao, F., Zheng, H., Zhang, M., Liu, Y.-M., et al. (2014). Latitudinal dependence of nonlinear interaction between electromagnetic ion cyclotron wave and terrestrial ring current ions. *Phys. Plasmas* 21, 052310. doi:10.1063/1.4880036

Summers, D. (2005). Quasi-linear diffusion coefficients for field-aligned electromagnetic waves with applications to the magnetosphere. *J. Geophys. Res.* 110, A08213. doi:10.1029/2005ja011159

Summers, D., and Thorne, R. M. (2003). Relativistic electron pitch-angle scattering by electromagnetic ion cyclotron waves during geomagnetic storms. *J. Geophys. Res.* 108, 1143. doi:10.1029/2002ja009489

Swanson, D. G. (2012). *Plasma waves*. New York: Elsevier.

Tao, X., Bortnik, J., Albert, J. M., and Thorne, R. M. (2012). Comparison of bounce-averaged quasi-linear diffusion coefficients for parallel propagating whistler mode waves with test particle simulations. *J. Geophys. Res.* 117, A10205. doi:10.1029/2012ja017931

Tao, X., and Bortnik, J. (2010). Nonlinear interactions between relativistic radiation belt electrons and oblique whistler mode waves. *Nonlinear process. geophys.* 17, 599–604. doi:10.5194/npg-17-599-2010

Vainchtein, D., Zhang, X.-J., Artemyev, A., Mourenas, D., Angelopoulos, V., and Thorne, R. (2018). Evolution of electron distribution driven by nonlinear resonances with intense field-aligned chorus waves. *JGR. Space Phys.* 123, 8149–8169. doi:10.1029/2018ja025654

Wang, B., Li, P., Huang, J., and Zhang, B. (2019). Nonlinear landau resonance between emic waves and cold electrons in the inner magnetosphere. *Phys. Plasmas* 26, 042903. doi:10.1063/1.5088374

Wang, B., Su, Z., Zhang, Y., Shi, S., and Wang, G. (2016). Nonlinear landau resonant scattering of near equatorially mirroring radiation belt electrons by oblique emic waves. *Geophys. Res. Lett.* 43, 3628–3636. doi:10.1002/2016gl068467

Zhang, J., Cai, Y., Xue, C., Xue, Z., and Cai, H. (2022). Leo mega constellations: Review of development, impact, surveillance, and governance. *Space Sci. Technol.* 2022, 9865174. doi:10.34133/2022/9865174

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Original Research
PUBLISHED 31 October 2022
DOI 10.3389/fspas.2022.1002697

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Gabriele Pierantoni,
University of Westminster,
United Kingdom
Baptiste Cecconi,
UMR8109 Laboratoire d'études spatiales et d'instrumentation en astrophysique (LESIA), France

*CORRESPONDENCE
A. R. A. Smith,
ashley.smith@ed.ac.uk

†The Swarm Data,
Innovation, and Science Cluster

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 25 July 2022
ACCEPTED 03 October 2022
PUBLISHED 31 October 2022

CITATION
Smith ARA and Pačes M (2022), Python tools for ESA's Swarm mission: VirES for Swarm and surrounding ecosystem. *Front. Astron. Space Sci.* 9:1002697. doi: 10.3389/fspas.2022.1002697

COPYRIGHT
© 2022 Smith and Pačes. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Python tools for ESA's Swarm mission: VirES for Swarm and surrounding ecosystem

A. R. A. Smith [1]* M. Pačes[2] and Swarm DISC†

[1]School of GeoSciences, University of Edinburgh, Edinburgh, United Kingdom, [2]EOX IT Services GmbH, Vienna, Austria

ESA's Swarm mission is a constellation probing both Earth's interior and geospace, delivering magnetic and plasma measurements which are used to generate many derived data products. From empirical magnetic field models of the core, crust, ionosphere, and magnetosphere, to multi-point estimates of ionospheric currents and *in-situ* plasma properties, these are challenging to navigate, process, and visualize. The *VirES for Swarm* platform (https://vires.services) has been built to tackle this problem, providing tools to increase usability of Swarm data products. The VirES (Virtual environments for Earth Scientists) platform provides both a graphical web interface and an API to access and visualise Swarm data and models. This is extended with a cloud-hosted development environment powered by JupyterHub (the "Virtual Research Environment/VRE"). VirES provides two API's: the full VirES API for which a dedicated Python client is provided, *viresclient*, and the more interoperable Heliophysics API (HAPI). The VRE is furnished with a bespoke Python environment containing thematic libraries supporting science with Swarm. This service aims to ease the pathway for scientists writing computer code to analyze Swarm data products, increase opportunities for collaboration, and leverage cloud technologies. Beyond simply providing data and model access to Python users, it is extremely helpful to provide higher-level analysis and visualization tools, and ready-to-use code recipes that people can explore and extend. Critically for space physics, this involves crossover with many other datasets and so it is highly valuable to embed such tools within the wider data and software ecosystems. Through Swarm DISC (Data, Innovation, and Science Cluster), we are tackling this through cookbooks and Python libraries. Cookbooks are built and presented using Jupyter technologies, and tested to work within the VRE. A new library we are building is *SwarmPAL*, which includes tools for time-frequency analysis and inversion of magnetic field measurements for electric current systems, among others, while relying on the VirES server to provide data portability and other utilities. This paper reviews the current state of these tools and services for Swarm, particularly in the context of the Python in Heliophysics Community, and the wider heliophysics and geospace data environment.

KEYWORDS

swarm, python, space weather, geospace, geomagnetism

1 Introduction

Swarm is a European Space Agency (ESA) spacecraft mission in Low Earth Orbit probing the electromagnetic environment around and within Earth[1]. Researchers making use of Swarm data products can face many challenges. These range from discovering, accessing, and comprehending an appropriate dataset for their research question, to forwards evaluation of various geomagnetic field models, to combining their analysis with external data sources, and bringing other algorithms and visualizations to the data. With the large and growing Python software ecosystem, it is of course crucial to place the Swarm data environment in close proximity to this. Considering the ever-increasing wealth of data sources (and volumes thereof), and potential tools to leverage such coming often from well beyond the domain of space sciences, while software systems are becoming increasingly complex, it is important to build sustainable solutions that can accommodate this complexity.

In this paper we outline the projects bringing Swarm products, tools and services to the community, with a focus on the Python ecosystem. These efforts are being carried out by EOX IT Services[2] under contract to ESA, and by Swarm DISC (Data, Innovation, and Science Cluster) which is a consortium of institutions involved in Swarm data services. In the first part of the paper, we introduce the Swarm mission and its data products, and the *VirES for Swarm* service which provides convenient access to them. In the second part, we review the elements which connect to the Python ecosystem: the *viresclient* and *SwarmPAL* packages which provide access to the data and algorithms to process the data; the JupyterLab-based *Virtual Research Environment (VRE)*: an easy way for researchers to immediately run code in the cloud; *Swarm Notebooks*, a cookbook of reproducible data recipes. We conclude with some discussion about how these elements fit into the wider software environment.

2 Swarm and VirES

2.1 The Swarm mission and its data products

Swarm was selected in 2004 in ESA's Living Planet Programme to be the fifth mission within the fleet of Earth Explorers (Friis-Christensen et al., 2006). Swarm primarily comprises three spacecraft in Low Earth Orbit, gathering measurements of the magnetic field and plasma environment, which were launched on 22 November 2013. Having undergone several mission extensions Swarm continues to deliver well on its diverse objectives to study the geomagnetic field and related phenomena, probing properties and behavior all the way from Earth's core to the magnetosphere. It is likely that the three primary Swarm spacecraft will continue to operate into the 2030s. However, there is an increasing focus on moving toward a "true Swarm of magnetic satellites" (Olsen et al., 2019), to integrate better with other current and future spacecraft to make more use of complementary data and to draw an even greater benefit from Swarm. One example of this is the introduction of the Canadian Space Agency's e-POP (enhanced Polar Outflow Probe) payload aboard the CASSIOPE (Cascade, Smallsat and Ionospheric Polar Explorer) spacecraft as the fourth element of Swarm under ESA's Earthnet Third Party Mission Programme, and similar synergy being found with the CSES (China Seismo-Electromagnetic Satellite) program of CNSA (the China National Space Administration). Another example is new processing and calibration of platform magnetometer data from selected spacecraft: currently Cryosat-2, GRACE A and B (Gravity Recovery and Climate Experiment), GRACE-FO 1 and 2 (GRACE Follow-On), GOCE (Gravity Field and Steady-State Ocean Circulation Explorer), and more planned, to create science-grade data products, delivered through the Swarm framework (see, for example, Olsen et al. (2020) and Stolle et al. (2021)).

The Swarm spacecraft carry several instruments. The Absolute Scalar Magnetometer (ASM) and Vector Field Magnetometer (VFM) together measure the intensity and vector of the magnetic field. The Electric Field Instrument (EFI) consists of the Thermal Ion Imager (TII) and Langmuir Probe (LP), measuring plasma density, drift and velocity. The instrument package is completed with accelerometers (ACC) and GNSS receivers (GPSR). These instruments all produce relatively simple low level data products (time series), but the proper calibration and correction of these is quite complex. Further to this, much of the utility of the collected data is only realized after the expert development and application of specialized algorithms to create higher level data products that can be more readily consumed and analyzed by the wider scientific community. These factors mean that there are currently approaching one hundred data products publicly available, in many cases multiplied by three (plus) spacecraft. The products range from electric current estimates inverted from the magnetic measurements (e.g., field-aligned currents: `FACxTMS_2F`), to estimated ion drifts and masses (e.g., `EFIxIDM_2_`), to spherical harmonic models of the core (e.g., the CHAOS - CHAMP, Ørsted, SAC-C, and Swarm—model, Finlay et al., 2020) or ionosphere (e.g., the AMPS—Average Magnetic field and Polar current System—model, Laundal et al., 2018). The journey for a

1 https://earth.esa.int/eogateway/missions/swarm

2 https://eox.at

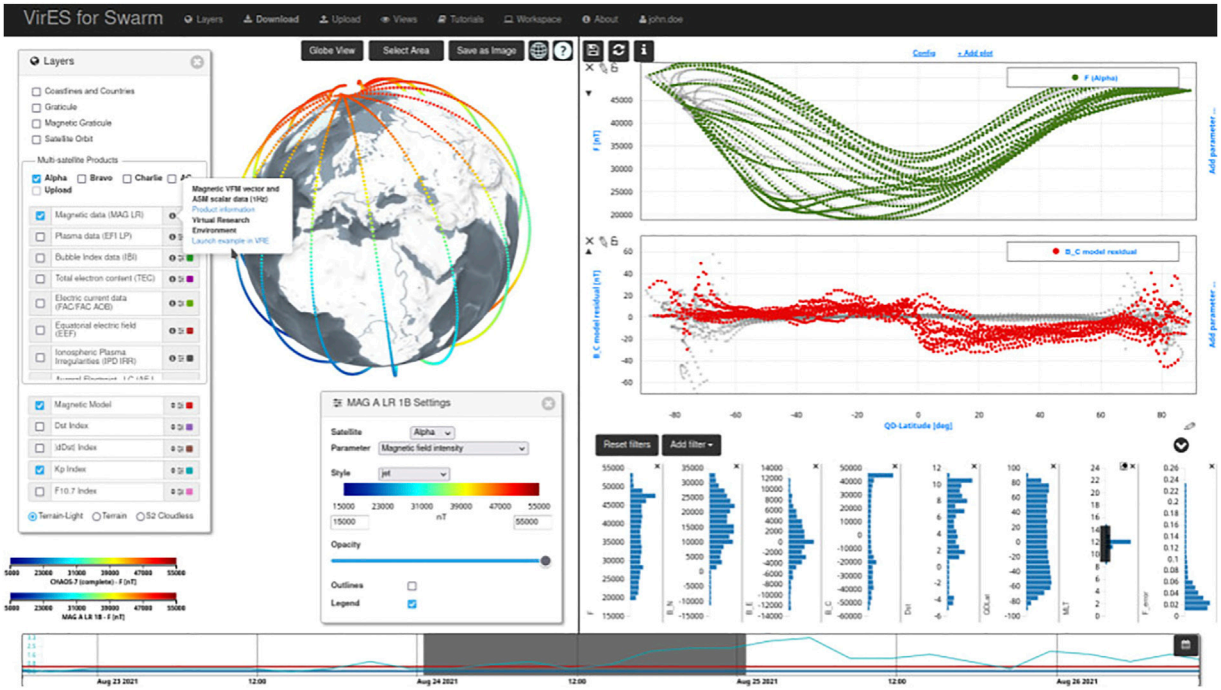


FIGURE 1
VirES web interface (https://vires.services): The layers panel on the left allows selection of multiple datasets together; the time slider on the bottom selects the time period of interest; a globe view and an "analytics" view can be used together to generate both 3D and 2D visualizations.

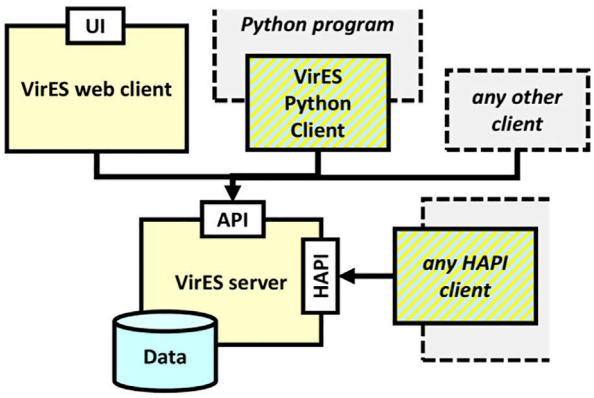


FIGURE 2
VirES architecture: The server holds data synchronized from external sources, which are accessible through the VirES API and the HAPI through a range of client software.

researcher to discover, access, and analyze such data can be quite long and difficult, though, as we hope to demonstrate in this paper, this situation is improved by availability of software and services that help researchers to navigate this landscape.

2.2 VirES data retrieval and visualization system

VirES for Swarm[3] started as an interactive data visualization and retrieval interface for the Swarm data products. Accessible directly through a web browser it allows anyone to view data and models both on a 3D globe and on 2D plots, in just a few mouse clicks (see Figure 1). Combinations of different data products can be selected and viewed together over a time window of interest, and plots have a degree of user-customizability to make them publication-ready. The web interface is not the focus of this paper, but it is a valuable part of the infrastructure because it is one of the foremost examples of accessible visualization tools in space physics and can still be very useful even when a particular user is comfortable processing data through Python. It can be used by newcomers to Swarm to gain familiarity and intuition about the data products, and by experienced users to quickly check events on particular days, for example.

3 https://vires.services

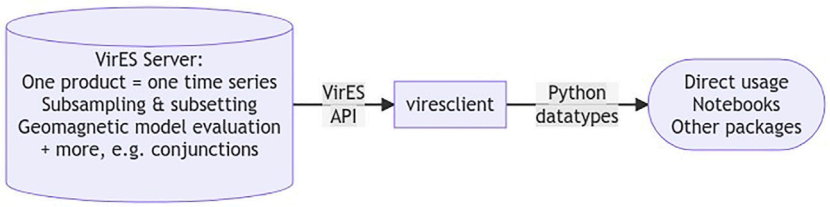


FIGURE 3
Connecting VirES to the Python ecosystem.

```
from viresclient import SwarmRequest

request = SwarmRequest()
request.set_collection("SW_OPER_MAGA_LR_1B")
request.set_products(
    measurements=["B_NEC"],
    models=["IGRF", "CHAOS-Core"],
    residuals=True,
    auxiliaries=["OrbitNumber", "MLT", "QDLat"],
    sampling_step="PT10S",
)
request.set_range_filter("Kp", 0, 2)
request.set_range_filter("Latitude", 50, 90)
data = request.get_between("2018-01-01", "2018-02-01")
ds = data.as_xarray()
```

FIGURE 4
Example usage of *viresclient* `.request` refers to an object that configures the request to the server `.data` refers to an object pointing to temporary data that has been created on-disk, delivered from the server `.ds` is an in-memory representation of the data loaded using the Xarray package.

A large part of what gives the web interface its functionality, and makes the infrastructure as a whole more sustainable, is that it is backed by the VirES Server (see Figure 2) which holds a database of the data products synchronized to their latest versions and offers various ways to interact with them. The server is built in part on the Django framework and other open source projects. It implements Open Geospatial Consortium (OGC[4]) standards for its Application Programming Interface (API) which allows it to communicate not only with the web interface but also other clients in a robust way. In particular, a component of the server is based on the OGC Web Processing Service (WPS) interface standard. This allows an external element to trigger one of a number of predefined processes on the service, passing custom parameters to the process to control its behavior, and thus retrieve customized data on-demand. This allows users to query subsets of the data, for example, by filtering by a range in Latitude or other available parameters. Another way in which is this is used is to evaluate auxiliary parameters such as magnetic local time (MLT) or to perform forward evaluation of geomagnetic field models at the same sample points as the data, without the code for such needing to be available on the client side. In 2022 the server was extended with an additional API, the Heliophysics API (HAPI, Weigel R. et al., 2021)[5], to allow smoother interoperability with other tools across heliophysics. Details of the HAPI implementation and the currently enabled datasets are available at the `/hapi` endpoint[6].

3 Tools and services to support scientific Python code

3.1 VirES Python client

Access to the VirES API was opened up in 2019 to allow machine-to-machine access outside of the graphical web interface. While it is possible to use the API directly, it is most convenient for

4 https://ogc.org

5 https://hapi-server.org

6 https://vires.services/hapi

the scientific audience to make use of the Python client which handles the communication with the server and loading of retrieved data into Python data types. This client is named *viresclient* (Smith et al., 2022)[7] and is available on the Python Package Index. The package allows the user to retrieve any data from the VirES service and bring it into the Python data ecosystem in just a few lines of code. It is very similar in usage pattern and ethos to broader data access libraries such as *hapiclient* (Weigel B. et al., 2021) or *speasy* (Jeandet and Schulz, 2022), though with specific features to accommodate possibilities with VirES.

The goal of *viresclient* is to make the data portable (see Figure 3). A user only needs to copy a short code example to bring the data into their own system, and adjust it for their needs. This works best for small volumes of data, but is more challenging with larger volumes where processing times, file management, and memory limits can become an issue. We keep the scope of the package intentionally limited to keep dependencies simple and ensure that it can be a reliable building block. The package also supports access to the "VirES for Aeolus" service[8].

While there are a few other features of the library (supporting other processes on VirES), the core usage is shown in Figure 4, which is demonstrative of the capabilities of the server side. A user first specifies which collection (i.e., dataset) they wish to access using `.set_collection()`, then they specify some configuration for that data using `.set_products()`: parameters to select from the dataset (`measurements=`), geomagnetic models to evaluate along the orbits (`models=`), to only return the data-model residuals (`residuals=True`), some auxiliary parameters that otherwise wouldn't be available within the source dataset (`auxiliaries=`), and an optional re-sampling of the data (`sampling_step=`). One can also specify to only retrieve a subset of the data according to values of a given parameter with `.set_range_filter()`—this can be useful for example for only retaining data with certain quality flags or to search within a geographical region. Finally one triggers the request to the server with `.get_between()`, specifying any time window desired. The request is processed on the server and if successful returns a data object which points to the downloaded data (stored as a temporary file, or set of files if a long time window was chosen). The user then has a choice about how to open the data for processing: currently we support the Pandas DataFrame and the Xarray Dataset, but more options could be added in the future.

3.1.1 Metadata and ontologies

The set of available data and models (i.e., those which have been defined on the server) can be found by users of *viresclient* using built-in methods such as `.available_collections()` giving a list of datasets, termed "collections" within VirES, `.available_measurements()` giving a list of parameters available within a given collection, or `.available_models()` giving a list of the available geomagnetic model names. The names of these entities (for example the strings "`SW_OPER_MAGA_LR_1B`" or "`IGRF`") are hard-coded into the server using either the official names of products as published by ESA, where available, or other names that can be understood by the user community (for example "IGRF" referring to the International Geomagnetic Reference Field). While these are defined systematically within the VirES database, they are not connected to a wider ontology such as those defined within ESPAS, the "near-Earth space data infrastructure for e-science" (Belehaki et al., 2016), or SPASE, Space Physics Archive Search and Extract (Roberts et al., 2018). This missing feature is likely to cause friction going forward as we anticipate to serve more datasets to more users, and seek interoperability with other systems to enable more multi-dataset multi-source science.

3.2 Swarm product algorithm laboratory and other thematic libraries

SwarmPAL (the Swarm Product Algorithm Laboratory) is currently in early development so the direction is highly likely to change, but it is worth outlining the current plan here. *SwarmPAL* is a new Python package containing a second level of tools over the top of data retrieved through *viresclient* and other sources, being developed openly on GitHub[9]. The goal of this package is to provide a range of analysis and visualization tools relevant to Swarm data products: it will hold proven algorithms that can be applied flexibly to the data, in effect producing new custom higher level data products on-demand, together with relevant visualizations for them. There are several Swarm DISC projects ongoing to deliver code-based toolboxes to work with Swarm products and *SwarmPAL* will be the home for these where appropriate. This will help us to centralize code quality improvements and maintenance, enable toolboxes to share elements, and present a coherent experience for the end users. Currently we are working on the addition of toolbox code from two Swarm DISC projects, TFA (time frequency analysis), and DSECS (dipolar spherical elementary current systems), with the potential to extend it further in the future.

As shown in Figure 5, the structure of the package is layered in order to support more flexible usage patterns. Swarm data may be loaded quasi-automatically and input directly to algorithms through special interfaces attached to internal data structures. This effectively allows users to follow a shortcut directly to a derived dataset and its visualization,

7 https://github.com/ESA-VirES/VirES-Python-Client

8 https://aeolus.services

9 https://github.com/Swarm-DISC/SwarmPAL

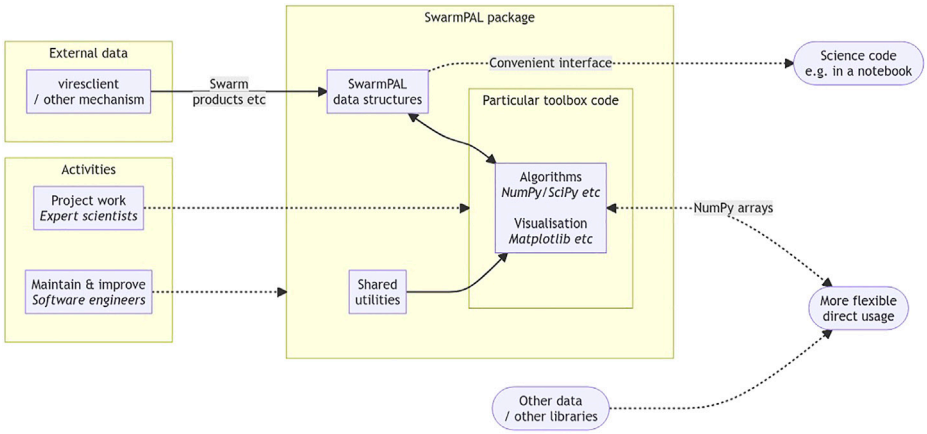


FIGURE 5
Schematic for the structure and usage patterns for *SwarmPAL*. While the package provides interfaces to run code immediately with Swarm data, hiding away the data access and processing steps, the underlying routines are accessible to use in more flexible ways through standard objects like NumPy arrays.

where aspects of the processing details can be abstracted away. Underneath that, however, the more functionally-designed routines performing individual steps are also exposed. These may be used in a more manual fashion together with other packages or datasets.

It should be noted at this point that there are several other Python packages closely related to Swarm, such as *ChaosMagPy* (Kloss, 2022)[10], providing tools for working with the CHAOS model and other field modeling utilities, and *pyAMPS* (Laundal and Toresen, 2018)[11], supporting the AMPS model. There is also *eoxmagmod*[12] which is used internally in the VirES server to provide optimized forward evaluation of geomagnetic models, but can also be used externally though is not yet very accessible. *SwarmPAL* is not meant to replace such libraries as they have different goals and requirements, but they could be brought in as dependencies for *SwarmPAL* or *vice versa*. On a related note, development of *SwarmPAL* must be very cognizant of other domain-specific libraries (such as those described within this issue) and concerns may be raised about interoperability and sustainability, particularly where usage scenarios may involve combining multiple packages. Furthermore, there is a received wisdom that scientific codes have a longer life when they are part of a larger non-project specific framework, so it may be that functionalities within *SwarmPAL* migrate to a different package in the future.

3.3 Virtual research environment

The most accessible software services are those which are cross-platform and require no installation by the user. JupyterHub and related Jupyter technologies such as the Jupyter Notebook format and paradigm have risen to dominance in recent years, lowering the barrier of entry for data science (and similar) workflows where writing code is necessary and embedding the human element into the way we approach interactive computing (Granger and Perez, 2021). Researchers often spend a long time configuring software and face many challenges in sharing their code with other researchers, greatly impeding time which can be better spent focusing on research questions. These factors strongly motivate the usage of shared JupyterHub systems for user groups with similar needs where software and hardware configuration can be performed centrally, giving rise to quicker on-boarding of new researchers and smoother options for collaboration. As an extension of the VirES service we provide such a Jupyter environment free to anyone with a user account on VirES, furnished with a selection of thematic libraries, and operating on cloud infrastructure. This service is called the Virtual Research Environment (VRE, or Swarm-VRE)[13], intended as a dynamic space to facilitate software

10 https://github.com/ancklo/ChaosMagPy
11 https://github.com/klaundal/pyAMPS/
12 https://github.com/ESA-VirES/MagneticModel
13 https://earth.esa.int/eogateway/tools/swarm-vre

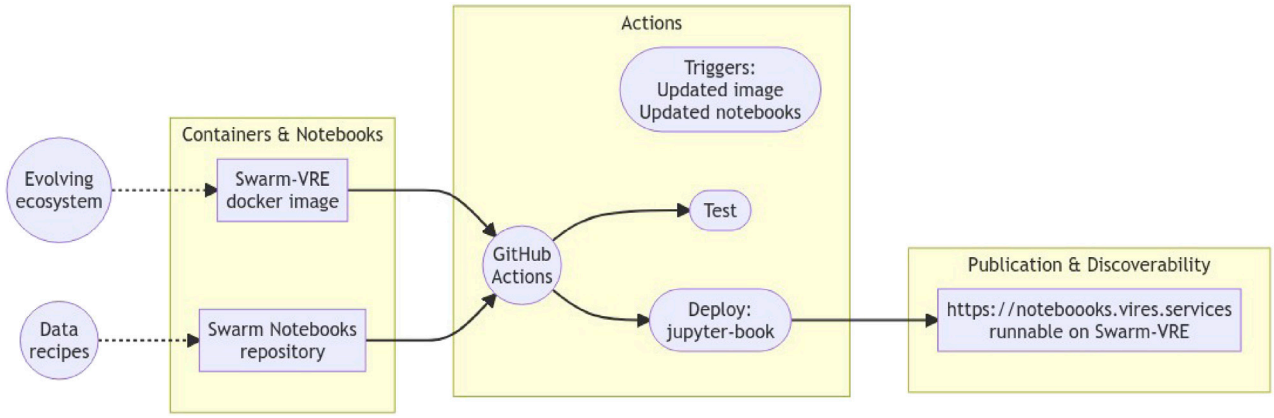


FIGURE 6
Architecture for maintaining notebooks. Containers provide a reliable execution environment both for testing the notebooks automatically and for end users interacting with notebooks on the JupyterHub. Notebooks are stored in a versioned repository to which multiple people can contribute, which are then tested on GitHub Actions and compiled into an HTML representation using Jupyter Book (verifying that they execute from top to bottom and allowing a human to manually review the output). The Jupyter Book deployment is the main way in which people can discover and read the notebooks, linked to the JupyterHub in order to interact with them.

development with Swarm. From the user perspective, it provides them with an Interactive Development Environment (IDE) in the form of JupyterLab that they can access directly through a web browser from any machine.

As an evolving service the future of the VRE depends on the users and their needs, and must strike a balance between ease of use (and ease of maintenance) and functionality when considering changes and additions to the environment. Likewise, it does not exist in a vacuum and will also be shaped by its surroundings. The system relies on many individual software components that themselves are evolving, from those supporting Jupyter (related to both the abstraction layers linking to the physical infrastructure, and the interface of the IDE and execution of notebooks), to the general-purpose Python scientific libraries, to the domain-specific libraries used within certain research areas. There are also many related initiatives, as different organizations move toward adoption of cloud-based services and/or containerization where they are useful for tackling the challenges of Big Data and collaboration in a world with increasingly complex software environments. Some of these are discussed in the Related Work section.

3.4 Swarm notebooks

A computational notebook, of which Jupyter Notebook is a prime contemporary example, is a natural way to express a research workflow where narrative, code, and results are interwoven in a medium which encourages rapid iteration and experimentation. There are also many more possibilities when combined with other utilities across the rapidly evolving Jupyter landscape, from parameterisation and automation with Papermill[14] to publication with Jupyter Book[15], as just two examples. Nevertheless, although achieving widespread adoption, notebooks do still present challenges in usability and sustainability. Here we discuss some of the ways in which we are using notebooks to support Swarm in the context of these challenges.

As part of the VRE, we include a collection of "Swarm Notebooks"[16] as a cookbook of reproducible recipes to work with Swarm. These perform several roles to assist people in using a new dataset or tool, showing exactly how to access a dataset alongside references that describe it, and how to bring different packages together to make use of that dataset. A precise recipe that can be executed immediately can be of great benefit to discoverability and accessibility of tools and data. We make use of continuous integration to ensure that these notebooks continue to run correctly and reliably within the VRE, as libraries get updated and other changes occur. By using Jupyter Book to compile the executed notebooks into a website[17], we enable them to be found on the web and

14 https://github.com/nteract/papermill
15 https://github.com/executablebooks/jupyter-book
16 https://github.com/Swarm-DISC/Swarm_notebooks
17 https://notebooks.vires.services

connected to the infrastructure that allows them to be executed. This approach has already been replicated to support the Aeolus mission as well[18]. The architecture is indicated in Figure 6.

In a similar fashion to the Swarm Notebooks repository, whose scope is to present a range of recipes to work with Swarm, we also extend the approach to other repositories with different goals. One example is FAC Exploration[19] (an earlier iteration of the SwarmFACE package; see Blagau et al. in this special issue), which is closer in intention to an executable paper, where specific phenomena are investigated and presented in a reproducible way for anyone to adjust and extend. By employing automation similar to the main Swarm Notebooks, we pave the way to enable maintenance of such resources so that they can continue to be accessed through the VRE as the service evolves, and aim to provide instructions enabling portability to other platforms. The creation of notebooks, which may also be connected to importable modules or scripts distributed together with the notebooks, is a reasonably accessible way for scientists to share their computational research in comparison to building full libraries or applications. When coupled with a centralized service that enables execution of these materials (subject to their adequate maintenance), this approach can be quite powerful, though it does not negate the need for well-engineered scientific libraries to provide the more re-usable building blocks.

Instead of bundling all notebook resources into one repository that would be challenging to maintain, or having many isolated projects with dissimilar methods for maintenance, we identify some common patterns that can be re-used and adjusted slightly across inter-linked projects. Namely, the use of containers, actions/automation, and notebooks in combination, where the particular tools we use for these are Docker, GitHub Actions, and Jupyter respectively. We use containerization to provide the execution environment (i.e., using the Docker image backing the VRE) which is used by GitHub Actions to systematically execute notebooks (on changes to either the notebooks or the environment) which are published to the web with Jupyter Book. Any issues found during updates can then be highlighted by this process, while simultaneously providing the web resource that people can use to discover and use the content. Such resources can be interlinked to form a knowledge network.

18 https://notebooks.aeolus.services

19 https://github.com/Swarm-DISC/FAC_exploration

4 Related work

We have presented our Swarm-VRE, but we have not yet discussed how this service fits into the wider landscape where many players are building and supporting similar services. Such initiatives include Pangeo[20], a community platform for Big Data geoscience; HelioCloud[21], similar in goals to Pangeo, but targeting heliophysics; Resen[22] (Reproducible Software Environment, Bhatt et al., 2020); AMDA[23] (Automated Multi-Dataset Analysis), a virtual research platform in operation since before the rise of Jupyter (Génot et al., 2021); PITHIA-NRF[24] (Plasmasphere Ionosphere Thermosphere Integrated Research Environment and Access services: a Network of Research Facilities), an upcoming platform to support geospace science and services; and AuroraX[25], a data platform for auroral science.

While the services listed above have different goals from Swarm-VRE (generally broader in scope, and domain-specific rather than mission-specific) and each has emerged from different frameworks and approaches, they share common threads. They aim to bridge the gap between complex layers of software and the end users, and they all rely on robust access to data, arriving at different solutions to provide that. The proliferation of different tools and services is inevitable, and we believe to achieve sustainability such projects always need to acknowledge that they exist within an ecosystem and so the sub-elements providing interoperability are the ones that will receive the greatest return on investment. Thus, while Swarm-VRE focuses on supporting applications with Swarm in particular, it should also be an engine for development and support of reusable components (i.e. libraries and notebooks) that can be utilized elsewhere, and its future will be shaped by close attention to and collaboration with other projects.

5 Future work

The evolution of VirES is directly connected to the expansion of the Swarm product portfolio and ongoing changes to the processing systems and publication and dissemination of these data.

ESA will soon launch a new “Swarm Data Handbook” as a tool for discovery and navigation of the Swarm products and their science level metadata (i.e., explanations of the data and links to other resources), giving a more convenient interface

20 https://pangeo.io/

21 https://heliocloud.org

22 https://ingeo.datatransport.org/home/resen

23 http://amda.irap.omp.eu/

24 https://pithia-nrf.eu/

25 https://aurorax.space/

than the current documentation. Persistent identifiers (i.e., Digital Object Identifiers, DOI) and JSON-LD (JavaScript Object Notation for Linked Data) will be attached to each item, which will help these products to follow the FAIR (Findable, Accessible, Interoperable, and Reusable) data principles (as explained by, for example, Masson et al., 2021). This is also an important step toward strengthening the ontology of objects within VirES (by including these references) and potentially integrating the products within systems such as SPASE. We expect to expand the publishing of data within VirES over the HAPI interface and through the Handbook activity increase utility of metadata in connecting with other data systems. We also foresee expansion of capabilities in VirES around handling of geomagnetic models (and potentially other types of model).

In coordination with scientists we will increase the coverage of tools and examples available within the VRE, while focusing on the aspects of portability (one can run the code anywhere) and modularity (one can flexibly use individual components). Sometimes this will mean adding functionality to *viresclient* and *SwarmPAL*, and improving the available notebooks (and encouraging community contribution thereto); sometimes supporting Swarm-adjacent Python tools (i.e., those within the orbit of Swarm but remaining autonomous from the VRE project); and sometimes connecting with tools from the wider Python ecosystem. Meanwhile, the interface of JupyterLab and other tooling around Jupyter will improve to help make notebooks more accessible and maintainable but it will remain a community effort to provide training and share ideas around using the tools and the data to accomplish science tasks.

6 Conclusion

With Swarm being an explorer of both the solid Earth and geospace, its data environment and applications are diverse. VirES provides interfaces to the data and models which are cognizant of this, for example handling near-Earth geomagnetic models so that space physicists do not need to interact with them directly, and ability to sub-select data by geographic region or other parameters. By handling some common requirements like these on the server accessible through a request-driven interface, the burden on the user side can be lightened (i.e., users do not need to have the full dataset downloaded in order to access subsets, and do not need to have the ability to perform the model forward evaluations). The server can be accessed through *viresclient* but also (with more limitations) through HAPI, which opens up pathways for different usages and integration through the Python ecosystem. In this paper we have shown how we build on this and make it more accessible through development of libraries and Jupyter notebooks backed by the VRE, but to achieve a broader impact and sustainability ongoing integration with other tools and services will be needed.

It is widely appreciated that data science will bring further momentum to discovery in geospace (e.g., McGranaghan et al., 2017), a trend that will accelerate with a growing number of available data sources, and so research funders should recognize the criticality of data systems in driving and supporting research activities. Likewise, developers of data systems need to remain aware of and engage with related activities, which can happen through fora such as the International Heliophysics Data Environment Alliance (IHDEA[26]). As is evident from the other articles in this issue, researchers need the ability to write analysis code that may draw from a number of different data systems or software tools, and Python is the most popular language and ecosystem for this (e.g., Python in Heliophysics Community[27]), being accelerated by the interactive computing paradigm supported by Jupyter and broader appreciation of the benefits of open science driven by open code. The Python software environment for Swarm is being developed in this context and seeks further alignment with related projects.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

ARAS has written the content of this article summarizing activities supported by Swarm DISC. MP is the lead software engineer on the VirES project. Both authors contributed to manuscript revision and approved the submitted version.

Funding

This work has been supported by the Swarm DISC project, funded by ESA through contract No. 4000109587/13/I-NB.

Conflict of interest

MP was employed by EOX IT Services GmbH.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

26 https://ihdea.net/

27 https://pyhc.org/

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Belehaki, A., James, S., Hapgood, M., Ventouras, S., Galkin, I., Lembesis, A., et al. (2016). The ESPAS e-infrastructure: Access to data from near-Earth space. *Adv. Space Res.* 58, 1177–1200. doi:10.1016/J.ASR.2016.06.014

Bhatt, A., Valentic, T., Reimer, A., Lamarche, L., Reyes, P., and Cosgrove, R. (2020). Reproducible software environment: A tool enabling computational reproducibility in geospace sciences and facilitating collaboration. *J. Space Weather Space Clim.* 10, 12. doi:10.1051/swsc/2020011

Finlay, C. C., Kloss, C., Olsen, N., Hammer, M. D., Tøffner-Clausen, L., Grayver, A., et al. (2020). The CHAOS-7 geomagnetic field model and observed changes in the South Atlantic Anomaly. *Earth Planets Space* 72, 156–231. doi:10.1186/s40623-020-01252-9

Friis-Christensen, E., Luhr, H., and Hulot, G. (2006). Swarm: A constellation to study the Earth's magnetic field. *Earth Planets Space* 58, 351–358. doi:10.1186/BF03351933

Génot, V., Budnik, E., Jacquey, C., Bouchemit, M., Renard, B., Dufourg, N., et al. (2021). Automated Multi-Dataset Analysis (AMDA): An on-line database and analysis tool for heliospheric and planetary plasma data. *Planet. Space Sci.* 201, 105214. doi:10.1016/J.PSS.2021.105214

Granger, B. E., and Perez, F. (2021). Jupyter: Thinking and storytelling with code and data. *Comput. Sci. Eng.* 23, 7–14. doi:10.1109/MCSE.2021.3059263

Jeandet, A., and Schulz, A. (2022). *Speasy*. Online, Zenodo. doi:10.5281/zenodo.4118780

Kloss, C. (2022). *ancklo/ChaosMagPy: ChaosMagPy*. Online, Zenodo. doi:10.5281/zenodo.3352398

Laundal, K. M., Finlay, C. C., Olsen, N., and Reistad, J. P. (2018). Solar wind and seasonal influence on ionospheric currents from Swarm and CHAMP measurements. *J. Geophys. Res. Space Phys.* 123, 4402–4429. doi:10.1029/2018JA025387

Laundal, K. M., and Toresen, M. (2018). *klaundal/pyAMPS: pyAMPS*. doi:10.5281/zenodo.1182930

Masson, A., De Marchi, G., Merin, B., Sarmiento, M. H., Wenzel, D. L., and Martinez, B. (2021). Google dataset search and DOI for data in the ESA space science archives. *Adv. Space Res.* 67, 2504–2516. doi:10.1016/J.ASR.2021.01.035

McGranaghan, R. M., Bhatt, A., Matsuo, T., Mannucci, A. J., Semeter, J. L., and Datta-Barua, S. (2017). Ushering in a new frontier in geospace through data science. *JGR. Space Phys.* 122, 12586–12590. doi:10.1002/2017JA024835

Olsen, N., Albini, G., Bouffard, J., Parrinello, T., and Tøffner-Clausen, L. (2020). Magnetic observations from CryoSat-2: Calibration and processing of satellite platform magnetometer data. *Earth Planets Space* 72, 48. doi:10.1186/S40623-020-01171-9

Olsen, N., Tøffner-Clausen, L., Kloss, C., and Finlay, C. (20192019). "Exploring earth from space – towards a true Swarm of magnetic satellites," in *Living Planet symposium* (Milan: Italy).

Roberts, D. A., Thieman, J., Génot, V., King, T., Gangloff, M., Perry, C., et al. (2018). The SPASE data model: A metadata standard for registering, finding, accessing, and using heliophysics data obtained from observations and modeling. *Space weather.* 16, 1899–1911. doi:10.1029/2018SW002038

Smith, A. R. A., Pačes, M., and Santillan, D. (2022). *ESA-VirES/VirES-Python-Client*. Online, Zenodo. doi:10.5281/zenodo.2554162

Stolle, C., Michaelis, I., Xiong, C., Rother, M., Usbeck, T., Yamazaki, Y., et al. (2021). Observing Earth's magnetic environment with the GRACE-FO mission. *Earth Planets Space* 73 (1 73), 51. doi:10.1186/S40623-021-01364-W

Weigel, B., Batta, H. N., Faden, J., and jvandegriff (2021a). *hapi-server/client-python*. Online, Zenodo. doi:10.5281/zenodo.5553155

Weigel, R., Vandegriff, J., Faden, J., King, T., Roberts, D. A., Harris, B., et al. (2021b). Hapi: An API standard for accessing heliophysics time series data. *JGR. Space Phys.* 126, e2021JA029534. doi:10.1029/2021JA029534

frontiers | Frontiers in Physics

TYPE Original Research
PUBLISHED 31 October 2022
DOI 10.3389/fphy.2022.1022475

Check for updates

OPEN ACCESS

EDITED BY
John Coxon,
Northumbria University,
United Kingdom

REVIEWED BY
Ciarán D. Beggan,
Natural Environment Research Council,
United Kingdom
Andrew Smith,
Mullard Space Science Laboratory,
Faculty of Mathematical and Physical Sciences, University College London,
United Kingdom
Robert Arritt,
Electric Power Research Institute,
United States
Gemma Richardson,
The Lyell Centre, United Kingdom

*CORRESPONDENCE
Shibaji Chakraborty,
shibaji7@vt.edu

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Physics

RECEIVED 18 August 2022
ACCEPTED 23 September 2022
PUBLISHED 31 October 2022

CITATION
Chakraborty S, Boteler DH, Shi X, Murphy BS, Hartinger MD, Wang X, Lucas G and Baker JBH (2022), Modeling geomagnetic induction in submarine cables.
Front. Phys. 10:1022475.
doi: 10.3389/fphy.2022.1022475

COPYRIGHT
© 2022 Chakraborty, Boteler, Shi, Murphy, Hartinger, Wang, Lucas and Baker. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Modeling geomagnetic induction in submarine cables

Shibaji Chakraborty[1]*, David H. Boteler[2], Xueling Shi[1,3], Benjamin S. Murphy[4], Michael D. Hartinger[5], Xuan Wang[6], Greg Lucas[7] and Joseph B. H. Baker[1]

[1]Center for Space Science and Engineering Research, Virginia Tech, Blacksburg, VA, United States, [2]Natural Resources Canada, Ottawa, ON, Canada, [3]High Altitude Observatory, National Center for Atmospheric Research, Boulder, CO, United States, [4]U.S. Geological Survey, Geomagnetism Program, Golden, CO, United States, [5]Space Science Institute, Boulder, CO, United States, [6]Department of Electrical Engineering Tsinghua University, Beijing, China, [7]SWX TREC, LASP, Boulder, CO, United States

Submarine cables have become a vital component of modern infrastructure, but past submarine cable natural hazard studies have mostly focused on potential cable damage from landslides and tsunamis. A handful of studies examine the possibility of space weather effects in submarine cables. The main purpose of this study is to develop a computational model, using *Python*, of geomagnetic induction on submarine cables. The model is used to estimate the induced voltage in the submarine cables in response to geomagnetic disturbances. It also utilizes newly acquired knowledge from magnetotelluric studies and associated investigations of geomagnetically induced currents in power systems. We describe the Python-based software, its working principle, inputs/outputs based on synthetic geomagnetic field data, and compare its operational capabilities against analytical solutions. We present the results for different model inputs, and find: 1) the seawater layer acts as a shield in the induction process: the greater the ocean depth, the smaller the seafloor geoelectric field; and 2) the model is sensitive to the Ocean-Earth layered conductivity structure.

KEYWORDS

magnetic induction, submarine cable, geomagnetic storm activity, space weather, conductivity model

1 Introduction

Submarine cables have become a vital component of our modern infrastructure. They carry more than 95% of international internet traffic [1], so any disruption to their operation can have wide-ranging consequences across the globe. Although studies focused on natural hazards that may damage cables such as submarine landslides and tsunamis have increased [1], the possibility of space weather effects has generally not been considered in such studies. Over the last 20 years - the lifetime of modern cable systems - space weather activity has been low; but more extreme space weather events have occurred in the past and it is uncertain how modern submarine cables systems would behave in an extreme (1 in 100 years) space weather event. Thus, submarine cables, like

other critical infrastructure, need to consider space weather as High Impact Low Frequency (HILF) events, which require an assessment of risk and preparation of mitigation measures if necessary.

During geomagnetic disturbances, geomagnetic field variations induce electric currents in the Earth, commonly referred to as geomagnetically induced currents (GIC), and in human-made conductors. Communication cables, power systems, pipelines and railway signaling are all susceptible to disturbances due to these currents, see Boteler et al. [2]. Geomagnetic induction first affected telegraph systems with widespread problems during the Carrington event of 1859 [3–5]. A new vulnerability was introduced in the 1950s with the development of cables with repeaters powered by a current fed along the cable. Phone calls over the first trans-Atlantic phone cable, TAT-1, during the magnetic storm of 10 February 1958, alternated between loud squawks and faint whispers as the naturally induced voltage acted with or against the cable supply voltage [6]. A later development was the introduction of fiber-optic cables in the 1980s, but they too include a copper conductor along the cable to carry power to the repeaters. Thus, they are vulnerable to GIC fluctuations and surges [7]. Recordings on the trans-Atlantic TAT-8 cable during a major magnetic storm in March 1989 showed that the cable's power feed could be affected during extreme geomagnetic disturbances [8]. Specifically, the study by Medford et al. [8] mentioned that, although there were no disruptions to service during the 1989 storm, the systems were very close to a threshold where they would be disrupted. Therefore, geomagnetic disturbances on submarine cables remain a concern to this day.

The aim of this study is to build a computational model of geomagnetic induction to calculate the induced voltages produced in submarine cables during geomagnetic disturbances. We present the theory for estimating the induced voltages experienced by the submarine cables during geomagnetic disturbances. The numerical model is capable of calculating geomagnetically induced cable voltages at different locations around the globe. Here we describe the Python-based software that enables the numerical model to be used by researchers or cable engineers. Specifically, we describe 1) the input and output formats, 2) the internal working of individual computational blocks, and 3) their interconnectivity. The model can digest 1) observed and synthetic geomagnetic field data, 2) cable route information, and 3) an Ocean-Earth conductivity model to calculate the induced electric fields and cable voltages. Users can provide the cable routes as input by providing the latitude and longitude information, or standard Route Position List (RPL) files used by the submarine cable industry. One subunit of the model is also dedicated to generating synthetic geomagnetic field and geoelectric field data, which is useful for validating the modeling software. This functionality can also be used to model behavior for an extreme geomagnetic storm that is rare in nature (1 in 100 years event). Users can input the conductivity model either by providing the data or selecting the lithospheric model (LITHO 1.0) of the Earth [9]. Java Script Object Notation (JSON) files are used to embed all the information and feed it into the software. We provide human-readable schema of the valid input JSON formats as reference for users. Model outputs are 1) transfer function to compute seafloor geoelectric field from the sea-surface geomagnetic field, 2) parameters for a transmission line model of the seawater and underlying resistive layers along the cable route, 3) result of the nodal network derived from the transmission line model, and 4) induced electric field and voltages on the cable. Finally, we discuss the assumptions for converting the theoretical model into this computational model, as well as the limitations of these assumptions.

The model has a few free parameters, such as ocean depth, length of cable, and earth conductivity model, which affect the estimate of the induced cable voltage. Note that among various parameters used in this model, seawater conductivity is a relatively fixed parameter, which realistically varies between about 0.28 and 0.33 Ωm. We conduct a sensitivity analysis of the model and comment on the factors that contribute to the geomagnetic hazard to submarine cables. A number of components are included in the software that can be used separately by end-users, each designed as a separate module for a specific study with specific scientific objectives. As part of the demonstration of the capabilities of the software and validation of the model, we describe several applications and examples of the software. Finally, we discuss the various capabilities and limitations of the current computational model and the potential extensions of the software in the future. We provide several *IPython* notebook examples as a user guide for the software.

2 Methodology: Computational model

This section presents a detailed description of the computational model and control flow diagrams of the *Python* software that estimates geomagnetic impact on submarine cables. The software is designed to perform the following two functions: 1) determination of induced electric fields on the seafloor at locations along a cable route [10–13], and 2) computation of the overall effect of geomagnetic induction on the submarine cable.

The control flow diagram of these operations is presented in Figure 1. The determination of the geomagnetic impact on submarine cables requires calculation of both the seafloor electric field experienced by the cable (computed by Module 02) and the Earth potentials produced at the ends of the cable by the induced currents flowing across the ocean (computed by Module 03). The software can be verified by making calculations for test cases for which analytic solutions are available (Module

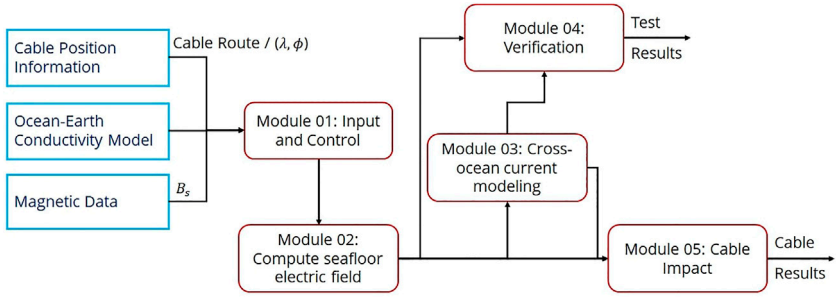


FIGURE 1
Control flow diagram of operations done by computational model. The boxes in red and blue define operational and input modules, respectively. Solid lines represents data flow interconnections between different modules.

TABLE 1 Example RPL file.

Pos no.	Event	Latitude	Longitude	Distance (km)		Cable type	Approx depth (m)	Target burial (m)
				Between Positions	Cumulative Total			
1		18°7′N	66°36′E		0.000		0	
				19.212		SA		0.0
2	AC	18°3.477′N	66°25.561′E		19.212		8	
				10.317		SA		0.0
3		18°0.199′N	66° 20.737′E		29.529		11	
				9.088				0.0
4	AC	17°57.312′N	66°16.489′E		38.616		22	
				96.993		SA		0.0
5	AC	17°37.527′N	66°324.704′E		135.609		28	
				23.700		SA		0.0
6		17°33.525′N	66°11.743′E		159.609		31	
				46.368		SA		0.0
7		17°25.695′N	65°46.401′E		205.678		31	
				4.992		SA		0.0

04). The seafloor electric fields along the route of the cable and the Earth potentials at the cable ends are then combined to give the total impact on the power feed for the cable (Module 05). User interaction with the program is provided through an Input and Control routine (Module 01) that inputs a specific location on the Earth or cable route, Ocean and Earth model details, and magnetic data; and controls whether the calculations are made using synthetic data for verification of the software or using recorded magnetic field data from actual space weather events.

TABLE 2 Parameters for test Ocean-Earth model (Atlantic model).

Layer	Thickness (in km)	Resistivity (in Ωm)
Seawater	4	0.3
Sediments	2	3
Crust	10	3,000
Mantle Lithosphere	140	1,000
Upper Mantle	254	100
Transition Zone	250	10
Lower Mantle	340	1

2.1 Module 01: Input and control

The operation of the program is governed by the Input and Control Module. This is where the cable to be studied is specified and the cable's Route Position List (RPL) is read as input. Table 1 shows a typical format for a RPL file, which contains location of the cable section edges, length of each section, cumulative

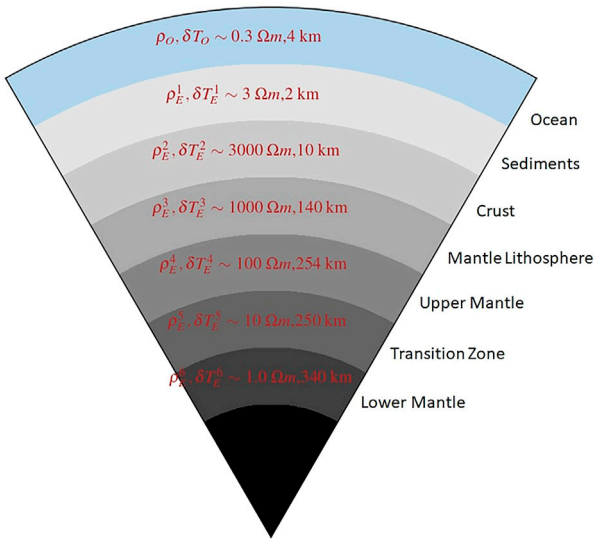


FIGURE 2
Schematic of layered Ocean-Earth conductivity model. Thickness and conductivity of each layer is provided on the right of the figure. Name of each layer is also mentioned in the figure.

distance in kilometers, type of each section, *etc.* The ocean depth along the path of the cable profile can be viewed and the user can specify how the cable route will be broken into sections that will be used in the subsequent modeling. Then the crust and lithosphere model, LITHO1.0 [9], is accessed to retrieve average thickness of the Ocean water and layers of the Earth for each section of the cable route. Conductivities of each layer then form a predefined table; one example is shown in Table 2. We used this process to define the layer conductivity model for each section of the cable route. In addition, the model can process magnetic field data/observations from magnetometer instruments. The magnetic observatory to be used in electric field calculations is also specified, and the files for specified days are retrieved from the INTERMAGNET database [14].

2.1.1 Ocean-Earth layered conductivity model

One of the primary components of the model is the Ocean-Earth layered conductivity model. This layered structure is used to determine the transfer function between the seafloor electric field and the surface magnetic field (T_x). Figure 2 presents a schematic cross-section of a layered Ocean-Earth model. Thickness and the resistivity of each layer is shown in red text. Properties of the layers used in this plot are listed in Table 2. The layered conductivity model holds thickness ($\delta T_{O/E}$ in km) and resistivity (ρ in Ωm) information of each layer of the Earth (including water bodies on the Earth). There are three different ways to input this information into the software: 1) request for in-build Ocean-Earth layered models (taken from previous studies), 2) user defined layers as a list/array, and 3) geodetic latitude, longitude position of the cable route. For the third option we used the LITHO1.0 model, which is a 1° tessellated representation of the crust and uppermost mantle of the Earth, including the lithospheric lid and underlying asthenospheric layer. LITHO1.0 model parameters are specified laterally as tessellated nodes and vertically as a series of geophysically identified layers, including water, ice, sediments, crystalline crust, lithospheric lid, and asthenosphere. As LITHO1.0 provides thickness of the Ocean-Earth layers, resistivity (conductivity) values described in Table 2 are only used by LITHO1.0 model. For a given latitude-longitude value LITHO1.0 provides thicknesses of the Ocean and Earth layers (center column of Table 2). We assign resistivity value to each of these layers as given in Table 2 (right column). These values are chosen as commonly recognized representative values for each layer, and they generally match observations from Atlantic Ocean magnetotelluric (MT) sites [15–17]. Note that the resistivity values for the LITHO1.0 model can also be modified by the user. Table 2 also presents resistivity values and thicknesses for the transition zone and lower mantle, which are fixed based on common understanding of where the mantle transition zone sits (410–660 km) with a fixed model bottom at 1,000 km. Hence, for a given cable section with edge location information, the model (LITHO1.0) provides the Ocean-Earth Conductivity structure at the center of the cable section [18].

2.2 Module 02: Seafloor electric field calculations

The calculations of seafloor electric fields are comprised of two parts: 1) calculate the transfer function, T_x, between the seafloor electric field and the surface magnetic field variations, and 2) use the transfer function with magnetic field data to calculate the seafloor electric fields. To calculate the transfer function, T_x, the Ocean-Earth conductivity model (from LITHO1.0) is used to calculate the transfer function, T_f, at the seafloor between the electric field, E_f, and the magnetic field, B_f, using the recursive formulas as shown by Boteler and Pirjola [13]. The seafloor transfer function (T_f) is then used with the formula from Boteler and Pirjola [13] to give the transfer function (T_x) that relates the seafloor electric field (E_f) and the surface magnetic field (B_s) variations:

$$T_f = \frac{E_f}{B_f} = \frac{Z_d}{\mu_0} \quad (1)$$

$$T_x = \frac{E_f}{B_s} = \frac{Z}{\mu_0} \frac{2}{\left(1 + \frac{Z}{Z_d}\right)e^{kd} - \left(1 - \frac{Z}{Z_d}\right)e^{-kd}} \quad (2)$$

where, $Z_d = \mu_0 \frac{E_f}{B_f}$ is the surface impedance at the seafloor, d is the depth of the ocean, μ_0 is the permeability at vacuum, Z and k are the characteristic impedance and propagation constant, respectively, of the seawater layer.

Calculation of the seafloor electric fields produced by a magnetic disturbance can be achieved by taking the Fourier transform of the magnetic field time series to obtain the magnetic field spectrum. Each spectral component is then multiplied by the corresponding value of the transfer function, T_x, to obtain the electric field spectrum. An inverse Fourier transform then gives the electric field variations in the time domain. Standard pre-processing (remove mean and trend and taper the ends of the time series) of the magnetic field data is applied before taking the Fourier transform. In addition, care is needed in assigning the correct values to the negative frequency components of the electric field spectrum before taking the inverse Fourier transform (see [19]).

Finally the electric field is integrated over the length of the cable section to give the electromotive force (emf) induced in each section. To simplify the calculations we have made the approximation that the electric field is uniform within each section, with index j. In this case the emf for each section is given by

$$\mathcal{E}_j = E_j^N L_j^N + E_j^E L_j^E \quad (3)$$

where E_j^N and E_j^E are the Northward and Eastward electric fields (in V/km), respectively, and L_j^N and L_j^E are the effective Northward and Eastward distances (in kilometers). To calculate the distances we use the 'WGS84' geodetic model, which is used for power system modeling [20]. This gives the following expression for the Northward and Eastward distances in kilometers:

$$L_j^N = \left(111.133 - 0.56\cos 2\hat{\lambda}_j\right).\delta\lambda_j \quad (4)$$

$$L_j^E = \left(111.5065 - 0.1872\cos 2\hat{\lambda}_j\right).\cos\hat{\lambda}_j.\delta\phi_j \quad (5)$$

Where $\delta\lambda_j$ and $\delta\phi_j$ are the difference in latitude (in degrees) and longitudes (in degrees) between the ends of the cable section (j), respectively, and $\hat{\lambda}_j$ is the average latitude of the ends of the cable section (j).

2.3 Module 03: Cross-ocean current modeling

The electric fields produced during geomagnetic disturbances drive electric currents across the ocean. When these electric currents encounter the more resistive land on either side of the ocean there is an accumulation of charge creating an electrical potential in the Earth that deflects the electric currents down through the Earth's lithosphere. The Earth potentials produced can be modeled using distributed source transmission line (DSTL) theory as shown by Wang et al. (2022), (also presented in the Supplementary Document). Based on the theory described in Wang et al. (2022), we can subdivide the ocean into multiple segments along the route of the cable, assuming the conductivity of the each segment depends only on vertical variations in stratified Ocean-Earth structures and their electrical properties. We then convert each transmission line segment to its equivalent-π circuit with equivalent admittance and current source. Furthermore, we connect equivalent-π circuits in series and perform a nodal analysis by applying Kirchoff's current law at each node. This leads to a set of equations (refer to the Supplementary Document for detailed analysis) involving the nodal voltages U which can be written in matrix form:

$$J = YU \quad (6)$$

where J represents the current sources in vectorized form. Elements of J represent the sum of equivalent current sources directed into each node. Y is the admittance matrix in which the diagonal elements, Y_{ii}, are the sum of all the admittances terminating at the node i including the admittance-to-ground at node i, and the off-diagonal elements, Y_{ki}, are equal to the negative of the admittance connected directly between nodes i and k, that are given by following equations:

$$Y_{ii} = y_i + \sum_{k=1}^{N} y_{ki}, \text{ for } k \neq i \quad (7)$$

$$Y_{ki} = -y_{ki}, \text{ for } k = i \quad (8)$$

The potentials at each node, in vector U, are then obtained by matrix inversion of the admittance matrix, Y, and multiplication by the nodal current sources, J:

$$U = Y^{-1}J \quad (9)$$

These potentials at the edges of each section between nodes i, k are then used to calculate the Earth potential along the cable route:

$$U_{i,k}(x) = \frac{U_k\left[e^{\gamma x} - e^{-\gamma x}\right] + U_i\left[e^{\gamma(L-x)} - e^{-\gamma(L-x)}\right]}{e^{\gamma L} - e^{-\gamma L}} \quad (10)$$

where: U_i, U_k, γ, L, x are the potential at nodes i, k, propagation constant, length of section, and location on the section along the transmission line, respectively.

2.4 Module 04: Verification

To test the accuracy of the calculations performed in Modules 02 and 03 these modules can be operated with test inputs for which analytic solutions are available. Module

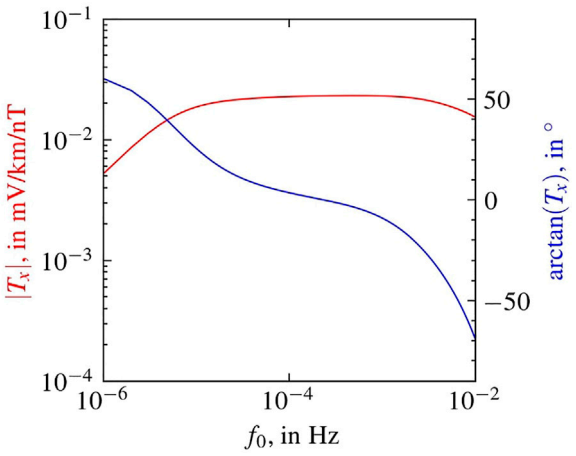


FIGURE 3
Amplitude ($|T_x|$) and phase (arctan (T_x)) of the transfer function, T_x, for the Ocean-Earth model described in Table 2. Example *Python* notebook for generating the transfer function is uploaded in Zenodo [21].

TABLE 3 Transmission line parameters for test model.

Parameter	Value
Series Impedance, Z	$7.14 \times 10^{-2}\ \Omega/km$
Parallel Admittance, Y	$5.88 \times 10^{-6}\ S/km$
Characteristic Impedance, Z_0	$110\ \Omega$
Propagation Constant, γ	$6.48 \times 10^{-4}/km$
Adjustment Distance, $\frac{1}{\gamma}$	1,543.21 km

04 performs the comparison between the outputs of Modules 02 and 03 and the analytic solutions.

To test the calculation of electric fields we follow the procedure described in Boteler and Pirjola [19]. A synthetic magnetic field variation, comprised of six frequencies, is created and a test case Ocean-Earth model is specified (Table 2). Table 2 describes an average stratified Ocean-Earth structure of the Atlantic Ocean, hence, from here onward, we refer this model as "Atlantic" model. The transfer function, T_x, between the seafloor electric field and surface magnetic field variations are then calculated for this Ocean-Earth model for the six frequencies. The amplitude and phase of the transfer functions (T_x) are presented in Figure 3. The six components of the magnetic field variations are then multiplied by the corresponding transfer function values to give the components of the electric field variations. These are then summed to give an analytic solution for the electric field. For testing, the synthetic magnetic field variations are used as input to Module 02, and the electric field output from Module 02 is compared with the analytic solution and the test results are output by Module 04. Compilation of synthetic magnetic field data, to test the model, is described in Section 2.4.2.

To test the cross-ocean modeling calculations, we used the Ocean-Earth model described in Table 2. For the test case, the same Ocean-Earth model is used for the testing of Module 02. The parameters of this model are used to calculate the transmission line properties listed in Table 3.

Expressions for end potentials are also required for these tests. For a transmission line of length L, with zero admittance to ground at each end, the general expression for the end potentials is:

$$U_A = -\left(\frac{\sinh \gamma L}{\cosh \gamma L + 1}\right)\frac{E}{\gamma} \tag{11}$$

$$U_B = \left(\frac{\sinh \gamma L}{\cosh \gamma L + 1}\right)\frac{E}{\gamma} \tag{12}$$

The potential as a function of distance along the transmission line is given by:

$$U(x) = \left(\frac{U_B e^{\gamma L} - U_A}{e^{\gamma L} - e^{-\gamma L}}\right)e^{-\gamma(L-x)} + \left(\frac{U_A e^{\gamma L} - U_B}{e^{\gamma L} - e^{-\gamma L}}\right)e^{-\gamma x} \tag{13}$$

These expressions depend only on the length of the transmission line segment L (corresponding to the width of the ocean) and the propagation constant of the transmission line model (γ) which be calculated for any specified ocean thickness.

2.4.1 Electrically-long and electrically-short Ocean-Earth sections

Consider an Ocean-Earth section with physical length, L, and propagation constant, γ. The section has an adjustment distance $\frac{1}{\gamma}$. For an electrically-long transmission line, where $L > \frac{4}{\gamma}$, (i.e. length is greater than four times of adjustment distance) we have the following scenario,

$$e^{\gamma L} \gg 1 \gg e^{-\gamma L}$$

Hence, for electrically-long sections, Supplementary Equations S7–S9, S11–S13 reduce to the following:

$$Y_E = \frac{2}{Z_0 e^{\gamma L}} \tag{14}$$

$$\frac{Y'}{2} = \frac{1}{Z_0} \tag{15}$$

$$I_E = \frac{E}{Z} \tag{16}$$

$$U_A = -U_B = -\frac{E}{\gamma} \tag{17}$$

$$U(x) = Ue^{-\gamma(L-x)} - Ue^{-\gamma x}, \text{where } |U_A| = |U_B| = U = \frac{E}{\gamma} \tag{18}$$

TABLE 4 Parameters of synthetic magnetic field variation and analytic electric field solution.

m	$A^{(m)}$ (nT)	$\Phi^{(m)}$ (°)	$\tau_m = \frac{1}{f^{(m)}}$ (min)	$\lvert E_a^{(m)} \rvert$ (mv/km)	$\phi^{(m)} + \theta^{(m)}$ (°)
1	200	10	180	10.35	20.66
2	90	20	80	4.86	25.0
3	30	30	36	1.65	31.22
4	17	40	15	0.95	36.88
5	8	50	8	0.45	41.93
6	3.5	60	3	0.19	36.37

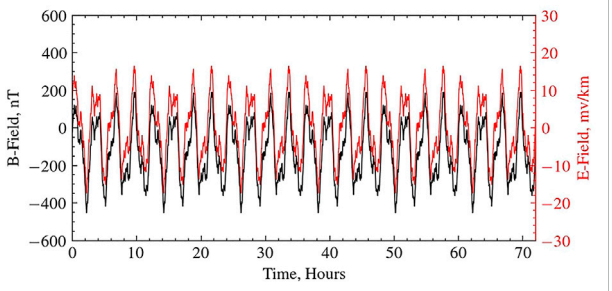


FIGURE 4
Synthetic magnetic field variations (in black) and analytic electric field solution (in red). Example *Python* notebook for generating the synthetic magnetic field is uploaded in Zenodo [21].

For an electrically-short section, where the physical length is less than the adjustment distance, i.e., $L < \frac{1}{\gamma}$, we have the following scenario,

$$e^{\pm\gamma L} \approx 1 \pm \gamma L$$

Hence, for electrically-short sections, Equations (S7-S9) from the Supplementary Equations S11–S13 reduce to the following:

$$Y_E = \frac{1}{Z_0 \gamma L} \tag{19}$$

$$\frac{Y'}{2} = 0 \tag{20}$$

$$I_E = \frac{E}{Z} \tag{21}$$

$$U_A = -U_B = -\frac{LE}{2} \tag{22}$$

$$U(x) = \frac{U}{2L}(2 + \gamma L)(2x - L), \text{where } |U_A| = |U_B| = U = \frac{LE}{2} \tag{23}$$

In the following we will use these analytical solutions to validate our methods.

2.4.2 Synthetic magnetic data and analytical solution to induced electric field

To test the calculation of geoelectric fields using the model, we need a specified geomagnetic field variation that we can use to generate an exact analytical solution for the geoelectric field that will be calculated. From Fourier's theorem, we know that any function can be represented as a sum of cosine and sine functions, and from studies of geomagnetic disturbances, we know that the disturbance amplitude generally decreases with increasing frequency [13,22]. To provide a synthetic magnetic field variation that reproduces this behaviour, we define a test magnetic field variation as the sum of six sine functions (properties are listed in Table 4) as follows:

$$B(t) = \sum_{m=1}^{6} A^{(m)} \sin\left(2\pi f^{(m)} t + \Phi^{(m)}\right) \tag{24}$$

where: $A^{(m)}$, $T^{(m)} = \frac{1}{f^{(m)}}$, and $\Phi^{(m)}$ are amplitude, period, and phase associated with the mth frequency component of the signal. Figure 4 shows the synthetic sea-surface magnetic field data $[B(t)]$ in black.

Following Boteler et al. [22], we know the analytic 1) solution of the electric field is given by:

$$E_a(t) = \sum_{m=1}^{6} |T_x^{(m)}| A^{(m)} \sin\left(2\pi f^{(m)} t + \Phi^{(m)} + \theta^{(m)}\right) \tag{25}$$

where: $|T_x^{(m)}|$ and $\theta^{(m)}$ are the amplitude and phase of each frequency component of the test transfer function. Figure 4 shows the analytic solution of the seafloor electric field data $[E_a(t)]$ in red, utilizing the transfer function showing in Figure 3. The amplitude and phase of each frequency component of the analytic solution are also listed in Table 4 along with the magnetic field values.

2.5 Module 05: Cable impact

During geomagnetic disturbances, a submarine cable will experience an emf equal to the integrated electric field along the

cable. For a cable route that is divided into *S* sections (in Module 01), the emfs calculated for each section [Eq. 2 in Module 02] are summed to give the total emf induced in the cable:

$$\mathcal{E}_C = \sum_{j=1}^{S} \mathcal{E}_j \tag{26}$$

In addition the geomagnetic induction in the seawater results in changes to the Earth potentials at the ends of the cables [Eq. 10 in Module 03]. In Module 2.5 these outputs from Modules 2.2 and 2.3 are combined to give the total voltage experienced by the cable power feed equipment:

$$U_C = \mathcal{E}_C + U_A - U_B \tag{27}$$

where: $\mathcal{E}_C$ is the total induced emf in the cable, U_A and U_B are the Earth potentials at cable end A, and B, respectively.

Module 05 stores the outputs of the simulations at various stages, such as T_x of different cable sections, *U*, *J*, *Y* from nodal analysis, $\mathcal{E}_j$, $\mathcal{E}_C$, and U_C, in the form of '.csv' (comma separated value) files, which can later be used for further analysis, comparison study, sensitivity analysis, and to produce analysis images.

3 Software specifications

This section introduces the *Python* software, its specifications, inputs, and output formats that computes the induced geoelectric field and voltages in the submarine cables associated with geomagnetic field variations. The software needs an input control JSON file that describes the operations, source of the input magnetic data, cable sections geometry and its properties, Ocean-Earth conductivity model, and output storage locations. In the following subsections, we describe the structure of the input control JSON file and its various components, layered conductivity model, and calculations of seafloor transfer function.

3.1 Input control java script object notation

JavaScript Object Notation is a lightweight data-interchange format that has been used for input and configuration by various software across different platforms and is easy for both humans and machine to read/parse and write/generate. JSON is built on two structures: 1) a collection of name/value pairs, alternatively realized as an object, record, structures, dictionary, hash table, keyed list, or associative array in different programming languages; 2) an ordered list of values alternatively realized as an array, vector, list, or sequence.

We have used JSON to provide control instructions to the model. One example structure of the Control JSON is shown in Listing 1.

```
{
    "sid": 1,
    "opcode": 0,
    "Bfield": {
        "structure": {
            "Am": [200, 90, 30, 17, 8, 3.5],
            "Tm_min": [180, 80, 36, 15, 8, 3],
            "Phim": [10, 20, 30, 40, 50, 60],
            "T_hours": 72.0
        },
        "tapering": 0.1
    },
    "cable": {
        "cable_sections": [
            {
                "sec_id": 0,
                "len_km": 1000,
                "elec_params": {
                    "earth_model": "DB1",
                    "ocean_depth": 100,
                    "ocean_resistivity": 0.25
                }
            }
        ]
    },
    "out_dir": "tmp/out/{sid}/"
}
```

Listing 1. Example Input Control JSON. A few example input control JSONs can be found here.

The parameters shown in Listing 1 are described in Table 5. In the listing the submarine cable related details are given under 'cable' parameter, which describes the cables into multiple cable section under 'cable_sections' parameter. Each entry in cable section holds the details about the cable segment, such as length of cable or the location information of cable section ends, and electrical properties *etc.* The electrical properties hold the information about the thickness and resistivity details of the Ocean-Earth layers. In addition, each cable section also holds information about the INTERMAGNET station name, which is used to extract B-field data and compute induced E-field and cable voltage using the DSTL model. This example Control JSON creates synthetic magnetic field data (see Section 2.4.2 for details) and transfer functions based on the electrical properties of the Ocean-Earth structure, along the cable section, provided by the *cable. cable_sections[0]. elect_params* JSON field. The *cable_sections[i]* fields also provide physical information on the cable, such as effective length (*len_km*) or Northward/Eastward geodetic lengths of the cable sections or geodetic location information of the cable route. In-built intelligence of the software can parse any of this information to estimate effective length of the cable. For a detailed description of all the available JSON fields, please refer to the Supplementary Document.

3.2 Compatibility & operating system

The software is written in Python-3.10 language, thus it is compatible with any *Python* code or interpretor or distribution.

TABLE 5 Description of the parameters presented in Control JSON Listing 1.

JSON parameter	Type	Description
sid	Integer	Simulation ID
opcode	Integer [0/1]	Operation ID
Bfield.structure.Am	Array	Magnitude (in nT) for *m*th component
Bfield.structure.Tm_min	Array	Period (in minutes) for *m*th component
Bfield.structure.Phim	Array	Phase (in °) for *m*th component
Bfield.structure.T_hours	Float	Total hours of the time-series (B_s)
Bfield.tapering	Float	Tapering coefficient
cable.cable_sections[i].sec_id	Integer	Cable section ID.
cable.cable_sections[i].len_km	Float	Cable section length in km
cable.cable_sections[i].elec_params.earth_model	String	Name of the Earth conductivity model
cable.cable_sections[i].elec_params.ocean_depth	Float	Ocean water depth in meters
cable.cable_sections[i].elec_params.ocean_resistivity	Float	Ocean water reisitivity in Ωm
out_dir	String	Output directory/folder

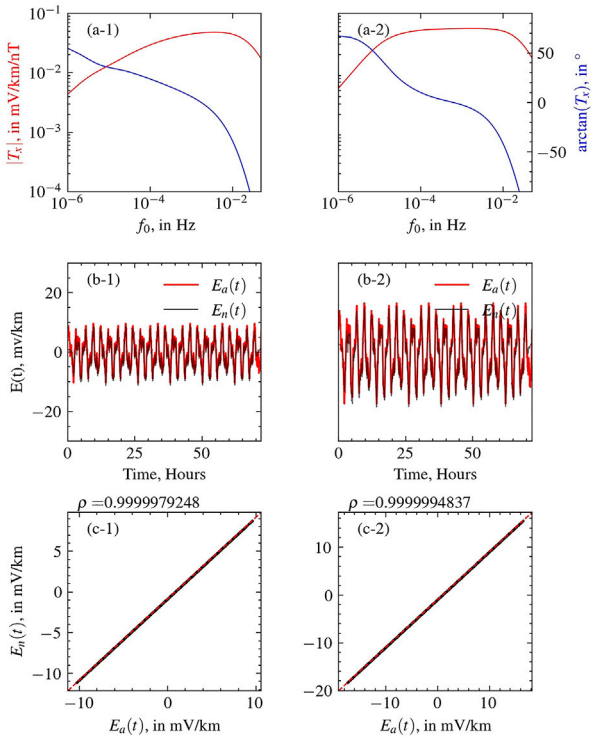


FIGURE 5
Model validation using following layered Ocean-Earth conductivity models: 1) Uniform and 2) Atlantic model described in Table 2. From top to bottom rows present: **(A)** Amplitude and phase spectrum of the transfer function, T_x. **(B)** Time series of analytically (red) and numerically (black) estimated induced electric field. **(C)** Correlation analysis between analytically ($E_a(t)$) and numerically ($E_n(t)$) calculated induced electric field. The correlation coefficient (ρ) between $E_a(t)$ and $E_n(t)$ is provided in the top left corner of the panel. Example *Python* notebook for this experiment is uploaded in Zenodo [21].

However, there are ways to invoke/embed the model functionalities from other languages. Specifically, codes written in Python-3.10 can be embedded into C/C++. Due to its complicated embedding structure, we encourage users to invoke the software from the Python-3.*.* environment. In addition, the code consumes less memory, computational unit, and can run in Unix, Windows, and MAC operating systems.

3.3 Helper python-packages used in the code

The majority of analysis and visualization was completed with the help of free, open-source software tools such as numpy [23], scipy [24], matplotlib [25], IPython [26], pandas (e.g. [27, 28]).

3.3.1 Calculation of seafloor transfer function (T_x) using BEZpy

The calculation of induced geoelectric fields during geomagnetic storms is a long studied problem. The transfer function described in Eq. 2 requires computation of effective reflection coefficients for each layer of the Earth to calculate the seafloor impedance. The calculation of effective reflection coefficients for layered Earth structure can be done using recursive function call. BEZpy [29,30] is primarily implemented for analysis of magnetic (B), electric (E), and impedance (Z) data within a geophysical framework. This library contains routines for calculating the geoelectric field from the geomagnetic field in multiple ways. BEZpy implements the recursive function procedure to compute effective reflection coefficients and impedance at the

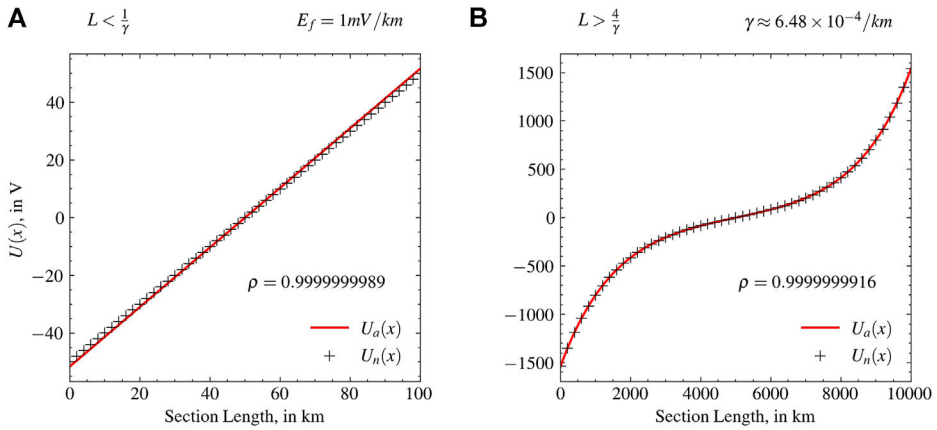


FIGURE 6
Distribution of voltage, estimated using theoretical approximation (red) and computational model (black "+"), along the **(A)** electrically-short and **(B)** electrically-long Ocean-Earth sections. The correlation coefficient between theoretical and numerical estimates are provided in each panel. L and $\frac{1}{\gamma}$ are the physical length and adjustment distance of the cable, respectively. Example *Python* notebook for this experiment is uploaded in Zenodo [21].

seafloor. The BEZpy *Python* code is released under a BSD-3 license, hence, we used this module as a plugin into our *Python* code, to simplify the effective seafloor impedance (Z) calculation.

4 Results

This section presents the outputs, namely, transfer functions T_x, induced emf $\mathcal{E}_C$, total voltage U_C for the following cases: 1) uniform and Atlantic Ocean-Earth conductivity model, 2) electrically-long and electrically-short Ocean-Earth sections, 3) effects of variations in seawater depth, d, on the transfer function, T_x and 4) effects of variations in seawater depth, d, on transmission line parameters. We used the synthetic magnetic field data presented in Section 2.4.2 for all the above-listed case study simulations. In addition, we link the GitHub code base for sample code and example Input Control JSON.

4.1 Uniform, and Atlantic Ocean-Earth conductivity model

To validate the model performance, we compare the estimations of the induced seafloor electric fields (E_f) from the model with corresponding analytical solutions for two different Ocean-Earth Conductivity models. For the analysis, we have used the following conductivity models: 1) uniform: consisting of equal σ for all the layers and 2) the Atlantic conductivity model presented in Table 2. Figure 5 presents the comparisons to validate the model forecast using synthetic magnetic field data (refer Section 2.4.2). From top to bottom of Figure 5, the rows present a) transfer function, b) induced electric field estimates from the model and their analytical solutions, and c) correlation analysis between the electric field estimates. For all the cases, the model is able to reproduce the analytical solution (refer to ρ provided in panels (c-1 ~2)). Note that, electric fields estimated using the model ($E_n(t)$) presented in panels (b-1 ~2) are tapered around their edges, which is due to the tapering of the input magnetic field. The tapering reduces the spurious frequency components in the data and produces a smooth output. Thus, while conducting the correlation analysis we removed the edges from both $E(t)$ estimates. The correlation coefficient (ρ) is shown in the top left corners in the bottom row (panels (c-1 ~2)). From correlation analysis we find that there is a 1 in 10^6 part difference between the numerical estimates and analytical solution, indicating a high confidence in the model calculations. This comparative analysis shows that the model is sensitive to the Ocean-Earth layered conductivity structure.

4.2 Electrically-long and electrically-short Ocean-Earth sections

In this case study, we validate Ocean-Earth potential estimates from the model against the standard theoretical solution presented in Section 2.4.1. Figure 6 presents the outputs for 1) electrically-short and 2) electrically-long Ocean-Earth sections. For this analysis we used the Ocean-Earth layered conductivity model described in Table 2, which has the

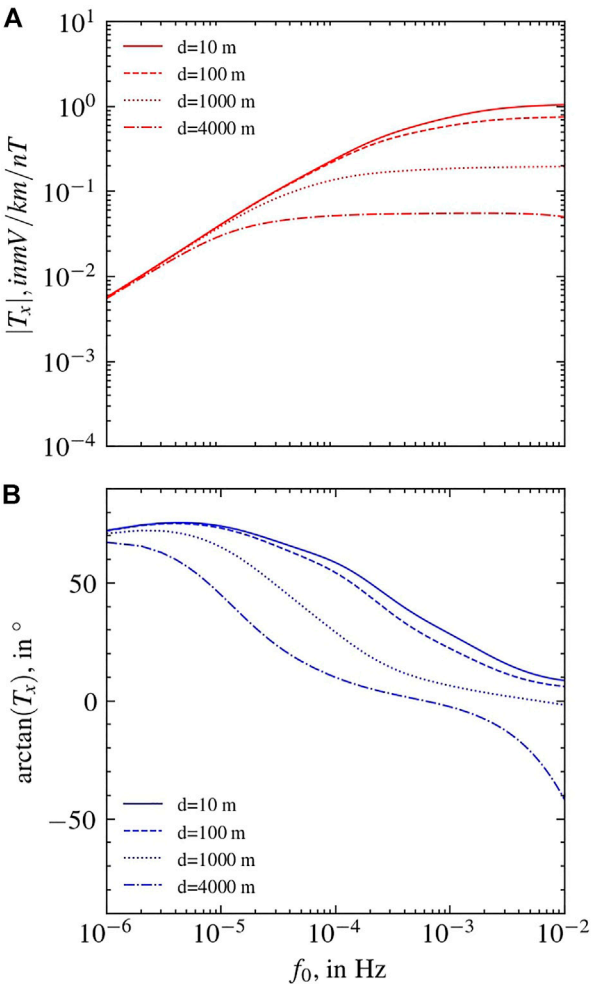


FIGURE 7
(A) Amplitude and **(B)** phase of the transfer functions T_x for different seawater depths **(D)**. Example *Python* notebook for this experiment is uploaded Zenodo [21].

amplitude and phase spectrum of the transfer function. We use an input electric field of E_f = 1 mV/km. The Earth potential, estimated from the model (in black "+"), is overlaid on top of the analytical solutions (in red). The correlation analysis indicates a 1 in 10^9 part difference between the numerical estimates and analytical solution for both the cases under consideration indicating a high confidence in the modeling process.

4.3 Effects of variations in seawater depth, *d*, on the transfer function, T_x

Among the various model parameters, seawater depth is the one that has the most influence on the seafloor electric field. Here, we conduct a sensitivity analysis, presented in Figure 7, that shows how change in sea-water depth impacts the amplitude (top panel) and phase (bottom panel) of the transfer function. This shows that changing the seawater depth can change the transfer function amplitudes by orders of magnitude. In addition, we observe the following features in the amplitude spectrum: 1) tapering for higher frequency increases with seawater depth, indicating a water shielding effect, 2) at lower frequencies all amplitude spectra converge and taper off to lower values, indicating an attenuation for the waves with very long periods, irrespective of the seawater depth. From the phase spectrum we observe a stable progression of phase change from 70° to 0° for the frequency range under consideration.

4.4 Effects of variations in seawater depth, *d*, on transmission line parameters Z_0 and $\frac{1}{\gamma}$

Variations in seawater depth also impact the transmission line model parameters (refer to Supplementary Document: Module 03). In Figure 8 we present the variations of characteristic impedance Z_0 (in red '+') and adjustment distance $\frac{1}{\gamma}$ (in blue squares) of the transmission line model for a fixed layered Earth presented in Table 2 (except the top row). Note that, as expected (refer to Supplementary Document: Module 03) the characteristic impedance (Z_0) and propagation constant (γ) of the transmission line model decreases with increasing seawater depth (d). Supplementary Equations S1–S6 indicate that an increase in seawater depth increases the horizontal conductance C and, subsequently, reduces the series impedance Z, which finally reduce the characteristic impedance (Z_0) and propagation constant (γ) of the transmission line. Supplementary Equations S1–S6 also indicate that the relation between series conductance C and Z_0, γ is hyperbolic in nature, hence, we observe the same trend in the output presented by Figure 8.

5 Discussion

Here, we presented a computational model that is capable of estimating geomagnetically induced electric fields at the seafloor and associated voltages in submarine cables and that can be used to examine the magnetic induction effects due to various types of geomagnetic field variations. We describe the *Python* software that does all the computation, all the inputs required to run the model, and show how the shielding effects of the seawater and the Ocean/Earth conductivity structure affects the results. To test the modeling software we make calculations using a synthetic magnetic perturbation as input, and validate the model output by comparison with analytic solutions. In future work we plan to use the model to make calculations of the

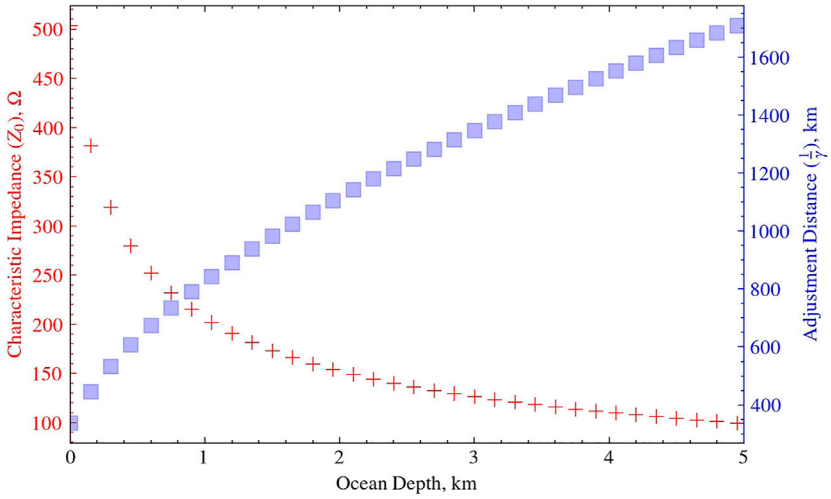


FIGURE 8
(Red) Characteristic impedance (Z_0) and (Blue) adjustment distance ($\frac{1}{\gamma}$) of the transmission line model for different seawater depths **(D)**. Example *Python* notebook for this experiment is uploaded in Zenodo [21].

induced voltages in the TAT-8 trans-Atlantic cable and compare these to measurements made during the March 1989 geomagnetic storm.

The magnetic induction model consists of two primary parts, first, a transfer function that calculates geomagnetically induced electric field at the seafloor; second, a calculation of seafloor Earth potential due to induced electric field using a transmission line model of the Ocean/Earth conductivity structure. There are a number of different types of magnetic field variations: all will induce an electric field in submarine cables, but not all of them have the characteristics to produce electric fields that may be important for submarine cable operation. From the transfer functions we can directly understand that the layered Ocean-Earth acts like a band-pass filter, that allows magnetic disturbance with specific periods to induce electric field under the sea. This helps us to infer what types of magnetic phenomena may affect submarine cable infrastructures. Thus this model can be used to study the effects of many types of geomagnetic disturbances on submarine cables, including storm sudden commencements (SSC) [31], magnetic storm main phase [32], and magnetic substorms [33].

5.1 Assumptions and limitations of the model

All the model assumptions and limitations contribute to the uncertainty of the model outputs. The one-dimensional (1-D) model assumes that the conductivity variations of the Ocean-Earth layers are only a function of depth and there are no lateral spatial conductivity variations within a given section. The seafloor geoelectric field calculations are made in a "piecewise" fashion [34] for each section, ignoring the sections on either side. The change between section are then handled by the transmission-line modeling. In addition, we assume that conductivity of the layers (including seawater and Earth) do not changes between each cable sections, which also contributes to the model uncertainty. One can reduce the section lengths to more closely follow the variations in ocean depth, but this violates the assumptions of 1-D induction theory, which assumes variations of conductivity and layer thickness in the east-west and north-south direction have negligible impact on the overall calculation. Moreover, calculating the Ocean-Earth layered conductivity model using LITHO1.0 interpolates thicknesses of layers at the middle point of a cable segment, which produces an interpolation error that also contributes to the model uncertainty. Note that, the cable route is split into sections to capture the large-scale changes in the cable depth, so that there are only small changes in depth within each section.

Several factors act together to influence the size of the voltage variations encountered by submarine cables. The magnetic field at the sea surface has a spectrum that falls off with frequency, whereas the transfer function between the seafloor electric field and surface magnetic field increases with frequency for shallow sea depths such as on the continental shelf, while the response to deep ocean seafloor is flatter. In the deep ocean, higher frequencies do not contribute as much to the electric fields on the seafloor. Another important factor is the spatial extent of the magnetic disturbances. Trans-oceanic submarine cables span large distances that cover many time zones, so the magnetic disturbances at opposite ends of a cable are not

necessarily the same. Besides, it is also hard to find magnetometer stations (observations) near the submarine cable route across the ocean. Thus, the model has to interpolate sparsely available magnetometer data along the cable route, which is another source of uncertainty in the calculations. These factors all needed to be examined on a case-by-case basis for specific cables. This is planned to be a topic of future work using the modeling tools described in this paper.

6 Conclusion

In this study, we present a computational model to estimate the geomagnetically induced electric field and voltage in submarine cables. This includes calculation of the transfer function relating the seafloor electric field to the surface magnetic field variations. Cross-ocean modeling describes a way to estimate induced voltage along the submarine cable due to the seafloor geoelectric field. A synthetic geomagnetic field has been defined, comprising six frequency components, to validate the model performance against theoretical outputs. We validated the model calculation processes by comparison of model results with analytic solutions for two different Ocean-Earth conductivity models and electrically-long and electrically-short Ocean-Earth sections. The induced geoelectric field at the seafloor and induced voltage within the cable are directly related to the seawater depth. The sensitivity study indicates that the seawater layer acts as a shield in the induction process: the greater the ocean depth, the smaller the seafloor geoelectric field. In addition, we found that the model is sensitive to the Ocean-Earth layered conductivity structure. Thus, the geomagnetic effect on submarine cables depends on the intensity of the geomagnetic disturbance at that location on the globe as well as the seawater depth and Ocean-Earth layered conductivity structure along the cable route. The model can be used to study the effects of different types of geomagnetic phenomena including SSC, storm main phase, substorms *etc.* The present modeling incorporates various simplifying assumptions, such as the use of 1-D Earth conductivity models. Possible improvements to the modeling could include use of 3-D finite element models to more accurately represent the variations of ocean depth and conductivity structure along the cable route.

Data availability statement

The datasets used in this study are synthetically produced inside the code and can be found in the GitHub code repository identified by the following https://github.com/shibaji7/submarine_cable_modeling.git. The latest version of the code is published in Zenodo [21]. The example code and IPython notebooks are also available in Zenodo and GitHub repositories. The crust and lithospheric model of the Earth LITHO1.0 model [9] is available on IRIS DMC Data Products website [35].

Author contributions

SC contributed to code writing, model validation, analysis, and drafting the manuscript. DB, XS, and MH contributed to conception and design of the study. BM contributed to conductivity modeling. XW, GL, and JB provided feedback on manuscript organization. All authors contributed to manuscript revision, read, and approved the submitted version.

Funding

SC, XS, MH, and JB were supported by NASA 80NSSC19K0907 and 80NSSC21K1677. DB was supported by Natural Resources Canada contribution number 20220247. SC and XS also thank the National Science Foundation for support under grants AGS-1935110. BM was supported by a Mendenhall Postdoctoral Fellowship through the U.S. Geological Survey.

Acknowledgments

Authors acknowledge Josh Rigler, Krissy Lewis, Ryan Gold, and Janet Carter for facilitating USGS internal review. Any use of trade, firm, or product names are for descriptive purposes only and does not imply endorsement by the U.S. Government.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fphy.2022.1022475/full#supplementary-material

References

1. Carter L, Burnett D, Drew S, Hagadorn L, Marle G, Bartlett-Mcneil D, et al. *Submarine cables and the oceans – connecting the world*. Kenya: UNEP-WCMC Biodiversity (2009).

2. Boteler D, Pirjola R, Nevanlinna H. The effects of geomagnetic disturbances on electrical systems at the Earth's surface. *Adv Space Res* (1998) 22:17–27. Terrestrial Relations: Predicting the Effects on the Near-Earth Environment. doi:10.1016/S0273-1177(97)01096-X

3. Prescott G. Observations made at boston, mass., and vicinity, in article on the great auroral exhibition of august 28th to september 4th, 1859. *Amer J Sci Arts* (1860) 92–7.

4. Prescott G. History, Theory and Practice of the electric telegraph *(ticknor and fields)*. London: the University of Michigan (1866).

5. Boteler D. The super storms of august/september 1859 and their effects on the telegraph system. *Adv Space Res* (2006) 38:159–72. The Great Historical Geomagnetic Storm of 1859: A Modern Look. doi:10.1016/j.asr.2006.01.013

6. Anderson CW. Magnetic storms and cable communications, solar system plasma Physics *(kennel, C)* (1978).

7. Lanzerotti LJ. Geomagnetic induction effects in ground-based systems. *Space Sci Rev* (1983) 34:347–56. doi:10.1007/BF00175289

8. Medford LV, Lanzerotti LJ, Kraus JS, Maclennan CG. Transatlantic Earth potential variations during the march 1989 magnetic storms. *Geophys Res Lett* (1989) 16:1145–8. doi:10.1029/GL016i010p01145

9. Pasyanos ME, Masters TG, Laske G, Ma Z. Litho1.0: An updated crust and lithospheric model of the Earth. *J Geophys Res Solid Earth* (2014) 119:2153–73. doi:10.1002/2013JB010626

10. Cox C. Electromagnetic induction in the oceans and inferences on the constitution of the Earth. *Geophys Surv* (1980) 4:137–56. doi:10.1007/BF01452963

11. Medford LV, Meloni A, Lanzerotti LJ, Gregori GP. Geomagnetic induction on a transatlantic communications cable. *Nature* (1981) 290:392–3. doi:10.1038/290392a0

12. Dawson TW, Weaver JT, Raval U. B-polarization induction in two generalized thin sheets at the surface of a conducting half-space. *Geophys J Int* (1982) 69:209–34. doi:10.1111/j.1365-246X.1982.tb04945.x

13. Boteler DH, Pirjola RJ. Magnetic and electric fields produced in the sea during geomagnetic disturbances. *Pure Appl Geophys* (2003) 160:1695–716. doi:10.1007/s00024-003-2372-6

14. Peltier A, Chulliat A. On the feasibility of promptly producing quasi-definitive magnetic observatory data. *Earth Planets Space* (2010) 62:e5–8. doi:10.5047/eps.2010.02.002

15. Wang S, Constable S, Rychert CA, Harmon N. A lithosphere-asthenosphere boundary and partial melt estimated using marine magnetotelluric data at the central middle atlantic ridge. *Geochem Geophys Geosyst* (2020) 21:e2020GC009177. doi:10.1029/2020gc009177

16. Rychert CA, Tharimena S, Harmon N, Wang S, Constable S, Kendall JM, et al. A dynamic lithosphere–asthenosphere boundary near the equatorial mid-atlantic ridge. *Earth Planet Sci Lett* (2021) 566:116949. doi:10.1016/j.epsl.2021.116949

17. Attias E, Evans RL, Naif S, Elsenbeck J, Key K. Conductivity structure of the lithosphere-asthenosphere boundary beneath the eastern north American margin. *Geochem Geophys Geosyst* (2017) 18:676–96. doi:10.1002/2016GC006667

18. Grayver AV. Global 3-d electrical conductivity model of the world ocean and marine sediments. *Geochem Geophys Geosyst* (2021) 22:e2021GC009950. doi:10.1029/2021gc009950

19. Boteler DH, Pirjola RJ. Numerical calculation of geoelectric fields that affect critical infrastructure. *Int J Geosciences* (2019) 10:930–49. doi:10.4236/ijg.2019.1010053

20. Horton R, Boteler D, Overbye TJ, Pirjola R, Dugan RC. A test case for the calculation of geomagnetically induced currents. *IEEE Trans Power Deliv* (2012) 27:2368–73. doi:10.1109/TPWRD.2012.2206407

21. [Dataset] Chakraborty S (2022). shibaji7/submarine_cable_modeling: Geomagnetic Induction in Submarine Cables. doi:10.5281/zenodo.7055321

22. Boteler DH, Pirjola RJ, Marti L. Analytic calculation of geoelectric fields due to geomagnetic disturbances: A test case. *IEEE Access* (2019) 7:147029–37. doi:10.1109/ACCESS.2019.2945530

23. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. *Nature* (2020) 585:357–62. doi:10.1038/s41586-020-2649-2

24. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat Methods* (2020) 17:261–72. doi:10.1038/s41592-019-0686-2

25. Hunter JD. Matplotlib: A 2D graphics environment. *Comput Sci Eng* (2007) 9:90–5. doi:10.1109/MCSE.2007.55

26. Perez F, Granger BE. Ipython: A system for interactive scientific computing. *Comput Sci Eng* (2007) 9:21–9. doi:10.1109/MCSE.2007.53

27. McKinney W. Data structures for statistical computing in Python. In: S van der Walt J Millman, editors. *Proceedings of the 9th Python in science conference* (2010). p. 56–61. doi:10.25080/Majora-92bf1922-012

28. Millman KJ, Aivazis M. Python for scientists and engineers. *Comput Sci Eng* (2011) 13:9–12. doi:10.1109/MCSE.2011.36

29. [Dataset] Lucas G. *greglucas/bezpy*. Hawaii: First release of bezpy (2020). doi:10.5281/zenodo.3765861

30. Love JJ, Lucas GM, Kelbert A, Bedrosian PA. Geoelectric hazard maps for the mid-atlantic United States: 100 year extreme values and the 1989 magnetic storm. *Geophys Res Lett* (2018) 45:5–14. doi:10.1002/2017GL076042

31. Zhang JJ, Yu YQ, Wang C, Du D, Wei D, Liu LG. Measurements and simulations of the geomagnetically induced currents in low-latitude power networks during geomagnetic storms. *Space Weather* (2020) 18:e2020SW002549. doi:10.1029/2020sw002549

32. Rajput VN, Boteler DH, Rana N, Saiyed M, Anjana S, Shah M. Insight into impact of geomagnetically induced currents on power systems: Overview, challenges and mitigation. *Electric Power Syst Res* (2021) 192:106927. doi:10.1016/j.epsr.2020.106927

33. Viljanen A, Tanskanen EI, Pulkkinen A. Relation between substorm characteristics and rapid temporal variations of the ground magnetic field. *Ann Geophys* (2006) 24:725–33. doi:10.5194/angeo-24-725-2006

34. Marti L, Yiu C, Rezaei-Zare A, Boteler D. Simulation of geomagnetically induced currents with piecewise layered-Earth models. *IEEE Trans Power Deliv* (2014) 29:1886–93. doi:10.1109/TPWRD.2014.2317851

35. Trabant C, Hutko AR, Bahavar M, Karstens R, Ahern T, Aster R. Data products at the IRIS DMC: Stepping stones for research and other applications. *Seismological Res Lett* (2012) 83:846–54. doi:10.1785/0220120032

36. Wang X, Boteler D, Pirjola R. Distributed-source transmission line theory for modeling the coast effect on geoelectric fields. *IEEE Trans Power Deliv* (2022).

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 04 November 2022
DOI 10.3389/fspas.2022.1005061

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Karl Laundal,
University of Bergen, Norway
Alexandra Fogg,
Dublin Institute for Advanced Studies (DIAS), Ireland
Katie Herlingshaw,
The University Centre in Svalbard, Norway

*CORRESPONDENCE
Kyle R. Murphy,
kylemurphy.spacephys@gmail.com

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 27 July 2022
ACCEPTED 02 September 2022
PUBLISHED 04 November 2022

CITATION
Murphy KR, Rae IJ, Halford AJ, Engebretson M, Russell CT, Matzka J, Johnsen MG, Milling DK, Mann IR, Kale A, Xu Z, Connors M, Angelopoulos V, Chi P and Tanskanen E (2022), GMAG: An open-source python package for ground-based magnetometers.
Front. Astron. Space Sci. 9:1005061.
doi: 10.3389/fspas.2022.1005061

COPYRIGHT
© 2022 Murphy, Rae, Halford, Engebretson, Russell, Matzka, Johnsen, Milling, Mann, Kale, Xu, Connors, Angelopoulos, Chi and Tanskanen. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

GMAG: An open-source python package for ground-based magnetometers

Kyle R. Murphy[1,2]*, I. Jonathan Rae[2], Alexa J. Halford[3], Mark Engebretson[4], Christopher T. Russell[5], Jürgen Matzka[6], Magnar G. Johnsen[7], David K. Milling[8], Ian R. Mann[8], Andy Kale[8], Zhonghua Xu[9], Martin Connors[10], Vassilis Angelopoulos[5], Peter Chi[11] and Eija Tanskanen[12]

[1]Independent Researcher, Thunder Bay, Ontario, Canada, [2]Department of Maths, Physics and Electrical Engineering, Northumbria University, Newcastle Upon Tyne, United Kingdom, [3]NASA Goddard Space Flight Center, Greenbelt, MD, United States, [4]Department of Physics, Augsburg University, Minneapolis, MN, United States, [5]Institute of Geophysics and Planetary Physics, University of California, Los Angeles, CA, United States, [6]GFZ German Research Centre for Geosciences, Potsdam, Germany, [7]Tromsø Geophysical Observatory, UiT the Arctic University of Norway, Tromsø, Norway, [8]Departement of Physics, University of Alberta, Edmonton, AB, Canada, [9]Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, United States, [10]Department of Physics and Astronomy, Athabasca University, Athabasca, AB, United States, [11]Department of Earth, Planetary and Space Sciences, University of California, Los Angeles, CA, United States, [12]Sodankylä Geophysical Observatory, University of Oulu, Oulu, Finland

Magnetometers are a key component of heliophysics research providing valuable insight into the dynamics of electromagnetic field regimes and their coupling throughout the solar system. On satellites, magnetometers provide detailed observations of the extension of the solar magnetic field into interplanetary space and of planetary environments. At Earth, magnetometers are deployed on the ground in extensive arrays spanning the polar cap, auroral and sub-auroral zone, mid- and low-latitudes and equatorial electrojet with nearly global coverage in azimuth (longitude or magnetic local time—MLT). These multipoint observations are used to diagnose both ionospheric and magnetospheric processes as well as the coupling between the solar wind and these two regimes at a fraction of the cost of *in-situ* instruments. Despite their utility in research, ground-based magnetometer data can be difficult to use due to a variety of file formats, multiple points of access for the data, and limited software. In this short article we review the Open-Source Python library GMAG which provides rapid access to ground-based magnetometer data from a number of arrays in a Pandas DataFrame, a common data format used throughout scientific research.

KEYWORDS

ground-based, magnetometers, open-source, python, pandas, dataframe

1 Introduction

Ground-based magnetometers have been used to study Earth's geomagnetic environment for over a century (e.g., Birkeland 1908). In initially controversial but ground-breaking work Birkeland (1908) postulated that perturbations of the Earth's magnetic field, observed by ground-based magnetometers, were driven by ionospheric currents connected to a system of currents flowing along Earth's magnetic field lines into and out of the polar ionosphere. It wasn't until the launch of spacecraft with onboard magnetometers in the 1960's and 1970's (e.g., Zmuda et al., 1966; Zmuda et al., 1967, and Iijima and Potemra 1976, Iijima and Potemra 1978) that Birkeland's theory of field aligned currents was proven correct, paving the way for the study of heliophysics and studies solar wind-magnetosphere-ionosphere coupling. Today, the ionospheric current systems studied by Birkland are known as the auroral electrojets and the field aligned current systems he first postulated are now termed Birkeland currents.

Following Birkeland's seminal work, ground-based magnetometers have become a fundamental tool in magnetospheric and ionospheric research providing the ability to study the spatio and temporal dynamics of Earth's geomagnetic field driven by the coupling of the solar wind-magnetosphere-ionosphere systems. For example, Heppner (1954) demonstrated that the brightening of auroral arcs was strongly associated with perturbations of the Earth's nightside magnetic field, highlighting a link between the aurora and the auroral electrojets during what is now known as the auroral substorm (Akasofu, 1964). Coordinated research utilizing multiple ground-based magnetometers expanded on these studies and allowed researchers to probe the spatial-temporal dynamics of processes coupling the magnetosphere and ionosphere. These studies have allowed researchers to determine the strength and location of the auroral electrojets (e.g., Rostoker and Phan, 1986), investigate the evolution of auroral substorms (e.g., McPherron, 1970), and define global magnetic indices to classify geomagnetic activity and which can be used as input to research and forecasting models (e.g., Kp, Dst, auroral electrojet indices; see Davis and Sugiura 1966; Iyemori 1990; Love and Remick, 2007, Kauristie, 2017, Yamazaki et al., 2022).

In the late 1970's and early 1980's the community began developing extended arrays and networks of ground-based magnetometers in response to International Solar-Terrestrial Physics Science (ISTP) (Jones 1986, see also Rostoker et al., 1995). Today, several magnetometer arrays exist spanning the polar cap, through the auroral oval, down to the equatorial electrojet and across the globe in longitude providing nearly continuous observations for as long as three solar cycles (e.g., Mann et al., 2008; Russell et al., 2008; Tanskanen 2009; Chulliat et al., 2017). These arrays allow researchers to conduct detailed studies of the spatial-temporal dynamics of the solar wind-magnetosphere-ionosphere system on global scales. This includes, for example: statistical (e.g. Mathie and Mann, 2000; Murphy et al., 2011) and event based studies of Ultra-low Frequency waves (e.g., Rae et al., 2019) which can be used in radiation belt modeling (e.g., Mann et al., 2016; Ozeke et al., 2017; Ma et al., 2018); studies of the spatial distribution and expansion of waves (e.g. Milling et al., 2008; Murphy et al., 2009), field aligned currents (e.g., Pulkkinen et al., 2003; Weygand et al., 2011), the substorm current wedge (e.g., Lester et al., 1983; Cramoysan et al., 1995), and geomagnetically induced currents (e.g., Pulkinen et al., 2005; Pulkinen et al., 2017; Smith et al., 2022); analysis of field line resonances (e.g., Mann et al., 2002; Rae et al., 2005); remote sensing of *in-situ* plasma properties (e.g., e.g., Mann et al., 2002; Rae et al., 2005); and magnetoseismology of ionospheric disturbances (e.g., Waters et al., 1996; Chi and Russell, 2005; Chi et al., 2013; Kale et al., 2009). For more examples see Mann et al. (2008), Russell et al. (2008), Menk and Waters (2013), Engebretson and Zesta (2017), and Murphy et al. (2022a)). Despite their utility in magnetospheric and ionospheric physics, the data from ground-based magnetometers can be difficult to use due to limited software, multiple file formats (e.g., text and CDF), varying sources and methods for obtaining the data, and varying temporal resolutions.

Recent community efforts to develop Open-Source software packages for scientists, including AstroPy (Robitaille et al., 2013; Price -Whelan et al., 2018), SunPy (Barnes et al., 2020), the THEMIS Data Analysis Software (TDAS) and its extension to the Space Physics Environment Data Analysis Software (SPEDAS) and pySPEDAS (Angelopoulos et al., 2019) have been extremely beneficial to the community. These resources reduce barriers to data access and provide well developed software that does not rely on the cost prohibitive licenses of programming languages such as MATLAB and IDL (see https://heliopython.org/projects/for an extensive list of Python packages for Heliophysics). In this short paper we detail the Open-Source Python package GMAG (Murphy, 2022b). GMAG is a simple package that provides access to over 200 ground-based magnetometers from multiple arrays in a Pandas DataFrame, a common data format used throughout data analysis and scientific research. We discuss installation of the package, the package modules, illustrate several examples, highlight the GMAG website, and discuss potential future extensions of the package.

2 GMAG

2.1 Python module

The GMAG Python package (Murphy, 2022b) provides access to just over 200 ground-based magnetometers (Figure 1) that are part of 12 arrays (Table 1). The GMAG directory and file structure is based on typical Python packages. The top-level directory is the GitHub repository GMAG. Within that directory is the Python package `gmag`, the docs directory, which contains the GMAG GitHub Pages website (below), and the notebooks directory which contains several examples. Figure 2 shows the structure of the GitHub directory and the GMAG package with short file descriptions. The main modules are the base `gmag` module and the `arrays` submodule (discussed below). GMAG can be installed by downloading or cloning the GitHub repository (https://github.com/kylermurphy/gmag) and using the single line of code `pin`

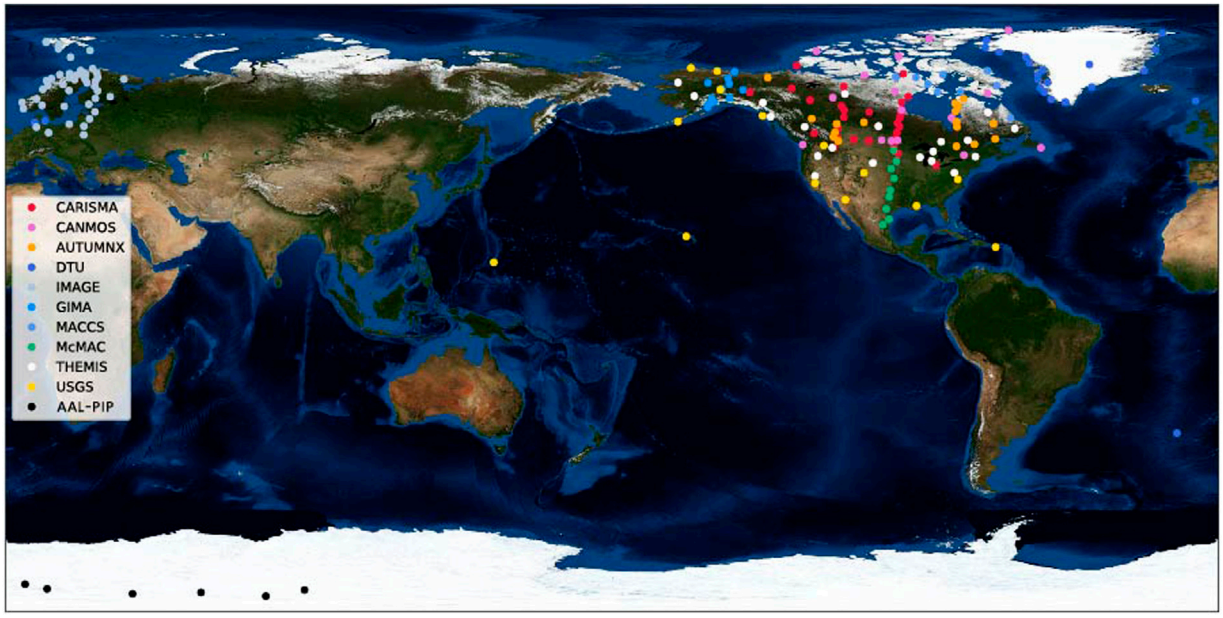


FIGURE 1
The geographic location of ground-based magnetometers that the GMAG package can download and load data for. The stations are color coded by their parent array (see legend and Table 1).

TABLE 1 The ground-based magnetometer arrays that can be accessed by the GMAG package.

Array	PI/institution	GMAG module	Secondary module
AUTUMN and AUTUMN X	Martin Connors/Athabasca University	themis	N/A
CANMOS	Natural Resources Canada	themis	N/A
CANOPUS	Ian Mann/University of Alberta	canopus	themis
CARISMA	Ian Mann/University of Alberta	carisma	themis
DTU	Anna Willer/Technical University of Denmark	themis	N/A
GIMA	Don Hampton/University of Alaska	themis	N/A
IMAGE	Liisa Juusola/Several	image	themis
MACCS	Mark Engebretson/Augsburg University	themis	N/A
McMAC	Peter Chi/University of California, Los Angeles	themis	N/A
AAL-PIP	Zhonghua Xu/Virginia Tech	themis	N/A
THEMIS EPO and GBO	Christopher Russell/University of California, Los Angeles	themis	N/A
USGS	US Geological Survey	themis	N/A

`install`. from the terminal and within the GMAG directory, this also installs all GMAG requirements. Once installed the `gmagrc_example` should be renamed `gmagrc` and the variable `data_dir`, defining the local directory to download data, should be set by the user based on their preferences. GMAG is platform and operating system independent.

The main `gmag` package provides a set of modules for parsing the configuration file and loading station data for various ground-based magnetometers (e.g., geographic, and altitude adjusted corrected geomagnetic coordinates - AACGM Burrell et al., 2018). The configuration file defines the local directory for downloading data, the variable `data_dir` and the web addresses of the data servers accessed by GMAG the variables `??_http` where `??` is shorthand for the server. Though GMAG provides access to eleven ground-based arrays it uses only three data servers, the CARISMA, IMAGE, and THEMIS data servers. This is because the THEMIS server provides direct access to stations from all the magnetometer arrays listed in Table 1, which also provides redundancy for the CARISMA and IMAGE arrays. In addition to the configuration routines the main package contains the `utils` module, a set of utility functions for reading in station data, namely the

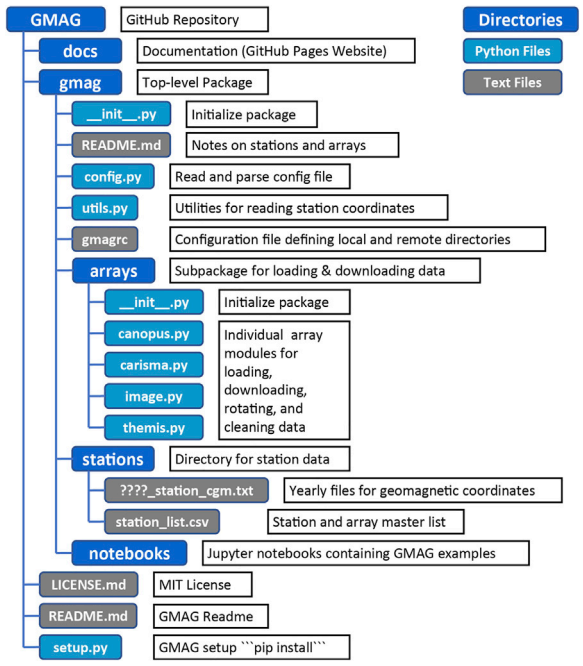


FIGURE 2
Structure of the GMAG GitHub Repository and gmag module.

coordinates for each station, and parent magnetometer array. The AACGM are calculated yearly using the Python packages `igrf12` (Hirsch, 2021) and `aacgmv2` (Burrell et al., 2018) and stored as text files in the format `gmag/stations/????_station_cgm.txt`, where `????` is the year, so they can be rapidly loaded. Note, installation of the `igrf12` and `aacgmv2` packages can be difficult; thus, the coordinate text files are generated using the notebook `notebook/convert_coords.ipynb` then saved to text files so to avoid difficulties of installing `igrf12` and `aacgmv2`.

The `arrays` subpackage is the bulk of the `gmag` package and contains the functionality to list local and remote files, and download, load, clean, and rotate magnetometer data for the arrays listed in Table 1. The `arrays` subpackage is broken-down into four array specific modules, the `carisma`, `image`, `themis`, and `canopus` modules. The code is structured in this way as data is retrieved for each module from a different data server in a different file format. The `carisma`, `canopus`, and `image` modules load data in geographic XYZ coordinates where X is north, Y is east, and Z is vertical. These data are scrubbed using the `clean()` routine which replaces bad or poor quality data with not-a-number. Poor quality data are identified by flags (CARISMA and CANOPUS) or specific numbers (IMAGE) in the data files; currently no other cleaning (e.g., removal of spikes) is performed. These data are then rotated using the stations declination into local magnetic coordinates HDZ. Here H is the north, D is the east, and Z is the vertical magnetic field in nT. The declinations are loaded with `gmag.utils.load_station_coor()` from the yearly AACGM files `stations/????_station_cgm.txt` described above. The data loaded *via* the `themis` module has undergone extensive processing and is typically returned in local geomagnetic coordinates HDZ and so no additional processing or scrubbing is done to data loaded *via* the `themis` module. The `canopus` module loads CANOPUS data (CARISMA pre 1 April 2005) manually downloaded from the CARISMA website, as an easily accessible CANOPUS data server is not currently available, and performs the same data rotation and cleaning as the `carisma` module. The `carisma`, `image`, and `themis` modules also have the utility to download data files. Finally, the `carisma`, `image`, and `canopus` modules return data in both geographic and local geomagnetic coordinates (XYZ, HDZ).

For each array module data is loaded using the `load()` function. Data files are first searched for locally and then downloaded if no local files are found. For simplicity all the `load()` functions have the same basic parameters and will load observations from multiple stations for extended periods. The `load()` functions return both the magnetometer data and metadata for the loaded stations in well-defined Pandas DataFrames (e.g., clear, and descriptive column names). The `download()` functions allow users to download data without loading the data. GMAGs array modules and functions are summarized in Table 2. The CARISMA module downloads and loads 1 s resolution data, the IMAGE module downloads 10 s resolution. The resolution of data loaded by the THEMIS module varies with magnetometer array. In each case the returned metadata contains the temporal resolution of each station loaded. Note the CARISMA module does not currently load the 8 Hz data or the searchcoil data.

It is important to note that there are a multitude of different data structures in Python, both built in and available *via* various packages. Python's built-in data structures (lists, dictionaries, tuples, and sets) generally do not offer the utility required for scientific research and data analysis, e.g., data structure arithmetic such as adding or multiplying two series/vectors/arrays or matrix multiplication. Other packages, such as Numpy, Astropy, Sunpy, SpacePy, and xarray, offer data structures whose utility extends beyond Python's built-in data structures and are more appropriate for research and data analysis. All these packages have their benefits. GMAG uses Pandas DataFrames as opposed to these other libraries as a DataFrame is ideal for handling both time-series and tabular data, further Pandas DataFrames provide built in functionality to rapidly plot and visualize data. The next section illustrates a number of use cases for the GMAG library.

2.2 GMAG, examples

The `load()` function in the `carisma`, `canopus`, `image`, and `themis` modules loads data from multiple magnetometer stations over extended periods. In general, analysis of short time periods or a small number of stations can be easily accomplished using the `load()` functions (e.g., case studies). For studies using a larger number of stations

TABLE 2 List of function available for each of the `arrays` module. * function part of the THEMIS module.

Functions	
list_files()*	Create list of filenames, local and remote directories for loading/downloading
download()*	Download files
load()*	Load data
clean()	Replace bad data with not-a-number
rotate()	Rotate data from geographic XYZ to local geomagnetic HDZ coordinates

TABLE 3 Example of the returned DataFrame; in this case the `carisma` module. In the case of CARISMA data, the flag column identifies ok (.) and bad (x) data (bad data are replaced with not-a-number).

t (index)	ISLL_X	ISLL_Y	ISLL_Z	ISLL_flag	ISLL_H	ISLL_D
2019-01-04 0:00:00	12098.82	52.05	57679.85	.	12098.80	57.26
2019-01-04 0:00:01	12098.84	52.03	57679.85	.	12098.82	57.25
2019-01-04 0:00:02	12098.84	52.04	57679.83	.	12098.82	57.25

over extended periods (e.g., storms or statistical studies) data can first be downloaded using the `download()` function and subsequently loaded. The load and download functions use the station code/abbreviations. These codes can be found in the master station list (`stations/station_list.csv`), loaded *via* the `utils` module `load_station_coor()` and `load_station_geo()` routines (see example) or on the GMAG website (described below).

The Python code below demonstrates how to load data from each of the `arrays` modules (see comments for additional details).

```
# load CARISMA module
import gmag.arrays.carisma as carisma

# load 2 days of data from the ISLL and PINA
# using ndays to define the time span
df, meta = carisma.load(['ISLL','PINA'],'2012-01-01',ndays=2)
# using edate to define the end date
df, meta = carisma.load(['ISLL','PINA'],'2012-01-01',edate='2012-01-03')

# load IMAGE module
import gmag.arrays.image as image
# load 21 days of data from the AND station
df, meta = image.load('AND','2012-01-01', ndays=21)

# load THEMIS module
import gmag.arrays.themis as themis
# load 22 days of THEMIS data from the KUUJ station
# force download even if local files exist
df, meta = themis.load('KUUJ',sdate='2012-01-01',ndays=22, dl=True, force=True)

#load CANOPUS module
#load a single day of data from the ISLL station
import gmag.arrays.canopus as canopus
df, meta = canopus.load('ISLL',sdate='2001-01-01',ndays=1)
```

In each case the `load()` functions return data in a Pandas DataFrame. The index of the DataFrame is the timestamp and each column is a component of the magnetic field for each station loaded, in either geographic coordinates (XYZ), local geomagnetic coordinates (HDZ), or both. We use the timestamp as the index as this allows for rapid plotting in Pandas as well as combining observations with different cadences into a single DataFrame. The `load()` function also returns station metadata. Table 3 and Table 4 show examples of the data and metadata DataFrames returned by the load functions.

The Magnetometer data DataFrame can be easily searched, accessed, and manipulated using a mix of Python's and Panda's indexing and DataFrame methods. DataFrames have the additional utility of being able to rapidly plot data using Pandas built in plotting and Matplotlib. The DataFrame can also be used with other Python packages such as Numpy, to rapidly produce publication quality plots or analyze the data using SciPy's signal processing library or scikit-learn's machine learning library. Figure 3 and the following block of code demonstrate how to rapidly load and plot data using the `themis` module during a substorm observed between 0530 and 0630 UT on 9 April 2011.

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
```

10.3389/fspas.2022.1005061

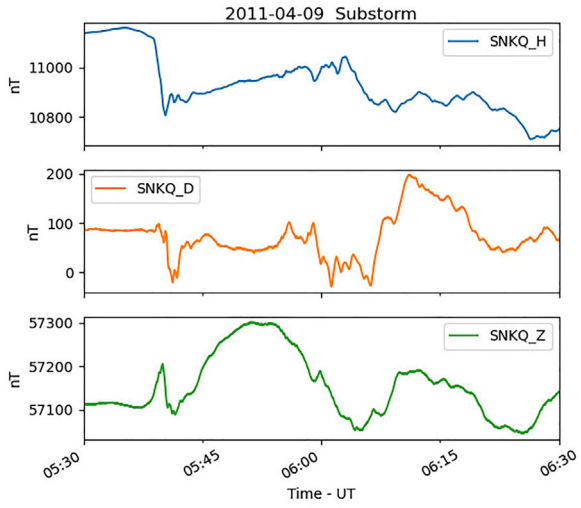


FIGURE 3
Example of a simple plot produced using Pandas built in plotting functionality highlighting a substorm observed by the SNKQ ground-based magnetometer station loaded using the `themis` module.

```
import gmag.arrays.themis as themis

# define start and end dates for plotting
sdate = '2011-04-09 05:30:00'
edate = '2011-04-09 06:30:00'

# load data
th_dat, th_meta = themis.load(['SNKQ'],
sdate,ndays=1)

# plot all data in the DataFrame between
# sdate and edate
th_dat[sdate:edate].plot(ylabel='nT',
xlabel='Time - UT', figsize=[6,6],
subplots=True)
plt.title(sdate[0:11]+' Substorm',y=3.35)
```

More complex multi-station plots of specific components can also be rapidly generated with a few lines of code using list comprehension and chaining together Pandas and Numpy methods. Two examples are shown in Figure 4 and Figure 5 along with the accompanying code.

Figure 4 shows the H component magnetic field (mean removed) from select CARISMA magnetometers in a multi-panel plot during a substorm observed on 17 July 2005. Figure 5 shows a stacked plot of a large amplitude Ultra-low Frequency (ULF) wave observed along the CARISMA Churchill line on 5 November 2011.

```
# Plot multi-panel plot of the H component
# magnetic field for select CARSIMA stations
# import required modules
import gmag.arrays.carisma as carisma
import numpy as np
import matplotlib.pyplot as plt
# define start and end date for plotting and
loading
# assume a single day is loaded
sdate = '2005-07-17 08:00:00'
edate = '2005-07-17 12:30:00'
# define component to be plotted
comp='H'
# load data
car_dat,car_meta=carisma.load(['GILL',
'ISLL','PINA','RABB','FSMI','FSIM',
'MCMU'],sdate)
```

TABLE 4 Example metadata returned by the load functions.

Array	CARISMA
code	ISLL
name	Island Lake
latitude	53.86
longitude	265.34
cgm_latitude	63.03
cgm_longitude	335.62
declination	−0.02
lshell	4.86
mlt_midnight	6.88
mlt_ut	17.12
year	2019
Time Resolution	1.0
Coordinates	Geographic North—X, East—Y, Vertical Down—Z, Geomagnetic North—H, East—D, Vertical Down—Z
PI	Ian Mann
Institution	University of Alberta

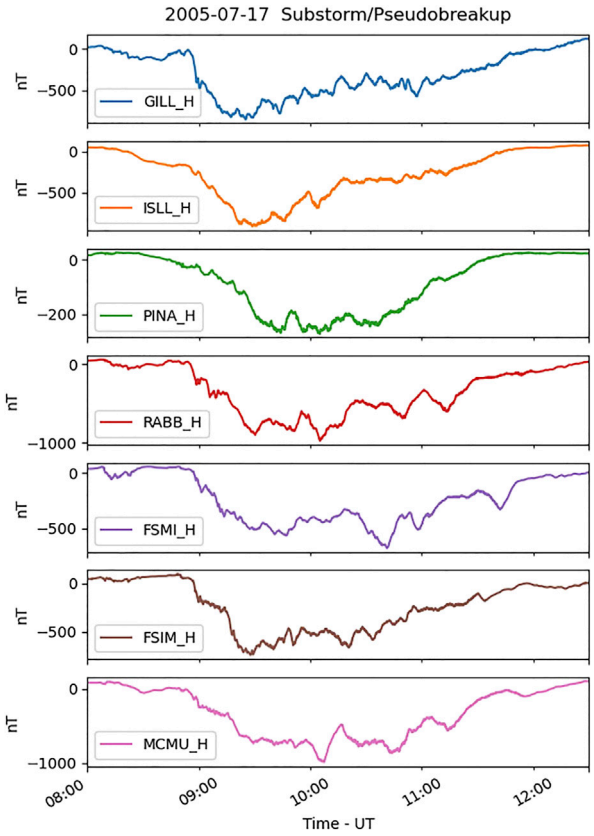


FIGURE 4
A multi-panel plot highlighting a substorm observed by select CARISMA magnetometer stations. Data was loaded using the `carisma` module and plotted using Pandas.

```
# find the correct columns of the DataFrame
p_col = [col for col in car_dat.columns if
col[-1] == comp]
# plot the DataFrame between sdate and edate
# plot only p_col columns and subtract the
mean from each column
# before plotting
car_dat[sdate:edate]
[p_col].subtract(car_dat[p_col].mean()).
plot(ylabel='nT', xlabel='Time - UT',
figsize=[6,10],subplots=True)
plt.title(sdate[0:11]+' Substorm/
Pseudobreakup',y=8.25)

# import required modules
import gmag.arrays.carisma as carisma
import numpy as np
import matplotlib.pyplot as plt
# define start and end date for plotting
# use start date for loading data
sdate = '2014-11-05 13:25:00'
edate = '2014-11-05 14:25:00'

# define component for plotting
comp='H'

# load data
car_dat, car_meta=carisma.load(['PINA',
'ISLL','GILL','FCHU','RANK'],sdate)

# find the columns from the loaded DataFrame
that have comp
# in the title, these are the columns that
will be plotted
p_col = [col for col in car_dat.columns if
col[-1] == comp]

# determine the shift to apply to each time
series so that they don't
# overlap

# the shift is determined using the DataFrame
returned by the describe()
# method which stores the DataFrame stats
including max and min of each column
# only use columns from p_col and values
between the start and end of plotting
# defined by sdate and edate
# the shift in the y direction is defined by 1.5
times the range of the series
y_shift = np.array([(val['max']-val
['min'])/1.5 for col_h, val in car_dat[sdate:
edate][p_col].describe().iteritems()])

# the cumsum() method determines the
cumalitative sum up
# to each index
# the cumsum() ensures timeseries don't
overlap
y_shift = (y_shift-y_shift.min()).cumsum()

#plot p_col columns of the data frame between
sdate and edate
# subtract the mean from each time series and
apply the y-shit
car_dat[sdate:edate]
[p_col].subtract(car_dat[sdate:edate]
[p_col].mean()-y_shift).plot(ylabel='nT',
xlabel='Time - UT', figsize=[6,10])
plt.title(sdate[0:11]+' ULF Wave')
```

GMAG also allows extended periods of data to be downloaded using any of the `download()` functions. This allows users to download data without also loading the data into memory.

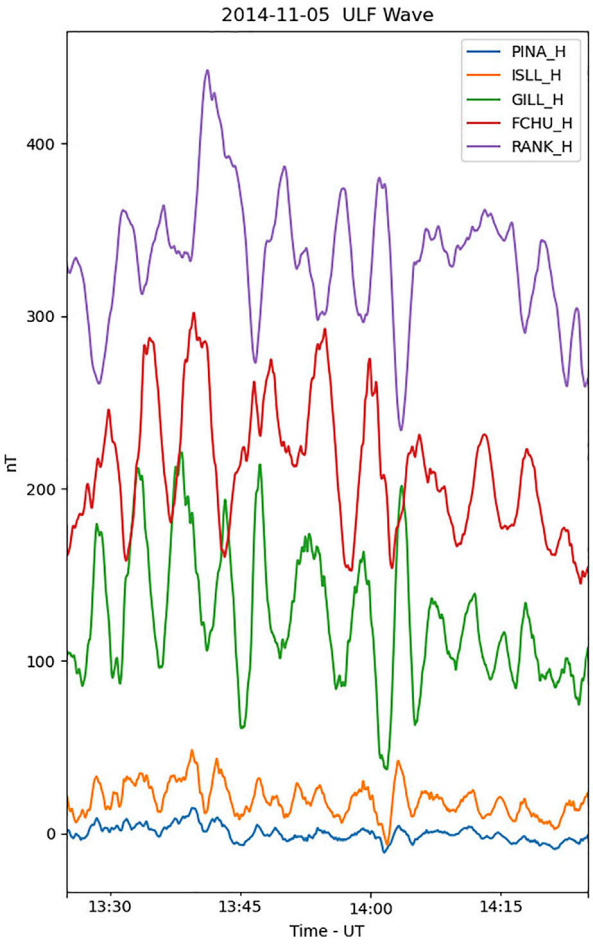


FIGURE 5
Multi-station stacked plot of a ULF wave observed by select CARISMA magnetometer stations that are a part of the Churchill line. The plot was generated using Pandas and Numpy.

```
# download CARISMA
# data from ISLL
import gmag.arrays.carisma as carisma
carisma.download('ISLL','2012-01-01',
ndays=21)

# download IMAGE data
import gmag.arrays.image as image
image.download('AND','2012-01-01',
ndays=21)

# force download THEMIS data
import gmag.arrays.themis as themis
themis.download('KUUJ',sdate='2012-01-01',
ndays=21, force= True)
```

Finally, the `utils` module provides two simple routines `load_station_coor()` and `load_station_geo()` for loading geographic and altitude adjusted corrected geomagnetic data for magnetometer stations. Both functions return data in DataFrame and in each case the `col` and `param` keywords can be used to filter a given DataFrame column by the `param` variable. Several examples are shown below. These functions can also be used in conjunction with the `load()` functions to load all stations from a particular magnetometer array.

```
from gmag import utils
#load geomagnetic data
#load all CARISMA station data for 2002
car_stn =
utils.load_station_coor(param='CARISMA',
col='array', year=2002)

#load GILL data from 2012
gill_stn =
utils.load_station_coor(param='GILL',
col='code', year=2012)

#load all data from 2012
all_stn =
utils.load_station_coor(param='ALL',
year=2012)

#load station geographic data
#load all CARISMA data
geo_stn =
utils.load_station_geo(param='CARISMA',
col='array')

#load all stations
all_stn =
utils.load_station_geo(param='ALL')
```

A more complex example shows how to identify stations in a particular region of interest and then load data from those stations. This example can be modified to identify stations within a range of any combination of coordinates (e.g., L shell, longitude). Note, the GMAG GitHub repository includes a notebooks folder with several Jupyter Notebooks showing examples of the GMAG code, including those here.

```
import pandas as pd
from gmag import utils
import gmag.arrays.carisma as carisma
import gmag.arrays.image as image
import gmag.arrays.themis as themis

#find all stations between 18-24 MLT
#and L shells 6-8 during an event observed
#on 2018-01-01/04:00:00 UT
date = pd.to_datetime('2018-01-01/04:00:00')
mlt_min = 18
mlt_max = 24
l_min = 6
```

```
l_max = 8

#load all stations for date of interest
all_stn = utils.load_station_coor(param='*',
year=date.year)

#calculate MLT of the stations for the date
#MLT at 0 UT is stored in the mlt_ut column
#mlt is then mlt at 0 UT plus current UT hour
all_stn['mlt'] = (all_stn['mlt_ut']+date
.hour) % 24

#create masks for the mlt and lshell regions
mlt_mask = (all_stn['mlt'] >= mlt_min) &
(all_stn['mlt'] <= mlt_max)
l_mask = (all_stn['lshell'] >= l_min) &
(all_stn['lshell'] <= l_max)

#create masks for the arrays
car_mask = all_stn['array'] == 'CARISMA'
img_mask = all_stn['array'] == 'IMAGE'

#identify stations from each array
car_stn = all_stn[car_mask & mlt_mask &
l_mask]
img_stn = all_stn[img_mask & mlt_mask &
l_mask]
# ~ is bitwise negation, not image or carisma
(1 if 0, 0 if 1)
thm_stn = all_stn[~img_mask & ~car_mask &
mlt_mask & l_mask]

#create an empty DataFrames
mag_meta = pd.DataFrame()
mag_data = pd.DataFrame()

#loop through stations and load data
for stn in [car_stn, img_stn, thm_stn]:
  #skip if no stations were identified
  if stn.shape[0] == 0:
      continue
  elif stn['array'].iloc[0] == 'CARISMA':
  l_dat, l_meta = carisma.load(car_stn
  ['code'],date,ndays=1,drop_flag=True)
  elif stn['array'].iloc[0] == 'IMAGE':
  l_dat, l_meta = image.load(img_stn
  ['code'],date,ndays=1,drop_flag=True)
  else:
  l_dat, l_meta = themis.load(thm_stn
  ['code'],date,ndays=1)

# add loaded data to
mag_data =
mag_data.join(l_dat,how='outer')
mag_meta = pd.concat([mag_meta,l_meta],
axis=0, sort=False, ignore_index=True)
```

2.3 GMAG website

The GMAG website is located at https://kylermurphy.github.io/gmag/. The website provides simple and up to date documentation for the GMAG package as well as quick access to array and magnetometer station information. The array page includes a list of magnetometer arrays that can be accessed by GMAG, links to webpages and data access, as well as the acknowledgements and a link to the Terms and Conditions of Use (if available). The stations page includes a map of all stations, a table of the station's geographic coordinates, and links to tables of the station's AACGM coordinates. The tables are searchable allowing researchers to rapidly find information on each magnetometer array and their stations (e.g., L-shells). Finally, the examples page includes various usage examples for the package.

3 Conclusion and future possibilities

Ground-based magnetometers are a corner stone of heliophysics research. Compared to flight hardware, ground-based magnetometers are less cost-prohibitive, and are easier to deploy and maintain. This allows extensive networks of ground-based magnetometers to be deployed to study the global spatial-temporal dynamics of ULF waves (Mann et al., 2002), ionospheric and magnetospheric current systems (Weygand et al., 2011), substorms (Murphy et al., 2009), storms (Rae et al., 2019), radiation belt dynamics (Ozeke et al., 2017), or magnetosphere plasma distributions (Chi et al., 2005). Generally, these networks are deployed by individual PIs and research groups from various research institutions and Universities across the world. This allows for many stations to be deployed around the globe; however, this also generally means that no common ground-based magnetometer framework exists, for example, data format, cadence, access point and method, that allows researchers to easily and rapidly access high-resolution near real-time data.

Recent work has begun unifying ground-based magnetometer data, providing a common data format, access point, and software to access the data. The recently updated PySPEDAS Python package now includes access to ground-based magnetometer data from the THEMIS data server *via* PyTplot (the Python counterpart to SPEDAS's IDL `tplot` routine) (Angelopoulos et al., 2019). The SuperMAG program provides ground-based magnetometer data from several magnetometer arrays in a common data format (Gjerlov, 2012); however, at the time of this writing, access to high cadence data *via* a simplified Python package is limited. Further, SuperMAG receives high resolution data on delayed schedule. Both the PySPEDAS and SuperMAG Python packages were developed along similar timelines as GMAG and the three packages offer similar,

albeit slightly different functionality; for example, the underlying data structures utilized. Nevertheless, the GMAG Python package described here has ample utility. GMAG returns data in a common data format, the Pandas DataFrame which allows for rapid plotting using Pandas and Matplotlib, and analysis *via* Pandas, Numpy, Scipy, Scikit-learn, and a host of other Python libraries. Further, GMAG provides the utility to download CARISMA and IMAGE magnetometer data from the source; this provides researchers with a more extensive dataset at higher cadence then those currently provided by PySPEDAS, and SuperMAG. This is especially useful for researchers who may want to use near real-time ground-based magnetometer data for space weather forecasting and mitigation or those looking to study a more recent geomagnetic event not available in the SuperMAG or PySPEDAS datasets.

GMAG was developed in parallel with ongoing research to provide rapid access to an extensive set of ground-based magnetometers. As such it is a relatively simple module for accessing data which can be used in conjunction with other code and packages for more sophisticated analysis. While GMAG's core functionality is achieved in its current incarnation there are several future developments that would enhance the package. This includes, for example:

- A wrapper routine for the magnetometer array `load` modules that would allow users to load data from any magnetometer regardless of which array the station was a part of.
- A set of formal documentation (e.g., Read the Docs, https://readthedocs.org/).
- Addition of other data sets and magnetometer arrays such as, SuperMAG, INTERMAGNET, MagIE (the Magnetometer Network of Ireland), SAMBA (South American Meridional B-Field Array), the 8 Hz CARISMA data, the searchcoil CARISMA data.
- Routines to analyse ground-based magnetometer data such as ULF wave power for use in radiation belt models (e.g., Ozeke et al., 2015), substorm onset timing and signal propagation (e.g., Murphy et al., 2009), and inference of plasma density using the cross-phase technique (e.g. Chi et al., 2005).
- Maintenance and updates as more stations and magnetometer arrays become available.

With regard to future developments, it is important to note GMAG is an Open-Source package and we encourage members of the community to not only utilize the package but submit issues, suggest new functionality, and include their improvements in the base package using the typical GitHub collaborative development model of forking the repository and generating a pull request.

In summary, GMAG provides a set of simple yet robust routines to download, load, and clean data from over 200 ground-based magnetometers a part of 12 arrays. This enables researchers to rapidly access high fidelity data, providing a basis for studying the system science of the solar wind-magnetosphere-ionosphere system from the unique vantage point of ground-based magnetometers.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: http://data.carisma.ca/, https://space.fmi.fi/image/www/data_download.php?, http://themis.ssl.berkeley.edu/data/themis/

Author contributions

KRM is responsible for the core development and maintenance of the GMAG Python package and the conceptualization and writing of the paper. IJR and AJH provided discussion in the development of the GMAG package. IJR provided editorial comments ME is PI of the MACCS array and provided editorial comments. CTR and VA are PIs of the THEMIS array; CTR provided editorial comments. JM provided editorial comments and helpful discussion and is the GFZ PI of the IMAGE array. MGJ provided comments and discussion and is the UiT PI of the IMAGE array. DKM provided editorial comments and discussion. IRM, DKM and AK are responsible for the continued operation of the CARISMA array. ZX is the PI of the AAL-PIP array. MC is PI of the AUTUMN and AUTUMN X arrays. PC is PI of the McMAC array. ET is the Sodankylä Geophysical Observatory PI of the IMAGE array. A significant amount of time, energy and work goes into securing funding for, deploying, and maintaining magnetometer arrays, in addition to processing and providing data to the community. Without these efforts this work would not exist.

Funding

KRM is partially supported by NERC grant NE/V002554/2. IJR is partially funded by NERC grants NE/P017185/2, NE/V002554/2, and STFC grant ST/V006320/1. AJH is partially funded by the SPI ISFM.

Acknowledgments

KRM is thankful to the PIs and teams of all magnetometer arrays. The data they provide is invaluable to the magnetospheric community. This includes: MC and Chris Russell and the rest of the AUTUMN/AUTUMNX team for use of the GMAG data. Natural Resources Canada and the Geological Survey of Canada as the source of the data CANMOS data. Ian Mann, David

Milling and the rest of the CARISMA team for use of CARSIMA data. CARISMA is operated by the University of Alberta, funded by the Canadian Space Agency. Anna Willer, the PI of the DTU magnetometers. Magnetometer data from the Greenland Magnetometer Array were provided by the National Space Institute at the Technical University of Denmark (DTU Space). Don Hampton, the PI of the GIMA magnetometer array, data is provided by the Geophysical Institute Magnetometer Array operated by the Geophysical Institute, University of Alaska. ME, Mark Moldwin, and the rest of the MACCS team for use of MACCS data. MACCS is operated by Augsburg University and funded by the U.S. National Science Foundation through grants AGS-2013648 and AGS-2013433. PC for use of the McMAC data and NSF for support through grant ATM-0245139. Chris Russell for use of the GMAG data and NSF for support through grant AGS-1004814. The United States Geological Survey for USGS data, original data provided by the USGS Geomagnetism Program (http://geomag.usgs.gov). The AAL-PIP array (formerly the Polar Experimental Network for Geospace Upper atmosphere Investigations; PENGUIn), PI, ZX, Virginia Tech; this effort is supported by the National Science Foundation.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Akasofu, S. I. (1964). The development of the auroral substorm. *Planet. Space Sci.* 12 (4), 273–282. doi:10.1016/0032-0633(64)90151-5

Angelopoulos, V., Cruce, P., Drozdov, A., Grimes, E. W., Hatzigeorgiu, N., King, D. A., et al. (2019). The space physics environment data analysis system (SPEDAS). *Space Sci. Rev.* 215 (1), 9–46. doi:10.1007/s11214-018-0576-4

Barnes, W. T., Bobra, M. G., Christe, S. D., Freij, N., Hayes, L. A., Ireland, J., et al. (2020). The SunPy project: Open source development and status of the version 1.0 core package. *Astrophys. J.* 890 (1), 68. doi:10.3847/1538-4357/AB4F7A

Birkeland, K. (1908). *The Norwegian aurora polaris expedition, 1902-1903*. Christiania H; Aschelhoug, 1902–1903. doi:10.5962/bhl.title.17857

Burrell, A., Meeren, C. van der, and Laundal, K. M. (2018). *aburrell/aacgmv2*, 1. AACGMV2 2. doi:10.5281/ZENODO.1469697

Chi, P. J., Engebretson, M. J., Moldwin, M. B., Russell, C. T., Mann, I. R., Hairston, M. R., et al. (2013). Sounding of the plasmasphere by mid-continent MAgnetoseismic chain (McMAC) magnetometers. *J. Geophys. Res. Space Phys.* 118 (6), 3077–3086. doi:10.1002/JGRA.50274

Chi, P. J., and Russell, C. T. (2005). Travel-time magnetoseismology: Magnetospheric sounding by timing the tremors in space. *Geophys. Res. Lett.* 32 (18), 1–4. doi:10.1029/2005GL023441

Chulliat, A., Matzka, J., Masson, A., and Milan, S. E. (2017). Key ground-based and space-based assets to disentangle magnetic field sources in the Earth's environment. *Space Sci. Rev.* 206, 123–156. doi:10.1007/s11214-016-0291-y

Cramoysan, M., Bunting, R., and Orr, D. (1995). The use of a model current wedge in the determination of the position of substorm current systems. *Ann. Geophys.* 13, 583–594. doi:10.1007/s00585-995-0583-0

Davis, T. N., and Sugiura, M. (1966). Auroral electrojet activity index AE and its universal time variations. *J. Geophys. Res.* 71 (3), 785–801. doi:10.1029/JZ071i003p00785

Engebretson, M., and Zesta, E. (2017). The future of ground magnetometer arrays in support of space weather monitoring and research. *Space weather.* 15 (11), 1433–1441. doi:10.1002/2017SW001718

Gjerloev, J. W. (2012). The SuperMAG data processing technique. *J. Geophys. Res.* 117(A9), doi:10.1029/2012ja017683

Heppner, J. P. (1954). Time sequences and spatial relations in auroral activity during magnetic bays at College, Alaska. *J. Geophys. Res.* 59 (3), 329–338. doi:10.1029/JZ059I003P00329

Iijima, T., and Potemra, T. A. (1976). Field-aligned currents in the dayside cusp observed by Triad. *J. Geophys. Res.* 81 (34), 5971–5979. doi:10.1029/JA081i034p05971

Iijima, T., and Potemra, T. A. (1978). Large-scale characteristics of field-aligned currents associated with substorms. *J. Geophys. Res.* 83 (A2), 599. doi:10.1029/JA083iA02p00599

Iyemori, T. (1990). Storm-time magnetospheric currents inferred from mid-latitude geomagnetic field variations. *J. Geomagn. Geoelec.* 42 (11), 1249–1265. doi:10.5636/jgg.42.1249

Jones, A. V., McNamara, A. G., McEwan, D. J., Samson, J. C., Cogger, L. L., Creutzber, F., et al. (1986). *CANOPUS an automatic ground-based instrument array to support space projects*, Canopus yellow-book. retrieved from Available at: https://cgsm.ca/doc/CANOPUS_yellow_book.pdf.

Kale, Z. C., Mann, I. R., Waters, C. L., Vellante, M., Zhang, T. L., and Honary, F. (2009). Plasmaspheric dynamics resulting from the Halloween 2003 geomagnetic storms. *J. Geophys. Res.* 114 (8), 1–12. doi:10.1029/2009JA014194

Kaursite, K. (2017), On the usage of geomagnetic indices for data selection in internal field modelling, *Space Sci. Rev.* 206, 61–90. doi:10.1007/s11214-016-0301-0

Lester, M., Hughes, W., and Singer, H. (1983). Polarization patterns of pi 2 magnetic pulsations and the substorm current wedge. *J. Geophys. Res.* 88, 7958–7966. doi:10.1029/JA088iA10p07958

Love, J. J., and Remick, K. J. (2007). Magnetic indices. *Encycl. Geomagnetism Paleomagnetism*, 509–512. doi:10.1007/978-1-4020-4423-6_178

Ma, Q., Li, W., Bortnik, J., Thorne, R. M., Chu, X., Ozeke, L. G., et al. (2018). Quantitative evaluation of radial diffusion and local acceleration processes during GEM challenge events. *JGR. Space Phys.* 1, 1938–1952. doi:10.1002/2017JA025114

Mann, I. R., Milling, D. K., Rae, I. J., Ozeke, L. G., Kale, A., Kale, Z. C., et al. (2008). The upgraded CARISMA magnetometer array in the THEMIS era. *Space Sci. Rev.* 141 (1–4), 413–451. doi:10.1007/s11214-008-9457-6

Mann, I. R., Ozeke, L. G., Murphy, K. R., Claudepierre, S. G., Turner, D. L., Baker, D. N., et al. (2016). Explaining the dynamics of the ultra-relativistic third Van Allen radiation belt. *Nat. Phys.* 12, 978–983. doi:10.1038/nphys3799

Mann, I. R., Voronkov, I., Dunlop, M., Donovan, E., Yeoman, T. K., Milling, D. K., et al. (2002). Coordinated ground-based and Cluster observations of large amplitude global magnetospheric oscillations during a fast solar wind speed interval. *Ann. Geophys.* 20, 405–426. doi:10.5194/angeo-20-405-2002

Mathie, R. A., and Mann, I. R. (2000). A correlation between extended intervals of Ulf wave power and storm-time geosynchronous relativistic electron flux enhancements. *Geophys. Res. Lett.* 27 (20), 3261–3264. doi:10.1029/2000GL003822

McPherron, R. L. (1970). Growth phase of magnetospheric substorms. *J. Geophys. Res.* 75 (28), 5592–5599. doi:10.1029/JA075i028p05592

Menk, F. W., and Waters, C. L. (2013). *Magnetoseismology*. Wiley. doi:10.1002/9783527652051

Hirsch, M. (2021). *space-physics/igrf: Robust build, matlab unit test*. doi:10.5281/ZENODO.5560949

Milling, D. K., Rae, I. J., Mann, I. R., Murphy, K. R., Kale, A., Russell, C. T., et al. (2008). Ionospheric localisation and expansion of long-period Pi1 pulsations at substorm onset. *Geophys. Res. Lett.* 35 (17), L17S20–5. doi:10.1029/2008GL033672

Murphy, K. R., Bentley, S. N., Miles, D. M., Sandhu, J. K., and Smith, A. W. (2022a). "Imaging the magnetosphere–ionosphere system with ground-based and *in-situ* magnetometers," in *Magnetospheric imaging* (Elsevier), 287–340. doi:10.1016/B978-0-12-820630-0.00002-7

Murphy, K. R., Jonathan Rae, I., Mann, I. R., Milling, D. K., Watt, C. E. J., Ozeke, L., et al. (2009). Wavelet-based ULF wave diagnosis of substorm expansion phase onset. *J. Geophys. Res.* 114 (2). doi:10.1029/2008JA013548

Murphy, K. R. (2022b). *kylermurphy/gmag: GMAG Frontiers Initial Release (v1.0.0)*. Zenodo. doi:10.5281/zenodo.6686035

Murphy, K. R., Mann, I. R., Jonathan Rae, I., and Milling, D. K. (2011). Dependence of ground-based Pc5 ULF wave power on F10.7 solar radio flux and solar cycle phase. *J. Atmos. Solar-Terrestrial Phys.* 73 (11–12), 1500–1510. doi:10.1016/j.jastp.2011.02.018

Ozeke, L. G., Mann, I. R., Murphy, K. R., Sibeck, D. G., and Baker, D. N. (2017). Ultra-relativistic radiation belt extinction and ULF wave radial diffusion: Modeling the september 2014 extended dropout event. *Geophys. Res. Lett.* 44, 2624–2633. doi:10.1002/2017GL072811

Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., et al. (2018). The Astropy project: Building an open-science project and status of the v2.0 core package. *Astron. J.* 156 (3), 123. doi:10.3847/1538-3881/aabc4f

Pulkkinen, A., Amm, O., Viljanen, A., Korja, T., Hjelt, S. E., Kaikkonen, P., et al. (2003). Ionospheric equivalent current distributions determined with the method of spherical elementary current systems. *J. Geophys. Res.* 108 (A2), 1–9. doi:10.1029/2001JA005085

Pulkkinen, A., Bernabeu, E., Thomson, A., Viljanen, A., Pirjola, R., Boteler, D., et al. (2017). Geomagnetically induced currents: Science, engineering, and applications readiness. *Space weather.* 15 (7), 828–856. doi:10.1002/2016SW001501

Pulkkinen, A., Lindahl, S., Viljanen, A., and Pirjola, R. (2005). Geomagnetic storm of 29–31 October 2003: Geomagnetically induced currents and their relation to problems in the Swedish high-voltage power transmission system. *Space weather.* 3 (8). doi:10.1029/2004SW000123

Rae, I. J., Donovan, E. F., Mann, I. R., Fenrich, F. R., Watt, C. E. J., Milling, D. K., et al. (2005). Evolution and characteristics of global Pc5 ULF waves during a high solar wind speed interval. *J. Geophys. Res.* 110 (A12), A12211–A12216. doi:10.1029/2005JA011007

Rae, I. J., Murphy, K. R., Watt, C. E. J., Sandhu, J. K., Georgiou, M., Degeling, A. W., et al. (2019). How do Ultra-Low Frequency waves access the inner magnetosphere during geomagnetic storms? *Geophys. Res. Lett.* 46, 10699–10709. doi:10.1029/2019gl082395

Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., et al. (2013). Astropy: A community Python package for astronomy. *Astron. Astrophys.* 558, A33. doi:10.1051/0004-6361/201322068

Rostoker, G., and Duc Phan, T. (1986). Variation of auroral electrojet spatial location as a function of the level of magnetospheric activity. *J. Geophys. Res.* 91 (A2), 1716. doi:10.1029/JA091IA02P01716

Rostoker, G., Samson, J. C., Creutzberg, F., Hughes, T. J., McDiarmid, D. R., McNamara, A. G., et al. (1995). Canopus: A ground-based instrument array for remote sensing the high latitude ionosphere during the ISTP/GGS program. *Space Sci. Rev.* 71 (1–4), 743–760. doi:10.1007/BF00751349

Russell, C. T., Chi, P. J., Dearborn, D. J., Ge, Y. S., Kuo-Tiong, B., Means, J. D., et al. (2008). THEMIS ground-based magnetometers. *Space Sci. Rev.* 141 (1–4), 389–412. doi:10.1007/s11214-008-9337-0

Sibeck, D. G. (1990). A model for the transient magnetospheric response to sudden solar wind dynamic pressure variations. *J. Geophys. Res.* 95, 3755–3771. doi:10.1029/JA095iA04p03755

Smith, A. W., Rodger, C. J., Mac Manus, D. H., Forsyth, C., Rae, I. J., Freeman, M. P., et al. (2022). The correspondence between sudden commencements and geomagnetically induced currents; Insights from New Zealand. *Space Weather* 20, e2021SW002983. doi:10.1029/2021SW002983

Tanskanen, E. I. (2009). A comprehensive high-throughput analysis of substorms observed by IMAGE magnetometer network: Years 1993–2003 examined. *J. Geophys. Res.* 114 (A5). doi:10.1029/2008ja013682

Waters, C. L., Samson, J. C., and Donovan, E. F. (1996). Variation of plasmatrough density derived from magnetospheric field line resonances. *J. Geophys. Res.* 101 (A11), 24737–24745. doi:10.1029/96ja01083

Weygand, J. M., Amm, O., Viljanen, A., Angelopoulos, V., Murr, D., Engebretson, M. J., et al. (2011). Application and validation of the spherical elementary currents systems technique for deriving ionospheric equivalent currents with the North American and Greenland ground magnetometer arrays. *J. Geophys. Res.* 116 (3), 1–8. doi:10.1029/2010JA016177

Yamazaki, Y., Matzka, J., Stolle, C., Kervalishvili, G., Rauberg, J., Bronkalla, O., et al. (2022). Geomagnetic activity index Hpo. *Geophys. Res. Lett.* 49, e2022GL098860. doi:10.1029/2022GL098860

Zmuda, A. J., Heuring, F. T., and Martin, J. H. (1967). Dayside magnetic disturbances at 1100 kilometers in the auroral oval. *J. Geophys. Res.* 72 (3), 1115–1117. doi:10.1029/JZ072I003P01115

Zmuda, A. J., Martin, J. H., and Heuring, F. T. (1966). Transverse magnetic disturbances at 1100 kilometers in the auroral region. *J. Geophys. Res.* 71 (21), 5033–5045. doi:10.1029/JZ071I021P05033

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 14 November 2022
DOI 10.3389/fspas.2022.1001166

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Mario D'Amore,
German Aerospace Center (DLR), Germany
Reinaldo Roberto Rosa,
National Institute of Space Research (INPE), Brazil

*CORRESPONDENCE
Corentin Louis,
corentin.louis@dias.ie

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 22 July 2022
ACCEPTED 21 October 2022
PUBLISHED 14 November 2022

CITATION
Louis C, Jackman C, Mangham S, Smith K, O'Dwyer E, Empey A, Cecconi B, Boudouma A, Zarka P and Maloney S (2022), The "SPectrogram Analysis and Cataloguing Environment" (SPACE) labelling tool.
Front. Astron. Space Sci. 9:1001166.
doi: 10.3389/fspas.2022.1001166

COPYRIGHT
© 2022 Louis, Jackman, Mangham, Smith, O'Dwyer, Empey, Cecconi, Boudouma, Zarka and Maloney. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

The "SPectrogram Analysis and Cataloguing Environment" (SPACE) labelling tool

Corentin Louis [1]*, Caitriona Jackman [1], Sam Mangham [2], Kevin Smith [1], Elizabeth O'Dwyer [1], Aaron Empey [1], Baptiste Cecconi [3], Adam Boudouma [3], Philippe Zarka [3] and Shane Maloney [1]

[1]School of Cosmic Physics, DIAS Dunsink Observatory, Dublin Institute for Advanced Studies, Dublin, Ireland, [2]Department of Electronics & Computer Science, University of Southampton, Southampton, United Kingdom, [3]LESIA, Observatoire de Paris, CNRS, PSL Research University, Meudon, France

The SPectrogram Analysis and Cataloguing Environment (SPACE) tool is an interactive python tool designed to label radio emission features of interest in a time-frequency map (called "dynamic spectrum"). The program uses Matplotlib's Polygon Selector widget to allow a user to select and edit an undefined number of vertices on top of the dynamic spectrum before closing the shape (polygon). Multiple polygons may be drawn on any spectrum, and the feature name along with the coordinates for each polygon vertex are saved into a ".json" file as per the "Time-Frequency Catalogue" (TFCat) format along with other data such as the feature id, observer name, and data units. This paper describes the first official stable release (version 2.0) of the tool.

KEYWORDS

python tools, labelling tool, planetary radio emission, solar radio emission, jupiter, saturn, earth, sun

1 Introduction

Non-thermal planetary radio emissions are produced by out-of-equilibrium populations of charged particles in planetary magnetospheres, and are observed at almost all strongly magnetized planets in our Solar System: the Earth, Jupiter, Saturn, Uranus and Neptune. The radio emissions can be divided into different classes, such as plasma waves, electromagnetic radio waves or electrostatic radio waves. It is highly desirable to select these distinct classes, which can often have characteristic frequency ranges, morphologies, or polarizations. Once catalogues of different emission types have been built up, that can enable large statistical studies unveiling both the average and extreme behaviour of planetary radio emissions.

There have been many long-running planetary spacecraft which have returned huge volumes of radio data and we have only scratched the surface of its analysis.

For example, the Cassini mission at Saturn spent 13 years studying the kronian system, revealing several components to its radio spectrum (Lamy et al., 2008; Taubenschuss et al., 2011; Ye et al., 2011; Lamy, 2017). Furthermore, the Wind spacecraft has spent almost 2 decades observing terrestrial (and solar) radio emissions from a range of vantage points near Earth (Bonnin et al., 2008; Waters et al., 2021; Fogg et al., 2022).

Significant efforts have been made in recent years to classify radio emissions from Jupiter, where the non-thermal radio emission is composed of half a dozen components (Louis et al., 2021c). These components overlap themselves in time and frequency, making automatic detection non-trivial. Therefore, manually cataloguing them is mandatory to be able to study them independently. Previous catalogues have been using square boxes to define the time and frequency intervals containing the radio signal (such as Leblanc, 2020a; Leblanc, 2020b; Leblanc, 2020c; Leblanc, 2020d; Leblanc, 2020e), but to be able to automatically disentangle the emissions when using the catalogue, it is then needed to construct catalogues with polygon vertices and with distinct labels. This was done by a few authors, primarily using tools built in IDL to construct such catalogues of Jupiter radio emissions Marques et al. (2017); Zarka et al. (2021). Once catalogues are built they can comprise training sets which form the basis of supervised machine learning approaches to classify larger samples of unseen data.

Here we present a Python user interface tool to allow the drawing of polygons around features in dynamic spectra. **Section 2** describes the package, and an example of the use of this package. **Section 3** summarises the version history of the code. **Section 4** gives examples of the application of the catalogues produced by the tool. **Section 5** presents some future avenues to continue to improve the tool.

2 The SPACE package

The SPectrogram Analysis and Cataloguing Environment (SPACE) tool is an interactive python tool designed to label radio emission features of interest in a time-frequency map (called a "dynamic spectrum"). The program enables users to create and edit the vertices of a polygon on the dynamic spectrum plot, before naming and saving it as a 'feature' in a catalogue for future analysis. Full usage documentation is available on the GitHub repository of the code (Louis et al., 2022) and directly at https://space-labelling-tool.readthedocs.io.

2.1 Installation

The code is available on an open-source GitHub repository (Louis et al., 2022), with full installation instructions present on the repository page and on the ReadTheDocs webpage, and packed with the tool. It can easily be downloaded using git and installed using pip, with all prerequisites included in the provided requirements.txt file:

```
1 git clone https://github.com/CorentinLouis/
  SPACE_labelling_tool.git
2 pip install -e SPACE_labelling_tool
```

Listing 1: Download and Install SPACE using git and pip.

Installing in editable mode ("-e") will allow the scripts to read any new or modified versions of the configuration files.

The code is also available from PyPi[1] and can be directly install using pip:

```
1 pip install SPACE-labelling-tool
```

Listing 2: Download and Install SPACE with pip.

2.2 Usage

Once installed using pip, the space labelling tool is available as a system-wide command spacelabel. It can then be used to view and label spacecraft observational data, by providing an input file in **HDF5** or **NASA CDF** format (see **Section 2.2.3** for specifics), and a time window to view. An example use is shown in **Figure 1**.

```
2 spacelabel[-h][-s SPACECRAFT] FILE DATE
  DATE [-options]
```

Listing 3: System-wide command to run the space labelling tool - see **Section 2.2.2** for more details on the -options.

Users may select any number of the measurement types present in the file (e.g., polarisation, flux and/or power), and view them all tiled on the screen. For example, **Figure 1A** displays Cassini Flux and Polarization radio data, while **Figure 1B** only display Juno Flux radio data. Users can then click to select polygonal regions of the observation to label as features (top panel of **Figure 1A**). To close the polygon, users should click on the first drawn vertex of the polygon. Once done, a window pops up and ask to name the drawn feature appropriately (see top panel of **Figure 1A**). Features labelled in one view (e.g., intensity) appear simultaneously on the other views once they have been named (see bottom panel of **Figure 1A**), allowing users to easily see how a feature presents in multiple measurement types.

Once a region has been labelled, a user can pan their viewing window back and forth through the time range within the dataset by clicking on the Prev or Next buttons (see **Figure 1**), with an overlap applied between each view in order to facilitate

1 https://pypi.org/project/SPACE-labelling-tool/

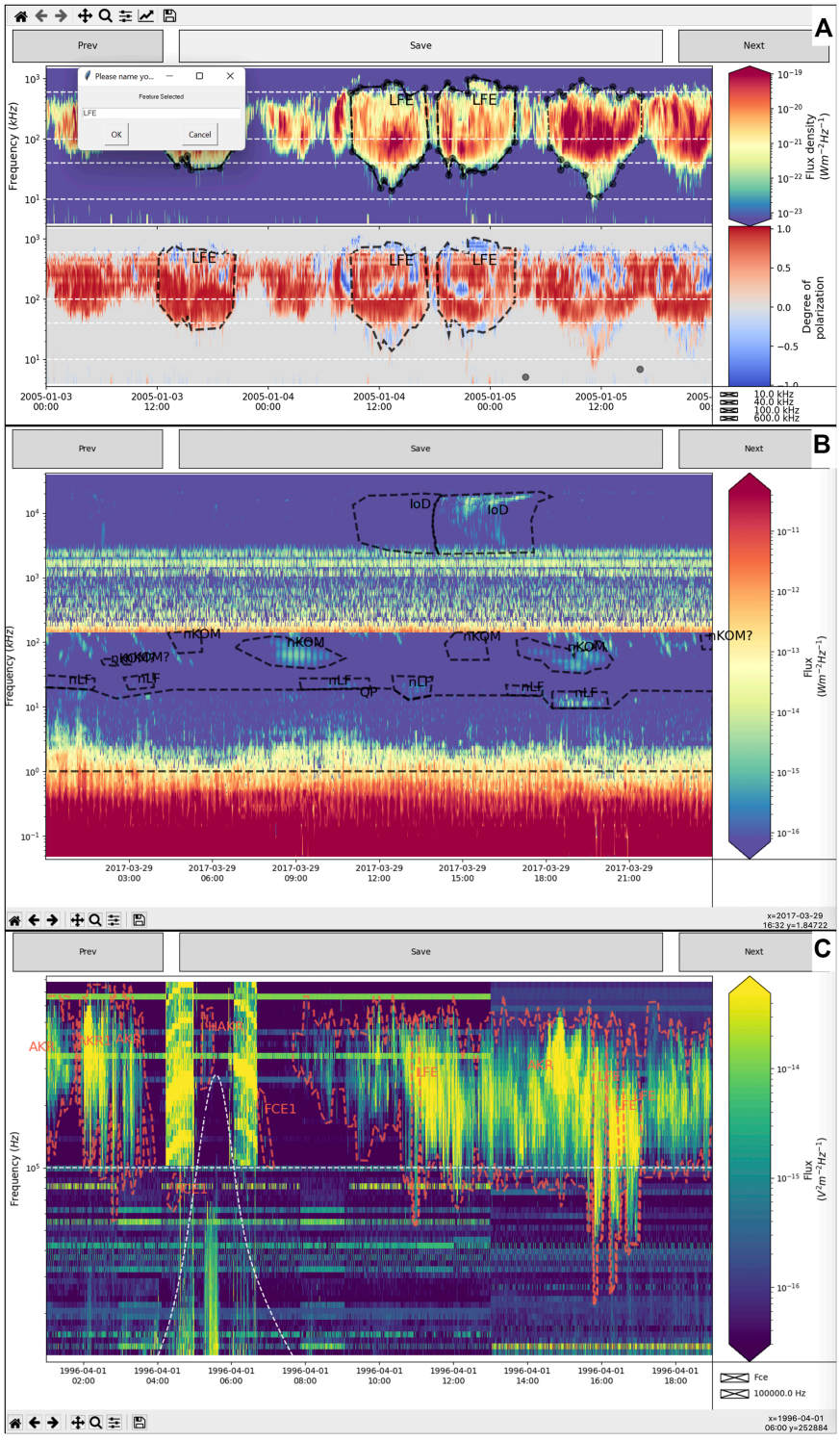


FIGURE 1
Examples of plots from the SPACE labelling Tool. Panel **(A)** displays Cassini/RPWS (Radio and Plasma Waves Science Gurnett et al., 2004) data (Lamy et al., 2008, 2009). The two panels show Intensity and Polarization data, respectively. At the top right of the top panel one can see a polygon that has just been drawn, with the window for naming the feature appearing at the top left of the graphics window. Other features have already been labelled, and appear in both intensity and polarisation views, with their names overlaid. The data displays in panel **(B)** are the estimated flux density (Louis et al., 2021a,c) from Juno/Waves measurements (Kurth et al., 2017), with the Louis et al. (2021b) catalogue overlaid. Panel **(C)** displays observations of Polar/PWI instrument (Gurnett et al., 1995) with the Smith et al. (2022) catalogue overlaid. The horizontal dashed-white line shows an example of the use of the -g [FREQUENCY GUIDE [FREQUENCY GUIDE ...]] option. The variable dashed-white line show that the tool is also able to read a 1D table from the CDF file (provided that this has been specified in the configuration file).

labelling features that lie on the edge of a window. Once finished, the labelled regions can be saved as a TFCat (Time-Frequency Catalogue) formatted *JSON*[2] file (Cecconi et al., 2022, this issue) by clicking on the Save button, and used later. If a user re-opens the same data file, or another data file with the same naming structure (e.g., observations_20180601_v02.cdf and observations_20180602_v02.cdf) saved features from previous sessions will be pre-loaded (see **Figure 1**).

Full usage documentation is available on the GitHub repository of the code (Louis et al., 2022) and directly at https://space-labelling-tool.readthedocs.io.

2.2.1 Procedure

When the code first opens a datafile, it compares the columns within to a selection of pre-made (and user-creatable) 'configuration' files for each type of input file (e.g., CDF, HDF5). Each describes a file in terms of the column names within it, and provides metadata for use in the tool - units and display names, and scaling factors that can be applied to change data stored in one unit into data viewed in another. If the data are in an unspecified format, the code will prompt the user to create a configuration file (see **Section 2.2.3**). Alternatively, if their data fit multiple configuration files, they will be prompted to select which they want to use (see **Section 2.2.2**).

Once a configuration has been determined, the code parses the observations and may re-bin the time into a coarser resolution in order to improve performance, taking the average of measurements in the new larger bins. It can also rescale the frequency data into evenly-spaced logarithmic bins between the minimum and maximum bins in the original data, as some data files have non-monotonic bin structures. When altering the frequency bins, measurements are logarithmically interpolated between the readings on the previous scale. Default adjustments can be defined in configuration files for file types, and may be over-ridden by command-line arguments to the tool. The parsed and adjusted data are then re-saved as a compressed HDF5 file, reducing both the size of the data and time to access.

The pre-processed data are then displayed to the screen using MatPlotLib (Hunter, 2007), in a window the size of the user's initial time range. The dynamic range of the data is constrained to improve visibility of features; displaying, by default, the 5th-95th quantiles of the signal for each measurement (to prevent anomalously low or high values overly-compressing the ranges of interest, reducing the ability to discern features). GUI buttons allow users to pan between time windows of equivalent size - with each window will overlap the previous window by 25% in the direction of travel. Features can then be defined by drawing polygons on the time window using the MatPlotLib Polygon Selector widget. There are the detailed interactive components of the MatPlotLib window:

- Measurements: Each panel displays a measurement, with name, scale and units on the right. Features can be drawn by clicking to add coordinates, and completed by clicking on the first coordinate added again. The vertices of the polygon can be modified before completed the polygon:
  - Hold the ctrl key and click and drag a vertex to reposition it before the polygon has been completed.
  - Hold the shift key and click and drag anywhere in the axes to move all vertices.
  - Press the esc key to start a new polygon.

Once selected, a feature can be named (see **Figure 1A**). Features can be selected on any pane, and will be mirrored on all other panes.

- Prev/Next buttons: These move through the data by an amount equal to the width of time range selected. This will also overlap 1/4 of the current window as 'padding'.
- Save button: This will save any features to TFcat JSON format, as catalogue_OBSERVER_NAME.json.
- Check boxes: If the option -g [FREQUENCY_GUIDE [FREQUENCY_GUIDE ...]] has been enabled by the users (e.g., fixed frequency lines in **Figures 1A,C**), or if a 1D variable is contained in the input data and configuration files (e.g., Fce **Figure 1C**) check boxes will appear in the lower right hand corner of the figure to make the white dotted lines appear or disappear.

Once finished, you can save and then close the figure using the normal close button.

2.2.2 Options

The user must specify the path to the file they want to visualise (or 'first' file in a collection, e.g., of **CDF** files), along with the start and end dates of their initial viewing window, in **ISO** *year-month-day hour:minute:second* format (e.g., 2018-06-12 18:00:01). However, there are further options available:

- -f FREQUENCY_BINS: Rescales the data to this many evenly-spaced logarithmic bins. Overrides any default set in the configuration files.
- -t MINIMUM_TIME_BIN: Rebins the data to time bins of this size, if it is currently more finely binned. The bin size need to be given in second.
- -s SPACECRAFT: Specifies the name of the spacecraft configuration file to use, if multiple describe the datafile the user has provided.
- -fig_size FIGURE_SIZE FIGURE_SIZE: x and y dimension of the matplotlib figure (by default: 15 9).
- -frac_dyn_range FRAC_MIN FRAC_MAX: Defines the dynamic range of the colour bar in the visualisation, as a fraction of the distribution of values in the data file. This

2 https://www.json.org/

must be numbers between 0 and 1. Default values are 0.05 and 0.95 (the 5th-95th quantiles of the displayed signal).
- -cmap CMAP: The name of the color map that will be used for the intensity plot (by default: viridis).
- -cfeatures CFEATURES: The name of the colour for the saved features of interest polygons (by default: tomato).
- -thickness_features TFEATURES: The thickness value for the saved features of interest polygons (by default: 2).
- -size_features_name SFEATURESNAME: The font size for the name of the saved features of interest polygons (by default: 14).
- -g [FREQUENCY_GUIDE [FREQUENCY_GUIDE ...]]: Draws horizontal line(s) on the visualisation at these specified frequencies to aid in interpretation of the plot.Values must be in the same units as the data.Lines can be toggled using check boxes.
- –not_verbose: If not_verbose is called, the debug log will not be printed. By default: verbose mode.

2.2.3 Input formats

The code is designed to cope with input files in a variety of file formats and column formats by use of configuration files, several of which are pre-provided. HDF5 input files require at least three datasets, corresponding to observation time (floats, in MJD), frequency range (floats, in any arbitrary unit) and at least one measurement, stored with frequency as the rows and observation as the columns. The names and units for each measurement (in LaTeX form) must be provided in a configuration file, in easily-editable JSON format. The appropriate configuration files are automatically-selected by the code from those available - making it easy to work with HDF5 files from a variety of collaborators with arbitrary naming schemes.

CDF files in NASA format are more structured, and can be read in either singly or as a collection, combining all files in the directory matching the naming scheme [...]_YYYMMDD_[...].cdf into a single pre-processed data file. As with the HDF5 files, CDF files must contain a frequency attribute (floats, in any arbitrary unit) and a time attribute (either in TT_2000 or CDF_EPOCH format, which is parsed using Astropy, Price-Whelan et al., 2018) and at least one measurement, stored with frequency as the rows and observation as the columns.

The code can easily be expanded to ingest other file formats (see **Section 2.3.2**).

2.3 Structure

2.3.1 'Model-View-Presenter' architecture

The code is designed using a standard object-oriented 'Model-View-Presenter' architecture, with strong separation between the data input and management, and the visualisation. This allows for easy development of both new input file-types (see **Section 2.3.2**) and pre-processing options, and alternative GUI front-ends and settings (see **Section 5** for suggested development building on top of this flexibility). A generic 'Presenter' controls the logic of the program, and feeds data from the data models to the selected GUI view, and requests from the GUI for changes to the data models. Either the 'View' or 'Model' can be easily interchanged as long as they conform to the API expected by the 'Presenter'.

Full development documentation is available on the GitHub repository of the code (Louis et al., 2022) and directly at https://space-labelling-tool.readthedocs.io.

2.3.2 Addition of new input formats

New input formats can be easily added by extending the base DataSet class included in the code. A developer only needs to define the routines for inputting the data from file; the code will then handle pre-processing and data access.

3 History of the code

The first version of the labelling code was developed in IDL by P. Zarka. It allowed users to read data from an IDL saveset (sav format), draw polygons around features of interest and label them. However, this IDL version had to be adapted to each new dataset. This code has been used to build many catalogues based on different observers (such as the Nançay Decameter Array (NDA) ground-based radio telescope (Marques et al., 2017), or the Cassini (Zarka et al., 2021) or Juno (Louis et al., 2021c,b) spacecraft).

The second version of the labelling tool was written in Python and was the first to be officially released (Empey et al., 2021). This version allowed to automatically read any dataset in sav or cdf format, based on the information requested from users from the terminal. The other main improvements compared to the previous version were the number of vertices in the polygons (unlimited) and the possibility to modify the vertices position during the polygon drawing (using the Matplotlib's Polygon Selector widget), as well as the production of the catalogue directly in TFCat format.

The current version (Louis et al., 2022) brings a large number of improvements, both in terms of architecture, usability and ergonomics, which are described in the previous sections.

The SPACE tool has also joined the MASER service (Cecconi et al., 2020).

4 Applications

Once a catalogue has been produced, it can also be displayed using the SPACE labelling Tool (see **Figure 1** or the Autoplot

Software (Faden et al., 2010). For an example, the reader is invited to visit the web page https://doi.org/10.25935/nhb2-wy29 where an autoplot template file is given to display the Juno data (Louis et al., 2021a) and the Louis et al. (2021b) catalogue overlaid in autoplot. See Cecconi et al. (2022, this issue) for more information about the display of a Catalogue using Autoplot. The catalogue can be used to study the different components of the radio emission spectrum, e.g. as done by Louis et al. (2021c), where the data can then be automatically selected using the catalogue *via* a mask or an inverse-mask. In the case presented in **Figure 1**, not every type of emission is labelled, but in each frequency range (kilometric or hectometric) only one radio component remains. We can then study the components one by one [e.g., their latitudinal distribution, as in Louis et al. (2021c), their distribution as a function of observer's or Sun's longitude, as in Zarka et al. (2021), or their distribution *versus* observer's longitude and satellite (Io) phase as in Marques et al. (2017)].

With the SPACE labelling Tool, we are also providing some useful routines to use the catalogue[3].

These catalogues can also then be used to train machine learning algorithm to detect automatically the radio emissions in past (Cassini, NDA) or future observation (such as Juno, JUICE, NDA).

5 Limitations and future work

The code is ready for distribution and use, but has some technical limitations. Potential works to address those limitations, and avenues of future development, are:

- Performance: The MatPlotLib-based front-end can struggle when provided with especially high resolutions of data, or over large time windows. Rebinning features exist to mitigate this, but an alternative rendering framework could be employed (e.g., datashader[4]), and/or the performance of the interactive front-end components could be improved if re-implemented in a more performant framework [e.g., Plotly Inc. (2015)].
- Scalability: The code loads all the data provided into memory at launch, limiting its applicability for large datasets. Whilst this can be mitigated by the feature to allow appending to **TFCat** files created by data files sharing filename formats, a 'deferred load' approach would be better. This would be best accomplished using the Dask and XArray libraries (Dask Development Team, 2016; Hoyer and Hamman, 2017).
- Configurations: The code depends heavily on pre-written configuration files, and can prompt users to create missing ones - but does not yet contain a 'wizard' or automatic walkthrough to aid users in creating them.
- Platform: Run the SPACE labelling tool in a notebook (launch on Binder).
- Catalogue integration: The modular format of the code would make it possible to create 'dataset' types that access and download data directly from online catalogues, maintaining local caches.

Data availability statement

The code of the SPACE labelling Tool is open-source and freely available on github (Louis et al., 2022). The Cassini/RPWS dataset displayed in **Figure 1A**, produced by Lamy et al. (2008) is available at https: //doi.org/10.25935/ZKXB-6C84 (Lamy et al., 2009). The Juno/Waves dataset displayed in **Figure 1B**, produced by Louis et al. (2021c), is accessible at https://doi.org/10.25935/6jg4-mk86 (Louis et al., 2021a), and the catalogue can be download at https://doi.org/10.25935/ nhb2-wy29 (Louis et al., 2021b). The Polar/PWI dataset displayed in **Figure 1C** is accessible through the CDAWeb at https://cdaweb.gsfc.nasa.gov/pub/data/polar/pwi/.

Author contributions

CL and CJ led the development of the SPACE Labelling Tool. CL wrote the paper and worked on the development of the code. PZ developed the first IDL version of the code. AE developed the first python version of the code. SM worked on the development of the second version of the code. SWM developed the current version of the code. EO tested the different versions of the code, and gave inputs to the developers of the code. AB helped run the code on a virtual machine environment, and made changes on the plotting functions to the latest version of the code. AB helped run the code on a virtual machine environment, and made changes on the plotting functions to the latest version of the code. KS added features to the latest version of the code. EO, AB, and KS wrote useful routines to read, modify and use the output catalogue. BC led the TFCat format of the catalogues. All the co-authors have contributed to the writing of the paper.

Funding

CL's, CJ's, EO's, and AE's work at the Dublin Institute for Advanced Studies was funded by Science Foundation Ireland Grant 18/FRL/6199. SWM's work at the University of

3 https://github.com/elodwyer1/Functions-for-SPACE-Labelling-Tool

4 datashader.org

Southampton Research Software Group was funded by a grant *via* the Alan Turing Institute. KS's work at the Dublin Institute for Advanced Studies was funded by a 2022 SCOSTEP/PRESTO Grant. The Europlanet 2024 Research Infrastructure project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871149.

Acknowledgments

CL thanks A. Loh and S. Lion for their very helpful comments on the development phase of the code. CL also gives special thanks to R. Prangée for finding errors that no one had seen.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Bonnin, X., Hoang, S., and Maksimovic, M. (2008). The directivity of solar type III bursts at hectometer and kilometer wavelengths: Wind-Ulysses observations. *Astronomy Astrophysic* 489, 419–427. doi:10.1051/0004-6361:200809777

Cecconi, B., Loh, A., Sidaner, P. L., Savalle, R., Bonnin, X., Nguyen, Q. N., et al. (2020). Maser: A science ready toolbox for low frequency radio Astronomy. *Data Sci. J.* 19, 1062. doi:10.5334/dsj-2020-012

Cecconi, B., Louis, C. K., Bonnin, X., Loh, A., and Taylor, M. B. (2022). *TFCat (Time-Frequency catalogue): JSON implementation and Python library*. Frontiers in astronomy and space sciences. This issue.

Dask Development Team (2016). *Dask: Library for dynamic task scheduling*. https://dask.org

Empey, A., Louis, C. K., and Jackman, C. M. (2021). *SPACE labelling tool*. version 1.1.0. [Code]. doi:10.5281/zenodo.5636922

Faden, J. B., Weigel, R. S., and Merka, J., (2010). Autoplot: A browser for scientific data on the web. *Earth Sci. Inf.* 3, 41–49. doi:10.1007/s12145-010-0049-0

Fogg, A. R., Jackman, C. M., Waters, J. E., Bonnin, X., Lamy, L., Cecconi, B., et al. (2022). Wind/WAVES observations of auroral kilometric radiation: Automated burst detection and terrestrial solar Wind - magnetosphere coupling effects. *JGR. Space Phys.* 127, e30209. doi:10.1029/2021JA030209

Gurnett, D. A., Kurth, W. S., Kirchner, D. L., Hospodarsky, G. B., Averkamp, T. F., Zarka, P., et al. (2004). The Cassini radio and plasma wave investigation. *Space Sci. Rev.* 114, 395–463. doi:10.1007/s11214-004-1434-0

Gurnett, D. A., Persoon, A. M., Randall, R. F., Odem, D. L., Remington, S. L., Averkamp, T. F., et al. (1995). The polar plasma wave instrument. *Space Sci. Rev.* 71, 597–622. doi:10.1007/bf00751343

Hoyer, S., and Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *J. Open Res. Softw.* 5, 10. doi:10.5334/jors.148

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Kurth, W. S., Hospodarsky, G. B., Kirchner, D. L., Mokrzycki, B. T., Averkamp, T. F., Robison, W. T., et al. (2017). The Juno waves investigation. *Space Sci. Rev.* 213, 347–392. doi:10.1007/s11214-017-0396-y

Lamy, L., Cecconi, B., and Zarka, P. (2009). *Cassini/RPWS/HFR LESIA/kronos SKR data collection*. Version 1.0. [Data set]. doi:10.25935/ZKXB-6C84

Lamy, L. (2017). "The saturnian kilometric radiation before the Cassini grand finale," in *Planetary radio emissions (PRE-VIII)*. Editors G. Fischer, G. Mann, and M. Panchenko, (Vienna: Austrian Academy of Sciences Press). 455–466. doi:10.1553/PRE8s455

Lamy, L., Zarka, P., Cecconi, B., Prangé, R., Kurth, W. S., and Gurnett, D. A. (2008). Saturn kilometric radiation: Average and statistical properties. *J. Geophys. Res.* 113, A07201. doi:10.1029/2007JA012900

Leblanc, Y. (2020a). *A catalogue of Jovian decametric radio observations from January 1978 to December 1979*. Digitised version. [Data set]. doi:10.25935/GXZF-ZT33

Leblanc, Y. (2020b). *A catalogue of Jovian decametric radio observations from January 1982 to December 1984*. Digitised version. Data set. doi:10.25935/403B-VA51

Leblanc, Y. (2020c). *A catalogue of Jovian decametric radio observations from January 1985 to December 1987*. Digitised version. [Data set]. doi:10.25935/gh59-py87

Leblanc, Y. (2020d). *A catalogue of Jovian decametric radio observations from January 1988 to December 1990*. Digitised version. [Data set]. doi:10.25935/cmqb-jj10

Leblanc, Y. (2020e). *A catalogue of Jovian radio observations from January 1980 to December 1981*. Digitised version. [Data set]. doi:10.25935/RAS0-ER93

Louis, C. K., Jackman, C. M., Mangham, S. W., Smith, K. D., O'Dwyer, E., Empey, A., et al. (2022). *SPACE labelling tool*. Version 2.0.0. [Code]. doi:10.5281/zenodo.6886528

Louis, C. K., Zarka, P., and Cecconi, B. (2021a). *Juno/Waves estimated flux density Collection*. Version 1.0. doi:10.25935/6jg4-mk86

Louis, C. K., Zarka, P., Cecconi, B., and Kurth, W. S. (2021b). *Catalogue of Jupiter radio emissions identified in the Juno/Waves observations*. Version 1.0. doi:10.25935/nhb2-wy29

Louis, C. K., Zarka, P., Dabidin, K., Lampson, P. A., Magalhães, F. P., Boudouma, A., et al. (2021c). Latitudinal beaming of jupiter's radio emissions from juno/waves flux density measurements. *JGR. Space Phys.* 126, e29435. doi:10.1029/2021JA029435

Marques, M. S., Zarka, P., Echer, E., Ryabov, V. B., Alves, M. V., Denis, L., et al. (2017). Statistical analysis of 26 yr of observations of decametric radio emissions from Jupiter. *Astron. Astrophys.* 604, A17. doi:10.1051/0004-6361/201630025

Plotly Inc (2015). *Collaborative data science*. https://plot.ly

Price-Whelan, A. M., Sipőcz, B., Günther, H., Lim, P., Crawford, S., Conseil, S., et al. (2018). The astropy project: Building an open-science project and status of the v2. 0 core package. *Astronomical J.* 156, 123. doi:10.3847/1538-3881/aabc4f

Smith, K. D., Louis, C. K., Jackman, C. M., Fogg, A. R., O'Dwyer, E., Waters, J. E., et al. (2022). Catalogue of Auroral Kilometric Radiations (AKR) observed by the Polar spacecraft (1.1). *Zenodo*. doi:10.5281/zenodo.7235986

Taubenschuss, U., Leisner, J. S., Fischer, G., Gurnett, D. A., and Nemec, F. (2011). "Saturnian low-frequency drifting radio bursts: Statistical properties and polarization," in *Planetary, solar and heliospheric radio emissions (PRE VII)*. Editors H. O. Rucker, W. S. Kurth, P. Louarn, and G. Fischer (Vienna: Austrian Academy of Sciences Press), 115–123.

Waters, J. E., Jackman, C. M., Lamy, L., Cecconi, B., Whiter, D. K., Bonnin, X., et al. (2021). Empirical selection of auroral kilometric radiation during a multipoint remote observation with Wind and Cassini. *JGR. Space Phys.* 126, e29425. doi:10.1029/2021JA029425

Ye, S. Y., Fischer, G., Menietti, J. D., Wang, Z., Gurnett, D. A., and Kurth, W. S. (2011). "An overview of Saturn narrowband radio emissions observed by Cassini RPWS," in *Planetary, solar and heliospheric radio emissions (PRE VII)*. Editors H. O. Rucker, W. S. Kurth, P. Louarn, and G. Fischer. (Vienna: Austrian Academy of Sciences Press), 99–113.

Zarka, P., Magalhães, F. P., Marques, M. S., Louis, C. K., Echer, E., Lamy, L., et al. (2021). Jupiter's auroral radio emissions observed by Cassini: Rotational versus solar Wind control, and components identification. *JGR. Space Phys.* 126, e29780. doi:10.1029/2021JA029780

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 18 November 2022
DOI 10.3389/fspas.2022.1000145

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Vladimir A. Sreckovic,
University of Belgrade, Serbia
Piers Jiggens,
European Space Agency (ESA), France

*CORRESPONDENCE
Samuel D. Walton,
sdwalton.space@gmail.com

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 21 July 2022
ACCEPTED 21 October 2022
PUBLISHED 18 November 2022

CITATION
Walton SD and Murphy KR (2022), Superposed epoch analysis using time-normalization: A Python tool for statistical event analysis.
Front. Astron. Space Sci. 9:1000145.
doi: 10.3389/fspas.2022.1000145

COPYRIGHT
© 2022 Walton and Murphy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Superposed epoch analysis using time-normalization: A Python tool for statistical event analysis

Samuel D. Walton[1]* and Kyle R. Murphy[2,3]

[1]Mullard Space Science Laboratory, University College London, Dorking, United Kingdom, [2]Self Independent Researcher, Thunder Bay, ON, Canada, [3]Department of Maths, Physics and Electrical Engineering, Northumbria University, Newcastle Upon Tyne, United Kingdom

A superposed epoch analysis (SEA) is a simple, yet powerful statistical analysis technique, used to identify patterns in the temporal evolution of observed quantities relative to defined epochs. In some cases, the event duration and time between epochs (epoch length) can be highly variable. If the measured response scales with the event duration or epoch length, then the underlying temporal patterns can be suppressed when analyzed in absolute time. In this article, we describe an adaptation of the traditional SEA, where we apply time-normalization to each event and present a Python package `sea_norm` which implements the time-normalized SEA. Rather than defining a singular epoch time, a start, epoch, and end time are defined for each event, separating each event into two intervals. For every event, the duration of both intervals is normalized to a common time axis, essentially stretching or compressing each interval, such that each respective epoch interval is the same length for all events. This technique has the advantage of identifying temporal patterns not observed in a traditional SEA. Given a time series, a list of event start, epoch, and end times, and specified binning dimensions the Python package `sea_norm` returns a time-normalized SEA analysis of the time-series. This technique is widely applicable across the Space Physics field, where events have defined start and end times, and where the response to those events may scale proportionally with event length. We provide examples demonstrating how the SEA code works with one-dimensional and two-dimensional time series, and how users can specify their own statistics to use in the superposed analysis (e.g., percentiles).

KEYWORDS

python, superposed epoch analysis, time-normalization, event analysis, statistics

1 Introduction

Many sciences and scientific fields of research use measurements of observed quantities in response to a specific event, usually with the aim of understanding the mechanics of the physical system which have led to that response, not least, within

the various fields of space science. Repeated events such as geomagnetic storms and substorms, can drive activity throughout Earth's magnetosphere, prompting the response of many measurable processes, including magnetic field dipolarizations, substorms injections, wave-particle interactions, and local or global precipitation of electrons and ions (e.g. O'Brien et al., 2003; Summers and Thorne, 2003; Meredith et al., 2011; Katsavrias et al., 2019; Wang et al., 2019). While these processes can typically be analyzed as individual events, sparse measurements and limited data availability may not provide sufficient temporal or spatial coverage required to develop a comprehensive understanding of the processes (e.g. Wang et al., 2019). For common and repeatable events, a statistical analysis, utilizing data obtained from many events can provide improved spatial and temporal coverage of the region of interest (e.g., the outer radiation belt), allowing researchers to gain additional insight into the events and more importantly, the underlying physical processes (e.g. Yokoyama, 1997; Halford et al., 2010; Hutchinson et al., 2011; Murphy et al., 2018; Murphy et al., 2020; Olifer et al., 2021; Walton et al., 2022).

A superposed epoch analysis (SEA) is one of the most powerful and widely used statistical analysis techniques for studying the temporal evolution of observed quantities in response to a specific type of event, and relative to a defined epoch. Studies have utilized a conventional SEA for a range of physical phenomena and Python packages have previously been released which perform this conventional SEA analysis (Larsen et al., 2010; Morley et al., 2011). For example, Turner et al. (2019) used near-equatorial measurements of energetic electrons in Earth's outer radiation belt during geomagnetic storm events. In their analysis, the point of minimum Sym-H in each event was used as the singular epoch time and a range of statistics (including median, mean, etc.) were calculated. In particular, Turner et al. (2019) demonstrated the existence of both an energy and storm driver dependence in the storm-time response of electrons in the outer radiation belt. In space physics, this conventional SEA has been extensively used in relation to geomagnetic storm activity. For example, authors have studied both the response of energetic particles during storms (e.g. Meredith et al., 2011; Whittaker et al., 2014; Olifer et al., 2021; Smirnov et al., 2022), as well as the storm-time response of various plasma waves (e.g. O'Brien et al., 2003). SEA studies are not limited to storms, rather any event which can be studied in large numbers is ideal for SEA studies. This includes, for example, substorms (e.g. Boakes et al., 2011; Liu et al., 2011; Katsavrias et al., 2019), the response of the radiation belt to varying solar wind drivers (e.g. Hietala et al., 2014) and nightside particle injections (e.g. Gabrielse et al., 2014).

In some cases, such as that of geomagnetic storms, event duration can be highly variable. If the measured response of an event scales with the event duration, or the event can be separated into phases which scale with phase duration (e.g. geomagnetic storm phases), then the underlying temporal patterns can be suppressed when analyzed in absolute time, as with a traditional SEA. A solution to this is an adaptation of the traditional SEA, where time-normalization is applied to each event. Such an analysis helps to identify temporal patterns not observed in a traditional SEA. In a time-normalized SEA, rather than defining a singular epoch time, a start, epoch and end time are defined for each event, separating each event into two intervals (or phases). For every event, the duration of the two intervals is normalized to a common time axis, essentially stretching or compressing each interval, such that each respective epoch interval is the same length for all events. The SEA statistics (e.g., mean) are then calculated along the time-normalized axis of each phase using a binning algorithm.

Studies in space physics have utilized the time-normalized SEA technique for geomagnetic storms in a number of ways. Halford et al. (2010) used a time-normalized SEA to examine electromagnetic ion cyclotron (EMIC) wave occurrence in the outer radiation belt during geomagnetic storms, clearly showing heightened EMIC wave occurrence during the main phase of storms. Time-normalized SEA has also revealed further characteristics of geomagnetic storms and shock events, including the response of energetic electrons (e.g. Murphy et al., 2018, 2020; Walton et al., 2022), geomagnetic indices (e.g. Yokoyama, 1997) and various solar wind parameters (e.g. Hutchinson et al., 2011; Kilpua et al., 2015). To our knowledge, there is no widely available Python code which performs a time-normalized SEA.

In this article, we present a new Python package, `sea_norm` (GitHub link: https://github.com/samwalton7645/SEA_Code), capable of performing the adapted time-normalized SEA, given a time series, specified binning dimensions and list of events with defined start, epoch and end times. In the following sections, we describe the Python code and it is functionality, before providing examples demonstrating both a one-dimensional and a two-dimensional time-normalized SEA. This technique is widely applicable across the Space Physics field, where events have defined start and end times, and where the response to those events may scale proportionally with event length.

2 The sea_norm module

In this section, we detail the methodology behind the time-normalized SEA implemented in the `sean()` function, within the `sea_norm` module. We further describe the prerequisites for the `sea_norm` module, the input parameters and return values of the `sean()` function.

The time-normalized SEA is executed as follows:

1) Each event in the superposed epoch analysis is split into two phases defined by three times, the start, epoch, and end of each event. The first phase is defined as the start of the event to the epoch time (phase 1). The second phase is defined from

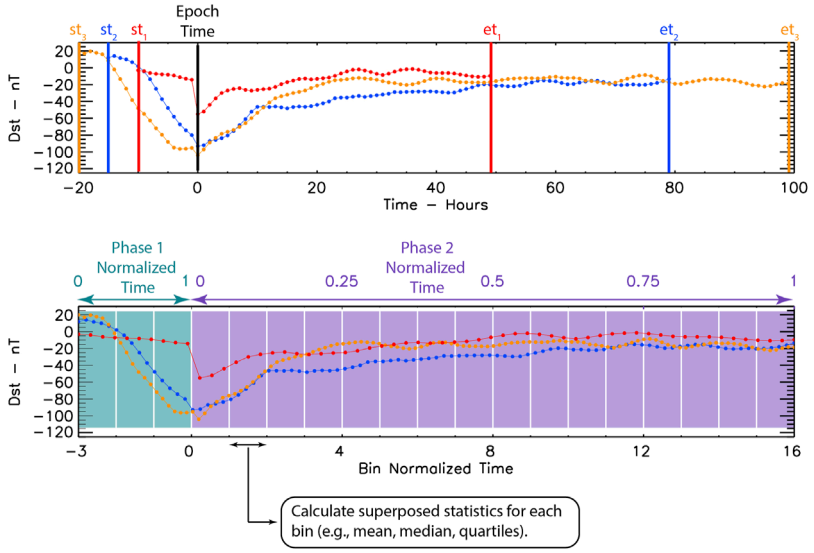


FIGURE 1
Illustration of how the time-normalized superposed epoch analysis is performed on the Dst index. The top panel shows the conventional SEA alignment, where $st_{1,2,3}$ and $et_{1,2,3}$ represent the start times and end times, respectively, of three different geomagnetic storms, aligned at their epoch times. The bottom panel shows the time-normalized SEA for the same storms. Phase 1 and phase 2 are highlighted as labeled, where each colored block represents a single normalized time bin.

the epoch time until the end of the event (phase 2). The phase 1 and 2 arrays are produced and input by the user.

2) For every event, phase 1 and 2 are then normalized between 0 and 1.
3) The normalized phases are then binned into a set of equally spaced bins, the number of bins being defined by the user. For example, if phase 1 was divided into four bins, the time axis would be divided into bins with edges [0, 0.25, 0.50, 0.75, 1.00].
4) For each phase, a set of statistics is then determined for the data residing in each bin (e.g. the mean and median).
5) If a 2D superposed epoch analysis is performed the data is binned in a second dimension before calculating the statistics of each bin.

The process described above is illustrated in **Figure 1**. The top panel shows the setup of a conventional SEA with Dst data for three storms of differing lengths, aligned at their respective epoch times. The bottom panel shows the time-normalized SEA setup. Each event is separated into phase 1 (`st` to the epoch time) and phase 2 (epoch time to `et`). For each event, phase 1 and 2 are normalized, and the time-normalized axes are binned into equally spaced bins. In the example here, phase 1 is binned into three bins and phase 2 into 16 bins (identified by the colored blocks), plotted with the time axis labeled relative to the epoch time and in terms of the number of normalized time bins. As **Figure 1**(bottom) illustrates, this process effectively stretches (e.g., the red event) or compresses (e.g., the orange event) events about the epoch time such that they are all the same length. Finally, a set of statistics is calculated for each bin, completing the time-normalized superposed epoch analysis.

The time-normalized SEA analysis described above is implemented in the `sea_norm` module (Walton and Murphy, 2022), which can be downloaded *via* the GitHub link: https://github.com/samwalton7645/SEA_Code. The user can download a .zip file of the repository to a local repository, extract the files, and then installed *via* the terminal using the command 'pip install', within the 'SEA_Code' directory. The prerequisite packages are Pandas v1.1.5 or later, Numpy v1.21.6 or later, Scipy v1.2.1 or later, and tqdm v4.36.1 or later, used within Python 3.6. `sea_norm` may work on earlier versions of Python and the respective packages, but is untested. Pandas and Numpy are used for data handling and manipulation; Scipy is used for the `.stats.binned_statistic()` and `.stats.binned_statistic_2d()` functions, which bin the time-normalized data and calculate the SEA statistics for each bin; tqdm is used to add progress bars to the display when using the `sea_norm` package, since statistical analysis can be somewhat time intensive.

The `sean()` function is the bulk of the time-normalized SEA code and implements the analysis described above and illustrated in **Figure 1**. `sean()` requires input data in the form of a Pandas DataFrame with a datetime index, as the index is used to normalize the data within phases. The list of events is specified using the events argument as a list of three arrays `[st, ep, et]`, containing start times in `st`, epoch times in `ep` and end times in `et`, in a datetime format. Unless otherwise specified using the `cols` argument, a 1D SEA is performed on

TABLE 1 An example array for when `sean()` is used to perform a 1D superposed epoch analysis on AE data only, using the default statistics.

t_norm	AE_mean	AE_median	AE_lowq	AE_upq	AE_cnt
-20.0	214.209,852	108.0	54.0	250.0	19651.0
-19.0	221.725,136	96.0	48.0	255.0	19580.0
-18.0	209.517,656	102.0	46.0	251.0	19590.0
-17.0	225.828,864	102.0	46.0	268.0	19562.0
-16.0	236.405,764	102.0	47.0	311.0	19559.0
⋮					
115.0	186.718,109	133.0	62.0	251.0	4,013.0
116.0	188.601,150	132.0	59.0	247.0	4,013.0
117.0	178.943,151	127.0	56.0	236.0	4,001.0
118.0	142.817,818	95.0	52.0	192.0	4,012.0
119.0	124.726,202	84.0	50.0	168.0	4,090.0

every column in the DataFrame. `x_dimensions` must also be specified as a list `[x1, x2]`, containing the number of bins in phase 1 and phase 2 of the superposed epoch analysis. If `sean()` is being used for a 2D SEA, the `y_col` argument must be used to specify the column which should be used for the *y*-axis, and a 2D SEA will be performed on the remaining columns. For a 2D SEA, `y_dimensions` must also be specified as a list [`y_min`, `y_max`, `y_spacing`], containing the minimum and maximum y boundaries, as well as the desired bin spacing for the second dimension.

The `sean()` function returns a Pandas DataFrame containing the time-normalized SEA time-series, as well as a dictionary of metadata for the performed analysis (e.g., which statistics were returned and which columns the analysis was run on). The returned DataFrame contains a column with the SEA for each data column and statistic calculated in the analysis. By default, `sean()` returns the mean, median, lower and upper quartiles, and counts for each data column. If a 2D SEA was performed, the returned DataFrame further contains a column for each bin of the second dimension and the metadata dictionary contains a dictionary of *y*-axis metadata (e.g., *y*-axis min, max and bin size). As an example, Table 1 shows the columns of a returned DataFrame when the input DataFrame contains a single time-series AE (see also the examples below). In a 2D analysis the same columns are returned but each column is further binned by the second dimension and the column names are appended with the second dimensions bin number. For example, in a 2D analysis the AE median column in Table 1 would become AE_median_n where n is the second dimension bin number. Finally, in both the 1D and 2D cases, the index of the returned DataFrame is the bin normalized time. For example if phase 1 was binned into 20 bins and phase 2 120 bins then the index goes from -20 to 119 in steps of 1.

For additional functionality, sean also allows users to define their own statistics *via* the `seastats` argument. This can simplify or speed up the SEA analysis by calculating only a subset of the default statistics. This can also be used for a more in-depth analysis by allowing users to define a more complex set of statistics to be calculated. The seastats argument is passed as a dictionary of the form: "`stat_name`":`stat_function`. `stat_function` can be a string e.g. as defined in `scipy.stats.binned_statistic()`, a callable, e.g., `np.nanmean`, or a lambda defined callable e.g., the 90th percentile: `p90 = lambda stat:np.nanpercentile(stat, 90)`. The '`stat_name`' key is used to label of the columns of the returned DataFrame (see Table 1).

3 Examples of use

In this section, we demonstrate how the `sea_norm` package can be used to analyze both 1D and 2D data for 168 geomagnetic storm events within a 12-year time period (1992–2004). For the purposes of the examples presented, we import the Pandas and Numpy packages as below, as well as the Matplotlib package, which is used for plotting the examples. Finally, we import the `sean()` function from the `sea_norm` package:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sea_norm import sean
```

We then load all data required for the examples in this section:

```
# load OMNI data
o_dat='https://zenodo.org/record/6835641/files/omnidata.csv.bz2'
omnidata = pd.read_csv(o_dat,parse_dates=True,
                       infer_datetime_format=True, header=0,
                       names=['t','B_Z_GSE','V','P','AE','SymH'],
                       index_col=0)
# load SAMPEX PET data
s_dat = 'https://zenodo.org/record/6835641/files/sampexflux.csv.bz2'
sampexdata = pd.read_csv(s_dat,parse_dates=True,
                       infer_datetime_format=True, header=0,
                       names=['t','ELO','EHI','L'],
                       index_col=0)
# load the event list and place the
# epoch times into the appropriate format
stormlist = pd.read_csv('StormList_short.txt', index_col=0,
                       parse_dates=[1, 2, 3, 4])
stormlist = stormlist.reset_index(drop=True)

starts = stormlist.IStart
epochs = stormlist.RStart
ends = stormlist.REnd
events=[starts, epochs, ends]
```

For the 1D examples, we use OMNIWeb parameters solar wind speed (V), dynamic pressure (P), Southward interplanetary magnetic field (B_z), Sym-H and the Auroral Electrojet (AE) index. For the 2D example, we use SAMPEX PET flux measurements from the low energy (ELO, 1.5–6.0 MeV) electron channel and the high energy (EHI, 2.5–14.0 MeV) electron channel, and L-shell (L) for the *y*-axis data. The list of events, '`StormList_short.txt`', is a list of geomagnetic storms identified by the algorithm described in Walach and Grocott (2019). All data is during the 1992–2004 time period.

The number of bins is defined for phase 1 and phase 2 as below, which remain the same for all examples:

```
# specify the number of bins in phase 1 and phase 2 as [nbins1, nbins2]
bins=[20, 120]
```

We use 20 bins for the pre-epoch phase and 120 bins for the post-epoch phase to loosely reflect the relative proportions of a geomagnetic storm.

3.1 One-dimensional SEA

The below Python code shows a simple example use of the `sean()` function to produce a 1D, time-normalized SEA for all of the OMNI parameters in our DataFrame (V, P, B_z, Sym-H and AE). A subset of columns can be specified by passing the column names *via* the `cols` argument. For example, to perform the analysis on only AE, use `sea_cols = ['AE']`.

```
# set the columns to run the analysis on
sea_cols = ['V','P','B_Z_GSE','SymH','AE']

# perform the time-normalized superposed epoch analysis
SEAarray, meta = sean(omnidata, events, bins, cols=sea_cols)

# get the columns that the SEA was performed
# on from the returned metadata
cols = meta['sea_cols']

# plot the superposed epoch analysis for each variable
# plot the mean, median, upper and lower quartiles
# ignore the cnts column

fig, axes = plt.subplots(nrows=len(cols), sharex=True,
                         squeeze=True,figsize=(5,8))

# loop over columns that were analyzed
for c, ax in zip(cols, axes):
    # for each column identify the column titles which
    # have 'c' in the title and those that don't have
    # 'cnt' in the title
    # e.g. for AE columns
    # AE_mean, AE_median, AE_lowq, AE_upq, AE_cnt
    # fine columns AE_mean, AE_median, AE_lowq, AE_upq
    mask = SEAarray.columns.str.startswith(c) & \
        ~SEAarray.columns.str.endswith('cnt')

    # plot the SEA data
    SEAarray.loc[:,mask].plot(ax=ax, style=['r-','b-','b--','b--'],
                              xlabel='Normalized Time',
                              ylabel=c.replace('_',' '),
                              legend=False, fontsize=8)
plt.show()
```

Figure 2 shows the plot resulting from the above code, presenting the mean, median and quartiles of, from top to bottom, V, P, B_z, Sym-H and AE. It is clear that a successful execution of the `sean()` function has produced the characteristic geomagnetic storm profile for the OMNI parameters. V, P and AE show characteristic increases before the storm epoch, while B_z shows a characteristic negative turn. Sym-H shows the typical storm shape, sharply turning negative pre-epoch, before gradually recovering post-epoch.

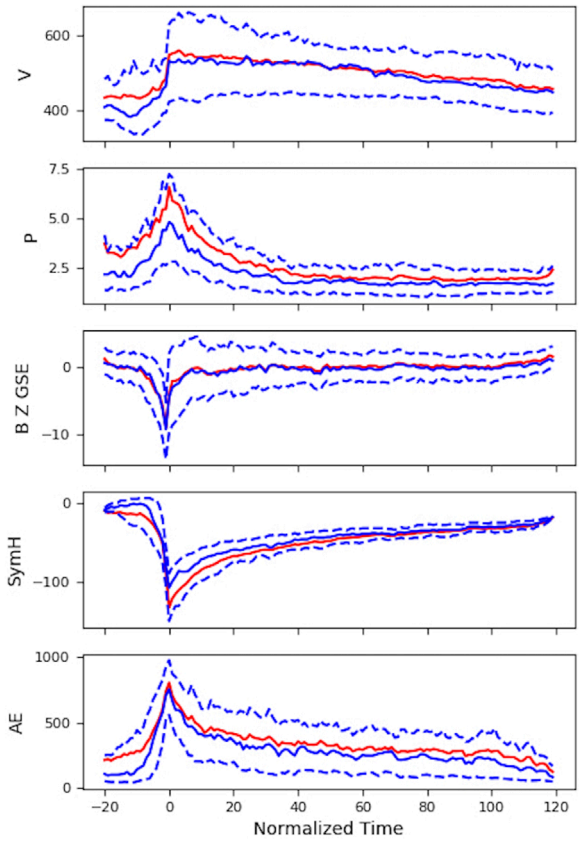


FIGURE 2
Result of the one-dimensional, time-normalised superposed epoch analysis for OMNIWeb parameters solar wind speed (V), dynamic pressure (P), southward interplanetary magnetic field (B_z), Sym-H and the Auroral Electrojet (AE) index. The statistics plotted are the mean (red), median (blue solid), upper and lower quartiles (blue dashed) for data within 168 geomagnetic storms from 1992–2004.

3.1.1 User-defined statistics

As mentioned in **Section 2**, the default for `sean()` is to return the mean, median, lower and upper quartiles and counts. However, `sean()` is capable of accepting user-defined statistics. In the below Python code, we define a set of lambda functions to be input into `sean()` as the user-defined statistic. In this case, the `np.percentile()` function is used to define the deciles using the `makepercentile()` function and a simple

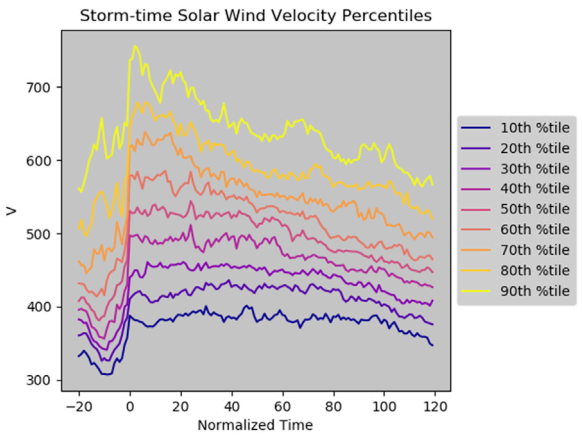


FIGURE 3
Result of the one-dimensional, time-normalised superposed epoch analysis for solar wind speed (V), for data within 168 geomagnetic storms. Deciles are calculated for data between 1992 and 2004. The color indicates the exact decile, referring to the legend plotted on the right.

for loop. The `seastats` variable is assigned to a dictionary containing ten lambda functions, which are input into `sean()` *via* the `seastat` parameter. To speed up the analysis, the deciles are only calculated for solar wind velocity.

```
# define a function that will return a lambda
# percentile function
def makepercentile(x):
    """Parameters
    ----------
    x : Percentile to calculate
    Returns
    -------
    Lambda function to calculate x'th percentile.
    """
    return lambda stat: np.nanpercentile(stat, x)

# use the makepercentile() function
# to return a lambda function of percentiles
seastats = {}
for x in np.arange(10,100,10):
    seastats[f'{x}th_%tile'] = makepercentile(x)

# set the columns to run the analysis on
sea_cols = ['V']
# perform the time-noramlized superposed epoch analysis
SEAarray, meta = sean(omnidata, events, bins, cols=sea_cols, seastats=seastats)

# set up plotting
fig = plt.figure()
ax = plt.subplot(111)
# get column names for labels and for setting colors
cols = SEAarray.columns.values.tolist()
# setup some colors
colors = plt.cm.plasma(np.linspace(0,1,len(cols)))
# make better legend labels from the column names
lab_col = []
for cc in cols:
    lab_col.append(cc[2:].replace('_',' '))

# plot the DataFrame
SEAarray.plot(ax=ax, ylabel=sea_cols[0], xlabel='Normalized Time',
              title='Storm-time Solar Wind Velocity Percentiles',
              color=colors)

# shrink current axis by 20% for the new axis
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
# place the legend to the right of the current axis
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5),labels=lab_col, facecolor='silver')
# change background color so it's easier to see
ax.set_facecolor('silver')
plt.show()
```

The resulting plot is shown in **Figure 3**. All ten deciles are plotted with a color assigned as displayed to the right of the plot.

The temporal storm profile is similar to the V panel in **Figure 3** median (50th percentile) and the upper and lower (25th and 75th percentiles, respectively), showing the characteristic rapid increase in solar wind speed V pre-epoch, followed by a gradual decrease post-epoch.

3.2 Two-dimensional SEA

To also demonstrate the `sean()` function's 2D capability, we present the below Python code, which produces the plot in **Figure 4** for SAMPEX PET flux data. For the `y-dimensions` (L-shell dimension), we set boundaries of L = 2.5 and L = 5.5, with a bin-spacing of L = 0.2. L-shell dimensions are chosen to reflect data availability in this particular data set.

```
# set the columns to run the analysis on
sea_cols = ['ELO','EHI']

# specify the y parameters for the 2D SEA
y_col = 'L'
ymin = 2.5
ymax = 5.5
y_spacing = 0.2
y_dim = [ymin, ymax, y_spacing]

# define the statistics to use
seastats = {'median':np.nanmedian}

# log the sampex data before performing SEA
# replace infinity values with nan to properly
# calculate statistics
logdata=sampexdata.copy()
logdata.iloc[:, 0:2]=np.log10(sampexdata.iloc[:, 0:2])
logdata.replace([np.inf, -np.inf], np.nan, inplace=True)

# perform the 2D SEA analysis
sea2d, meta =  sean(logdata, events, bins, cols=sea_cols,
                    seastats=seastats,
                    y_col=y_col,y_dimensions=y_dim)
# get the columns that the SEA was performed
# on from the returned metadata
cols = meta['sea_cols']

# grab the y metadata for plotting
ymeta = meta['y_meta']

fig, axes = plt.subplots(nrows=len(cols), sharex=True,
                         squeeze=True,figsize=(5,3.5))

# loop over columns that were analyzed
for c, ax in zip(cols, axes):
    # for each column identify the column titles which
    # have 'c' in the title
    # a more complex mask would need to be used if multiple
    # statistics where returned
    mask = sea2d.columns.str.startswith(c)

    # plot the data from the mask
    # transform to a 2D numpy array and transpose for plotting
    hb = ax.imshow(sea2d.loc[:,mask].to_numpy().transpose(), cmap='inferno',
               origin='lower', aspect='auto',
               extent =[sea2d.index.min(),sea2d.index.max(),
                        min(ymeta['edges']),max(ymeta['edges'])])
    ax.set_ylabel(c)
    cb = fig.colorbar(hb, ax=ax, label='log(flux)')

axes[len(cols)-1].set_xlabel('Normalized Time')
plt.tight_layout()
plt.show()
```

Figure 4 shows the result of the above Python code. Once again, the time-normalized SEA has produced the characteristic temporal storm profile for the ELO and EHI (relativistic) energies. Pre-epoch, electron flux shows rapid decreases at outside L ≈ 3.5, before recovering post-epoch.

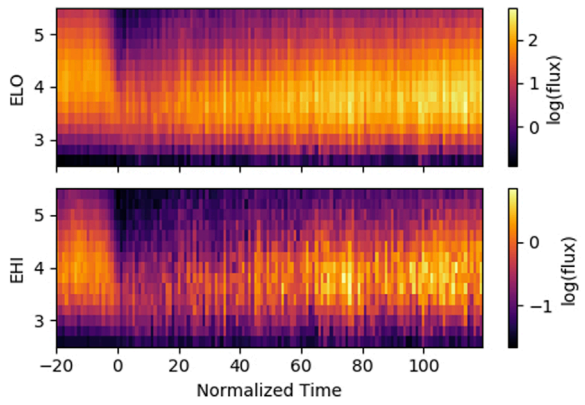


FIGURE 4
Result of the two-dimensional (L-shell vs. time), time-normalised superposed epoch analysis for SAMPEX PET fluxes between 1992 and 2004, for 168 geomagnetic storms. The colors are on a logarithmic scale, and show flux from the low energy (ELO, 1.5–6.0 MeV) electron channel on the top panel and from the high energy (EHI, 2.5–14.0 MeV) electron channel on the bottom panel.

4 Conclusion and future possibilities

In this article, we have presented `sea_norm`, a new python package which is able to perform a time-normalized SEA on both 1D and 2D data. The bulk of `sea_norm` is contained within the `sean()` function, where the only required inputs are a time series, a list of events and specified binning dimensions for the normalized time-axis (and for the y-dimensions if performing a 2D analysis). `sean()` returns a Pandas DataFrame with the completed SEA, along with the relevant metadata.

We have demonstrated three potential uses of `sean()`: a 1D SEA, performed on multiple time series parameters; a 1D SEA with user-defined statistics; and a 2D SEA for two time series binned by L-shell.

While the 1D and 2D functionality of the `sea_norm` package covers the vast majority of common SEA uses, future developments of `sea_norm` could incorporate a 3D version. This would provide `sea_norm` the capability of producing more in-depth analysis of events where more than one spatial dimension is of interest. For example, a 3D SEA could be used with SAMPEX flux data in both the L-shell and magnetic local time (MLT) dimensions throughout a geomagnetic storm. An animated L vs. MLT plot could then be produced, depicting the 2D spatiotemproal evolution of electrons in the radiation belts during a storm.

The superposed epoch analysis (SEA) has been used as an effective tool in time-series data analysis for over 100 years (Chree, 1913) and extensively used in Space Physics. The simple nature of the SEA makes it a powerful tool for statistical analysis whose results are easy to interpret and analyze. The time-normalized SEA discussed here provides a solution to a potential short-coming of the conventional SEA, whereby events of differing length can smear the underlying dynamics researchers wish to study. `sea_norm` provides a convenient method to perform a time-normalized SEA, allowing researchers to circumvent potential pitfalls in a traditional SEA analysis. Overall, `sea_norm` provides any researcher in any field who utilizes time series in their work to rapidly perform a time-normalized superposed epoch analysis and identify the underlying statistical patterns in their data.

Data availability statement

The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Author contributions

SW and KM equally contributed to development of the code, writing of the manuscript and production of figures.

Funding

SW was supported by Science and Technology Facilities Council (STFC) studentship ST/S50578X/1 and by the UKRI/NERC and Mitacs Canada grant NE/T014164/1 as part of the UK-Canada Globalink doctoral exchange scheme. KM is partially supported by NERC grant NE/V002554/2.

Acknowledgments

The authors would like to thank all of the individuals involved in the upkeep of the OMNIWeb and SAMPEX data websites.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Boakes, P. D., Milan, S. E., Abel, G. A., Freeman, M. P., Chisham, G., and Hubert, B. (2011). A superposed epoch investigation of the relation between magnetospheric solar wind driving and substorm dynamics with geosynchronous particle injection signatures. *J. Geophys. Res.* 116. doi:10.1029/2010JA016007

Chree, C. (1913). Some phenomena of sunspots and of terrestrial magnetism at kew observatory. *Philosophical Trans. R. Soc. Lond. Ser. A* 212, 75–116. doi:10.1098/rsta.1913.0003

Gabrielse, C., Angelopoulos, V., Runov, A., and Turner, D. L. (2014). Statistical characteristics of particle injections throughout the equatorial magnetotail. *JGR. Space Phys.* 119, 2512–2535. doi:10.1002/2013JA019638

Halford, A. J., Fraser, B. J., and Morley, S. K. (2010). EMIC wave activity during geomagnetic storm and nonstorm periods: CRRES results. *J. Geophys. Res.* 115. doi:10.1029/2010JA015716

Hietala, H., Kilpua, E. K., Turner, D. L., and Angelopoulos, V. (2014). Depleting effects of ICME-driven sheath regions on the outer electron radiation belt. *Geophys. Res. Lett.* 41, 2258–2265. doi:10.1002/2014GL059551

Hutchinson, J. A., Wright, D. M., and Milan, S. E. (2011). Geomagnetic storms over the last solar cycle: A superposed epoch analysis. *J. Geophys. Res.* 116. doi:10.1029/2011JA016463

Katsavrias, C., Daglis, I. A., and Li, W. (2019). On the statistics of acceleration and loss of relativistic electrons in the outer radiation belt: A superposed epoch analysis. *J. Geophys. Res. Space Phys.* 124, 2019JA026569–2768. doi:10.1029/2019JA026569

Kilpua, E. K., Hietala, H., Turner, D. L., Koskinen, H. E., Pulkkinen, T. I., Rodriguez, J. V., et al. (2015). Unraveling the drivers of the storm time radiation belt response. *Geophys. Res. Lett.* 42, 3076–3084. doi:10.1002/2015GL063542

Larsen, B. A., Morley, S. K., Niehof, J. T., and Welling, D. T. (2010). Spacepy. doi:10.5281/zenodo.3252523

Liu, J., Gabrielse, C., Angelopoulos, V., Frissell, N. A., Lyons, L. R., McFadden, J. P., et al. (2011). Superposed epoch analysis of magnetotail flux transport during substorms observed by THEMIS. *J. Geophys. Res.* 116. doi:10.1029/2010JA015886

Meredith, N. P., Horne, R. B., Lam, M. M., Denton, M. H., Borovsky, J. E., and Green, J. C. (2011). Energetic electron precipitation during high-speed solar wind stream driven storms. *J. Geophys. Res.* 116. doi:10.1029/2010JA016293

Morley, S. K., Koller, J., Welling, D. T., Larsen, B. A., Henderson, M. G., and Niehof, J. T. (2011). "Spacepy - a Python-based library of tools for the space sciences," in Proceedings of the 9th Python in Science conference, Austin, TX, June 28–July 3, 2011.

Murphy, K. R., Mann, I. R., Sibeck, D. G., Rae, I. J., Watt, C., Ozeke, L. G., et al. (2020). A framework for understanding and quantifying the loss and acceleration of relativistic electrons in the outer radiation belt during geomagnetic storms. *Space weather.* 18, e2020SW002477. doi:10.1029/2020SW002477

Murphy, K. R., Watt, C. E. J., Mann, I. R., Jonathan Rae, I., Sibeck, D. G., Boyd, A. J., et al. (2018). The global statistical response of the outer radiation belt during geomagnetic storms. *Geophys. Res. Lett.* 45, 3783–3792. doi:10.1002/2017GL076674

O'Brien, T. P., Lorentzen, K. R., Mann, I. R., Meredith, N. P., Blake, J. B., Fennell, J. F., et al. (2003). Energization of relativistic electrons in the presence of ULF power and MeV microbursts: Evidence for dual ULF and VLF acceleration. *J. Geophys. Res.* 108, 1329. doi:10.1029/2002JA009784

Olifer, L., Mann, I. R., Kale, A., Mauk, B. H., Claudepierre, S. G., Baker, D. N., et al. (2021). A tale of two radiation belts: The energy-dependence of self-limiting electron space radiation. *Geophys. Res. Lett.* 48, e2021GL095779. doi:10.1029/2021GL095779

Smirnov, A., Shprits, Y., Allison, H., Aseev, N., Drozdov, A., Kollmann, P., et al. (2022). Storm-time evolution of the equatorial electron pitch angle distributions in Earth's outer radiation belt. *Front. Astron. Space Sci.* 9, 62. doi:10.3389/fspas.2022.836811

Summers, D., and Thorne, R. M. (2003). Relativistic electron pitch-angle scattering by electromagnetic ion cyclotron waves during geomagnetic storms. *J. Geophys. Res.* 108, 1143. doi:10.1029/2002JA009489

Turner, D. L., Kilpua, E. K. J., Hietala, H., Claudepierre, S. G., O'Brien, T. P., Fennell, J. F., et al. (2019). The response of Earth's electron radiation belts to geomagnetic storms: Statistics from the van allen probes era including effects from different storm drivers. *JGR. Space Phys.* 124, 1013–1034. doi:10.1029/2018JA026066

Walach, M., and Grocott, A. (2019). SuperDARN observations during geomagnetic storms, geomagnetically active times, and enhanced solar wind driving. *JGR. Space Phys.* 124, 5828–5847. doi:10.1029/2019JA026816

Walton, S. D., Forsyth, C., Rae, I. J., Meredith, N. P., Sandhu, J. K., Walach, M.-T., et al. (2022). Statistical comparison of electron loss and enhancement in the outer radiation belt during storms. *JGR. Space Phys.* 127, e2021JA030069. doi:10.1029/2021JA030069

Walton, S. D., and Murphy, K. R. (2022). samwalton7645/SEA_Code: sea_norm Frontiers initial release. doi:10.5281/zenodo.6867860

Wang, H., He, Y., Lühr, H., Kistler, L., Saikin, A., Lund, E., et al. (2019). Storm time EMIC waves observed by swarm and van allen probe satellites. *J. Geophys. Res. Space Phys.* 124, 293–312. doi:10.1029/2018JA026299

Whittaker, I. C., Clilverd, M. A., and Rodger, C. J. (2014). Characteristics of precipitating energetic electron fluxes relative to the plasmapause during geomagnetic storms. *J. Geophys. Res. Space Phys.* 119, 8784–8800. doi:10.1002/2014JA020446

Yokoyama, N., and Kamide, Y. (1997). Statistical nature of geomagnetic storms. *J. Geophys. Res.* 102, 14215–14222. doi:10.1029/97JA00903

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 01 December 2022
DOI 10.3389/fspas.2022.1022690

Check for updates

OPEN ACCESS

EDITED BY
Sophie A Murray,
Dublin Institute for Advanced Studies (DIAS), Ireland

REVIEWED BY
Stefano Markidis,
KTH Royal Institute of Technology, Sweden
Ze-Jun Hu,
Polar Research Institute of China, China

*CORRESPONDENCE
Xueling Shi,
xueling7@vt.edu

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 18 August 2022
ACCEPTED 11 November 2022
PUBLISHED 01 December 2022

CITATION
Shi X, Schmidt M, Martin CJ, Billett DD, Bland E, Tholley FH, Frissell NA, Khanal K, Coyle S, Chakraborty S, Detwiller M, Kunduri B and McWilliams K (2022), pyDARN: A Python software for visualizing SuperDARN radar data.
Front. Astron. Space Sci. 9:1022690.
doi: 10.3389/fspas.2022.1022690

COPYRIGHT
© 2022 Shi, Schmidt, Martin, Billett, Bland, Tholley, Frissell, Khanal, Coyle, Chakraborty, Detwiller, Kunduri and McWilliams. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

pyDARN: A Python software for visualizing SuperDARN radar data

Xueling Shi[1,2]*, Marina Schmidt[3], Carley J. Martin[3], Daniel D. Billett[3], Emma Bland[4], Francis H. Tholley[5], Nathaniel A. Frissell[6], Krishna Khanal[7], Shane Coyle[1], Shibaji Chakraborty[1], Marci Detwiller[3], Bharat Kunduri[1] and Kathryn McWilliams[3]

[1]Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, United States, [2]High Altitude Observatory, National Center for Atmospheric Research, Boulder, CO, United States, [3]Institute of Space and Atmospheric Studies, University of Saskatchewan, Saskatoon, SK, Canada, [4]Department of Arctic Geophysics, The University Centre in Svalbard, Longyearbyen, Norway, [5]Department of Computing Sciences, The University of Scranton, Scranton, PA, United States, [6]Department of Physics and Engineering, The University of Scranton, Scranton, PA, United States, [7]Center for Space Plasma and Aeronomic Research, University of Alabama in Huntsville, Huntsville, AL, United States

The Super Dual Auroral Radar Network (SuperDARN) is an international network of high frequency coherent scatter radars that are used for monitoring the electrodynamics of the Earth's upper atmosphere at middle, high, and polar latitudes in both hemispheres. pyDARN is an open-source Python-based library developed specifically for visualizing SuperDARN radar data products. It provides various plotting functions of different types of SuperDARN data, including time series plot, range-time parameter plot, fields of view, full scan, and global convection map plots. In this paper, we review the different types of SuperDARN data products, pyDARN's development history and goals, the current implementation of pyDARN, and various plotting and analysis functionalities. We also discuss applications of pyDARN, how it can be combined with other existing Python software for scientific analysis, challenges for pyDARN development and future plans. Examples showing how to read, visualize, and interpret different SuperDARN data products using pyDARN are provided as a Jupyter notebook.

KEYWORDS

python, Super Dual Auroral Radar Network, radar, ionosphere, space weather

1 Introduction

The Super Dual Auroral Radar Network (SuperDARN) is an international network of more than 30 high frequency (HF, 3–30 MHz) coherent scatter radars that provide continuous, coordinated observations of the electrodynamics of the Earth's upper atmosphere from middle to polar latitudes (Greenwald et al., 1995). The radars detect

coherent backscatter from decameter-scale electron density irregularities which act as tracers for the large-scale plasma circulation in the F-region ionosphere. The radars also detect backscatter from meteor plasma trails at around 90 km altitude, and backscatter from the ground or sea surface, which occurs when the transmitted radio waves undergo total internal reflection by the ionosphere and then scatter on the ground/sea. These three types of backscatter (ionospheric, ground/sea, meteor) can be used for studying the ionospheric signatures of a wide range of phenomena including plasma convection, high latitude plasma structures, gravity waves, magnetohydrodynamic waves, and substorm processes (Chisham et al., 2007; Nishitani et al., 2019, and references therein).

As with many geophysical datasets, effective and accessible data visualization tools are essential for enabling scientists to use the data. The SuperDARN community has a long history of developing and sharing data visualization tools among SuperDARN institutions and with the wider space physics community, which is described further in **Section 3**. However, the software needs of data users change over time and the dataset appears to present a number of challenges to new users. One of the major challenges is that SuperDARN produces several types of multidimensional data products, and these are stored using a highly-customized file format that cannot easily be read using common scientific analysis software. Furthermore, many of the common methods for visualizing SuperDARN data require additional data preparation or assumptions that are standard within the SuperDARN community but are not necessarily obvious to other users, such as the reflection height assumptions required for assigning geographic coordinates to a scattering target (Chisham et al., 2008). While these challenges are distinct from data visualization, they present significant barriers to non-expert users who want to use SuperDARN as a supporting dataset in their scientific work.

To make SuperDARN data more accessible to the wider space physics community, the SuperDARN community has developed a new open-source Python3 library for SuperDARN data visualization called *pyDARN* (SuperDARN Data Visualization Working Group et al., 2022). This new library provides a user-friendly way to visualise SuperDARN data in all the ways that have become standard in the SuperDARN scientific community, including time series plots, range-time plots, field of view plots, scan plots, and global convection maps. pyDARN is maintained on GitHub (https://github.com/SuperDARN/pydarn) by the SuperDARN Data Visualization Working Group (DVWG), which includes volunteer members from many institutions. The package is licensed under the GNU Lesser General Public License version 3 (https://www.gnu.org/licenses/lgpl-3.0.en.html). In this paper, we review the different types of SuperDARN data products, pyDARN's development history and goals, the current structure and functionality of the software, and future development plans. We provide examples demonstrating how the current implementation of pyDARN makes it easy for students, researchers, and other interested individuals to access, visualize, and analyze different SuperDARN data products. We believe that pyDARN as a Python software tool will help advance space science by making the SuperDARN dataset accessible to a broader scientific community.

2 SuperDARN data products

SuperDARN radars operate in the 8–20 MHz frequency range. Each radar consists of a linear array of antennas that are phased electronically to produce a steerable beam that is narrow in azimuth (~5°) and wide in vertical extent (~40°). This beam is divided into 70–110 *range gates* which are 45 km wide in the standard operational mode. Depending on the number of beams and range gates used, the total field of view covers a 52–78° azimuthal sector that extends to 3,500–5,000 km in range. This allows the radars to detect coherent backscatter from scattering targets over a very wide geographical area, including two-dimensional observations from pairs of radars with overlapping fields of view. In addition to the main antenna array, most SuperDARN radars have a smaller 'interferometer' array of antennas that is displaced by about 100 m from the main array, and this is used to determine the angle-of-arrival (elevation angle) of the backscatter returns. The radars transmit sequences of unevenly-spaced pulses, and the returns from 20 to 30 sequences obtained during the standard ~3 s dwell time are averaged together before further analysis. In the standard operational mode, a complete scan of all 16–24 beams is completed every minute. When the data from multiple radars are combined to produce convection maps, the standard time resolution of the data product is 2 min.

The primary data products of SuperDARN are power (signal-to-noise ratio), Doppler velocity, and spectral width. For radars with an interferometer array, the elevation angle of the received backscatter can also be calculated after careful calibration (Chisham et al., 2021). These data products are determined in a two-stage process commencing with the level-zero-type data product consisting of the in-phase and quadrature (I&Q) samples measured by the receiver for each multipulse sequence. These samples are then processed into a complex-valued autocorrelation function (ACF) for each range gate along the sampling beam. These ACFs are the 'raw' data product that are distributed to the scientific community, as described in the *Data Availability Statement*. The power, velocity, spectral width and elevation angle in each range gate are then determined using a program called *FitACF*, which is one of the core routines in the standard SuperDARN

data analysis software, the *Radar Software Toolkit* (RST) (SuperDARN Data Analysis Working Group et al., 2022). This software also includes routines for combining the line-of-sight (LoS) velocity measurements from individual radars onto a hemispheric grid of equal-area cells, and for applying a global fitting technique called *Map Potential* to determine the global convection pattern (Ruohoniemi and Baker, 1998).

SuperDARN data are stored using a self-describing file format called 'DataMap' (DMap). This file format was developed to suit the operational environment of the radar sites, including encoding the real-time data stream provided by some radars. A description of the data fields included in each format is included in the RST documentation (https://radar-software-toolkit-rst.readthedocs.io/en/latest/). The DMap files have a different structure depending on the type of data stored, and they are named as follows:

- *.iqdat: I&Q voltage samples, named IQDat file
- *.rawacf: Raw autocorrelation functions, named RawACF file
- *.fitacf: Fitted parameters including power, Doppler velocity, spectral width, elevation angle, and their errors, named FitACF file.
- *.grid/*.grd: Fitted parameters from all radars in one hemisphere placed on a grid of equal-area cells spanning 1° of geomagnetic latitude, named Grid file.
- *.map: Contains the same data as *.grid files, as well as the fitted coefficients describing the convection pattern, named Map file.

Note that prior to 2006, SuperDARN data were stored in a binary data format, and these files cannot currently be read using the Python tools described in this paper. However, the RST includes routines for converting these binary files to the DMap format so that they can be used in pyDARN.

In addition to the data formats above, the SuperDARN community maintains set of text files containing basic technical information about each radar that is required for data analysis and visualization. These 'hardware files' include information such as the radar location, scanning boresight direction, the number of beams and the beam separation. These files are stored in a GitHub repository (https://github.com/SuperDARN/hdw) and pyDARN can automatically access them as required.

3 pyDARN development

3.1 History

Data visualisation tools have been shared within the international SuperDARN community since the early years of the collaboration. Early versions of the RST (then known as *radops*) were primarily used for processing of the radar data on-site, with basic plotting functionality included for range-time-intensity (RTI) and field-of-view (FOV) plots. This code, however, could not handle non-standard radar operating modes or coordinate systems beyond rectilinear. In an effort to expand SuperDARN plotting functionalities and to make it more flexible for the growing SuperDARN user base, S. Milan at the University of Leicester led the development of a dedicated SuperDARN data visualisation software package called *Go* in the early 2000s. *Go* was written in the *Interactive Data Language* (IDL), which was the most commonly used programming language in the community at the time. As a result, *Go* became popular throughout the SuperDARN community and received contributions from numerous scientists from many institutions.

In 2009, a postdoctoral fellow at Virginia Tech, L. Clausen, developed *Go* into a new IDL package, the *Data Visualisation Toolkit* (DaViT). Over time, research students at Virginia Tech added new analysis and visualisation tools to the package including HF raytracing (de Larquier et al., 2013a), total electron content (TEC) maps overlayed with SuperDARN data (Thomas et al., 2013), and data diagnostics. *DaViT* also included the *THEMIS Data Analysis Software* (TDAS, http://themis.igpp.ucla.edu/software.shtml, see also Angelopoulos et al. (2019)) for downloading space- and ground-based data from the THEMIS mission. This allowed users to plot THEMIS all-sky imager data overlaid with radar data (e.g., Gallardo-Lacourt et al., 2014). The *DaViT* source code was distributed online, and it was also used to run a suite of online data visualisation tools on the Virginia Tech SuperDARN website (http://vt.superdarn.org). These widely-used online visualisation tools are connected to a SuperDARN data server at Virginia Tech, allowing anyone in the world to generate plots of SuperDARN data from any time period since late 1993.

In 2012, the growing popularity of the Python language in the space physics community prompted research students at Virginia Tech to port *DaViT* from IDL to Python2, creating a new package called *DaViTPy* (de Larquier et al., 2013b). This effort was led by S. de Larquier, A. J. Ribeiro, and N. A. Frissell. *DaVitPy*'s place among other space physics Python packages is described in Burrell et al. (2018). The package was developed collaboratively on Github, which facilitated code contributions from the wider scientific community and allowed development to continue after the original developers graduated. *DaViTPy* provided FTP access to the Virginia Tech data server, so the data necessary for producing a particular plot could be downloaded automatically. Python tools for performing coordinated studies with incoherent scatter radars and satellite missions, as well as Python wrappers for various Fortran-based models and analysis routines (IGRF, AACGM, NRLMSISE, HWM, IRI, and HF raytracing) were also added to *DaVitPy*. The SuperDARN data visualisation tools, the provision of processed SuperDARN data (FitACF, Grid, and Map files) *via* the Virginia Tech data server, and the integration with

other space physics software and models in *DaVitPy* led to a substantial increase in the use of SuperDARN data in the wider space physics community.

By 2018 it was apparent that *DaVitPy* had become challenging to maintain due to its broad scope, and the package had a long list of dependencies that made it difficult for users to install. The planned deprecation of Python2.7 at the beginning of 2020 also provided the impetus to review the package structure and scope while transitioning to Python3. At the same time, the SuperDARN Principal Investigators decided that processed SuperDARN data (FitACF, Grid, and Map) would no longer be distributed, instead favouring a citable archive of the raw data that users must process themselves with the RST. While this was a necessary step to support scientific reproducibility, this decision has created an extra barrier for users who must now install the RST and work through multiple steps of data analysis before they can visualise the data. All of these factors led to a community decision to deprecate *DaVitPy* and develop a new package in Python3 that would focus solely on SuperDARN data visualisation. *DaVitPy* was officially deprecated and archived in 2020 (Ribeiro et al., 2020).

The *pyDARN* project was then initiated at the University of Saskatchewan, with M. Schmidt and M. Detwiller as the lead developers of the data visualisation code and the I/O code respectively. Development began in 2018 and the first official release was published in March 2020 (Detwiller et al., 2020).

3.2 Development goals

pyDARN is developed with four primary goals in mind: usability, accessibility, maintainability and flexibility. These goals, or objectives that would fall within these goals, are often used when describing programming 'best practices'. For example, in the list of good Python design principles of PEP 20 *The Zen of Python* (Tim Peters, 2004), tenets such as 'Readability counts' (accessibility), 'If the implementation is hard to explain, it is a bad idea' (usability, maintainability) and 'Sparse is better than dense' (flexibility) are included. pyDARN's goals are upheld during development through extensive code review and unit testing.

For usability, it is vital that pyDARN is easy to download, install and use. The pyDARN software package is listed on PyPi, making it easy to install from any Linux or Mac terminal using pip: pip install pydarn. pyDARN functions can then be accessed in the same way as other popular software packages. For example, the 'Maps' and 'RTP' modules can be imported using: from pydarn import Maps, RTP. Complete examples are provided in the accompanying Jupyter notebook. PyDARN now includes all of the standard SuperDARN data visualisation types, and these are described further in **Section 5**. The pyDARN developers seek user feedback on new features as part of the code review process, and welcome new code contributions and other input from the scientific community. However, since pyDARN relies on scientists and engineers volunteering their time to contribute to the package, requests for new functionalities can only be accommodated if someone is willing to volunteer their time to develop them.

SuperDARN data visualisation often requires additional analysis tools to prepare the data for plotting. These include determining the distance (slant range) to the backscatter, projecting the field of view onto geographic coordinates, and performing coordinate transforms. Therefore, pyDARN includes a wide range of functions that are not directly related to plotting. These functions are not typically called by users, but will be called by the plotting code in an 'under-the-hood' fashion. For example, pyDARN includes several virtual height models (e.g., Chisham et al., 2008) that are an essential part of assigning geographic coordinates to the backscatter. In all cases, the standard virtual height model that is most widely accepted by the SuperDARN community is called by default if no model is specified by the user. This greatly improves pyDARN's ease of use and accessibility for non-expert users and for those who wish to quickly view the data before fine-tuning the results later.

pyDARN is developed in a way that allows for simple maintainability and expansion. Code is modularized as much as possible to avoid duplicating functionality across multiple different modules and methods. For example, fan, grid and convection map plots all share access to methods that generate map projections, radar FOV locations, and Python colour maps, meaning that any bug fixes can be applied without needing to consider the intricacies of all the functions that call them.

Expansion of capability is also very important for pyDARN. SuperDARN users may wish to build upon existing functionality, or add new features entirely, so pyDARN must be designed in such a way that code changes or expansions are easy to achieve. For example, new plotting map projections can be developed directly into the separate module dedicated to projections. pyDARN methods that already use projections can then use the new projections immediately without any further code modification, as they already contain a call to the 'projections' enumeration members. Similarly, new virtual height models for geolocating data can be appended into the corresponding module, making it available for range and coordinate estimations. More information about the layout and structure of pyDARN's code can be found in **Section 4**.

Finally, pyDARN is flexible such that users can easily integrate and overlay data from non-SuperDARN sources, such as from satellites or ground-based instruments. Plots are returned as regular Matplotlib axis objects with their inherent ability to be customised. In addition, data and their corresponding coordinates can also be returned to be utilised in any Python plotting code.

3.3 Workflow and testing

To achieve the goals described in **Section 3.2**, the pyDARN development team has a published workflow and testing regime, which is described fully in the documentation. The team uses Pull Requests and Code Reviews, requiring that at least two developers approve the code to maintain standards. Anyone is welcome to develop in pyDARN, and the team fosters a welcoming and inclusive environment though a code of conduct.

At present the library has users on Windows, MacOS, openSUSE, CentOS, Ubuntu, and many other Linux operating systems. In general, all new components are tested on Windows, MacOS and at least one Linux based system. The majority of our new users are based in Windows, and many of the developers use MacOS, but users based in operations rather than research use numerous Linux systems and as such pyDARN is required to be system-independent.

4 Structure and components

pyDARN is structured as an application containing several internal packages. The structure of the current version, *pyDARN 3.0* (SuperDARN Data Visualization Working Group et al., 2022), is illustrated in **Figure 1**. There are three top-level directories: *pydarn*, *docs* and *test*. The *pydarn* directory contains the main codebase that is shipped and installed for users. The remaining top-level directories are containers for the documentation and the testing suite.

4.1 Main codebase

The *pydarn* directory contains four packages. There are three main packages used for data visualisation, and an additional package, *exceptions*, which provides useful warnings and troubleshooting information that might be required to visualise a particular dataset. The exceptions are grouped into modules relating to generic plotting, specific types of plots, reading data files, and accessing radar hardware information. In addition, the team uses *warnings* to advise the user that optional dependencies are not installed, or that the data file is not suitable for plotting in the requested format. For example, a warning message is also displayed on import to remind the user to cite *pyDARN* in published work.

The central package in the pyDARN library is the *plotting* package. This package allows the user to visualise SuperDARN data in all the formats that have become standard in the SuperDARN scientific community (see **Section 2**), such as range-time plots and convection maps. Each type of visualisation has its own module, and examples are provided in **Section 5**. The plotting directory also contains two utilities modules, *color_maps.py* and *projections. py*, which provide special color maps developed especially for SuperDARN data visualisation, and polar or Cartesian *matplotlib* axis objects, respectively. Note that for any plotting function with a color bar, the color map can also be customized by changing the *cmap* parameter.

The *utils* package provides a range of utilities that support the plotting package. Some modules are merely used for storing physical constants (*constants.py*) or dictionaries for storing radar control program information (*superdarn_cpid.py*). Other modules contain methods that are essential to plotting, such as determining the geocentric coordinates of each cell in the radar field of view (*geo.py*), and organising the data into 'scans' (*scan.py*) so that a sequence of observations covering the entire azimuthal field of view can be visualised as a 'fan' plot (see **Section 5**). These modules replicate similar routines available in the RST. A number of modules (*coordinates.py*, *range_estimations.py* and *virtual_heights.py*) are used as callable classes, where the user selects an option in the plotting method call, which then automatically selects the chosen method from the module. For example, the *virtual_heights.py* module includes two *virtual height models* that can be used for mapping the location of backscatter targets in the ionosphere (Chisham et al., 2008). A class within the *virtual_heights.py* module will be called and this option will be entered into the module, and the corresponding virtual height model, *Standard* or *Chisham*, will be used. This design allows the user to easily select virtual height models, slant range estimates for ionospheric or ground scatter, and geographic or geomagnetic coordinate systems for use in any compatible plotting method. It also allows for greater flexibility and extensibility in development, since additional options, models or coordinate systems can be added and integrated with ease.

The *utils* package also includes the *superdarn_radar.py* module, which is designed to retrieve, update and format information about each SuperDARN radar that is required for data visualisation such as the radar location, the direction of the centre beam (boresight), and the dates of operation. This information is provided by each radar's Principal Investigator though the SuperDARN hardware files repository on GitHub (https://github.com/SuperDARN/hdw). Upon installation of pyDARN, the hardware repository is copied from GitHub and placed within the library under the *utils* package. The user can update the hardware files to the latest version at any time using *get_hdw_files*, which re-installs the hardware files from source. The hardware file contents can be read using *read_hdw_files*, which formats the data into a class object. The *superdarn_radar.py* module also contains general information about each radar, including the full name, 3-letter station code, and the name and institution of the principal investigator. The *superdarn_radar.py* module is designed to work outside of the data visualisation and will provide hardware information to the

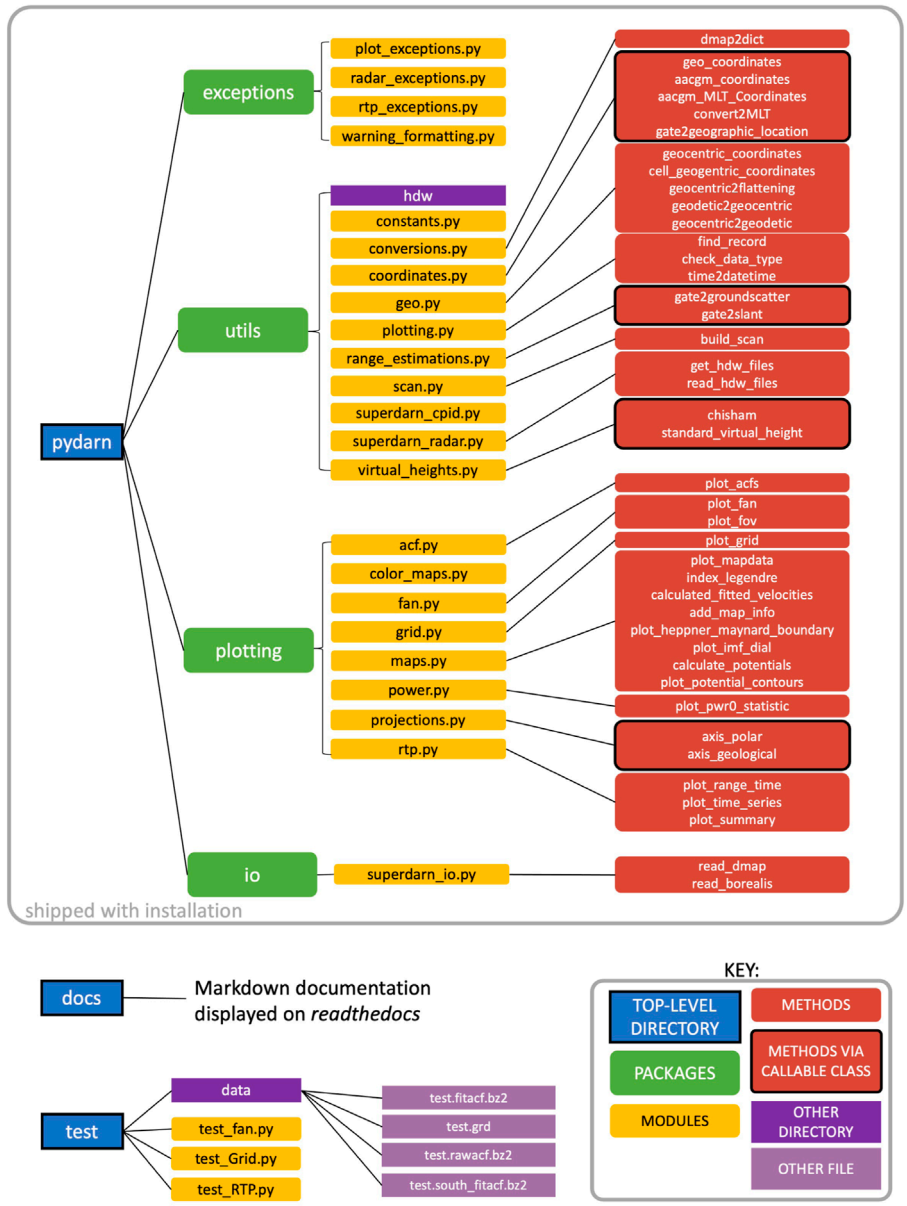


FIGURE 1
Layout and structure of the pyDARN v3.0.0 library, including the packages, modules and some of the more commonly used methods that are shipped with the installation. Also shown are other peripheral directories that are useful in development and discussion.

user through the *SuperDARNRadars* class. Further information and examples are given in **Section 5** and the Jupyter notebook.

4.2 Documentation

To enhance the usability of the pyDARN library, extensive documentation and tutorials are written for each module. These are updated for each version release and can be found at https://pydarn.readthedocs.io/en/main/. The documentation is written in *Markdown* and built using *MkDocs*. To increase the accessibility, usability and flexibility of the code, the team uses standard PEP8 styling for the codebase. This styling is checked frequently using the flake8 library (Cordasco, 2022). The NumPy docstring convention is used to document modules, methods, and classes within the codebase. This helps the developers maintain the codebase and makes it easier for new developers to join the team.

4.3 Testing suite

The testing suite, found in the GitHub repository, is not included in the library at installation and is only used for development purposes. It allows the developers to determine whether a change to the codebase has altered or broken any of the library's functionality. The testing suite uses *pytest* (Krekel et al., 2022) and checks for multiple different inputs to each of the plotting methods. These tests can be performed by cloning the repository and running *pytest* in the top-level directory, which will run all of the tests within the directory at once.

4.4 Dependencies

pyDARN has several dependencies that are required to use the library. When installing with pip, the pyDARN setup will check that all dependencies are installed and update them if necessary. pyDARN currently requires Python version 3.6 or higher and the following packages to be installed: pyDARNio version 1.1.0 or higher (SuperDARN Data Standards Working Group et al., 2022), AACGMv2 (Burrell et al., 2021), Matplotlib version 3.3.4 or higher (Hunter, 2007), PyYAML (Simonov, 2021), and NumPy (Harris et al., 2020). pyDARNio requires that the h5py (Collette, 2008), deepdish (Larsson, 2021) and pathlib2 (Troffaes, 2022) libraries are installed. Finally, Cartopy (Met Office, 2010–2015) is an optional dependency of pyDARN that is used for displaying coastlines on some types of plots.

4.4.1 The IO package and pyDARNio

pyDARNio is an open source Python library for reading, writing and reformatting SuperDARN data. pyDARNio provides read support for the standard SuperDARN DMap data format and the HDF5-format files produced by the *Borealis* digital radar system (SuperDARN Canada, 2022). It also contains methods to write HDF5-format files and convert them to the DMap format.

pyDARNio was originally developed as an internal package of pyDARN. However, it was found that the IO sections of code were very useful for other Python-based SuperDARN software, including the *Borealis* radar control system which was developed alongside. Therefore, pyDARNio was separated from pyDARN to create a light-weight IO package that could be integrated into other packages that did not require the data visualisation tools in pyDARN. As a result, pyDARNio is now a dependency for pyDARN, and it is automatically installed during the pyDARN installation. Methods within pyDARNio are called by pyDARN's internal *io* package without the user having to import pyDARNio separately. The *io* package copies the pyDARNio method *SDarnRead* and converts it to *SuperDARNRead* in pyDARN, allowing all DMap files including IQDat, RawACF, FitACF, Grid, and Map files to be read by pyDARN.

Besides enabling data to be read for plotting purposes, pyDARNio has some additional functionalities for reading the HDF5-format *Borealis* data and compressing time-sequenced data into large arrays. This reduces the file size and allows the data to be read quickly. The pyDARNio package therefore serves the greater purpose of providing read and write support for a larger set of SuperDARN filetypes like those used in Borealis. The modularization of the Python file manipulation functions for SuperDARN into the pyDARNio package constrains all file-related tasks to a single package.

4.4.2 Altitude-Adjusted Corrected Geomagnetic and other coordinate systems

pyDARN produces a range of azimuthal projection plots that use either geographic or geomagnetic coordinate systems. Fan plots are produced from FitACF files using a projection of the radar FOV onto a geographic or geomagnetic coordinate system. The method for determining the geographic coordinates of each range–beam cell in the FOV is included in *geo.py*. This is a translation from the original *cnvtcoord.c* procedure in the RST that was written in *C* by K. Baker, R.J. Barnes and D. Andre. These geographic coordinates are then converted into geomagnetic coordinates depending on the input parameters used to produce the fan plot. The data in Grid and Map files are given in geomagnetic coordinates, and pyDARN does not perform any coordinate transformations when generating grid plots or convection maps. Therefore, these plots cannot currently be produced in geographic coordinates.

The geomagnetic coordinate system used is version two of the *Altitude-Adjusted Corrected Geomagnetic* (AACGM) coordinate system (Shepherd, 2014). The forward and inverse transformations between geographic and AACGM coordinates are performed using the AACGMv2 Python library (Burrell et al., 2021), which is a Python wrapper for the AACGM-v2 *C* library (Shepherd, 2019). This library also calculates magnetic local time (MLT), which is commonly used in SuperDARN azimuthal plots instead of magnetic longitude. Example conversions between geographic and AACGM coordinates are available in the Burrell et al. (2021) documentation.

4.4.3 Cartopy (optional)

Cartopy is a Python package for processing geospatial data (Met Office, 2010–2015). In pyDARN it is used to plot the outlines of the continents in fan plots, grid plots and map plots (see **Section 5**). At present, the implementation in pyDARN is available only when plotting in geographic coordinates, where a return of the ax object allows the ax.coastlines() option to be used. The next minor release of pyDARN will allow coastlines from Cartopy to be plotted in geomagnetic coordinates as well.

In the future, the coastlines option will be integrated into the projections.py module for use with all geospatial plotting as a keyword rather than an addition to the axis.

The installation process for Cartopy is more complicated than the other pyDARN dependencies due to different requirements for each operating system and some users may prefer a more streamlined version. Therefore, pyDARN does not install Cartopy by default and instead directs users to the Cartopy installation instructions at https://scitools.org.uk/cartopy/docs/latest/. If a user attempts to use a Cartopy-dependent method without installing Cartopy, pyDARN will raise an exception and display a message in the terminal window.

TABLE 1 Commonly used data product and the corresponding string name in SuperDARN DMap files.

Data product	String name
Sky Noise	noise.sky
Transmission Frequency (MHz)	tfreq
Number of averages	nav
Control Program ID	cp
Line-of-sight (LoS) velocity (m/s)	v
Spectral Width (m/s)	w_l
Power (dB)	p_l
Elevation angle (degrees)	elv

5 Functionalities

This section provides examples of pyDARN's primary and auxiliary functionalities and descriptions of scientific analysis that can be done using these plotting functions. Detailed documentation on pyDARN installation and all functionalities mentioned below can be found at https://pydarn.readthedocs.io/en/latest/.

5.1 Primary functionalities

In this subsection we provide examples of various functions for reading SuperDARN data files, and for generating different types of data visualizations including the time series plots, range-time parameter plots, FOV and fan plots, grid plots, and convection plots. The code required to reproduce each plot is provided in the accompanying Jupyter notebook, and information about accessing the underlying data is given in the Data Availability Statement.

5.1.1 Reading SuperDARN files

pyDARN reads SuperDARN data files through the IO package pyDARNio as described in **Subsection 4.4.1**. pyDARN reads SuperDARN data from the DMap files as follows:

Import pydarn
File = "path/to/file"
SDarn_Data = pydarn. SuperDARNRead (file).

This puts the contents of the file into a Python object called *SDarn_data*. Then the user will need to specify the type of SuperDARN data that was provided, e.g.

Fitacf_data = SDarn_Data.read_fitacf ()
Rawacf_data = SDarn_Data.read_rawacf ()
Map_data = SDarn_Data.read_map ()
Grid_data = SDarn_Data.read_grid ()
Iqdat_data = SDarn_Data.read_iqdat ()

It is also possible to read DMap files without knowing the file type using *read_dmap*, as shown in the example below. The resulting variable is a Python dictionary. This function also demonstrates the ability to stack the object creation and data reading into one line of code:

Dmap_data = pydarn. SuperDARNRead (). read_dmap (file).

SuperDARN RawACF and FitACF files usually contain 2 h of data and the files are compressed for distribution using Bzip2 (Seward, 2010). Therefore, multiple input files are required to produce some types of plots. Examples of generic SuperDARN file reading, opening multiple FitACF data files and combining them into one variable for plotting are provided in the Jupyter notebook in the plotting function demonstrations.

In addition to SuperDARN data files, pyDARN can also be used to access radar and hardware information and obtain coordinates for radar FOV. The function *pydarn. read_hdw_file* can extract radar hardware data based on the 3 letter abbreviation of the radar (e.g., 'sas' for the Saskatoon radar). The *pydarn. Coords.GEOGRAPHIC* and *pydarn. Coords.AACGM* functions provide easy ways to obtain the coordinates of a specific radar's FOV in geographic and AACGM coordinates. Examples on accessing radar and hardware information are also provided in the Jupyter notebook.

5.1.2 Time-series plots

Time-series plots are simple line plots of a single fitted parameter from a specific beam and range gate. The parameters that can be plotted are shown in **Table 1**. **Figure 2** shows an example of the time series plot for the LoS velocity parameter from beam 12 and gate 13 of the Prince George (PGR) radar on 25 January 2016 at 00:00–02:00 UT. Clear signatures of ultra-low frequency (ULF) waves are present during this time interval as reported by Shi et al. (2018).

5.1.3 Range-time parameter plots

Range-Time Parameter (RTP) plots, also known as range-time-intensity (RTI) plots, are two-dimensional representations of one fitted parameter plotted as a function of range and time for a specific beam. This is the most common way of viewing data from a single radar. The plotting function *plot_range_time* includes the keyword 'parameter' that allows the user to specify which parameter from **Table 1** will be plotted. This is specified

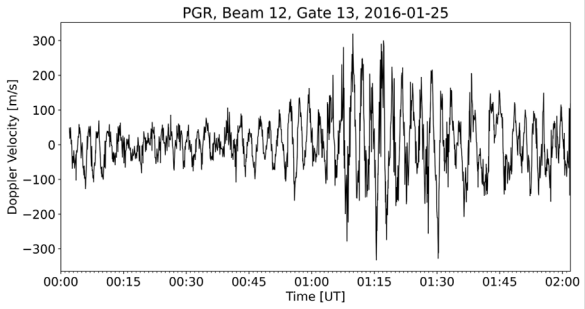


FIGURE 2
Doppler velocity time series from beam 12 and gate 13 of the Prince George radar at 00:00–02:00 UT on 25 January 2016.

using the doc string names shown in the second column, and the possible choices are LoS velocity (v), spectral width (w_l), elevation angle (elv), and power (p_l). By default, the velocity parameter is plotted.

Figure 3 shows a RTP plot from beam two of the Clyde River radar when the beam is looking from dawn to dusk at 08:00–10:30 UT on 18 December 2017. Positive (negative) velocities indicate motion towards (away from) the radar. As shown by the arrows (added manually after the figure is generated using pyDARN), there are several sequential flow structures that are moving towards the radar location. These structures correspond to dawnward return flows of the auroral forms resulting from the flux transfer events reported by Hwang et al. (2020).

pyDARN provides several ways to modify the vertical axis by specifying the keyword *range_estimation*. SuperDARN radar beams usually have 75 range gates of length 45 km. In order to have range gates on *y* axis, *range_estimation = pydarn.RangeEstimation.RANGE_GATE* can be used. We can also convert the range gates numbers into a slant range in kilometres using *range_estimation = pydarn.RangeEstimation.SLANT_RANGE*. Ground Scatter Mapped Range is another option that utilizes ground scatter to improve the slant range estimations (Bristow et al., 1994). This can be done by using *range_estimation = pydarn.RangeEstimation.GSMR*.

5.1.4 Summary plots

Summary plots allow the user to generate a set of RTP plots in which the backscatter is colour-coded by the power (p_l), velocity (v), spectral width (w_l) and, if available, the elevation (elv). This type of plot is useful when more than one parameter is needed to interpret the data, for example to distinguish between ionospheric scatter and ground scatter based on the Doppler velocity and spectral width. The plots also include the sky noise (noise.sky), transmission frequency (tfreq), the number of multipulse sequences transmitted during the dwell time ('Nave', see **Section 2**), and the control program identification number (CPID), which provide an overview of the operational parameters during the time interval.

An example summary plot is shown in **Figure 4**. This plot shows data from beam 15, channel one of the Hankasalmi SuperDARN radar on 13 February 2014. The top three panels show the summary information described above. The remaining four panels are RTP plots where the backscatter is colour-coded in turn by signal-to-noise ratio (SNR), Doppler velocity, spectral width and elevation. In the velocity panel, a positive velocity indicates motion towards the radar (blue shift) and a negative velocity indicates motion away from the radar (red shift). Backscatter classified as ground scatter using criteria defined in the RST are shaded grey in this panel.

Several features can be identified in this plot based on the power, velocity and spectral width of the backscatter. A wide band of 1-hop ground scatter spanning more than 30

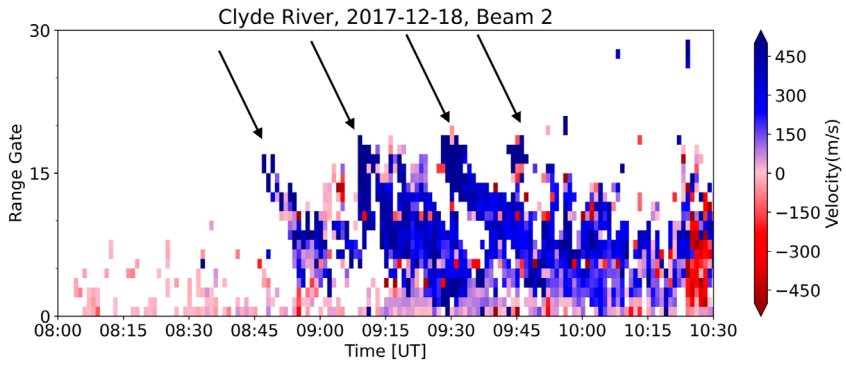


FIGURE 3
Range-time-parameter (RTP) plot of the line-of-sight velocity from beam two of the Clyde River radar at 08:00–10:30 UT on 18 December 2017. Arrows show ionospheric signatures of flux transfer events.

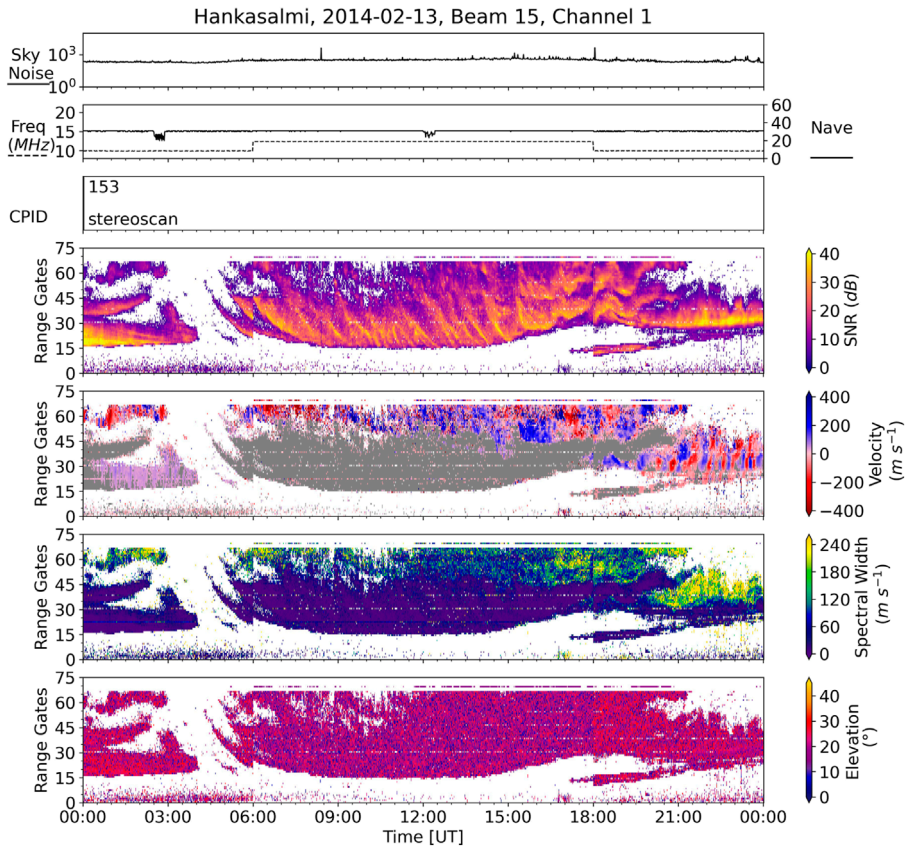


FIGURE 4
Summary plot for beam 15 of the Hankasalmi SuperDARN radar showing power, velocity, spectral width and elevation angle parameters on 13 February 2014.

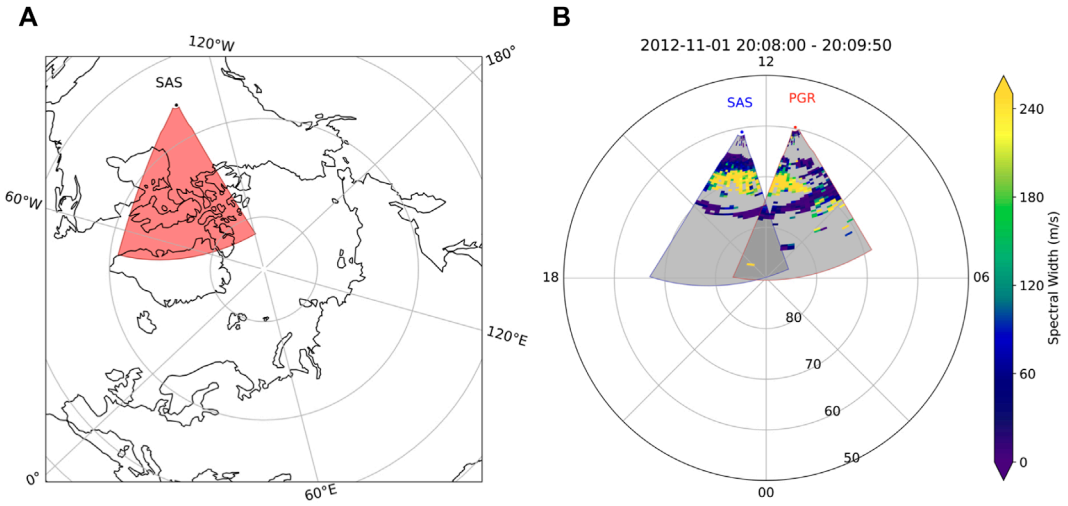


FIGURE 5
FOV plot (left) for the Saskatoon radar and fan plot (right) for the Saskatoon and Prince George radars at 20:08 UT on 1 November 2012.

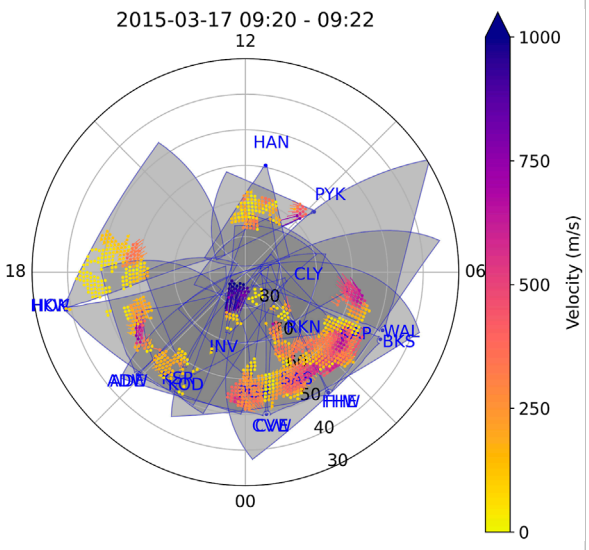


FIGURE 6
Gridded velocity map derived from multiple northern hemisphere radars at 09:20–09:22 UT on 17 March 2015.

TABLE 2 Common parameters in a MAP file that can be plotted by pyDARN.

Name	Parameter name	Map field name
Fitted Velocity (m/s)	MapParams.FITTED_VELOCITIES	NA
Modeled Velocity (m/s)	MapParams.MODEL_VELOCITIES	model.vel.median
Raw Velocity (m/s)	MapParams.RAW_VELOCITIES	vector.vel.median
Power (dB)	MapParams.POWER	vector.pwr.median
Spectral Width (m/s)	MapParams.SPECTRAL_WIDTH	vector.wdt.median

range gates is present from about 05:00 UT to 20:00 UT. This population of backscatter has been marked as ground scatter using grey shading in the velocity panel. This backscatter population exhibits striations in the SNR associated with F-region electron density perturbations caused by atmospheric gravity wave activity (*cf.* Samson et al., 1990). At ranges beyond the 1-hop ground scatter is a population of 1.5-hop ionospheric scatter that can be distinguished from the ground scatter based on the higher velocity and spectral width values. A smaller population of 1.5-hop ionospheric scatter is also present in range gates 55–65 at 00:00–03:00 UT. Backscatter from the 0.5-hop propagation mode is detected in gates 15–30 at 00:00–04:00 UT, and also from 20:00 UT onwards. In this latter population, the Doppler velocity oscillations are evidence of ULF wave activity (*cf.* Ponomarenko et al., 2003). Although elevation angle measurements can be useful for identifying different propagation modes in SuperDARN data, the elevation angle data for this time period appear to be unreliable. If desired, this panel can be removed from the plot by setting plot_elv = False.

5.1.5 Field of view and fan plots

FOV plots show the scanning region of specific radar(s) given the radar station ID. This method provides options for plotting colored FOVs, boundary lines, the radar locations, and labelling the radar names, as well as plotting in different coordinate systems (geographic, AACGM, and MLT). To actually visualize the data from a full radar scan, Fan plot should be used. Fan plot supports the four parameters: velocity (v), spectral width (w_l), power (p_l) and elevation (elv).

Figure 5A shows the FOV of the Saskatoon radar in geographic coordinates where Cartopy has been used to display the coastlines. **Figure 5B** shows the spectral width data from the Saskatoon (SAS) and Prince George (PGR) radars at 20:08 UT on 1 November 2012 in MLT coordinates. The region of high spectral width values (yellow tiles in **Figure 5B**) are associated with particle precipitation in the cusp region (*cf.* Baker et al., 1995). Therefore, the boundary where the spectral width transitions from low to high values indicates the equatorward edge of the particle precipitation region, or equivalently the open close field line boundary (Chisham and Freeman, 2003).

5.1.6 Grid plots

Grid data is a highly processed data product derived from FitACF data, which has been used with various models to calculate ionospheric electric fields and Joule heating rates (e.g., Matsuo et al., 2021; Wu and Lu, 2022). The LoS vectors are mapped within the cells of an equal-area grid, which is defined in the geomagnetic coordinates system with each cell measuring 1° in latitude, to eliminate biases that would derive from the much denser sampling over nearer radar range gates. The vectors contributed by a radar to a particular cell are averaged over a fixed period of time to obtain the Grid LoS data product. More details of Grid data processing can be found in Ruohoniemi and Baker (1998). The Grid plotting function is suitable for visualizing the combined data from multiple radars. This function can be used to plot the following gridded parameters: gridded LoS velocity, power, and spectral width in AACGM coordinate system. **Figure 6** shows an example of a gridded velocity map with good data coverage in the northern hemisphere at 09:20–09:22 UT on 17 March 2015.

5.1.7 Convection map plots

SuperDARN Map files are produced from the combined Grid data and contain the fitted coefficients from a statistical convection model describing the ionospheric convection pattern. Historically, different techniques have been developed to fit the gridded LOS velocities with various background statistical models (Ruohoniemi and Baker, 1998; Cousins et al., 2013; Thomas and Shepherd, 2018). Common parameters in a MAP file that can be plotted by pyDARN is summarized in **Table 2**.

The default plotting parameter of a Map file is the fitted velocity, which represent the fitted convection pattern. Another

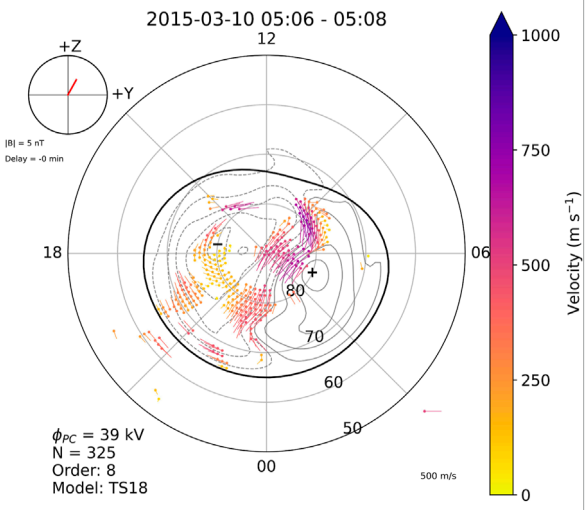


FIGURE 7
Ionospheric convection map with color-coded fitted velocities at 05:06–05:08 UT on 10 March 2015.

velocity that can be plotted is the modeled velocity, which is the velocity median of the model vectors from a statistical background model (e.g., Thomas and Shepherd, 2018). Other plotting parameters include the raw velocity, power, and spectral width, which are the weighted average velocity, power, and spectral width within every equal-area MLAT/MLON grid cell, respectively. **Figure 7** shows an example of the fitted velocity in the northern hemisphere 05:06–05:08 UT on 10 March 2015. The Jupyter notebook contains further examples of plotting options such as color-coded potential contours and convection pattern from the southern hemisphere.

5.1.8 Other plots

pyDARN also provides functions to plot auto-correlation functions and statistics of parameters from RawACF files. Specifically, the function *pydarn.ACF.plot_acfs* plots the real and imaginary parts of the auto-correlation function from a RawACF file for the specified beam and range gate. The function *pydarn.Power.plot_pwr0_statistic* applies a statistical function (e.g., the mean using numpy. mean) to the lag-0 power for each range/record and plots the results from all records as a function of time. This is a useful function to analyze background radio interference. Examples on how to use these functions can be found in the provided Jupyter notebook and pyDARN documentation.

5.2 Auxiliary functions

In addition to primary functionalities, pyDARN also offers some auxiliary functions to support SuperDARN data analysis. Note that the two functions described below are still under development and are not part of pyDARN as of now. Thus, **Figure 8** and **Figure 9** cannot currently be produced using pyDARN.

5.2.1 MUSIC for traveling ionospheric disturbances parameter estimation

Traveling Ionospheric Disturbances (TIDs) are quasi-periodic variations in ionospheric electron density that propagate horizontally with time. Medium Scale TIDs (MSTIDs) have periods of ∼15–60 min, horizontal wavelengths of several hundred kilometers, and horizontal phase speeds of $\lesssim$ 300 m s^{-1} (e.g., Ogawa et al., 1987; Samson et al., 1990). MSTIDs may be associated with atmospheric gravity waves (AGWs) (e.g., Hines, 1960; Bristow et al., 1994) or electrodynamic processes (e.g., Kelley, 2011; Atilaw et al., 2021) and therefore are important for understanding atmosphere-ionosphere-geospace coupling. MSTIDs propagating through a SuperDARN radar FOV cause concave and convex structures in the bottomside ionosphere that focus and de-focus HF radio waves returning to the ground, causing the bands of ground scatter detected by the radar to move with time (Samson et al., 1989; Samson et al., 1990; Frissell et al., 2014; Frissell et al., 2016). The ground scatter band movement caused by TIDs also has implications for the operation of terrestrial HF communications systems (Frissell et al., 2022).

In order to measure the wave parameters (i.e., period, horizontal wavelength, phase speed, and propagation direction) of TIDs moving through a SuperDARN FOV, Samson et al. (1990) first implemented the MUltiple SIgnal Classification (MUSIC) algorithm for use with SuperDARN data. MUSIC is a general-purpose algorithm developed by Schmidt (1986) to produce unbiased parameter estimations of wave-like structures detected by an array of sensors. In the case of SuperDARN, each range-beam cell is considered an individual sensor. MUSIC was re-implemented in Python 2 for use with SuperDARN by Frissell et al. (2014) and incorporated as part of the DaViTPy library (Ribeiro et al., 2020). MUSIC for SuperDARN has now been ported to Python 3 by Tholley and Frissell (2022) and is available as an add-on package for use with pyDARN.

The new Python 3 SuperDARN MUSIC package (hereafter referred to as the MUSIC package) uses an object-oriented approach to loading, processing, and visualizing the data in order to facilitate signal processing and ensure that the results, metadata, and provenance of each processing step are always available to the researcher. To begin analysis, FitACF data is loaded into a MUSIC object using pyDARN/pyDARNio routines. The data is stored in a 3D (2D horizontal space + time) NumPy array as a "dataset" attached to the MUSIC object, along with relevant metadata. Methods are provided for processing and visualizing the data stored in the dataset. Every time a signal processing algorithm is applied, a new dataset is created with

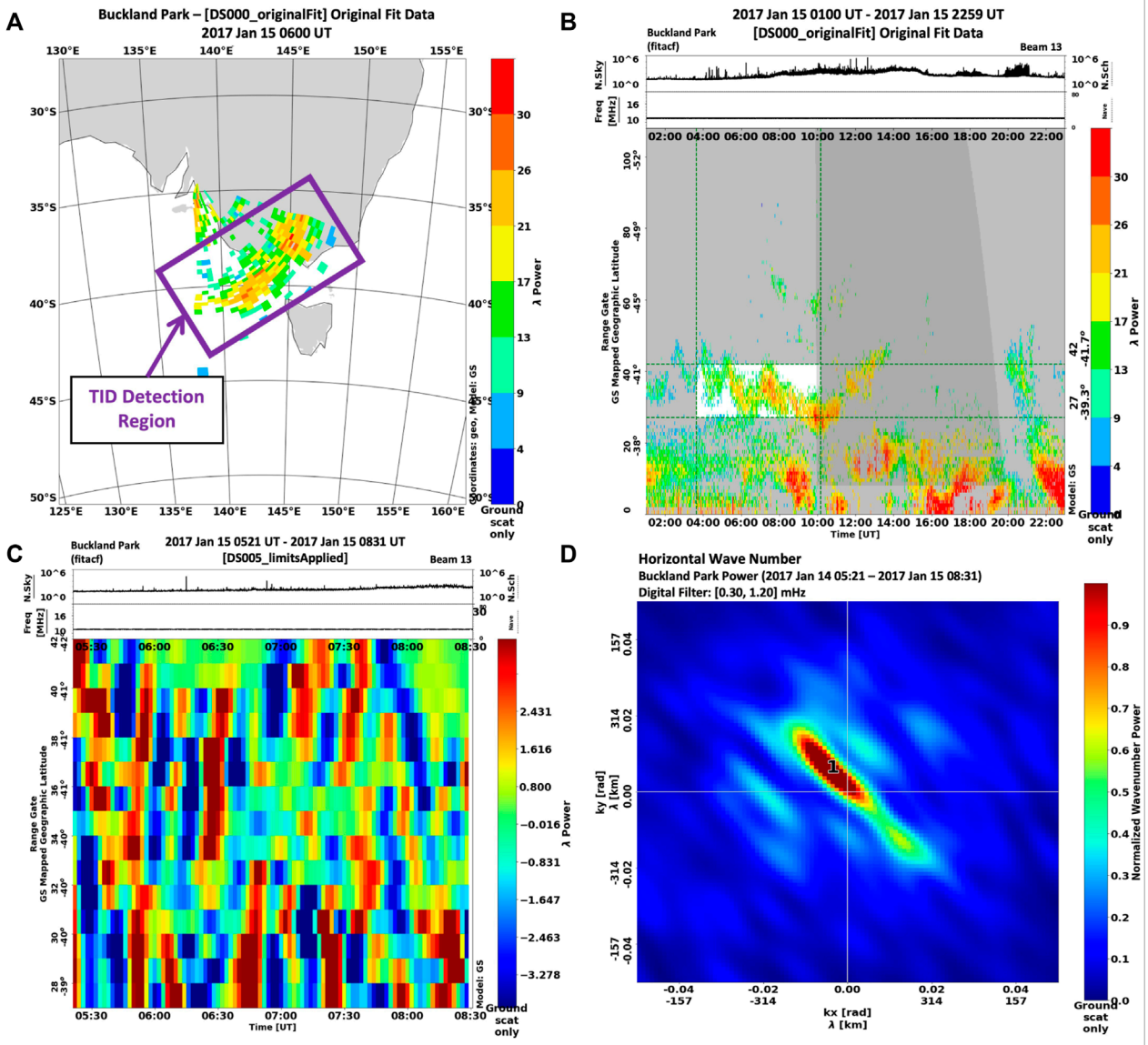


FIGURE 8
Selected outputs from an analysis conducted with the Tholley and Frissell (2022) MUSIC package using ground scatter power observations from the Buckland Park (BPK) SuperDARN radar made on 15 January 2017. **(A)** Fan plot of one scan of raw data using Ground Scatter Mapped Range estimation. The band of ground scatter where the TID can be detected is indicated by a purple box. **(B)** RTP plot with times and ranges selected for TID processing highlighted in white. **(C)** RTP plot showing data interpolated in space and time and filtered with a $T = 14 - 56$ min bandpass filter. **(D)** Final wavenumber array indicating the presence of TIDs observed in the radar data. Distance from the origin indicates wavelength, with shorter wavelengths farther away from the origin. Clockwise angle relative to the +y axis indicates propagation direction relative to geographic North. The hotspot marked 1 has a horizontal wavelength $\lambda_h \approx 890$ km, horizontal phase speed $v_h \approx 430$ m s^{-1} propagation azimuth $\phi_{azm} \approx -45°$, and period $T \approx 35$ min.

the result and attached to the MUSIC object. The MUSIC object automatically keeps track of the processing history and allows any prior stage of processing to be easily retrieved. Example Jupyter Notebooks are included with the MUSIC package as documentation and to demonstrate package functionality.

In addition to its data analysis capabilities, the MUSIC package also contains a forward-modeling simulation routine that can be used to test and validate results produced by the MUSIC algorithm. An example of this functionality is provided in a Jupyter Notebook within the MUSIC package. In this routine, the user can generate synthetic radar data with TID-like characteristics using a known sinusoidal function. The user can specify the sinusoid amplitude, period, wavelength, phase, and geographic source location for an arbitrary number of TIDs.

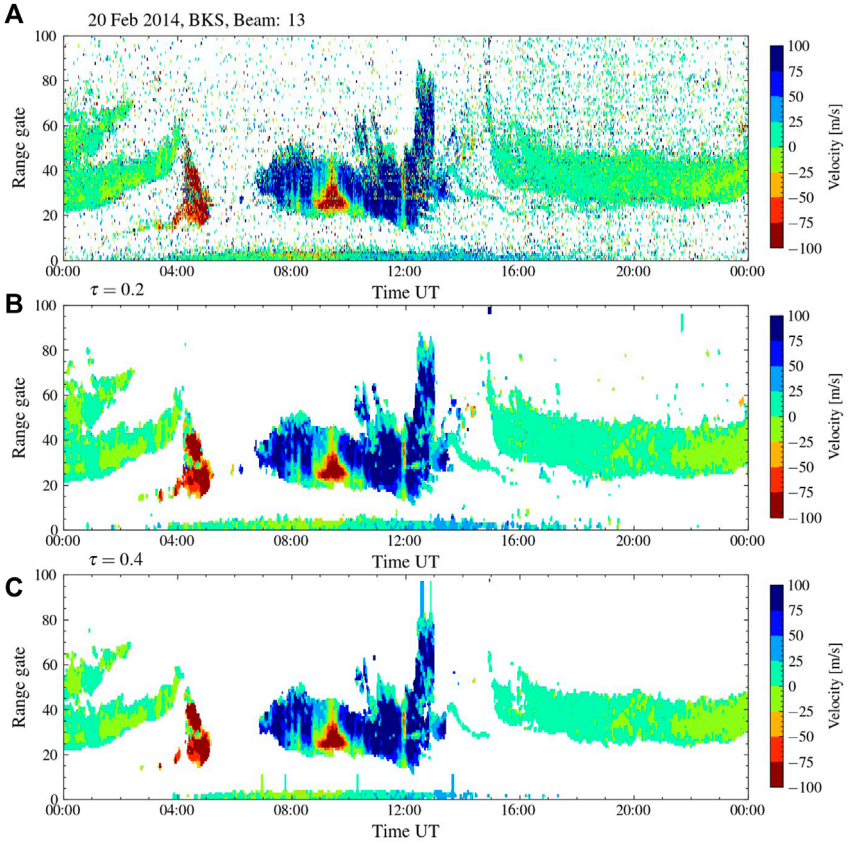


FIGURE 9
RTP plots showing SuperDARN Blackstone radar LoS velocity observations from beam 13 **(A)** without filtering and filtered using different weights **(B)** $\tau =$ 0.2, and **(C)** $\tau =$ 0.4 on 20 February 2014.

Gaussian white noise can also be added to allow for sensitivity testing.

Figure 8 presents selected outputs from an analysis conducted with the MUSIC package using ground scatter power observations from the Buckland Park (BPK) SuperDARN radar made on 15 January 2017. Once the data is loaded into a MUSIC object, the MUSIC fan plotting method can be used to generate a spatial view of the raw data as shown in Figure 8A. The band of ground scatter where the TID can be detected is indicated by a purple box. Next, a MUSIC object method allows the user to select a time and range of data for further TID analysis, as shown in the RTP plot in Figure 8B. The data within the white region will then be interpolated in space and time to remove data gaps and provide regularly gridded observations for efficient signal processing. After interpolation, the data is filtered with a $T =$ 14–56 min bandpass filter to select the MSTIDs in the data. This interpolated, filtered data is plotted in Figure 8C. The next step in the MUSIC analysis involves calculating the Fast Fourier Transform (FFT) and the cross-spectral matrix. The FFT and cross-spectral matrix are then be used to calculate the horizontal wave number array, shown in Figure 8D.

Hotspots in Figure 8D horizontal wavenumber array indicate the presence of TIDs observed in the radar data. Distance from the origin indicates wavelength, with shorter wavelengths farther away from the origin. Clockwise angle relative to the +y axis indicates propagation direction relative to geographic North. That is, +y is northwards, −y is southwards, +x is eastwards, and −x is westwards. The MUSIC package will automatically detect hotspots and report the wave parameters for that particular hotspot. In Figure 8D, the hotspot marked 1 has a horizontal wavelength $\lambda_h \approx$ 890 km, horizontal phase speed $v_h \approx$ 430 m s^{-1} propagation azimuth $\phi_{azm} \approx -45°$, and period $T \approx$ 35 min.

5.2.2 Boxcar (median) filter

Large-scale imaging of the global convection maps is generated using velocity observations. Velocity data from a single radar scan can be mapped to geographic or geomagnetic

coordinates using standard SuperDARN mapping algorithms and the magnetic coordinate system described by Baker and Wing (1989). Previous studies using SuperDARN data have shown that performing a "boxcar" (or median) filtering involving spatiotemporal sampling is sometimes necessary to reduce salt and pepper noise in the data (Ruohoniemi and Baker, 1998; Ribeiro et al., 2011; Ponomarenko et al., 2022). Specifically, for a scan collected at time $t_{\mathcal{I}}$, the filtering sample includes the scans performed at $t_{\mathcal{I}-1}$ and $t_{\mathcal{I}+1}$, where the subscripts indicate the scan order. The spatial sampling is performed over a 3×3 beam/gate template centered on the cell of interest. Thus the filtering sample for a cell indexed $\mathcal{B}$ for beam number and $\mathcal{G}$ for gate number, i.e., $(\mathcal{B},\mathcal{G})_{\mathcal{I}}$, encompasses the velocity data contained in the 3×3 beam/gate template centered on this cell through the three successive scans centered on $\mathcal{I}$. This suggests that the sampling is modified if a limiting scan, beam, or gate is encountered. We also assigned weights to favor the samples collected at the exact position of the target cell $(\mathcal{B},\mathcal{G})$ or at the target time, $\mathcal{I}$. The sample is median-filtered to produce a "best" estimate of the LoS velocity for the target cell. The particular merit of weighted boxcar-filtering over simple averaging in this application is its rejection of anomalous data. The assignment of a velocity value to the target cell depends on the weighted number (τ) of velocity values obtained for the sampling; if too low, no value is determined, and the cell is turned off; if sufficiently high, a value is determined, and the cell is turned on.

Example LoS velocity output from the boxcar filter with $\tau = 0.2, 0.4$ for Blackstone radar is presented in **Figure 9**. The top panel presents the original FitACF data from beam 7, while the bottom two panels present outputs from the boxcar filter with the two τ values. Note that with an increase in τ value we are observing a smother pattern in the velocity distribution, suggesting a reduction in noise, small-scale structures, and the number of data points.

6 Discussion

The necessity for developing pyDARN was primarily motivated by two key factors. First, to increase the accessibility of different SuperDARN data products by developing an open source software tool that removes the challenges associated with the DMap file format. Unlike other more commonly used file formats like NetCDF or HDF5, the software to work with the DMap format is not widely available, especially in Python. Secondly, to make SuperDARN, a widely used space science dataset with a rich history of providing scientific breakthroughs (Chisham et al., 2007; Nishitani et al., 2019), interoperable with other Python based space science toolkits such as SpacePy (Morley et al., 2011) and pySPEDAS. pyDARN addresses both these points successfully, and in doing so makes SuperDARN compliant with several aspects of FAIR (Findability, Accessibility, Interoperability, and Reusability) principles (Wilkinson et al., 2016). In particular, pyDARN significantly increases the "accessibility" of SuperDARN data to the wider space science community by providing the capability to generate publication-quality plots with only a few lines of code. Furthermore, pyDARN increases the "interoperability" between different space science datasets by making it easy to compare and plot SuperDARN data with others such as solar wind magnetic field and plasma data from the OMNI dataset (obtained from SpacePy). Lastly, pyDARN can leverage Python's rich ecosystem of tools and libraries for developing web frameworks to generate high quality plots. This feature makes it significantly easier for researchers and students to quickly access and browse through different SuperDARN data products without writing any code. The SuperDARN research groups at the University of Saskatchewan and Virginia Tech are leading two such efforts, with an online pyDARN platform available at https://superdarn.ca/pydarn.

The development of pyDARN was led by researchers from the SuperDARN group at the University of Saskatchewan, with major contributions from several other early-career scientists and students from other SuperDARN institutions across the world. Several developers have been employed using funding from research grants that support SuperDARN operations and research, and many other developers have simply volunteered their working time or personal time to contribute. As a result, the development and maintenance of pyDARN is limited by the availability of grants and volunteers. In fact, this has been a serious limitation for several other open source software projects as well. To overcome this problem, organizations like the Apache Software Foundation have been actively funding the development of large-scale open source projects. More recently, funding agencies like the National Aeronautics and Space Administration (NASA) have recognized the importance of open source software in advancing Earth and space sciences, and have been directly funding such efforts. The "Support for Open Source Tools, Frameworks, and Libraries" is one such opportunity released by NASA in 2020. Overall, it is important to ensure that open source tools such as pyDARN are continuously funded to serve the evolving needs of the broader space science community.

In the near future pyDARN will continue to serve the needs of the SuperDARN as well as the broader space science community. The pyDARN developers plan to create a suite of tools such as histograms and superposed epoch analysis that enable statistical analysis of SuperDARN data. Other development work will focus on increasing interoperability with other space science datasets to increase research productivity. For example, plotting SuperDARN convection patterns along with other data such as Active Magnetosphere and Planetary Electrodynamics Response Experiment (AMPERE) and Global Navigation Satellite System (GNSS) Total Electron Content (TEC) may provide crucial

insights into magnetosphere-ionosphere coupling processes. Such capability would make it easy to compare observations with the predictions of physics-based models and increase the utility of pyDARN to the modeling community. While working towards improving interoperability with other datasets, the narrow scope of pyDARN as a SuperDARN data visualization package will be upheld in line with the development goals described in **Section 3.2**.

7 Summary

pyDARN is a Python-based tool for SuperDARN data visualization that is actively developed by the SuperDARN Data Visualization Working Group. In this paper, we have presented the pyDARN development history, goals, and workflow. We have described the software structure and components and provided examples of the main functionalities. In addition, we discussed applications, challenges, and future plans for pyDARN development. pyDARN as an open source library initiated by the SuperDARN community for data visualization, and it attracts users from other communities to join the code testing and development. The pyDARN development team encourages community-driven development to make this SuperDARN data visualization software better in the future.

Data availability statement

The SuperDARN radar datasets and Jupyter notebook used to generate figures in this paper can be found https://doi.org/10.5281/zenodo.7005203 in Zenodo. Raw SuperDARN data (RawACF and IQDat format) from 1993 to 2019 are published in the Federated Research Data Repository (https://www.frdr-dfdr.ca/repo/collection/superdarn) with a separate DOI for each year. There are also three SuperDARN data mirrors maintained by the University of Saskatchewan (USask), the British Antarctic Survey (BAS) and the National Space Science Center, Chinese Academy of Sciences (NSSC). For access to the USask data mirror *via* Globus, contact superdarn@usask.ca. Information about accessing the BAS data mirror is available at https://www.bas.ac.uk/project/superdarn/#data. The NSSC mirror can be accessed at https://superdarn.nssdc.ac.cn.

Author contributions

All authors wrote sections of the manuscript, contributed to manuscript revision, read, and approved the submitted version.

All authors have contributed equally to this work and share last authorship.

Funding

MS, CM, and MD are supported by funding from the Canada Foundation for Innovation, the Canadian Space Agency's Geospace Observatory (GO) Canada program, and Innovation Saskatchewan. DB is funded *via* the European Space Agency Living Planet Fellowship under project "HLPF-SSA". XS is supported by National Science Foundation (NSF) grants AGS-1935110 and AGS-2025570 and National Aeronautics and Space Administration (NASA) grant 80NSSC21K1677. BK is supported by NSF under grants AGS-1822056 and AGS-1839509. FT is supported by NASA grant 80NSSC21K0002, and NF is supported by NASA grant 80NSSC21K0002 and NSF grant AGS-2045755. SC is supported by NSF grants OPP-1744828 and AGS-2027168.

Acknowledgments

We acknowledge J Michael Ruohoniemi, Steve Milan, and Kevin Stern who provide useful feedback on the history of SuperDARN data visualization tools for the manuscript. The authors acknowledge the use of SuperDARN data. SuperDARN is a collection of radars funded by national scientific funding agencies of Australia, Canada, China, France, Italy, Japan, Norway, South Africa, United Kingdom and the United States of America.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Angelopoulos, V., Cruce, P., Drozdov, A., Grimes, E., Hatzigeorgiu, N., King, D., et al. (2019). The Space Physics Environment Data Analysis System (SPEDAS). *Space Sci. Rev.* 215, 9–46. doi:10.1007/s11214-018-0576-4

Atilaw, T. Y., Stephenson, J. A., and Katamzi-Joseph, Z. T. (2021). Multitaper analysis of an mstid event above Antarctica on 17 march 2013. *Polar Sci.* 28, 100643. SuperDARN/Studies of Geospace Dynamics - Today and Future. doi:10.1016/j.polar.2021.100643

Baker, K. B., Dudeney, J. R., Greenwald, R. A., Pinnock, M., Newell, P. T., Rodger, A. S., et al. (1995). HF radar signatures of the cusp and low-latitude boundary layer. *J. Geophys. Res.* 100, 7671–7695. doi:10.1029/94JA01481

Baker, K. B., and Wing, S. (1989). A new magnetic coordinate system for conjugate studies at high latitudes. *J. Geophys. Res.* 94, 9139–9143. doi:10.1029/JA094iA07p09139

Bristow, W., Greenwald, R., and Samson, J. (1994). Identification of high-latitude acoustic gravity wave sources using the Goose Bay HF radar. *J. Geophys. Res.* 99, 319–331. doi:10.1029/93ja01470

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—An overview of python in heliophysics. *J. Geophys. Res. Space Phys.* 123, 10384–10402. doi:10.1029/2018JA025877

Burrell, A. G., van der Meeren, C., Laundal, K. M., and van Kemenade, H. (2021). aburrell/aacgmv2: Version 2.6.2. [software]. doi:10.5281/zenodo.4437229

Chisham, G., Burrell, A. G., Marchaudon, A., Shepherd, S. G., Thomas, E. G., and Ponomarenko, P. (2021). Comparison of interferometer calibration techniques for improved SuperDARN elevation angles. *Polar Sci.* 28, 100638. SuperDARN/Studies of Geospace Dynamics - Today and Future. doi:10.1016/j.polar.2021.100638

Chisham, G., and Freeman, M. P. (2003). A technique for accurately determining the cusp-region polar cap boundary using SuperDARN HF radar measurements. *Ann. Geophys.* 21, 983–996. doi:10.5194/angeo-21-983-2003

Chisham, G., Lester, M., Milan, S. E., Freeman, M. P., Bristow, W. A., Grocott, A., et al. (2007). A decade of the Super Dual Auroral Radar Network (SuperDARN): Scientific achievements, new techniques and future directions. *Surv. Geophys.* 28, 33–109. doi:10.1007/s10712-007-9017-8

Chisham, G., Yeoman, T. K., and Sofko, G. J. (2008). Mapping ionospheric backscatter measured by the SuperDARN HF radars—Part 1: A new empirical virtual height model. *Ann. Geophys.* 26, 823–841. doi:10.5194/angeo-26-823-2008

Collette, A. (2008). h5py. [software].

Cordasco, I. S. (2022). Flake8 v5.0.4. [software].

Cousins, E. D. P., Matsuo, T., and Richmond, A. D. (2013). SuperDARN assimilative mapping. *J. Geophys. Res. Space Phys.* 118, 7954–7962. doi:10.1002/2013JA019321

de Larquier, S., Ponomarenko, P., Ribeiro, A. J., Ruohoniemi, J. M., Baker, J. B. H., Sterne, K. T., et al. (2013a). On the spatial distribution of decameter-scale subauroral ionospheric irregularities observed by SuperDARN radars. *J. Geophys. Res. Space Phys.* 118, 5244–5254. doi:10.1002/jgra.50475

de Larquier, S., Ribeiro, A., Frissell, N., Spaleta, J., Kunduri, B., Thomas, E., et al. (2013b). "A new open-source Python-based space weather data access, visualization, and analysis toolkit," in AGU Fall Meeting Abstracts 2013, San Francisco, CA, December 9–13, 2013. IN51B–1545.

Detwiller, M., Schmidt, M., Billett, D., Reimer, A., Peters, D., Burrell, A., et al. (2020). SuperDARN/pyDARN: Pydarn v1.0.0 release. [software]. doi:10.5281/zenodo.3727270

Frissell, N. A., Baker, J. B. H., Ruohoniemi, J. M., Greenwald, R. A., Gerrard, A. J., Miller, E. S., et al. (2016). Sources and characteristics of medium-scale traveling ionospheric disturbances observed by high-frequency radars in the north American sector. *J. Geophys. Res. Space Phys.* 121, 3722–3739. doi:10.1002/2015JA022168

Frissell, N. A., Baker, J., Ruohoniemi, J. M., Gerrard, A. J., Miller, E. S., Marini, J. P., et al. (2014). Climatology of medium-scale traveling ionospheric disturbances observed by the midlatitude Blackstone SuperDARN radar. *J. Geophys. Res. Space Phys.* 119, 7679–7697. doi:10.1002/2014JA019870

Frissell, N. A., Kaeppler, S. R., Sanchez, D. F., Perry, G. W., Engelke, W. D., Erickson, P. J., et al. (2022). First observations of large scale traveling ionospheric disturbances using automated amateur radio receiving networks. *Geophys. Res. Lett.* 49, e2022GL097879. doi:10.1029/2022gl097879

Gallardo-Lacourt, B., Nishimura, Y., Lyons, L. R., Zou, S., Angelopoulos, V., Donovan, E., et al. (2014). Coordinated SuperDARN THEMIS ASI observations of mesoscale flow bursts associated with auroral streamers. *JGR. Space Phys.* 119, 142–150. doi:10.1002/2013JA019245

Greenwald, R. A., Baker, K. B., Dudeney, J. R., Pinnock, M., Jones, T. B., Thomas, E. C., et al. (1995). DARN/SuperDARN: A global view of the dynamics of high-latitude convection. *Space Sci. Rev.* 71, 761–796. doi:10.1007/BF00751350

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Hines, C. O. (1960). Internal atmospheric gravity waves at ionospheric heights. *Can. J. Phys.* 38, 1441–1481. doi:10.1139/p60-150

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Hwang, K.-J., Nishimura, Y., Coster, A., Gillies, R., Fear, R. C., Fuselier, S. A., et al. (2020). Sequential observations of flux transfer events, poleward-moving auroral forms, and polar cap patches. *JGR. Space Phys.* 125, e2019JA027674. doi:10.1029/2019ja027674

Kelley, M. C. (2011). On the origin of mesoscale TIDs at midlatitudes. *Ann. Geophys.* 29, 361–366. doi:10.5194/angeo-29-361-2011

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., Bruhin, F., et al. (2022). Pytest 7.1.2. [software].

Larsson, G. (2021). Deepdish 0.3.7. [software].

Matsuo, T., Fan, M., Shi, X., Miller, C., Ruohoniemi, J. M., Paul, D., et al. (2021). Multiresolution modeling of high-latitude ionospheric electric field variability and impact on joule heating using SuperDARN data. *J. Geophys. Res. Space Phys.* 126, e2021JA029196. doi:10.1029/2021ja029196

Met Office (2010–2015). *Cartopy: A cartographic python library with a matplotlib interface*. Exeter, Devon: Met Office. [software]. doi:10.5281/zenodo.1182735

Morley, S. K., Koller, J., Welling, D. T., Larsen, B. A., Henderson, M. G., and Niehof, J. T. (2011). "Spacepy - a Python-based library of tools for the space sciences," in Proceedings of the 9th Python in science conference (SciPy 2010), Austin, TX, June 28 - July 3, 2010.

Nishitani, N., Ruohoniemi, J. M., Lester, M., Baker, J. B. H., Koustov, A. V., Shepherd, S. G., et al. (2019). Review of the accomplishments of mid-latitude Super Dual Auroral Radar Network (SuperDARN) HF radars. *Prog. Earth Planet. Sci.* 6, 27–57. doi:10.1186/s40645-019-0270-5

Ogawa, T., Igarashi, K., Aikyo, K., and Maeno, H. (1987). NNSS satellite observations of medium-scale traveling ionospheric disturbances at southern high-latitudes. *J. Geomagn. Geoelec.* 39, 709–721. doi:10.5636/jgg.39.709

Ponomarenko, P. V., Bland, E. C., McWilliams, K. A., and Nishitani, N. (2022). On the noise estimation in Super Dual Auroral Radar Network data. *Radio Sci.* 57, e2022RS007449. doi:10.1029/2022RS007449

Ponomarenko, P. V., Menk, F. W., and Waters, C. L. (2003). Visualization of ULF waves in SuperDARN data. *Geophys. Res. Lett.* 30, 1926. doi:10.1029/2003GL017757

Ribeiro, A. J., Ruohoniemi, J. M., Baker, J. B. H., Clausen, L. B. N., de Larquier, S., and Greenwald, R. A. (2011). A new approach for identifying ionospheric backscatter in midlatitude SuperDARN HF radar observations. *Radio Sci.* 46, RS4011. doi:10.1029/2011RS004676

Ribeiro, A., Sterne, K., de Larquier, S., Reimer, A., Wessel, M., Maimaiti), M. R. M., et al. (2020). vtsuperdarn/davitpy: Final release of davitpy. [software]. doi:10.5281/zenodo.3824466

Ruohoniemi, J. M., and Baker, K. B. (1998). Large-scale imaging of high-latitude convection with Super Dual Auroral Radar Network HF radar observations. *J. Geophys. Res.* 103, 20797–20811. doi:10.1029/98JA01288

Samson, J. C., Greenwald, R. A., Ruohoniemi, J. M., and Baker, K. B. (1989). High-frequency radar observations of atmospheric gravity waves in the high-latitude ionosphere. *Geophys. Res. Lett.* 16, 875–878. doi:10.1029/GL016i008p00875

Samson, J. C., Greenwald, R. A., Ruohoniemi, J. M., Frey, A., and Baker, K. B. (1990). Goose Bay radar observations of earth-reflected, atmospheric gravity waves in the high-latitude ionosphere. *J. Geophys. Res.* 95, 7693–7709. doi:10.1029/JA095iA06p07693

SuperDARN Data Analysis Working Group, Burrell, A., Thomas, E., Schmidt, M., Bland, E., Coco, I., Sterne, K.T., et al. (2022). Superdarn/rst: Rst 4.7. [software]. doi:10.5281/zenodo.6473603

SuperDARN Data Standards Working Group, Rohel, R., Chartier, A., Krieger, K., Billett, D., Burrell, A., Martin, C.J, et al. (2022). Superdarn/pydarnio: Pydarnio v1.1.1. [software]. doi:10.5281/zenodo.5914211

SuperDARN Data Visualization Working Group, Schmidt, M., Martin, C., Shi, X., Tholley, F., Billett, D., Peters, D., et al. (2022). Superdarn/pydarn: Pydarn v3.0. [software]. doi:10.5281/zenodo.6473574

Schmidt, R. (1986). Multiple emitter location and signal parameter estimation. *IEEE Trans. Antennas Propag.* 34, 276–280. doi:10.1109/TAP.1986.1143830

Seward, J. (2010). Bzip2/libbz2. [software].

Shepherd, S. G. (2019). AACGM-v2. [software].

Shepherd, S. G. (2014). Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations. *J. Geophys. Res. Space Phys.* 119, 7501–7521. doi:10.1002/2014JA020264

Shi, X., Baker, J. B. H., Ruohoniemi, J. M., Hartinger, M. D., Murphy, K. R., Rodriguez, J. V., et al. (2018). Long-lasting poloidal ULF waves observed by multiple satellites and high-latitude SuperDARN radars. *JGR. Space Phys.* 123, 8422–8438. doi:10.1029/2018JA026003

Simonov, K. (2021). PyYAML. [software].

SuperDARN Canada (2022). Borealis—a control system for USRP based digital radars. Available at: https://borealis.readthedocs.io/en/latest/.

Tholley, F. H., and Frissell, N. A. (2022). MUltiple SIgnal classification (MUSIC) for SuperDARN in Python 3. [software] Github Repo: Available at: https://github.com/HamSCI/pyDARNmusic.

Thomas, E. G., Baker, J. B. H., Ruohoniemi, J. M., Clausen, L. B. N., Coster, A. J., Foster, J. C., et al. (2013). Direct observations of the role of convection electric field in the formation of a polar tongue of ionization from storm enhanced density. *J. Geophys. Res. Space Phys.* 118, 1180–1189. doi:10.1002/jgra.50116

Thomas, E. G., and Shepherd, S. G. (2018). Statistical patterns of ionospheric convection derived from mid-latitude, high-latitude, and polar SuperDARN HF radar observations. *J. Geophys. Res. Space Phys.* 123, 3196–3216. doi:10.1002/2018JA025280

Tim Peters (2004). Pep 20 - the zen of Python. Available at: https://peps.python.org/pep-0020/.

Troffaes, M. C. M. (2022). pathlib2. [software].

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Sci. Data* 3, 160018–160019. doi:10.1038/sdata.2016.18

Wu, H., and Lu, X. (2022). Data assimilation of high-latitude electric fields: Extension of a multi-resolution Gaussian process model (lattice kriging) to vector fields. *Space Weather* 20, e2021SW002880. doi:10.1029/2021SW002880

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 02 December 2022
DOI 10.3389/fspas.2022.1066480

Check for updates

OPEN ACCESS

EDITED BY
Guozhu Li,
Institute of Geology and Geophysics (CAS), China

REVIEWED BY
Zhipeng Ren,
Institute of Geology and Geophysics (CAS), China
Tong Dang,
University of Science and Technology of China, China

*CORRESPONDENCE
Jeff Klenzing,
jeff.klenzing@nasa.gov

†These authors have contributed equally to this work and share first authorship

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 10 October 2022
ACCEPTED 04 November 2022
PUBLISHED 02 December 2022

CITATION
Klenzing J, Smith JM, Halford AJ, Huba JD and Burrell AG (2022), sami2py—Overview and applications.
Front. Astron. Space Sci. 9:1066480.
doi: 10.3389/fspas.2022.1066480

COPYRIGHT
© 2022 Klenzing, Smith, Halford, Huba and Burrell. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

sami2py—Overview and applications

Jeff Klenzing[1]*, Jonathon M. Smith[1,2†], Alexa J. Halford[1], J. D. Huba[3] and Angeline G. Burrell[4]

[1]ITM Physics Laboratory NASA Goddard, Greenbelt, MD, United States, [2]Department of Physics, Catholic University of America, Washington, DC, United States, [3]Syntek Technologies Inc, Fairfax, VA, United States, [4]Space Science Division, Naval Research Laboratory, Washington, DC, United States

sami2py is a *Python* module that runs the SAMI2 (Sami2 is Another Model of the Ionosphere) ionospheric model, as well as load and archive the results. SAMI2 is a model developed by the Naval Research Laboratory to simulate the motions of plasma in a two-dimensional ionospheric environment along a dipole magnetic field. SAMI2 solves for the chemical and dynamical evolution of seven ion species in this environment (H^+, He^+, N^+, O^+, N_2^+, NO^+, and O_2^+). The *Python* implementation allows for additional modifications to the empirical models within SAMI2, including the exospheric temperature in the empirical thermosphere and the input of E×B ion drifts. The code is open source and available to the community on GitHub. The work here discusses the implementation and use of *sami2py*, including integration with the pysat ecosystem and the *growin* python package for ionospheric calculations. As part of the Application Usability Level (AUL) framework, we will discuss the usability of this code in terms of several ionospheric applications.

KEYWORDS

ionosphere, ionospheric model, SAMI2 model, python (programming language), open source software, software, plasma instability

1 Introduction

SAMI2 is a model developed at the Naval Research Laboratory to simulate the motions of plasma in a two dimensional (2D) ionospheric environment along dipole magnetic field lines (Huba et al., 2000). The model itself is written in FORTRAN (Backus and Heising, 1964) and distributed under an open source license. It has been applied to a variety of low-latitude ionospheric physics problems, including longitudinal variation of airglow measurements (England et al., 2008), the effect of neutral winds on instability growth rates (Zhan and Rodrigues., 2018), and plasma bubble refilling rates (Otsuka et al., 2021). Because of the open source nature of the code, other variations have been built with additional physics considerations such as photoelectron transport (Varney et al., 2012; Krall and Huba, 2019).

The *sami2py* software package (Klenzing et al., 2022) is an interface built in *Python* (Van Rossum and Drake, 2009) designed to initiate, modify, and manage runs of the SAMI2 model for ionospheric studies. The original version was written in MatLab (Higham and Higham, 2016) as part of a systematic study of solar minimum

(Klenzing et al., 2013), but has been rewritten and modified to comply with the Heliophysics *Python* ecosystem (e.g., Annex et al., 2018; Burrell et al., 2018). The software has been made open source and available to the community for modification to better improve reproducibility of ionospheric research (e.g., Gil et al., 2016). **Section 2** will discuss the implementation of *sami2py*. **Section 3** will discuss a brief overview of a standard workflow of the code, including example output and plots. **Section 4** will describe several ongoing applications of the *sami2py* project using the Application Usability Level (AUL) Framework (Halford et al., 2019). This framework was recently developed to help track the progress of a product and ensure that it will be usable by the intended user community. The framework matches the progress to similar frameworks such as the technology readiness levels used by the space hardware community and the readiness levels used by the National Oceanic and Atmospheric Administration (NOAA).

2 The sami2py project

The *sami2py* project will be discussed in terms of the three major components: the core ionospheric solver, the component models, and the *Python* interface.

2.1 SAMI2 core code

The core of the code is the FORTRAN ionospheric dynamics engine. At this stage of development, this is numerically unchanged from the original release of the SAMI2 model, though the handling of some variables has been updated to accommodate compilation using GNU compilers (e.g., gfortran team, 2022). SAMI2 solves for the chemical and dynamical evolution of seven ion species in this environment (H^+, He^+, N^+, O^+, N_2^+, NO^+, and O_2^+). The temperature equation is solved for three ion species (H^+, He^+ and O^+) and for the electrons. Ion inertia is included in the ion momentum equation for motion along the geomagnetic field. This is important in modeling the topside ionosphere where the plasma transitions from collisional to collisionless. SAMI2 uses a nonorthogonal, nonuniform, fixed grid. The grid is designed to optimize the numerical mesh so that the spatial resolution decreases with increasing altitude. The plasma is transported along the magnetic field using a semi-implicit transport algorithm, and transverse to the geomagnetic field using a finite volume method in conjunction with the donor cell method (Huba, 2003). The numerical solutions are well documented in Huba et al. (2000). A brief summary follows.

The SAMI2 model simulates the production, motion, and loss of ions along a two-dimensional slice of Earth's ionosphere, as shown in **Figure 1**. This slice is aligned with magnetic field lines as calculated for an offset tilted dipole field. The continuity, momentum, and temperature equations for ions and electrons are solved. The model is initialized and driven by empirical models, as discussed in **Section 2.2**. A series of scaling factors can be used to alter the magnitude of these empirical values through the namelist file. In general, the model is run for 24 h before modelled values are output to files. This is done to clear transients from the system.

2.2 Component models

The *sami2py* software builds on the modular nature of the SAMI2 model. In the original release, SAMI2 used four key empirical models to prime the ionospheric solutions: NRLMSISe-00 (Picone, 2002) to provide the neutral atmosphere, EUVAC (Richards et al., 1994) to provide the EUV spectrum, HWM-93 to provide neutral winds (Hedin et al., 1993b,a), and the Fejer-Scherliess model of low-latitude $\mathbf{E}\times\mathbf{B}$ drifts (Scherliess and Fejer, 1999). *sami2py* updates these component models, whose acronyms are defined below, to the latest versions and includes the older versions as optional inputs. Additionally, the number of scalable parameters has been expanded. A full list of the available models and scalable parameters is included in **Table 1**.

The Naval Research Laboratory Mass Spectrometer and Incoherent Scatter radar (NRLMSIS) model is a semi-empirical model representing multiple decades of neutral atmospheric measurements, including mass spectrometer, radar, and satellite drag data (Picone, 2002). The version implemented in *sami2py* is a modification of the extended version of the model released in 2000 (NRLMSISe-00). During the solar minimum between cycles 23 and 24, record low densities in the thermosphere were observed through satellite drag measurements Emmert et al. (2010) and direct measurement of neutral pressure density (Haaser et al., 2010). These measurements were outside of the underlying database used to construct the model. Solomon et al. (2010) suggested that anomalously low Extreme Ultraviolet (EUV) radiation during this period resulted in a much cooler thermosphere than expected from the radio flux proxy for solar activity ($F_{10.7}$). Since $F_{10.7}$ rather than EUV is used to drive the thermospheric model, Klenzing et al. (2013) implemented a scalar factor for the exospheric temperature in their empirical study of altered electrodynamics during extreme solar minima. The SAMI2 model already allows users to scale the resultant density profiles independently for each species after NRLMSISe-00 has run. The modification implemented here adds the capability to scale the exospheric temperature directly in NRLMSISe-00 in addition to constantly scaling each species. An example of the effect of this reduced temperature run is shown in **Figure 2**.

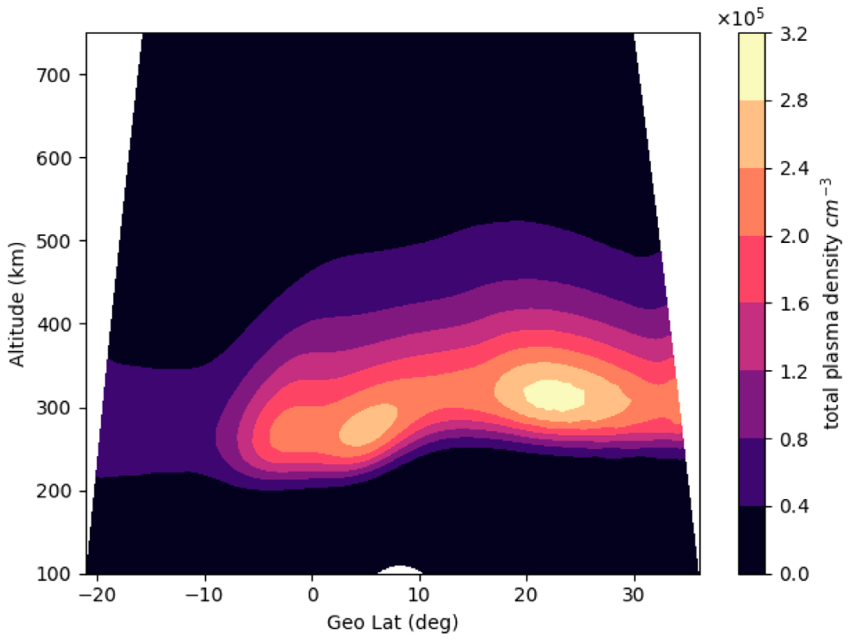


FIGURE 1
Example output of the SAMI2 model.

TABLE 1 Component Models in sami2py 0.3.0.

Physical Mechanism	Model Name	Scalable Parameters
Neutral Atmosphere	NRLMSISe-00	Neutral Species, Exospheric Temperature
Photoionization Rate	EUVAC	Total Ionization
Neutral Winds	HWM-14 (default)	Wind Magnitude
	HWM-07	
	HWM-93	
ExB drifts	Fejer-Scherliess (default)	Drift magnitude, offset from zero
	Fourier coefficients F(SLT)	

The Extreme Ultraviolet for Aeronomic Calculations (EUVAC) model provides a calculation of the EUV flux as a function of the solar radio flux proxy $F_{10.7}$ (Richards et al., 1994). For SAMI2, the model is used to calculate the photo-ionization rate of the ionosphere. While the implementation is unchanged from the SAMI2 1.00 release, a scalar parameter has been added to the code to allow sensitivity studies for directly changing the total photo-ionization rate.

The Horizontal Wind Model (HWM) provides a statistical view of neutral winds gathered from world-wide Fabry-Perot Interferometers, Incoherent Scatter Radars, satellites, and rockets (Drob et al., 2015). The latest version (HWM14) is incorporated as the default, thought users can run numerical experiments with HWM07 (Drob et al., 2008) and HWM93 as options.

The Fejer-Scherliess model of **E**×**B** drift climatology (e.g., Scherliess and Fejer, 1999) provides the two-dimensional drifts perpendicular to the magnetic field lines as a function of local time, solar activity, day of year, and longitude. This is done through cubic spline fits to data from the Jicamarca Incoherent Scatter Radar and the Atmospheric Explorer E satellite. The model is unchanged in the *sami2py* implementation. As in SAMI2, scalar parameters allow users to directly change the magnitude and offset of the drifts.

An alternative **E**×**B** is provided for users wanting to investigate alternate drift climatologies. Since the model is constrained to a local series of flow tubes in a single magnetic meridian, the alternate model is incorporated as a series of Fourier coefficients that are user-specified that describe a function of Solar Local Time (SLT), as shown in **Eq. 1**.

$$E \times B_{total}(SLT) = \Sigma_{i=1}^{10} C_{i0} cos\left(\frac{i\pi SLT}{12}\right) + C_{i1} sin\left(\frac{i\pi SLT}{12}\right) \quad (1)$$

This allows users with direct measurements to create a localized drift model. Examples of this type of usage are presented in Klenzing et al. (2013) and Smith and Klenzing (2022). An

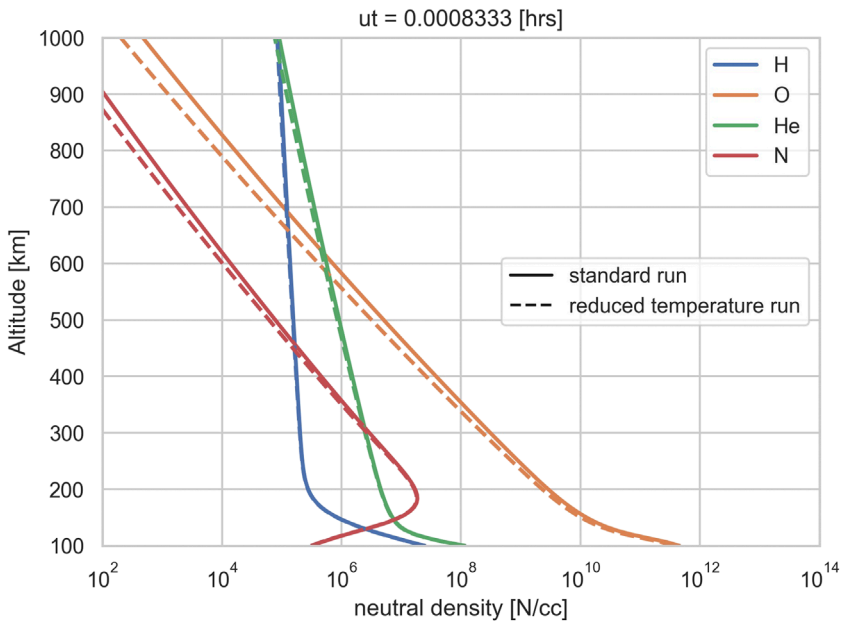


FIGURE 2
Modification of the NRLMSISe exospheric temperature.

additional input file to the FORTRAN code names *exb.inp* was added so that the localized model can be changed without recompiling the FORTRAN engine. Note that this creates a function that averages to zero over all local times, ensuring that there is no net upward or downward drift over the course of a day.

2.3 *Python* interface

The *sami2py Python* code wraps the compiled SAMI2 FORTRAN engine (see **Figure 3**) in a standardized *Python* package. It provides an interface for users to directly update the namelist and **E**×**B** input files *via* keywords, and returns the results in an *xarray.Dataset* object (Hoyer and Hamman, 2017).

The core SAMI2 code in *sami2py* is compatible with FORTRAN 90 and is suitable for compilation under multiple compilers. The variable parameters, such as geographic location, solar activity, and season, are input *via* a namelist file, and the resulting modelled parameters are sent to binary output files. An additional *exb.inp* file is included to generate alternate **E**×**B** drift models *via* a Fourier series over solar local time. The *sami2py* code provides a user interface to both the input namelist files (through the *sami2py.run_model* method) and the output binaries (through the *sami2py.Model* class).

The method *sami2py.run_model* allows the user to directly run the compiled FORTRAN executable. The namelist that specifies the parameters of the model run can be adjusted *via* keyword arguments, which are fully documented in the code docstrings and in the detailed documentation that is available in the GitHub repository and online at readthedocs. This includes a user-specified "tag" to quickly describe the run for archival purposes (e.g., "solarmin"). The FORTRAN executable saves each variable as a separate file. By default, this method will move all of the output files, as well as the input namelist and *exb.inp* files, to an archival directory. All files are grouped under subdirectories by the tag name, longitude, and date in case a user runs multiple dates or locations for the same input conditions.

The *sami2py.Model* class loads the raw output of the model run. It loads each individual file and reshapes them into a single *xarray. Dataset* object for convenience of use. This class will also load the namelist info as metadata to allow inspection of input parameters, as well as any custom **E**×**B** input that was used. When working within *sami2py*, this information is stored in the *model.MetaData* object as a dictionary. The parameters are reshaped as 4D arrays with appropriate coordinates. Examples are shown in the sample code in **Section 3**. Users may run analysis directly from the *Model* object or save to a single file.

For portability and reproducability, both data and metadata can be exported to a netCDF4 file (Whitaker et al., 2020) using the *to_netcdf* method on the model. The metadata will be included as top-level attributes in the output file, documenting

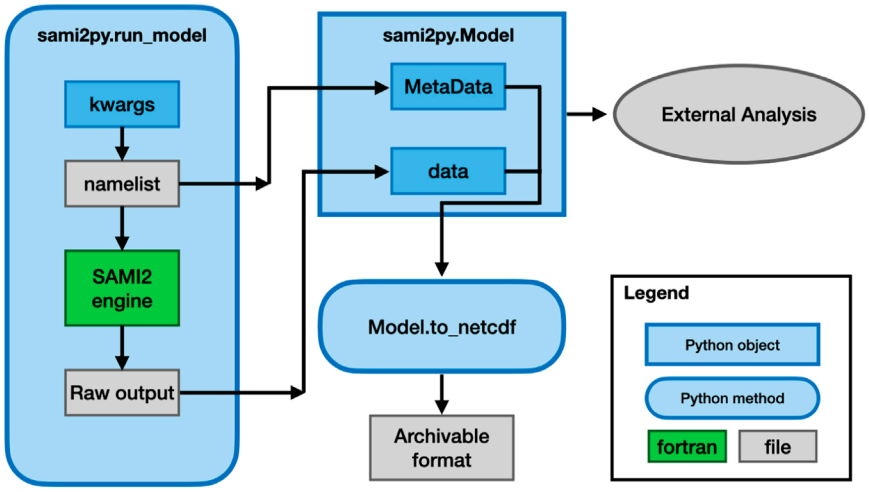


FIGURE 3
Block diagram of the sami2py workflow.

how the run was initialized and including both the sami2py version number and commit hash (in case a custom branch based on an official version was created). The netCDF4 versions of the file are constructed to be compatible with pysat.

2.4 Integration into the pysat ecosystem

The pysat ecosystem (Stoneback et al., 2018) has evolved to support management and analysis of a number of data sets throughout the space science community. The core pysat engine provides a framework to manage data sets, including acquisition, archival, and management. As a management tool, it has been used operationally in missions and analysis projects, including the ICON and COSMIC2 missions. A series of libraries has been written to translate between the core pysat commands and individual data sets. This standardization allows pysat to manage the metadata as well.

These files can be integrated into the pysat ecosystem by using the custom *sami2py* instrument module at pysatModels (Burrell et al., 2022). This package includes a number of other tools to compare observational data with models.

3 Sample workflow

This section demonstrates how *sami2py* can be used in a research workflow to run and analyze the SAMI2 model and output.

TABLE 2 Environments currently tested for sami2py 0.3.0.

Requirement	Versions tested	NEP029 tests
Operating System	Ubuntu 20.04.5 Mac OS 12.6 Windows Server 2022	Ubuntu 20.04.5
Python	3.9, 3.10	3.8
gcc	12.2.0	12.2.0
netCDF4	1.6.1	1.6.1
numpy	1.23.4	1.20.0
pandas	1.5.1	1.4.4
scipy	1.9.3	1.9.3
xarray	2022.10.0	2022.10.0

3.1 Environment and compilation

The code here has been tested in linux, Mac, and Windows environments through Github Actions. Each environment is tested through a unit test suite with 97.6% code coverage as of version 0.3.0. The unit tests are configured to use the latest python packages under python 3.9 and 3.10 environments, as well as a version limited to numpy 1.20 under python 3.8. The specific versions used for the core requirements as of the publication of this paper are listed in Table 2.

3.2 Preparing to run the model

The *sami2py.run_model* method and *sami2py.Model* class provide the core functionality of *sami2py*. The following code snippet prepares the archive directory, and specifies the time

and location for the run as well as declaring custom **E×B** input.

```
import datetime as dt
import os

import sami2py

# Check for archive directory and set if necessary
if not sami2py.archive_dir:
    home_dir = os.path.expanduser("~")
    path = os.path.join([home_dir, "data", "sami2py"])
    sami2py.utils.set_archive_dir(os)

# Set date to winter solstice
date = dt.datetime(2009, 6, 4)
doy = date.timetuple().tm_yday

# Set the longitude
lon = 22

# Set the fourier coefficients obtained from observations
exb_drifts = [[-1.27399486e+01, 4.84811390e+00],
              [ 5.75459367e+00, -1.39196171e+01],
              [ 1.16307457e+01, -1.78058791e+00],
              [ 7.09914415e+00, -3.28817843e+00],
              [-1.09464044e-02, 1.90632011e+00],
              [-9.40307626e-01, -4.54870858e-01],
              [ 1.62144077e-01, -3.54108276e+00],
              [ 2.30221902e+00, 1.05182704e-01],
              [ 4.97016102e-03, 2.47216869e+00],
              [-1.40601689e+00, 0.00000000e+00]]
```

Note that setting the user archive directory only needs to be run when the package is first installed.

3.3 Running the model

Now that the custom input has been declared and the environment is prepared for archival, the model can now be executed. The time, location, F10.7 and **E×B** are provided to the *sami2py.run_model* method. Upon completion the model output is loaded as a *sami2py.Model* object and archived as a netCDF file.

```
# Run basic model
sami2py.run_model(tag="fass_solarmin", lon=lon, year=date.year,
    day=doy, f107=70, f107a=70, fejer=False, exb_drifts=exb_drifts)

# Load and archive models
solarmin = sami2py.Model(tag="fass_solarmin", lon=lon,
    year=date.year, day=doy)
solarmin.to_netcdf("fass_solarmin.nc")
```

3.4 Plotting the model output

The following code snippet loads the archived model run, adds a new variable to the data set which consists of the total plasma density, and then plots the total plasma density as a function of local time and altitude with the **E×B** drift superimposed over the density. Note that by default, the ion density variable (*deni*) is a four-dimensional object, with one of the dimensions (retrievable as *ion*) specifies the individual ion species. A summation over this third axis is needed to extract total ion density.

```
import matplotlib.pyplot as plt
import xarray as xr

# Load archived model using xarray
model = xr.load_dataset("fass_solarmin.nc")

# Sum over all ions for total ion density
model["Ni"] = model["deni"].sum(dim="ion")

# Set time step and density range
step = 30
denmin = model["Ni"][:, :, step].min().values
denmax = model["Ni"][:, :, step].max().values

# Shift model data so that the lowest time value is at the 0th
    position
model = model.roll(ut=1, roll_coords=True)

# Create figure
fig = plt.gcf()

# Plot the plasma density
plt.contourf(model["ut"], model["zalt"].interp(z=51),
             model["Ni"].interp(z=51),
             cmap="magma", vmin=denmin, vmax=denmax)
cbar = plt.colorbar(pad=.15)
cbar.formatter.set_powerlimits((0, 0))
cbar.formatter.set_useMathText(True)
cbar.set_label("total plasma density $cm^{-3}$")
plt.ylabel("Altitude (km)")
plt.xlabel("slt (hours)")

# Plot the model drift on top of the density
host = plt.gca()
new = host.twinx()
new.set_ylabel(r"Meridional E$\times$B Drifts (m s$^{-1}$)")
new.plot(model["ut"], model["exb"], color="w")
new.set_ylim(-140, 75)
title = " ".join("F$_{10.7}$", "-", str(model.F10_7), "sfu"])
plt.title(title)
plt.tight_layout()

plt.show()
```

The resulting figure is shown in **Figure 4**.

4 Application overview

The AUL framework is divided into three phases with three levels each as shown in **Table 3** Halford et al. (2019). Examples of use are in the paper and a full example of the AUL framework applied to the development of a project can be found in Cid et al. (2020). The first phase focuses on basic research, the identification of the user, and agreement between the researcher and users of the intended application and requirements. The second phase develops and tests the application in a similar environment to where it will be operational. In the case of a software development such as *sami2py* this may include common operating systems and *Python* installations. The third phase includes the delivery of the application into the operational environment for routine use. The definitions of these AUL parameters are defined in the context of *sami2py* in **Table 4**.

At this phase in project development, we have identified three core use cases of the software: The use of early-phase research projects to perform key sensitivity studies, as a key dependency in the *growin* software package (Smith and Klenzing, 2020), and as an educational tool for classes to teach ionospheric electrodynamics. We will discuss each of these individually through the framework of the AUL framework summarized in **Table 3** as each as different users and requirements. The AUL framework provides a standardized scale for software and other

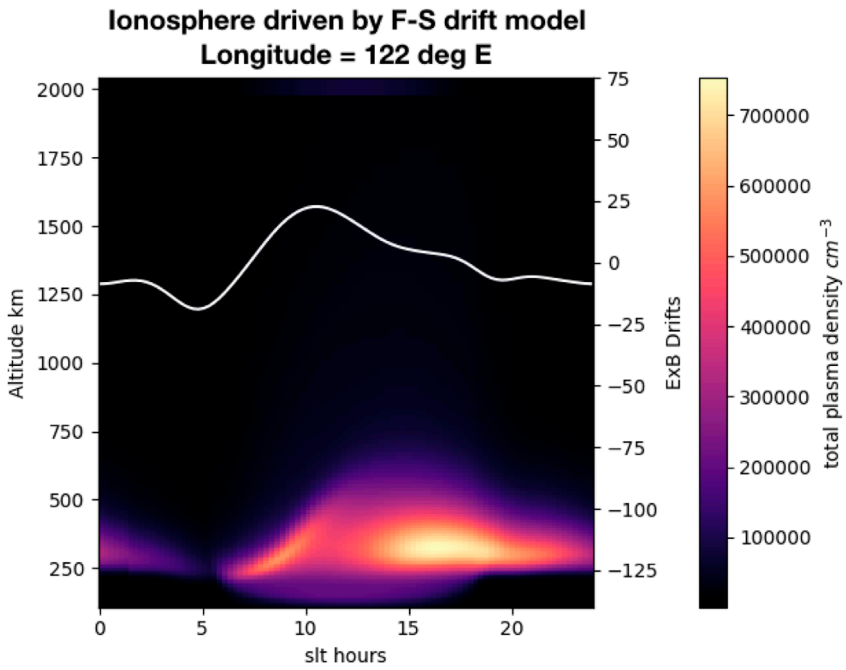


FIGURE 4
Example output ionosphere driven by custom drifts from the Fejer-Scherliess model.

TABLE 3 A brief description of the AUL phases and levels as outlined in Halford et al. (2019).

Phase	Phase definition	AUL	Level description
Phase 1	Discovery and Viability	1	Basic research
		2	Establishment of users and requirements
		3	Assess viability and current state of the art
Phase 2	Development, Testing, and Validation	4	Initial integration and verification
		5	Demonstration in the relevant context
		6	Completed validation
Phase 3	Implementation and Integration into Operation	7	Application prototype
		8	Validation in relevant context
		9	Approved for on-demand use

TABLE 4 AUL definitions for sami2py.

AUL parameter	Definition for *sami2py*
End User	Scientific researcher or Course Instructor
Operational Environment	End User's computer workstation (unix/mac/windows)
Simulated Operational Environment	GitHub Actions Continuous Integration environment

projects on a scale of 1–9, analogous to the Technology Readiness Levels often used for flight hardware projects. The first two applications have been identified as having completed validation (AUL 6), whereas the third application (use as an educational tool) is still at an AUL 1. This section will document the steps we have taken to reach these AUL levels.

4.1 Application: Early phase research test projects–AUL 7

One of the applications of *sami2py* is for early-phase research projects. The user is the broader ionospheric research community who are communicated with on a direct basis with the development team and at conferences such as CEDAR. The operational environment is then considered to be an individual's work computer.

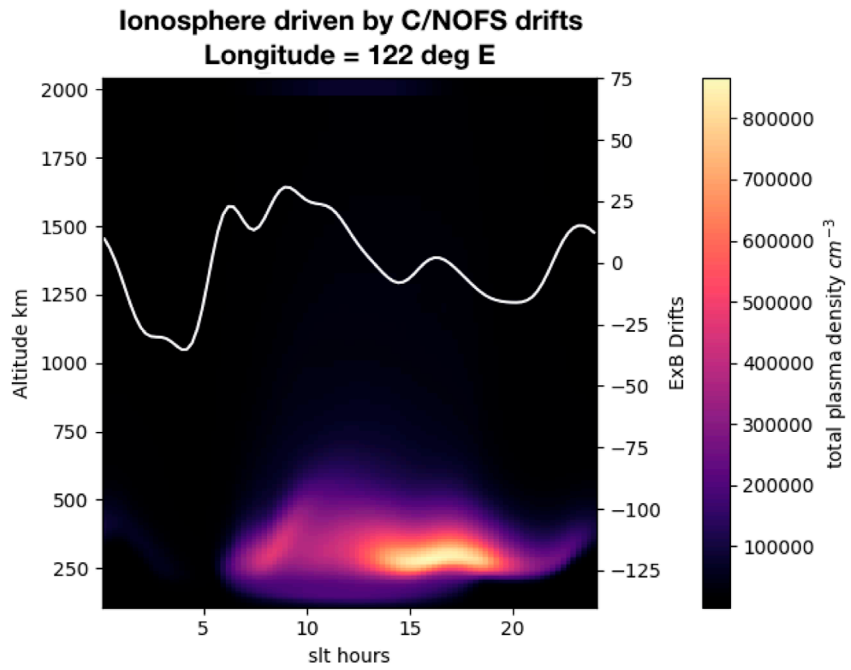


FIGURE 5
Example output ionosphere driven by custom drift climatology fit to C/NOFS data.

TABLE 5 Requirements and Metrics for the sami2py project.

Requirements	
Application 1	Generate a 2-D ionospheric slice in the geomagnetic plane Modify and switch between available empirical models *via Python* keywords Archive model runs for a user to access later, including code commit hash Load and return the resultant modeled ionosphere *via* an xarray object Do so consistently under a variety of possible computer configurations
Application 2	All of the above The code should output neutral density background in addition to the ions
Metrics	Unit tests capturing above requirements Continuous integration support under Linux and windows configurations Continuous integration testing compatible with NEP 029 (Caswell et al., 2019) Unit test coverage >95% Documentation consistent with PyHC Standards (Annex et al., 2018)

An example of the early-phase research projects is running sensitivity studies on proposed physical forcing mechanisms. For this paper, an example of an identified user for this application is Klenzing et al. (2013) where the early phase research includes a series of sensitivity studies for proposed modifications to ionospheric drivers under extremely low levels of solar activity. This study was originally conducted using a prototype of the *sami2py* model written in MatLab, but the functionality applies to the *Python* version as well. Each empirical model that drives the SAMI2 ion dynamics engine can be modified to reflect proposed changes to the forcing of the ionosphere, including reductions in exospheric temperature for the MSIS model and the direct input of user-specified **E**×**B** drift profiles as a function of local time.

Examples of how the ionospheric density changes by altering the **E**×**B** drift assumptions are shown in **Figures 4**, **5**. Each plot shows the evolution of the vertical ionospheric density profile over time. The white line plotted above the ionospheric density represents the driving **E**×**B** timeseries used in *sami2py*, with **Figure 4** driven by the Fejer-Scherliess model (Scherliess and Fejer, 1999) and **Figure 5** driven by climatology measured by the Coupled Ion-Neutral Dynamics Investigation (CINDI) mission of opportunity (Smith and Klenzing, 2022).

The work discussed above has shown how a *Python* version of SAMI2 will provide a path beyond the current state of the art capabilities for individual research projects. The *Python* interface for the SAMI2 model also provides a new capability making it easier for more researchers to access and use this model, as well as document results. Moving from a MatLab interface to an open source language improves the accessibility of the work. Incorporation of the resulting modeled data into an *xarray. Dataset* object improves the usability of the output. The primary requirement for this application at this phase is to ensure that

this *Python* package is open access and works across computer operating systems. We have satisfied the milestones for AUL 3 with the release of *sami2py* version 0.2.0 in December 2019 (Klenzing et al., 2019).

The AUL four to six milestones require improved documentation and testing of the beta prototype of the model. Changes incorporated since version 0.2.0 include docstrings for all functions, improved Continuous Integration (CI) testing, and improved compatibility with external *Python* packages, including numpy, xarray, and pysat. The model undergoes continuous integration tests in the GitHub Actions environment with >97% coverage, fulfilling simulation in an operational environment. The CI tests are run for Linux, mac, and windows systems to satisfy AUL 5 (demonstration in a relevant context). Additionally, tests for older versions of numpy are included to maintain compliance with NEP029 (Caswell et al., 2019). Since *sami2py* is being developed on GitHub, it is easily transferred from the development environment to the operational environment (end user's workstation) across the community. Regular updates are given at community workshops. With the documentation of the code, including the online documentation at *readthedocs* and the examples within this paper, and the release of version 0.2.5 (Klenzing et al., 2021) all milestones through AUL six have been completed.

AUL level 7 is the Application Prototype of the project. This requires demonstration of the prototype and dissemination of results. Both of these goals are achieved with the release of version 0.3.0 (Klenzing et al., 2022) and the publication of this paper. Improvements to the user interface and code style have been implemented in version 0.3.0 to maintain PyHC standards and improve code maintainability.

For AUL 8 and 9, a finalized project for on-demand usage needs to be released. In the context of this application for *sami2py*, a series of updates focusing on an improved workflow and code maintainability have been identified. These are demarcated as a future 0.4.0 release. Input from the community will be evaluated alongside these updates as the user base grows.

4.2 As a core dependency of the growin software tools–AUL 7

As an additional demonstration of the prototype, the *sami2py* module is a central dependency for the *growin* python module which was written to compute the Rayleigh-Taylor instability (RTI) growth rate. The calculation of the RTI growth rate is central to the development and growth of plumes of depleted plasma, or plasma bubbles, in the bottomside of the equatorial ionosphere. The *growin* module uses the *sami2py* module to run the SAMI2 model, archive the output, and load the output into *Python* data structures (Klenzing et al., 2022). Similar to the example code above, drift measurements are used to create a climatological drift profile from *in-situ* measurements. These drifts are then passed to *sami2py* and an ionosphere is simulated with the typical ionospheric indices for the corresponding time period. Subsequently the produced ionospheric plasma densities, drifts, and winds are used to compute flux-tube integrated quantities necessary to compute the RTI growth rate. These growth rates have been previously used to discuss bubble occurrence frequencies obtained from the CINDI (Smith and Klenzing, 2022) and Global Observations of the Limb and Disk (GOLD) (Martinis et al., 2021) missions.

Similar to the previous application, the broader ionospheric research community is the user and will benefit from a *Python* version of *growin* and the inclusion of *sami2py* within it. The feasibility, viability, and expected improvements can all be found within Smith and Klenzing (2022). Thus many of the milestones have been completed for this application through the previously discussed application in **Section 4.1**. As shown in **Table 5**, the key additional requirement here is the output of neutral atmospheric data, which is required to perform the RTI calculations. This has been added to *sami2py* as an optional output. As the other components *growin* were already within the operational/end user environment, the final AUL is now dependent on the progress of *sami2py*. Similar to the previous application, the usage of *sami2py* in the *growin* package is at an AUL of 7.

4.3 Application: Educational tool–AUL 1

Beyond the research community, another user community has been identified but not yet contacted. The code here can also be used as an educational tool as part of a Space Weather of Ionospheric Electrodynamics curriculum. The straightforward and modular nature of the code makes it practical to incorporate into homework or class projects as needed. As this application has been identified, but specific requirements have not been defined and incorporated into the code, this is defined as an AUL 1 project. Work is ongoing, and interested parties should contact the authors to help better refine this project and requirements for these purposes.

5 Summary and further work

This work documents an overview of the *sami2py* code and several potential applications. The proposed applications are documented here and their progress towards on-demand use using the Application Usability Level framework. Ongoing assessment and progress of these AULs will be updated online at the projects page of the GitHub repository.

Full documentation of the code including examples is available at https://sami2py.readthedocs.io.

Data availability statement

The data sets generated for the figures in this study can be found at zenodo: https://doi.org/10.5281/zenodo.7182786.

Author contributions

JK and JS wrote the *Python* interface to SAMI2, as well as modified the FORTRAN code. JH is the original author (with Dr. Glenn Joyce) of the FORTRAN SAMI2 code. AB contributed to overall design and interface of the code, as well as the integration into the pysat ecosystem. JK wrote the first draft of the manuscript. JS and AH wrote sections of the manuscript. All authors contributed to manuscript revision, read, and approved the submitted version.

Funding

JK and AH are supported by the Space Precipitation Impacts project at Goddard Space Flight Center through the Heliophysics Internal Science Funding Model. JS is supported by NASA NNH20ZDA001N-NASA. The research of JH was supported by NSF (AGS-1931415). AB is supported by the Office of Naval Research. This work uses the SAMI2 ionosphere model written and developed at the Naval Research Laboratory. The *sami2py Python* model is freely available to the community at www.github.com/sami2py/sami2py.

Conflict of interest

Author JH was employed by company Syntek Technologies Inc..

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Annex, A., Alterman, B. L., Azari, A., Barnes, W., Bobra, M., Cecconi, B., et al. (2018). *Python in heliophysics community (pyhc) standards (v1.0)*. Available at: https://doi.org/10.5281/zenodo.2529131

Backus, J. W., and Heising, W. P. (1964). Fortran. *IEEE Trans. Electron. Comput.* 13 (4), 382–385. doi:10.1109/pgec.1964.263818

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—An overview of python in heliophysics. *J. Geophys. Res. Space Phys.* 123 (1210384–10), 402. doi:10.1029/2018ja025877

Burrell, A. G., Klenzing, J., and Stoneback, R. (2022). *pysat/pysatModels: v0.1.0 Release (v0.1.0)* Zenodo. Available at: https://doi.org/10.5281/zenodo.6567105.

Caswell, T. A., Mueller, A., Granger, B., Munk, M., Gommers, R., Haberland, M., et al. (2019). *Nep 29–recommend python and numpy version support as a community policy standard*. Available at: https://numpy.org/neps/nep-0029-deprecation_policy.html

Cid, C., Guerrero, A., Saiz, E., Halford, A. J., and Kellerman, A. C. (2020). Developing the ldi and lci geomagnetic indices, an example of application of the auls framework. *Space weather.* 18, e2019SW002171. doi:10.1029/2019sw002171

Drob, D., Emmert, J., Meriwether, J., Makela, J., Doornbos, E., Conde, M., et al. (2015). An update to the horizontal wind model (hwm): The quiet time thermosphere. *Earth Space Sci.* 2 (7), 301–319. doi:10.1002/2014ea000089

Drob, D. P., Emmert, J. T., Crowley, G., Picone, J. M., Shepherd, G. G., Skinner, W., et al. (2008). An empirical model of the earth's horizontal wind fields: Hwm07. *J. Geophys. Res.* 113 (A12), A12304. doi:10.1029/2008ja013668

Emmert, J. T., Lean, J., and Picone, J. M. (2010). Record-low thermospheric density during the 2008 solar minimum. *Geophys. Res. Lett.* 37. doi:10.1029/2010gl043671

England, S. L., Immel, T. J., and Huba, J. D. (2008). Modeling the longitudinal variation in the post-sunset far-ultraviolet oi airglow using the Sami2 model. *J. Geophys. Res.* 113 (A1). doi:10.1029/2007ja012536

gfortran team (2022). *The gnu fortran compiler*. Available at: https://gcc.gnu.org/onlinedocs/gcc-12.2.0/gfortran/

Gil, Y., David, C. H., Demir, I., Essawy, B. T., Fulweiler, R. W., Goodall, J. L., et al. (2016). Toward the geoscience paper of the future: Best practices for documenting and sharing research from data to software to provenance. *Earth Space Sci.* 3 (10), 388–415. doi:10.1002/2015ea000136

Haaser, R. A., Earle, G. D., Heelis, R. A., Coley, W. R., and Klenzing, J. H. (2010). Low-latitude measurements of neutral thermospheric helium dominance near 400 km during extreme solar minimum. *J. Geophys. Res.* 115 (11), 1–5. doi:10.1029/2010ja015325

Halford, A. J., Kellerman, A. C., Garcia-Sage, K., Klenzing, J., Carter, B. A., McGranaghan, R. M., et al. (2019). Application usability levels: A framework for tracking project product progress. *J. Space Weather Space Clim.* 9, A34. doi:10.1051/swsc/2019030

Hedin, A. E., Fleming, E. L., Manson, A. H., Schmidlin, F. J., Avery, S. K., Clark, R. R., et al. (1993a). Empirical wind model for the middle and lower atmosphere – part 2: Local time variations." in *NASA technical memorandum 104592*. Greenbelt, MD, United States: NASA. Available at: https://ntrs.nasa.gov/citations/19940017389

Hedin, A. E., Schmidlin, F. J., Fleming, E. L., Avery, S. K., Manson, A. H., and Franke, S. J. (1993b). *NASA technical memorandum 104581*. Greenbelt, MD, United States: NASA. Empirical wind model for the middle and lower atmosphere – part 1: Local time average. Available at: https://ntrs.nasa.gov/citations/19930015971

Higham, D. J., and Higham, N. J. (2016). *MATLAB guide*, Vol. 150. Philadelphia, PA, United States: Society for Industrial and Applied Mathematics.

Hoyer, S., and Hamman, J. (2017). xarray: N-d labeled arrays and datasets in python. *J. Open Res. Softw.* 5, 10. doi:10.5334/jors.148

Huba, J. (2003). *A tutorial on hall magnetohydrodynamics*. Space Plasma Simulation. Lecture Notes in Physics, Vol. 615. Heidelberg: Springer. https://doi.org/10.1007/3-540-36530-3_9

Huba, J. D., Joyce, G., and Fedder, J. A. (2000). Sami2 is another model of the ionosphere (Sami2): A new low-latitude ionosphere model. *J. Geophys. Res.* 105 (A10), 23035–23053. doi:10.1029/2000ja000035

Klenzing, J., Burrell, A. G., Heelis, R. A., Huba, J. D., Pfaff, R. F., and Simões, F. (2013). Exploring the role of ionospheric drivers during the extreme solar minimum of 2008. *Ann. Geophys.* 31 (12), 2147–2156. doi:10.5194/angeo-31-2147-2013

Klenzing, J., Smith, J., and Hirsch, M. (2019). *sami2py/sami2py: Version 0.2.0 – Support for xarray (v0.2.0)*. Zenodo. https://doi.org/10.5281/zenodo.3581453

Klenzing, J., Smith, J. M., Burrell, A. G., Kitano, R., and Hirsch, M. (2022). *sami2py/sami2py: v0.3.0 (v0.3.0)*. Zenodo. https://doi.org/10.5281/zenodo.7277517

Klenzing, J., Smith, J. M., Kitano, R., Hirsch, M., and Burrell, A. G., (2021). *sami2py/sami2py: Version 0.2.5 (v0.2.5)*. Zenodo. https://doi.org/10.5281/zenodo.5575825

Krall, J., and Huba, J. D. (2019). Simulation of counterstreaming h+ outflows during plasmasphere refilling. *Geophys. Res. Lett.* 46 (6), 3052–3060. doi:10.1029/2019gl082130

Martinis, C., Daniell, R., Eastes, R., Norrell, J., Smith, J., Klenzing, J., et al. (2021). Longitudinal variation of post-sunset plasma depletions from the global-scale observations of the limb and disk (gold) mission. *JGR. Space Phys.* 6 (2), 1–10. doi:10.1029/2020ja028510

Otsuka, Y., Shinbori, A., Sori, T., Tsugawa, T., Nishioka, M., and Huba, J. D. (2021). Plasma depletions lasting into daytime during the recovery phase of a geomagnetic storm in may 2017: Analysis and simulation of gps total electron content observations. *Earth Planet. Phys.* 5 (5), 1–8. doi:10.26464/epp2021046

Picone, J. M., Hedin, A. E., Drob, D. P., and Aikin, A. C. (2002). NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *J. Geophys. Res.* 107 (A12), SIA 15-21–SIA 15-16. SIA 15–16. doi:10.1029/2002ja009430

Richards, P., Fennelly, J. A., and Torr, D. G. (1994). Euvac - a solar euv flux model for aeronomic calculations. *J. Geophys. Res.* 99 (A5), 8981–8992. doi:10.1029/94ja00518

Scherliess, L., and Fejer, B. G. (1999). Radar and satellite global equatorial f region vertical drift model. *J. Geophys. Res.* 104 (A4), 6829–6842. doi:10.1029/1999ja900025

Smith, J., and Klenzing, J. (2020). *JonathonMSmith/growin: Beta (0.1-beta.1)* Zenodo. https://doi.org/10.5281/zenodo.3678866

Smith, J. M., and Klenzing, J. (2022). Growin: Modeling ionospheric instability growth rates. *J. Space Weather Space Clim.* 12, 26. doi:10.1051/swsc/2022021

Solomon, S. C., Woods, T. N., Didkovsky, L. V., Emmert, J. T., and Qian, L. (2010). Anomalously low solar extreme-ultraviolet irradiance and thermospheric density during solar minimum. *Geophys. Res. Lett.* 37 (16), 1–5. doi:10.1029/2010gl044468

Stoneback, R. A., Burrell, A. G., Klenzing, J., and Depew, M. D. (2018). Pysat: Python satellite data analysis toolkit. *JGR. Space Phys.* 123 (6), 5271–5283. doi:10.1029/2018ja025297

Van Rossum, G., and Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

Varney, R. H., Swartz, W. E., Hysell, D. L., and Huba, J. D. (2012). Sami2-pe: A model of the ionosphere including multistream interhemispheric photoelectron transport. *J. Geophys. Res.* 117 (A6), 7280. doi:10.1029/2011ja017280

Whitaker, J., Khrulev, C., Huard, D., Paulik, C., Hoyer, S., FilipePastewka, L., et al. (2020). *Unidata/netcdf4-python: version 1.5.5 release (v1.5.5rel2)* Zenodo. https://doi.org/10.5281/zenodo.4308773

Zhan, W., and Rodrigues, S. (2018). June solstice equatorial spread f in the American sector: A numerical assessment of linear stability aided by incoherent scatter radar measurements. *JGR. Space Phys.* 123 (1), 755–767. doi:10.1002/2017ja024969

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Original Research
PUBLISHED 05 December 2022
DOI 10.3389/fspas.2022.1005977

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Bruce Fritz,
United States Naval Research Laboratory, United States
Scott England,
Virginia Tech, United States

*CORRESPONDENCE
Rebecca Ringuette,
Rebecca.ringuette@gmail.com

SPECIALTY SECTION
This article was submitted to Space Physics,
a section of the journal
Frontiers in Astronomy and Space Sciences

RECEIVED 28 July 2022
ACCEPTED 02 November 2022
PUBLISHED 05 December 2022

CITATION
Ringuette R, De Zeeuw D, Rastaetter L, Pembroke A, Gerland O and Garcia-Sage K (2022), Kamodo's model-agnostic satellite flythrough: Lowering the utilization barrier for heliophysics model outputs.
Front. Astron. Space Sci. 9:1005977.
doi: 10.3389/fspas.2022.1005977

COPYRIGHT
© 2022 Ringuette, De Zeeuw, Rastaetter, Pembroke, Gerland and Garcia-Sage. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Kamodo's model-agnostic satellite flythrough: Lowering the utilization barrier for heliophysics model outputs

Rebecca Ringuette[1,2]*, Darren De Zeeuw[2,3], Lutz Rastaetter[2], Asher Pembroke [4], Oliver Gerland[4] and Katherine Garcia-Sage [2]

[1]ADNET Systems Inc, Bethesda, MD, United States, [2]The Community Coordinated Modeling Center, NASA Goddard Space Flight Center, Greenbelt, MD, United States, [3]Catholic University of America, Washington, D.C., DC, United States, [4]Ensemble Government Services LLC, Hyattsville, MD, United States

Heliophysics model outputs are increasingly accessible, but typically are not usable by the majority of the community unless directly collaborating with the relevant model developers. Prohibitive factors include complex file output formats, cryptic metadata, unspecified and often customized coordinate systems, and non-linear coordinate grids. Some pockets of progress exist, giving interfaces to various simulation outputs, but only for a small set of outputs and typically not with open-source, freely available packages. Additionally, the increasing array of tools built upon these sporadic interfaces are typically model-specific. We present Kamodo's model-agnostic satellite flythrough capabilities as the solution to the utilization barrier for heliophysics model outputs. Developed at the Community Coordinated Modeling Center, these flythrough capabilities are built in Python upon a network of model-agnostic interfaces developed in collaboration with model developers, providing interpolation results the community can trust. Kamodo's flythrough capabilities present the user with a growing variety of flythrough tools based upon a rapidly expanding library of heliophysics model outputs in several domains, currently including a variety of Ionosphere-Thermosphere-Mesosphere and global magnetosphere model outputs. Each capability is designed to be easily accessible *via* simplistic model-agnostic syntax, with the entire package freely available in the cloud on Github. Here, we describe the tools developed, include several sample applications for common science questions, demonstrate interoperability with selected packages, and summarize ongoing developments.

KEYWORDS

data, heliophysics, python (programming language), simulation, model-data comparisons, ensemble modeling, flythrough, data functionalization

1 Introduction

Developing a flythrough capability for *in-situ* observations simulated by a given model is straightforward. All one must do is create a routine to read in the given dataset, a routine to interpolate between points on the given grids, including time, and some sort of interface that combines them. More advanced versions of this also account for conversion to a commonly-used coordinate system, especially if the model output is given on a coordinate system custom to the model. This single shot approach is what is currently used at the Community Coordinated Modeling Center to provide a flythrough functionality for the various models hosted there. As part of the resources offered for various simulation outputs, CCMC's users can select a sample of satellite tracks to fly through the simulation output, which are then offered to the public. These resources have been used to compare observational data to multiple model outputs (e.g. Ridley et al., 2016).

The satellite tracing for each of the various simulations used in the example study referenced were done with code custom to each model. Also, only two options for satellite tracks are available. Users can request those available through SSCWeb[1] or upload their own, which precludes users from implementing custom trajectories or trajectories from other sources and in other formats. As the number of hosted models increases at CCMC, and the number of satellites in orbit drastically increases, the software providing this capability requires drastic changes in order to keep up with the increasing variety of user needs.

The model flythrough services offered by CCMC are further limited by the interface itself. The run-on-request visualization interface does offer static visualization and data downloads for flythrough results, but does not offer model-data or multi-model comparisons[2]. A second lesser known interface, called the Virtual Model Repository Tools, offers model-data comparisons, also as static visualizations, but does not offer the option to download the data used to create the plots[3]. Although these capabilities would be beneficial to offer for all models and satellite trajectories, a pre-defined interface for this capability such as the one provided by CCMC is difficult to maintain. The current library of code also sometimes requires regridding before using the flythrough in IDL, which is not satisfactory for all applications. A more flexible solution is required.

Pysat, the Python satellite data analysis toolkit (Stoneback et al., 2018), offers a partial solution to this problem. The *pysat.models* interface provides a model-agnostic interface for two ITM models[4]. However, there are no coordinate conversions incorporated into the interface. This becomes problematic when comparing the simulated data to observational data in a different coordinate system, and prohibitive when dealing with simulation outputs in custom or model-specific coordinate systems. In its current state, pysat does not have the software structure to accommodate a large variety of simulation outputs.

Kamodo solves this problem by using a 'plug-and-play' interface design as a framework to incorporate any simulation output desired and in any coordinate system. At its roots, Kamodo is a Python open-source software package that provides a powerful array of capabilities for a given functionalized dataset (Pembroke et al., 2022). The capabilities include quick, interactive visualizations, easy function composition, unit conversions, and automatic LaTeX rendering of syntax all with simplistic syntax[5]. Applying these capabilities to simulation outputs provides for the first time a direct functionalized access method for users to interact with these data. This direct access abstracts away the custom and often complex coordinate grids, verified interpolation on these grids, the large range of data file formats, and the translation of cryptic variable names to standard representations. In order to enable this access, a network of model-specific interfaces called 'model readers' was created (Ringuette et al., 2022a). These model readers were designed to provide identical interfaces to a large set of simulation outputs (see Figure 1 for an example).

Eleven types of simulation outputs are currently represented in this network, with numerous additions in process. The simulation outputs in the ITM domain include the CTIPe (Coupled Thermosphere Ionosphere Plasmasphere Electrodynamics model, Codrescu et al., 2008), IRI (International Reference Ionosphere model, Bilitza 2018), GITM (Global Ionosphere Thermosphere Model, Ridley et al., 2006), SWMF (Space Weather Modeling Framework, Toth et al., 2007, ionosphere electrodynamics portion only), TIE-GCM (Thermosphere Ionosphere Electrodynamics General Circulation Model, Qian et al., 2013), SuperDARN (Super Dual Auroral Radar Network, both the default and equal-area grids, Thomas and Shepherd, 2018), WACCM-X (Whole Atmosphere Community Climate Model with thermosphere and ionosphere extension, Liu et al., 2018), DTM (Drag Temperature Model, Bruinsma, 2015), WAM-IPE (the coupled Whole Atmosphere Model and Ionosphere Plasmasphere Electrodynamics model, Maruyama et al., 2016; Fang et al., 2022), and AMGeO (Assimilative Mapping of

1 https://sscweb.gsfc.nasa.gov/.

2 See https://ccmc.gsfc.nasa.gov/results/index.php, search for existing simulation runs, and use one of the timeseries visualization links (e.g., "Timeseries in Magnetosphere") in each run's page, see https://ccmc.gsfc.nasa.gov/results/viewrun.php?domain=GM&runnumber=Junying_Yang_042021_1 for a specific example and click on one of the links at the bottom of the page/

3 See https://ccmc.gsfc.nasa.gov/ungrouped/GM_IM/GM_analysis.php, and https://ccmc.gsfc.nasa.gov/ungrouped/GM_IM/GM_analysis.php?Pid=21127&Pt=BO&Ps=Cluster-1 for a specific example.

4 https://pysat.readthedocs.io/en/latest/instruments/pysatModels.html.

5 https://ensemblegovservices.github.io/kamodo-core/.

```
# Retrieve model reader and access model data for the GITM model.
reader = MW.Model_Reader('GITM')
kamodo_object_gitm = reader(gitm_file_dir)
kamodo_object_gitm
```

$$\mathrm{TEC}\left(\vec{r}_{GDZsph3D}\right)\left[\frac{10000000000000000}{m^2}\right] = \lambda\left(\vec{r}_{GDZsph3D}\right)$$

$$\mathrm{TEC}_{ijk}(time[hr], lon[deg], lat[deg])\left[\frac{10000000000000000}{m^2}\right] = \lambda(time, lon, lat)$$

$$\rho_H\left(\vec{r}_{GDZsph4D}\right)\left[\frac{kg}{m^3}\right] = \lambda\left(\vec{r}_{GDZsph4D}\right)$$

$$\rho_{Hijk}(time[hr], lon[deg], lat[deg], height[km])\left[\frac{kg}{m^3}\right] = \lambda(time, lon, lat, height)$$

$$\rho_{H+}\left(\vec{r}_{GDZsph4D}\right)\left[\frac{kg}{m^3}\right] = \lambda\left(\vec{r}_{GDZsph4D}\right)$$

$$\rho_{H+ijk}(time[hr], lon[deg], lat[deg], height[km])\left[\frac{kg}{m^3}\right] = \lambda(time, lon, lat, height)$$

$$\rho_{He}\left(\vec{r}_{GDZsph4D}\right)\left[\frac{kg}{m^3}\right] = \lambda\left(\vec{r}_{GDZsph4D}\right)$$

$$\rho_{Heijk}(time[hr], lon[deg], lat[deg], height[km])\left[\frac{kg}{m^3}\right] = \lambda(time, lon, lat, height)$$

$$\rho_{He+}\left(\vec{r}_{GDZsph4D}\right)\left[\frac{kg}{m^3}\right] = \lambda\left(\vec{r}_{GDZsph4D}\right)$$

FIGURE 1
An example of a model reader interface in Kamodo. After an import statement, the user retrieves the model reader for the desired model with the first command, and then functionalizes the data with the second command of the same block. The only user-supplied information needed is which model is desired and the location of the data. The output shows the functionalized data for the variables found in the files indicated. Once the data is functionalized, all of the capabilities of Kamodo are available through simple commands. See Pembroke et al., 2022 and Ringuette et al., 2022a for more information.

Geospace Observations, AMGeO Collaboration 2019) models. The eleventh simulation output is from the OpenGGCM model (Open Geospace General Circulation Model, Raeder et al., 2001, global magnetosphere outputs only). Several simulation outputs are in the process of being added: ADELPHI (AMPERE-Derive ELectrodynamic Properties of the High-latitude Ionosphere model, Robinson et al., 2021), SWMF (global magnetosphere portion), CIMI (Comprehensive Inner-Magnetosphere Ionosphere model, Fok et al., 2014), MARBLE (Bard and Dorelli, 2021, development *via* collaboration), and GAMERA (Grid Agnostic MHD for Extended Research Applications, Zhang et al., 2019, development *via* collaboration). Many more simulation outputs are planned to be added in the coming months.

We have built the first model-agnostic flythrough capability in Kamodo based on this growing network of model readers. We further add compatibility with numerous satellite trajectories by linking to repositories providing them (e.g. SSCWeb), and include coordinate conversions for a wide array of coordinate systems. The code used to provide the flythrough capability is open source and available at the official NASA Kamodo repository[6]. The CCMC collaborates with Ensemble Consultancy to maintain the core capabilities of Kamodo, which the described capabilities depend on (also available on GitHub[7]).

The remainder of this paper describes the features included in the flythrough capability, then demonstrates a few science cases it can be used for. Section 2 provides a description of the flythrough capability, including the various functions and features. This is followed by a few sample science applications in section 3, such as a model-data comparison and an ensemble modeling example. Also included in the section is an example workflow demonstrating interoperability of Kamodo with pysat, the last core PyHC (Python in Heliophysics Community[8]) Python package lacking such an example. Some concluding remarks and an outline of our future plans are given in section 4. Throughout the text, this paper will use *italics* when referring to a function or variable.

2 Kamodo's flythrough description

Kamodo's *in-situ* flythrough capability is built upon a network of model-agnostic model interfaces. These scripts provide direct access to the chosen simulation outputs with simplistic syntax that remains the same regardless of the simulation data chosen (see Figure 1 for an example). In addition to the features already described, the model readers perform linear interpolation in all time+spatial dimensions

6 https://github.com/nasa/Kamodo.

7 https://github.com/EnsembleGovServices/kamodo-core.

8 https://heliopython.org/.

between data files and align each simulation output with a coordinate system defined by either SpacePy or AstroPy (SpacePy: Morley et al., 2011, AstroPy: AstroPy Collaboration et al., 2013 and AstroPy Collaboration et al., 2018). These uniform features allowed for a straightforward software design in the flythrough scripts.

The base function that is used for all variations of Kamodo's flythrough capability is *ModelFlythrough*. This function takes the trajectory as four one-dimensional arrays of time and spatial positions and 'flies' the trajectory in the given coordinate system through the chosen model data in a sequence of five steps. The required inputs to the function include the trajectory arrays, coordinate system information, the chosen model and variable names, and the directory containing the model data. After some initial checks, the *ModelFlythrough* function (1) retrieves the model reader for the chosen model without executing the model reader script. The function then (2) compares the time values in the trajectory with the time ranges associated with the set of files in the given directory, and discards all times and their associated positions not covered by the chosen model data. The given positions are then (3) converted to the coordinate system for the given model. Next, (4) the retrieved model reader uses Kamodo to functionalize the chosen variable data for each of the model output files associated with a trajectory position, and the value of each requested variable is calculated *via* interpolation at the given trajectory positions. The calculated values are then (5) returned in a Python dictionary with the trajectory times and positions in the original coordinate system. Users also have the option of saving the trajectory and the corresponding variable calculations in one of three output file formats, which are all compatible with various complementary routines included in the flythrough software, such as those used in the examples in section 3. Additional output format options are planned for enhanced interoperability with other CCMC services, including CAMEL (Rastaetter et al., 2019).

The logic contained in the *ModelFlythrough* function is model-agnostic both in syntax and in software design as a result of the model-agnostic syntax of the model reader library. However, the various model outputs are based on quite different coordinate systems, some specific to the model, which presented a challenge. Additionally, users desired more flexibility on the trajectory input options and in the automatically-produced publication-quality visualization capabilities, so we built a small library of functions to accommodate those requests. Our current solutions to these challenges are described in the next three subsections.

2.1 Coordinate systems

Satellite trajectories are often not available in the same coordinate system as the simulation outputs. Similarly, simulation outputs are often in varying coordinate systems, making comparisons across different models difficult. For example, the AMGeO empirical model produces output in the solar magnetic coordinate system at a constant altitude, while the TIE-GCM physics-based model output instead uses geodetic coordinates with pressure level as the vertical coordinate. Directly comparing data from the two models without converting the coordinates is difficult at best due to the different rotating frames and different vertical scales of the two coordinate systems. (Note that the corresponding altitude for a given pressure level changes with location and time.) Kamodo's flythrough simplifies this complex problem by incorporating the coordinate systems defined in AstroPy and SpacePy and the coordinate conversion functions in those packages. Model-specific coordinate conversions are handled individually within a standard framework.

Standard coordinate conversions are implemented by calling the *ConvertCoord* function automatically during step 3 from the *utils.py* script in the *kamodo_ccmc/flythrough* directory on GitHub. This function is a simple wrapper for the coordinate conversion capabilities in the AstroPy and SpacePy packages, and is written both for automatic execution by the flythrough functions and for direct user interaction. The input variables include the one dimensional arrays for the time and spatial coordinate values, and a few strings for the user to indicate the input and output coordinate systems. *ConvertCoord* determines which software package the input and output coordinate systems belong to, creates the appropriate coordinate object using the input data, performs the coordinate conversion, and returns the new coordinate values in one dimensional arrays (see the *Trajectory_Coords_Plots* notebook for more details[9] and Figure 2 below for a concept map). However, the function can only be used when both the input and output coordinate systems are defined in one of the two packages mentioned.

Some simulation outputs are given in a coordinate system specific to the model and not defined in either the AstroPy or SpacePy software packages. In those cases, we rely on custom coordinate conversion functions in the model readers to link between a coordinate system defined in either package and the coordinate system defined in the model. Current examples of this work can be found in the CTIPe and TIE-GCM model readers for conversions between pressure level and altitude. This approach uses Kamodo's function composition capabilities, and will soon be applied to models with more complex coordinate systems, such as those based on pitch angle and energy (Ringuette et al., 2022a). By linking the model-specific coordinate system to a coordinate system defined in one of the linked packages, the flythrough functionality will then be able to convert to any desired coordinate system. Collaborations with model developers on this challenge have begun for a few models.

9 https://github.com/nasa/Kamodo/tree/master/docs/notebooks.

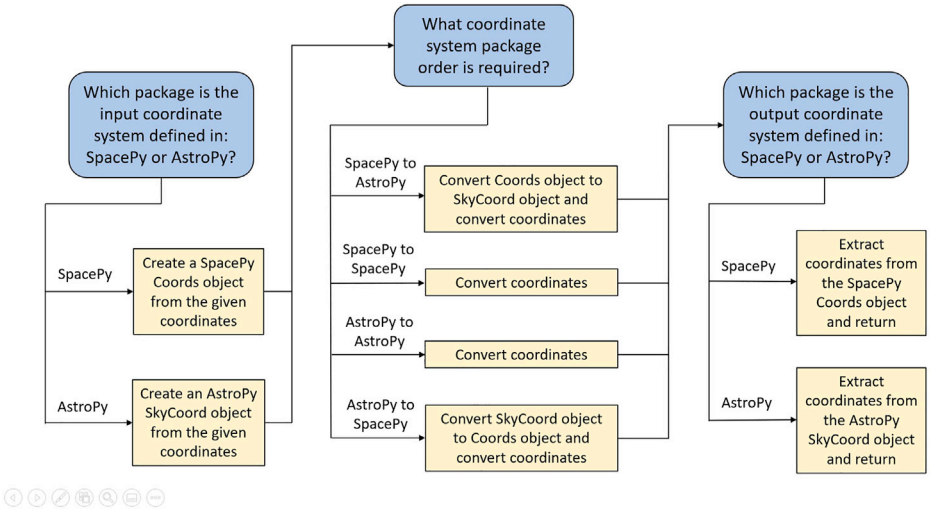


FIGURE 2
Concept map for the *ConvertCoords* function. Blue boxes indicate conditional arguments, yellow boxes show tasks executed depending upon the evaluation result of the preceding conditional argument. Error catching logic is not included in the figure for simplicity. The inputs to the argument include one dimensional arrays of the time and spatial coordinates (4 1D arrays), and four strings indicating the input and output coordinate systems and whether each is Cartesian or spherical. The values returned are one dimensional arrays of the spatial coordinates (3 1D arrays) and a list of strings containing the units of the output coordinates. Model-specific coordinates are handled external to this function.

2.2 Input trajectories

During the initial development, we identified five possible types of desired input methods for satellite trajectories: one from satellite ephemerides obtained from SSCWeb, one from a file of two-line elements (TLEs), one from a file of previously obtained satellite positions, one from a variable defined in the user's Python session memory, and one from a sample trajectory generator. To avoid duplicated code across the five possibilities, we chose the session memory example as the simplest example and designed the base functionality accordingly (the *ModelFlythrough* function described above). The remaining flythrough functions call the various specific trajectory functions and then call the *ModelFlythrough* function to perform the flythrough.

The five flythrough functions are the *RealFlight, TLEFlight, FakeFlight, MyFlight,* and *ModelFlythrough* functions (see Table 1). As mentioned above, the *ModelFlythrough* function takes the given trajectory, provided as four one-dimensional arrays, and flies it through the chosen simulation output. The *RealFlight* function calls the *SatelliteTrajectory* function to retrieve a real satellite trajectory in the form of four one-dimensional arrays from the SSCWeb[10] through an existing HAPI interface for this resource (Weigel et al., 2021). The function then calls the *ModelFlythrough* function to fly that trajectory through the data. An alternative option is provided with the *TLEFlight* function, which calls the *TLETrajectory* function to convert TLEs into a trajectory of the same structure as above using the SGP4 propagator (Simplified General Perturbations[11]: Vallado and Crawford 2008). The four one-dimensional arrays containing the trajectory information are then fed to the *ModelFlythrough* function as before. Similarly, the *MyFlight* function allows the user to provide the trajectory through a simply structured file either in comma-separated, tab-separated, or netCDF4 format options, and then calls *ModelFlythrough*. This function was specifically designed to easily fly a previously saved trajectory through a different set of model data to simplify comparisons across multiple models and model outputs. This capability has proved especially useful for users desiring to use a real satellite trajectory previously used but finding themselves without internet access.

Finally, the *FakeFlight* function calls the *SampleTrajectory* function, which constructs a sample satellite trajectory based on a variety of input parameters, and then flies that trajectory through the simulation output using the *ModelFlythrough* function. The simplest call to the *SampleTrajectory* function only requires start and end times in UTC to create a synthetic trajectory in geodetic spherical coordinates similar in nature

10 https://sscweb.gsfc.nasa.gov/.

11 https://pypi.org/project/sgp4/,https://help.agi.com/stk/index.htm#stk/vehSat_orbitProp_msgp4.htm.

TABLE 1 Flythrough Functions. The function names are followed by the minimum syntax required for each of the four flythrough functions. The source of each of the trajectories is indicated in the last column. *SatelliteTrajectory, TLETrajectory*, and *SampleTrajectory* are functions available through the flythrough software. See text for a basic description of each, and the example notebooks and documentation for more information[8].

Function name	Minimum syntax required	Trajectory source
ModelFlythrough	*ModelFlythrough* (model, file_dir, variable_list, sat_time, c1, c2, c3, coord_sys)	Session memory
RealFlight	*RealFlight* (dataset, start, stop, model, file_dir, variable_list)	SSCWeb *via SatelliteTrajectory*
TLEFlight	*TLEFlight* (tle_file, start, stop, time_cadence, model, file_dir, variable_list)	Text file containing TLEs *via TLETrajectory*
FakeFlight	*FakeFlight* (start_time, stop_time, model, file_dir, variable_list)	*SampleTrajectory*
MyFlight	*MyFlight* (traj_file, model, file_dir, variable_list)	File previously produced by a flythrough function

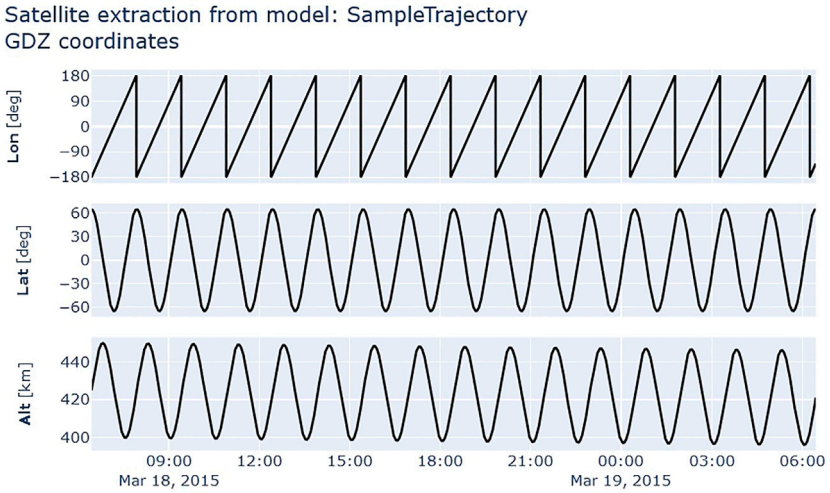


FIGURE 3
The default trajectory produced by the *SampleTrajectory* function. Together, the plots show the position of the imaginary satellite in geodetic spherical coordinates (longitude, latitude, and altitude) for a 24-h period. Note the longitude begins at -180 but ends slightly above that value due to the precession in longitude, which can be chosen by the user. The default latitude range is centered on the equator as shown, but any continuous range can be chosen. By default, the altitude varies in each orbit and degrades slowly over time. The rate of degradation can also be adjusted *via* an input parameter. See documentation for more details[9].

to a low earth orbit (see Figure 3). Various default values can be adjusted in the function call to change the longitudinal precession rate per orbit, the maximum and minimum latitudes, the maximum and minimum initial heights, the decay rate of the height, and the time cadence of the returned positions. The trajectory produced by the *SampleTrajectory* function is not meant to fully simulate any actual satellite orbit, but to simply give users a reasonable starting point if no actual trajectory is decided upon yet. The three trajectory functions, *SatelliteTrajectory*, *TLETrajectory*, and *SampleTrajectory*, may also be called independent of the other functions for additional uses, such as retrieving and modifying a real satellite trajectory before flying it through a set of model data, or visualizing a trajectory created from TLEs. Because these five flythrough functions are based on a network of readers with model-agnostic syntax, the flythrough function syntaxes are also model-agnostic.

An example of Kamodo's *RealFlight* function is presented in Figure 4. The first block shows the two required import statements followed by the definition of various input values. The input values related to the satellite should be determined using the SSCWeb website[9]. The parameters related to the simulation output data, such as the time values in UTC timestamps (*start_utcts* and *end_utcts*, the number of seconds since 1 January 1970 at midnight UTC) and the variable names, can be obtained using the various functions demonstrated in the *SF_IntroFunctions* notebook[8]. The second block shows the syntax of the *RealFlight* function, which is notably identical regardless of the model chosen. The final block shows a simple method to functionalize the object returned by the

A

```
# Import satellite flythrough code
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
import kamodo_ccmc.flythrough.model_wrapper as MW
import datetime as dt

# Choose input values for RealFlight function call.
model = 'TIEGCM'  # Choose the model.
# Full file path to model output data.
file_dir = 'C:/Users/rringuet/Kamodo_Data/TIEGCM/Data/'
# Example files from https://ccmc.gsfc.nasa.gov/RoR_WWW/output_files/KAMODO_DEMO/
dataset = 'gracel'
start_utcts = dt.datetime(2017, 9, 21, 20, 20).replace(tzinfo=dt.timezone.utc).timestamp()
end_utcts = dt.datetime(2017, 9, 22, 20, 0).replace(tzinfo=dt.timezone.utc).timestamp()
# Use https://sscweb.gsfc.nasa.gov/ to find the satellite name and time range desired.
variable_list = ['rho','u_n','T_e']  # List of desired variable names from model.
coord_type = 'GEO'  # Desired coordinate system for retrieved trajectory.
output_name = 'C:/Users/rringuet/Kamodo_env/RealFlightExample_TIEGCM.txt'
plot_coord = 'GSE'  # Coordinate system chosen for output plots (from SpacePy or AstroPy).
```

B

```
# Run RealFlight flythrough command.
results = SF.RealFlight(dataset, start_utcts, end_utcts, model, file_dir, variable_list,
                        coord_type=coord_type, output_name=output_name, plot_coord=plot_coord)
# Open plots in separate internet browser window for interactivity. Nothing will open here.
```

```
Output saved in C:/Users/rringuet/Kamodo_env/RealFlightExample_TIEGCM.txt.
Generating interactive plots...
-saving html div file:  C:/Users/rringuet/Kamodo_env/RealFlightExample_TIEGCM_T_e_3D
-saving html div file:  C:/Users/rringuet/Kamodo_env/RealFlightExample_TIEGCM_T_e_1D
-saving html div file:  C:/Users/rringuet/Kamodo_env/RealFlightExample_TIEGCM_rho_3D
-saving html div file:  C:/Users/rringuet/Kamodo_env/RealFlightExample_TIEGCM_rho_1D
```

C

```
kamodo_object = WO.Functionalize_SFResults(model, results)
kamodo_object
```

$$T_e\,(time)[K] = \lambda(time)$$

$$\rho(time)[\frac{g}{cm^3}] = \lambda(time)$$

FIGURE 4
Example notebook demonstrating the model-agnostic syntax of the *RealFlight* flythrough function. The first block **(A)** shows the two required import statements and chosen variable values. The datetime package is imported to deal with the generation of the UTC timestamps. The second block **(B)** presents the model-agnostic syntax of the function call, and the last block **(C)** shows a simple way to functionalize the Python object returned by the flythrough function.

function call. Note that the TIE-GCM model outputs use pressure level as the vertical coordinate, but no additional commands or parameters are needed as the conversions between pressure level and altitude (or radius) are handled internally and automatically.

Regardless of the flythrough function chosen, the object returned by each function call is identical in structure. Consequently, the *Functionalize_SFResults* function can be used to functionalize any of these objects. Once the returned object is functionalized, all of the previously described core capabilities of Kamodo are available to the user, including simplistic access to interactive plotting (see the bottom section of Figure 7 below).

2.3 Custom visualizations

If the user defines the name of the output file in any of the flythrough function calls, then the resulting data is written to the named file and two custom visualizations are automatically generated for each variable (see Figures 5, 6 below). Both visualizations are designed using Plotly (Plotly Technologies Inc, 2015), and are produced by calling the *SatPlot4D* function with different parameters (see the *SF_Traj_Coords_Plots* notebook for details[8]).

Figure 5 shows the full interactive spatial representation of the entire chosen trajectory in GSE coordinates. The colors indicate the total electron content (TEC) at the given location and time, as indicated by the colorbar at right. Users can pan, zoom, and hover over the image to inspect more closely. All interactivity is saved to an html file. Figure 6 shows the same information plotted as four time series. Zooming into one of the four plot sections results in the same zooming for the other three plots. Similar versions of these plots are saved on the NASA Kamodo repository[8]. Several options are available, especially for plots similar to Figure 5, and are explained in the referenced notebook. For additional visualization capabilities, see Ringuette et al., 2022a.

3 Example science workflows

Kamodo's model-agnostic flythrough capability reduces data-model, model-model comparisons, and even ensemble modeling analysis to a few lines of code, regardless of the

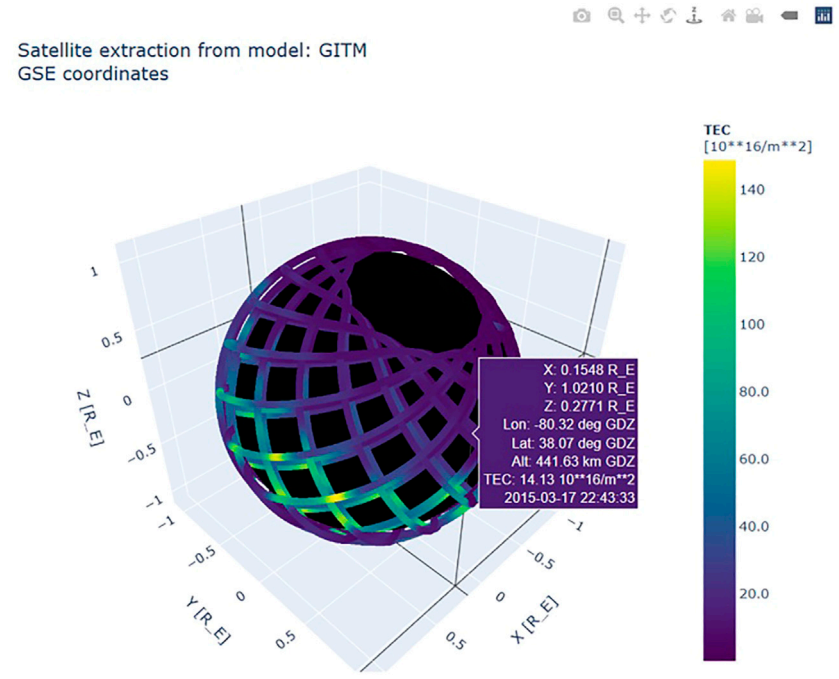


FIGURE 5
An example of a fully interactive 3D plot. In this case, the total electron content (TEC) is plotted for the entirety of a given satellite trajectory. TEC values are indicated by the colorbar at right, and the model source and coordinate system are printed in the top left corner. The purple box contains the coordinate information in GSE coordinates, both in Cartesian and spherical, the TEC value, also indicated by the color of the box, and the date and time in UTC for the data point hovered over by the mouse. The symbols on the top right are the typical buttons provided by Plotly. The plot is saved as a fully interactive html file. An example is included on the NASA Kamodo repository[5].

model or trajectory desired. Examples of these applications are given in the following subsections, followed by an example of interoperability of Kamodo with pysat. Links to examples of interoperability of Kamodo with the remaining PyHC core packages are also given in this section. In addition to being used in the example workflows, the flythrough capability is being used from the command line to incorporate physics-based models into GEODYN II, an orbit propagator written in FORTRAN (Luthcke et al., 2006; Garcia-Sage et al., 2021).

3.1 Model-data comparison workflow

Simplistic data-model comparisons using *in-situ* data are now possible with a few lines of code for an expanding range of models and a wide range of trajectories. Combining the *RealFlight* function demonstrated in Figure 4 with the CDAWeb[12] interface in Kamodo results in the simple workflow displayed in Figure 7. The top block in the figure uses the syntax taken from Figure 4 to fly the chosen satellite through the model data produced by the TIE-GCM model. The time range of the model data was chosen to cover the time range of the desired observational data. We note that any of the flythrough functions can be interchanged in place of the *RealFlight* function shown in the example without changing the syntax of the remaining blocks due to the identical data structures returned.

In this case, the observational dataset is the ion temperature observed by the Coupled Ion-Neutral Dynamics Investigations (CINDI) mission (Coley et al., 2010). Retrieval of the data is demonstrated in the second block. The parameter values shown in that block can be determined through the CDAWeb website. Once the data is retrieved, it can be easily functionalized using the *Functionalize_TimeSeries* function as shown in the same block. Note that this function can only be used to functionalize one dimensional time series data. Other methods are available to functionalize higher dimensionality data (see Kamodo documentation[4]). Once both the simulated and observed data are functionalized, all of the core capabilities available through Kamodo are easily accessible, including the fully interactive plotting shown in the bottom block. We note the two datasets are plotted using their

12 https://cdaweb.gsfc.nasa.gov/index.html.

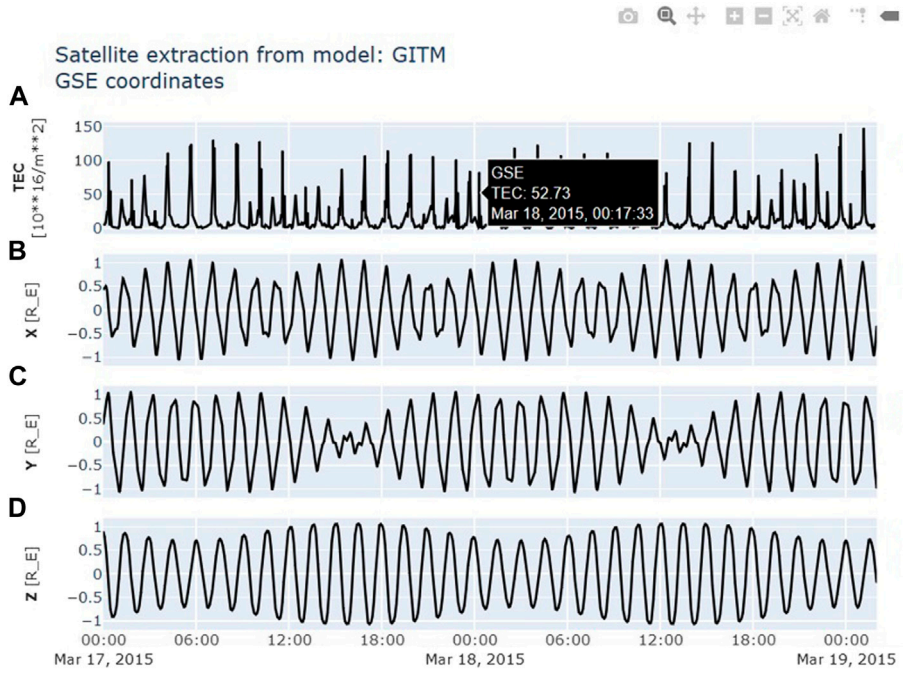


FIGURE 6
An example of a fully interactive 1D plot. As in Figure 5, the total electron content (TEC) is plotted for the entirety of a given satellite trajectory at top **(A)**, followed by the trajectory parameters in Cartesian coordinates in the remaining three plots **(B–D)**. The model source and coordinate system are printed in the top left corner. Hovering over a value in any of the plots gives a box containing the value and the date and time in UTC. The symbols on the top right are the typical buttons provided by Plotly. The plot is saved as a fully interactive html file. An example is included on the NASA Kamodo repository[5].

original time resolutions - every minute for the TIE-GCM simulation output and every second for the CINDI data. The 'DIFF' dataset generated by function composition in the second block is calculated and plotted using the finer time resolution of the CINDI dataset. Additional observational and simulated datasets can be added and functionalized using the same syntax to extend the comparison.

3.2 Ensemble modeling example workflow

Expanding the model-data comparison to include multiple models leads to another science use case called ensemble modeling. (We use the term 'ensemble modeling' in the same sense as in hurricane track prediction, which means to predict a given variable based on the distribution of predictions given by various models.) In this use case, the desired flythrough is called once for each simulation output per desired trajectory. Once the outputs are functionalized, function composition can be used to combine the results in a weighted average, even if the weights depend on another parameter.

Figure 8 below demonstrates a simple example of such a workflow. Two simulation outputs from different models were prepared for the same time range. For simplicity, the example uses the same trajectory function as in Figure 4 to fly the C/NOFS trajectory through both simulation outputs (blocks 1 and 2). The next block shows how to pull the results from each *RealFlight* function call into the same Kamodo object. This block also shows how to use the function composition capability in Kamodo-core to combine the results from each into a custom average calculation. Plotting any or all of the functions is similarly simplistic as in Figure 7, but is not shown for brevity. This workflow can be easily expanded to include additional simulation outputs, more complex functions, unit conversions, and other useful features.

3.3 Interoperable workflows

Kamodo is now one of the core packages in the PyHC group and is also interoperable with all of the other core packages. An example workflow of analyzing MMS data (Polson et al., 2022[13]) demonstrates the interoperability

13 https://deepnote.com/workspace/shawn-polson-c095a0fb-f02d-416d-9c94-c4a9c4e8e54d/project/PyHC-Paper-101b9646-3fd0-4978-a48e-a4f3e708a0ac/%2FMaking_an_Executable_Paper_with_the_Python_in_Heliophysics_Community_to_Foster_ Open_Science_and_Improve_Reproducibility_ipynb.

A

```
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
import datetime as dt

# Set input values for RealFlight function call.
model = 'TIEGCM'
file_dir = 'C:/Users/rringuet/Kamodo_Data/TIEGCM/Uriel_Ramirez_012517_IT_1/'
dataset = 'cnofs'
start_utcts = dt.datetime(2015, 3, 18, 0).replace(tzinfo=dt.timezone.utc).timestamp()
end_utcts = dt.datetime(2015, 3, 21, 0).replace(tzinfo=dt.timezone.utc).timestamp()-1
variable_list = ['T_i'] # list of desired variable names
coord_type = 'GEO'  # GEO cartesian coordinates for the trajectory.
results = SF.RealFlight(dataset, start_utcts, end_utcts, model, file_dir,
                        variable_list, coord_type)

# Functionalize the results
kamodo_object = SF.WO.Functionalize_SFResults(model,results)
kamodo_object
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$

B

```
# Get CINDI data through HAPI
from kamodo_ccmc.readers.hapi import HAPI

server = 'https://cdaweb.gsfc.nasa.gov/hapi'
dataset = 'CNOFS_CINDI_IVM_500MS'
parameters = 'ionTemperature'
start      = dt.datetime.utcfromtimestamp(start_utcts).isoformat('T')
stop       = dt.datetime.utcfromtimestamp(end_utcts).isoformat('T')
hapiCDA = HAPI(server, dataset, parameters, start, stop)

# Add the new time series into the Kamodo object
kamodo_object = SF.WO.Functionalize_TimeSeries(hapiCDA.tsarray, 'T_iCINDI',
                                               'K', hapiCDA.ionTemperature(),
                                               kamodo_object=kamodo_object)
kamodo_object['DIFF'] = 'abs(T_i - T_iCINDI)'  # Analyze.
kamodo_object
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{T_{iCINDI}}\,(time)[K] = \lambda(time)$$
$$\mathrm{DIFF}\,(time) = \mathrm{abs}\,(\mathrm{T_i}\,(time) - \mathrm{T_{iCINDI}}\,(time))$$

C

```
# Now we can plot model and data on the same figure with the difference
kamodo_object.plot('T_i','T_iCINDI','DIFF')
```

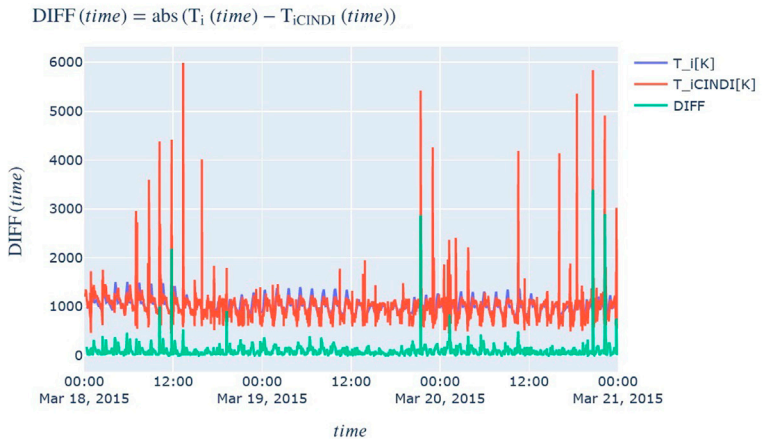


FIGURE 7
Example of model-data comparison. The first block **(A)** uses Kamodo's *RealFlight* function to fly the chosen satellite trajectory through the chosen simulation output. The second block **(B)** retrieves the ion temperature data from the same satellite and adds it to the previously created *kamodo_object* variable. The difference between the two datasets is easily calculated *via* function composition. A fully interactive plot showing the three variables is easily created in the third block **(C)**.

A

```
# Import satellite flythrough code
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
import kamodo_ccmc.flythrough.model_wrapper as MW
```

B

```
# Choose input values for RealFlight function call
modelA, file_dirA = 'CTIPe', 'C:/Users/rringuet/Kamodo_Data/CTIPe/Storm_201303/'
modelB, file_dirB = 'GITM', 'C:/Users/rringuet/Kamodo_Data/GITM/Storm_201303/'
dataset = 'cnofs'
start_utcts, end_utcts = 1363392000, 1363478400
variable_list = ['T_i']  #list of desired variable names
coord_type = 'GEO'  #GEO cartesian coordinate system for trajectory.

# perform flythrough for both models and functionalize.
resultsA = SF.RealFlight(dataset, start_utcts, end_utcts, modelA, file_dirA,
                         variable_list, coord_type=coord_type)
resultsB = SF.RealFlight(dataset, start_utcts, end_utcts, modelB, file_dirB,
                         variable_list, coord_type=coord_type)
kamodo_objectA = SF.WO.Functionalize_SFResults(modelA, resultsA)
kamodo_objectB = SF.WO.Functionalize_SFResults(modelB, resultsB)
kamodo_objectA
```

```
Attribute/Key names of return dictionary: dict_keys(['sat_time', 'c1', 'c2', 'c3'])
{'T_i': 'K', 'utc_time': 's', 'net_idx': '', 'c1': 'R_E', 'c2': 'R_E', 'c3': 'R_E'}
Attribute/Key names of return dictionary: dict_keys(['sat_time', 'c1', 'c2', 'c3'])
{'T_i': 'K', 'utc_time': 's', 'net_idx': '', 'c1': 'R_E', 'c2': 'R_E', 'c3': 'R_E'}
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$

C

```
# Combine into one kamodo object.
from kamodo import Kamodo
kamodo_object = Kamodo()
kamodo_object['CTIPeT_i[K]'] = kamodo_objectA['T_i']
kamodo_object['GITMT_i[K]'] = kamodo_objectB['T_i']
# Perform analysis.
kamodo_object['DIFF[K]'] = 'abs(CTIPeT_i-GITMT_i)'
kamodo_object['MEAN[K]'] = '2 * CTIPeT_i * GITMT_i/(CTIPeT_i + GITMT_i)'
kamodo_object
```

$$\mathrm{CTIPeT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{GITMT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{DIFF}\,(time)[K] = \mathrm{abs}\,(\mathrm{CTIPeT_i}\,(time) - \mathrm{GITMT_i}\,(time))$$
$$\mathrm{MEAN}\,(time)[K] = \frac{2\,\mathrm{CTIPeT_i}\,(time)\,\mathrm{GITMT_i}\,(time)}{\mathrm{CTIPeT_i}\,(time) + \mathrm{GITMT_i}\,(time)}$$

FIGURE 8
Example of ensemble modeling workflow. The first two blocks **(A and B)** use Kamodo's *RealFlight* function to fly the chosen satellite trajectory through two simulation outputs. In the third block **(C)**, the functions are collected into a single Kamodo object for further computation *via* function composition.

of Kamodo with several other Python packages, including SpacePy, PlasmaPy, and pySPEDAS[14] (PlasmaPy Community 2022). Another workflow shown at the 2022 PyHC Summer School demonstrated using SunPy and Kamodo together[15]. Finally, an interface exists between the pysat and kamodo packages[16], but no workflow using the two packages is known to exist.

In this section, we feature a workflow combining pysat and Kamodo to compare simulated and observed data (Figure 9). The workflow presented in Figure 9 is identical to the workflow presented in section 3.1, but instead uses pysat to retrieve the desired data. We adapted code from the pysat example notebooks given at the 2022 PyHC Summer School to implement pysat in this manner. One advantage to this choice is to use the data filtering in pysat to clean the observed data before comparing it with the simulated data, which results in a better comparison to the simulated data.

Comparing the final plots at the bottom of Figures 7, 9 shows the differences between the two analysis methods, especially the reduction and removal of several spikes in the data in Figure 9 as compared to Figure 7 (orange in both plots). As noted in the model-data comparison presented in section 3.1, the simulated data is plotted using its original time cadence of every minute, and the CINDI data and 'DIFF' function are plotted every second, with identical timestamps as in Figure 7. The exception to this [17]results from the data removed by the cleaning process. Figure 10 shows the

14 https://pyspedas.readthedocs.io/en/latest/.

15 https://github.com/heliophysicsPy/summer-school/tree/main/kamodo-tutorial.

16 https://pypi.org/project/pysat-kamodo/.

17 https://github.com/heliophysicsPy/summer-school.

A

```
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
import datetime as dt

# Set input values for RealFlight function call.
model = 'TIEGCM'
file_dir = 'C:/Users/rringuet/Kamodo_Data/TIEGCM/Uriel_Ramirez_012517_IT_1/'
dataset = 'cnofs'
start_utcts = dt.datetime(2015, 3, 18, 0).replace(tzinfo=dt.timezone.utc).timestamp()
end_utcts = dt.datetime(2015, 3, 21, 0).replace(tzinfo=dt.timezone.utc).timestamp()-1
variable_list = ['T_i'] # list of desired variable names
coord_type = 'GEO'  # GEO cartesian coordinates for the trajectory.
results = SF.RealFlight(dataset, start_utcts, end_utcts, model, file_dir,
                        variable_list, coord_type)

# Functionalize the results
kamodo_object = SF.WO.Functionalize_SFResults(model,results)
kamodo_object
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$

B

```
# Get CINDI ion temperature data using pysat.
import pysat  # pip install pysat
import pysatNASA  # pip install pysatNASA

pysat_datadir = 'C:/Users/rringuet/Kamodo_Data/pysat_data/'
pysat.params['data_dirs'] = pysat_datadir
pysat.utils.registry.register(['pysatNASA.instruments.cnofs_ivm'])
ivm = pysat.Instrument('cnofs', 'ivm', inst_id='', clean_level='clean')
start_download_date = dt.datetime.utcfromtimestamp(start_utcts)
stop_download_date = dt.datetime.utcfromtimestamp(end_utcts)
ivm.download(start_download_date, stop_download_date)  # already on machine
ivm.load(date=start_download_date, end_date=stop_download_date)
```

C

```
# Convert pandas timestamps into utc timestamps
import numpy as np
utc_time = ivm.index.values.astype(np.int64)/1e9

# Add to kamodo object.
kamodo_object = SF.WO.Functionalize_TimeSeries(utc_time, 'CINDIT_i', 'K',
                                               ivm.data['ionTemperature'].values,
                                               kamodo_object)
kamodo_object['DIFF'] = 'abs(CINDIT_i - T_i)'
kamodo_object
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{CINDIT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{DIFF}\,(time) = \mathrm{abs}\,(\mathrm{CINDIT_i}\,(time) - \mathrm{T_i}\,(time))$$

D

```
# Now we can plot model and data on the same figure with the difference
kamodo_object.plot('T_i','CINDIT_i','DIFF')
```

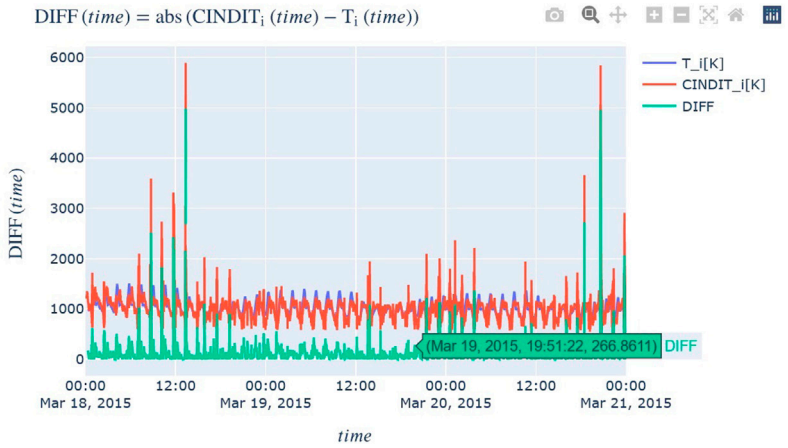


FIGURE 9
Example of model-data comparison using pysat and Kamodo. The first block **(A)** is identical to the first block of Figure 7. The second block **(B)** proceeds to use pysat to retrieve the same data as in Figure 7, but applies the data cleaning routines for the data implemented in pysat. The next block **(C)** converts the times to UTC timestamps, functionalizes the data, and adds the functionalized data to the existing Kamodo object. The output of the final block **(D)** is similar to that of Figure 7, but now contains the cleaned observational data instead of the raw data.

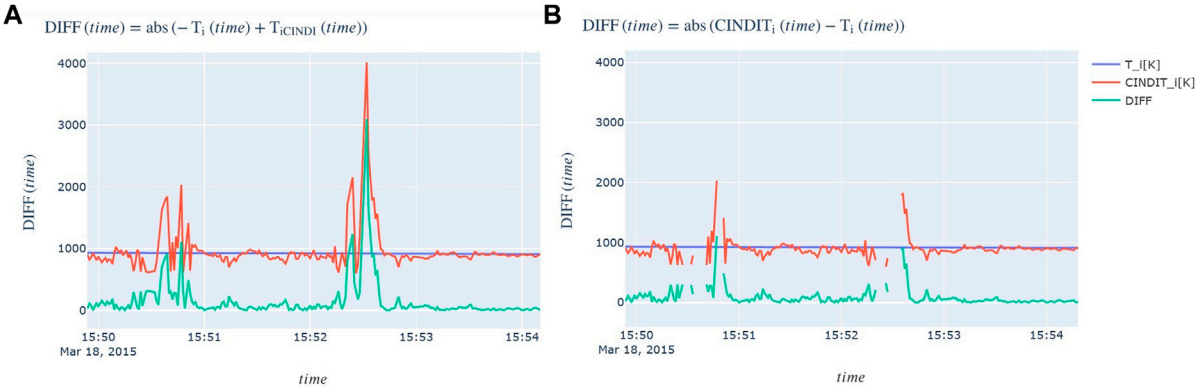


FIGURE 10
Comparison of the model-data comparison workflow results using only Kamodo (left, **(A)** and pysat+Kamodo (right, **(B)**. The data shown in both plots are approximately 4 min sections of the plots shown in Figure 7 and Figure 9 (less than 1/20 of an orbit). The axes in each plot are nearly identical, with the TIE-GCM dataset shown in blue (nearly horizontal lines near 1000 K), the CINDI dataset represented in orange, and the DIFF calculated function in green. The time cadences and values in each plot are identical, except for the data gaps in the right plot. These gaps result from the data cleaning routines developed for the CINDI data in pysat, and are automatically propagated to the DIFF function in the function composition performed using Kamodo.

data gaps in the cleaned CINDI data in a zoomed-in section of the graph centered on the spike just before 1600 UTC on 18 March 2015 as compared to the original data retrieved *via* Kamodo's CDAWeb HAPI interface. These data gaps are propagated to the DIFF function automatically as part of the function composition.

The simulated data and the chosen observational dataset in this workflow can both be changed to the user's specific goal, along with the dates and the analysis function. In the future, we are eager to use pysat's developing interface to orbit propagators as input to Kamodo's flythrough functions, and to use Kamodo to incorporate physics-based models into those orbit propagators.

4 Summary

Kamodo's satellite flythrough capability decreases the utilization barrier for heliophysics model outputs by providing a model-agnostic utility for the entire community. As the library of models implemented in Kamodo expands, so will the capability of the flythrough tools. Our first focus has been to add a variety of ionosphere-thermosphere-mesosphere simulation outputs to Kamodo in support of the upcoming Geospace Dynamics Constellation mission. We are continuing to add more simulation outputs to Kamodo in this domain, and are now expanding our efforts into the geospace domain. Expanding to include geospace model outputs requires a new approach to coordinate conversions due to the often model-specific coordinate systems involved (see Ringuette et al., 2022a). However, this is easily incorporated into the current software architecture by using the function composition capability inherent in Kamodo. Further expansion into heliosphere and solar physics model outputs is also planned. We intend for Kamodo to become the new user interface for interacting with all model outputs on the CCMC website, including the Instant Run, Runs-on-Request, and real-time model output visualization interfaces. Accomplishing this goal will require increasing the capability well beyond what is currently offered. Kamodo's continuing development as open-source software will also deliver the same powerful solution for users on their own computers and on the cloud.

Additional capabilities based on the flythrough are also in development. For example, we have developed a satellite constellation mission planning tool in support of the Geospace Dynamics Constellation mission (Pfaff, 2016) that is now available on GitHub as part of the Kamodo CCMC software package for user testing. To increase accessibility of model outputs, we are beginning work on adding HAPI as a layer on top of the *RealFlight* and *TLEFlight* flythrough functions and on top of the model reader interfaces. This will enable the community to more easily access and download model outputs by only serving the desired portions of the data instead of serving the often prohibitively large files. We are also working to improve CCMC's Kamodo documentation, especially including sample workflows similar to the ones included in this work. Simple additions planned in the short-term include code refactoring to generalize the treatment of model-specific coordinate conversions, and additional sample workflows. Long term development goals include a line of sight

calculation tool and a simulated imagery tool to expand beyond *in-situ* applications, cloud capabilities, nowcasting with continuous runs, satellite drag modeling[18], and reduced code and code-free interfaces to model outputs (see Figure 6, Figure 7 of Ringuette et al., 2022b).

Building these capabilities on top of Kamodo provides a powerful foundation for a large range of exciting capabilities, especially for the complex case of simulation outputs. Since Kamodo works for any data, our work can be easily extended into other areas, even outside of the science and research communities entirely. Applying Kamodo to heliophysics simulated and observed data is beginning to lower the utilization barrier for the entire community. Collaborators are welcome!

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: The simulated datasets analyzed for this study can be found in the Kamodo Demo directory at CCMC (https://ccmc.gsfc.nasa.gov/RoR_WWW/output_files/KAMODO_DEMO/). Additional model outputs can be accessed through the CCMC (https://ccmc.gsfc.nasa.gov/). The observational dataset from the CINDI instrument analyzed in this work is available through NASA SPDF's CDAWeb HAPI service (https://cdaweb.gsfc.nasa.gov/index.html).

Author contributions

RR wrote the text of the article, developed and tested the flythrough functions described, and generated figures. DD developed the visualization codes and coordinate conversion codes to support the flythrough functions. LR directed the development of the flythrough functions. DD and LR assisted in testing of the software and feedback on its functionality during development. AP and OG supported the required functionality by developing and maintaining the Kamodo-Core software package that the CCMC-Kamodo package depends on. KG-S assisted in the development of the *TLETrajectory* function. LR, DD, OG, and KG-S also provided feedback on the article content.

18 See http://alpha.drag.ensemblespacelabs.com:1234/, and https://www.youtube.com/watch?v=vpJgtAMCVvc&ab_channel=Ensemble for a tutorial video.

Funding

RR's work was funded by the CCMC through ADNET Systems, Inc. DD's work was funded by the CCMC through the Catholic University of America. LR's and KG-S's work was funded directly by the CCMC. AP and OG's work were funded *via* a NASA SBIR Phase 2: Space Weather R2O/O2R Technology Development grant titled "Kamodo Containerized Space Weather Models" (Contract #80NSSC20C0290) awarded to Ensemble Government Services, LLC. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

Acknowledgments

RR acknowledges Russell Stoneback for helpful comments on the pysat-Kamodo notebooks. This work uses simulation results from the Community Coordinated Modeling Center at Goddard Space Flight Center. The CCMC is a multi-agency partnership between NASA, AFMC, AFOSR, AFRL, AFWA, NOAA, NSF, and ONR. We also acknowledge the use of CINDI data from the C/NOFS mission, which was provided by the CINDI team at the University of Texas at Dallas, made available through NASA SPDF's CDAWeb HAPI service. The CINDI team was supported by NASA grant NAS5-01068. See https://hpde.io/NASA/NumericalData/CNOFS/CINDI/IVM/PT0.5S.html.

Conflict of interest

Author RR was employed by ADNET Systems Inc., a contractor for NASA GSFC. Authors AP and OG were employed by Ensemble Government Services.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

AMGeO Collaboration (2019). *A collaborative data science platform for the geospace community: Assimilative mapping of geospace observations (AMGeO) v1.0.0*. Zenodo. doi:10.5281/zenodo.3564914

AstroPy Collaboration, Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., et al. (2013). Astropy: A community Python package for astronomy. *Astron. Astrophys.* 558, A33. doi:10.1051/0004-6361/201322068

AstroPy Collaboration, Sipocz, B. M., Gunther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., et al. (2018). The astropy project: Building an open-science project and status of the v2.0 core package. *Astron. J.* 156 (3), 123. doi:10.3847/1538-3881/aabc4f

Bard, C., and Dorellli, J. C. (2021). Magnetotail reconnection asymmetries in an ion-scale, Earth-like magnetosphere. *Ann. Geophys.* 39, 991–1003. doi:10.5194/angeo-39-991-2021

Bilitza, D. (2018). IRI the international standard for the ionosphere. *Adv. Radio Sci.* 16, 1–11. doi:10.5194/ars-16-1-2018

Bruinsma, S. (2015). The DTM-2013 thermosphere model. *J. Space Weather Space Clim.* 5, A1. doi:10.1051/swsc/2015001

Codrescu, M. V., Fuller-Rowell, T. J., Munteanu, V., Minter, C. F., and Millward, G. H. (2008). Validation of the coupled thermosphere ionosphere plasmasphere electrodynamics model: CTIPe-mass spectrometer incoherent scatter temperature comparison. *Space* 6, S09005. doi:10.1029/2007SW000364

Coley, W. R., Heelis, R. A., Hairston, M. R., Earle, G. D., Perdue, M. D., Power, R. A., et al. (2010). Ion temperature and density relationships measured by CINDI from the C/NOFS spacecraft during solar minimum. *J. Geophys. Res.* 115, A02313. doi:10.1029/2009JA014665

Fang, T.-W., Kubaryk, A., Goldstein, D., Li, Z., Fuller-Rowell, T., Millward, G., et al. (2022). Space weather environment during the spaceX starlink satellite loss in february 2022. *Space Weather* 20, e2022SW003193. doi:10.1029/2022SW003193

Fok, M.-C., Buzulukova, N. Y., Chen, S.-H., Glocer, A., Nagai, T., Valek, P., et al. (2014). The comprehensive inner magnetosphere-ionosphere model. *J. Geophys. Res. Space Physics* 119, 7522–7540. doi:10.1002/2014JA020239

Garcia-Sage, K., Waldron, Z., Rowlands, D., Lemoine, F., Sutton, E., Thayer, J., et al. (2021). *Expanding and testing orbit propagation capabilities using CCMC-hosted models*. Washington, DC: American Geophysical Union. Presented at the AGU Fall Meeting 2021, 13-17 December 2021, id. SA31B-04. Available at: https://ui.adsabs.harvard.edu/abs/2021AGUFMSA31B.04G/abstract.

Liu, H.-L., Bardeen, C. G., Foster, B. T., Lauritzen, P., Liu, J., Lu, G., et al. (2018). Development and validation of the whole Atmosphere community climate model with thermosphere and ionosphere extension (WACCM-X 2.0). *J. Adv. Model. Earth Syst.* 10, 381–402. doi:10.1002/2017MS001232

Luthcke, S. B., Rowlands, D. D., Lemoine, F. G., Klosko, S. M., Chinn, D., and McCarthy, J. J. (2006). Monthly spherical harmonic gravity field solutions determined from GRACE inter-satellite range-rate data alone. *Geophys. Res. Lett.* 33, L02402. doi:10.1029/2005GL024846

Maruyama, N., Sun, Y.-Y., Richards, P. G., Middlecoff, J., Fang, T.-W., Fuller-Rowell, T. J., et al. (2016). A new source of the midlatitude ionospheric peak density structure revealed by a new Ionosphere-Plasmasphere model. *Geophys. Res. Lett.* 43. doi:10.1002/2015GL067312

Morley, S. K., Welling, D. T., Koller, J., Larsen, B. A., Henderson, M. G., Niehof, J., et al. (2011). "SpacePy - a python-based library of tools for the Space sciences," in Proceedings of the 9th Python in Science Conference, Austin, TX, July 11–17, 2010, p. 67–72. Editors S. vad der Walt and J. Millman. doi:10.25080/Majora-92bf1922-00c

Pembroke, A., De Zeeuw, D., Rastaetter, L., Ringuette, R., Gerland, O., Patel, D., et al. (2022). Kamodo: A functional api for space weather models and data. *J. Open Source Softw.* 7, 4053. doi:10.21105/joss.04053

Pfaff, R. F., Jr. (2016). *An overview of the scientific and space weather motivation for the "Notional" geospace dynamics constellation mission*. American Geophysical Union, Fall General Assembly. Available at: https://ui.adsabs.harvard.edu/abs/2016AGUFMSA23C..01P

PlasmaPy Community (2022). *PlasmaPy, version 0.8.1*. Zenodo. doi:10.5281/zenodo.6774350

Plotly Technologies Inc (2015). *Collaborative data science*. Montréal, QC: Plotly Technologies Inc. https://plot.ly.

Polson, S., Ringuette, R., Rastaetter, L., Grimes, E., Niehof, J., Murphy, N. A., et al. (2022). Making an executable paper with the python in heliophysics community to foster open science and improve reproducibility. *Front. Astron.* Under review.

Qian, L., Burns, A., Emery, B., Foster, B., Lu, G., Maute, A., et al. (2013). The ncar TIE-GCM: A community model of the coupled thermosphere/ionosphere system. *Geophys. Monogr. Ser.* 201, 73–83. doi:10.1029/2012GM001297

Raeder, J., Wang, Y. L., Fuller-Rowell, T. J., and Singer, H. J. (2001). Global simulation of space weather effects of the Bastille Day storm. *Sol. Phys.* 204, 325. doi:10.1023/A:1014228230714

Rastaetter, L., Wiegand, C., Mullinix, R., and MacNiece, P. J. (2019). Comprehensive assessment of models and events using library tools (camel) framework: Time series comparisons. *Space Weather* 17 (6), 845–860. doi:10.1029/2018SW002043

Ridley, A., Deng, Y., and Tóth, G. (2006). The global ionosphere–thermosphere model. *J. Atmos. Solar-Terrestrial Phys.* 68, 839–864. doi:10.1016/j.jastp.2006.01.008

Ridley, A. J., De Zeeuw, D. L., and Rastätter, L. (2016). Rating global magnetosphere model simulations through statistical data-model comparisons. *Space* 14, 819–834. doi:10.1002/2016SW001465

Ringuette, R., Engell, A., Gerland, O., McGranaghan, R. M., and Thompson, B. (2022b). The DIARieS ecosystem – a software ecosystem to simplify discovery, implementation, analysis, reproducibility, and sharing of scientific results and environments in Heliophysics. *Adv. Space Res.* doi:10.1016/j.asr.2022.05.012

Ringuette, R., Rastätter, L., De Zeeuw, D. L., Pembroke, A., and Gerland, O. (2022a). Simplifying model data access and utilization. *Adv. Space Res.* under review.

Robinson, R. M., Zanetti, L., Anderson, B., Vines, S., and Gjerloev, J. (2021). Determination of auroral electrodynamic parameters from AMPERE field-aligned current measurements. *Space* 19, e2020SW002677. doi:10.1029/2020SW002677

Stoneback, R., Burrell, A. G., Klenzing, J., and Depew, M. D. (2018). PYSAT: Python satellite data analysis toolkit. *JGR. Space Phys.* 123, 5271–5283. doi:10.1029/2018JA025297

Thomas, E. G., and Shepherd, S. G. (2018). Statistical patterns of ionospheric convection derived from mid-latitude, high-latitude, and polar SuperDARN HF radar observations. *J. Geophys. Res. Space Phys.* 123 (4), 3196–3216. doi:10.1002/2018JA025280

Toth, G., De Zeeuw, D. L., Gombosi, T. I., Manchester, W. B., Ridley, A. J., Sokolov, I. V., et al. (2007). Sun-to-thermosphere simulation of the 28-30 october 2003 storm with the space weather modeling framework. *Space* 5, S06003. doi:10.1029/2006SW000272

Vallado, D., and Crawford, P. (2008). "SGP4 orbit determination," in AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Honolulu, Hawaii, August 18–21, 2008 (Reston, VA: American Institute of Aeronautics and Astronautics). doi:10.2514/6.2008-6770

Weigel, R. S., Vandegriff, J., Faden, J., King, T., Roberts, D. A., Harris, B., et al. (2021). HAPI: An API standard for accessing heliophysics time series data. *JGR. Space Phys.* 126, 12. doi:10.1029/2021JA029534

Zhang, B., Sorathia, K. A., Lyon, J. G., Merkin, V. G., Garretson, J. S., and Wiltberger, M. (2019). Gamera: A three-dimensional finite-volume MHD solver for non-orthogonal curvilinear geometries. *Astrophys. J. Suppl. Ser.* 244, 20. doi:10.3847/1538-4365/ab3a4c

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 06 December 2022
DOI 10.3389/fspas.2022.1028042

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Mourad Djebli,
USTHB, Algeria
Jun Liang,
University of Calgary, Canada

*CORRESPONDENCE
Stephen R. Kaeppler,
skaeppl@clemson.edu

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 25 August 2022
ACCEPTED 24 October 2022
PUBLISHED 06 December 2022

CITATION
Kaeppler SR, Marshall R, Sanchez ER, Juarez Madera DH, Troyer R and Jaynes AN (2022), pyGPI5: A python D- and E-region chemistry and ionization model. *Front. Astron. Space Sci.* 9:1028042. doi: 10.3389/fspas.2022.1028042

COPYRIGHT
© 2022 Kaeppler, Marshall, Sanchez, Juarez Madera, Troyer and Jaynes. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

pyGPI5: A python D- and E-region chemistry and ionization model

Stephen R. Kaeppler[1]*, Robert Marshall[2], Ennio R. Sanchez[3], Diana H. Juarez Madera[4], Riley Troyer[5] and Allison N. Jaynes[5]

[1]Department of Physics and Astronomy, Clemson University, Clemson, SC, United States, [2]Department of Aerospace Engineering Sciences, University of Colorado Boulder, Boulder, CO, United States, [3]Center for Geospace Studies, SRI International, Menlo Park, CA, United States, [4]Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, United States, [5]Department of Physics and Astronomy, University of Iowa, Iowa City, IA, United States

We present a Python implementation of a D- and E-region chemistry and ionization code called pyGPI5. Particle precipitation that penetrates into the E- and D-region of the ionosphere-thermosphere causes significant enhancements of the electron density. Dissociative recombination of molecular ions with electrons is the primary electron loss mechanism in the E-region, down to approximately 85 km. However, below 85 km, chemical processes become significantly more complicated with positive and negative ions being generated in addition to electrons. The complex D-region ion chemistry has been known for many decades. We present a formulation to quantify the concentrations of four ion species composed of positive and negative, light and heavy ions, and the electrons. The implementation we describe in this investigation solves five ordinary stiff differential equations simultaneously. We present an overview of the code, along with discussions of the reaction rates, and assumptions used in the model. We describe an implementation of the electron transport model to quantify the altitude ionization profile caused by energetic particle precipitation. We show how to instantiate the model, generate the ion and electron profiles as a function of altitude for background conditions, how to generate altitude ionization profiles, and running the code to produce ion and electron profiles caused by energetic particle precipitation. Recent investigations that have used a D-region chemistry model are discussed, along with some applications.

KEYWORDS

aurora, particle precipitation, D-region, E-region, ion chemistry, ionization

1 Introduction

It has been well-established that the ion chemistry in the D-region of the ionosphere is significantly more complicated than the ion chemistry in the E- or F-regions (Rishbeth and Garriott, 1969; Brekke, 2013). Early rocket observations from ion mass spectrometers resolved positive and negatively charged ions (Narcisi and Bailey, 1965). The D-region

electron density has strong impacts on the absorption of high frequency (HF) radio wave propagation (Davies, 1990; Zawdie et al., 2017).

The D-region electron density can be significantly enhanced by energetic particle precipitation. These electron density enhancements can be observed with VLF networks (Cummer et al., 1997; Marshall et al., 2019; Hendry et al., 2020), optical techniques (Marshall et al., 2014; Marshall et al., 2019), at X-ray wavelengths (Marshall et al., 2020), riometers (Marshall and Cully, 2020), and using incoherent scatter radar (Marshall et al., 2014; Miyoshi et al., 2015; Sivadas et al., 2017; Kaeppler et al., 2020; Sanchez et al., 2022). Marshall and Cully (2020) provides an excellent review of observational techniques. However, quantifying observational parameters requires a model that describes the chemistry of the D-region.

There are relatively few models that describe the D-region ion chemistry. To date, the Sodankylä Ion Chemistry (SIC) model is the most comprehensive model of D-region chemistry (Verronen et al., 2005; Turunen et al., 2009; Turunen et al., 2016). SIC is a 1-D vertical transport code that is valid between 20–150 km, with 1 km resolution, and takes into account hundreds of chemical reactions that are driven by solar UV, X-ray, or energetic particle precipitation. SIC solves for the concentrations of at least 65 ions, 36 of which are positive ions, 29 of which are negative ions, and 15 minor neutral species (Turunen et al., 2009, and references therein). There have been recent efforts to incorporate the chemical reactions described in the SIC model into global whole-atmosphere models (Kovács et al., 2016; Verronen et al., 2016).

A second, more simplified, model was originally written by Glukov, Pasko, and Inan (GPI) (Glukhov et al., 1992) which solved four simultaneous differential equations representing the time evolution of the electrons, light positive and negative ions, and heavy positive cluster ions. The GPI model makes the important simplification that ion species and the electrons are modeled as individual fluids; the methodology does not track the evolution of individual ion constituents. The GPI model also does not compute the time evolution of the neutrals. The reaction rates used in the equations are effective reaction rates for the fluid. Lehtinen and Inan (2007) went on to modify the GPI model to include heavy negative cluster ions; therefore, GPI5 solves simultaneously for five ordinary differential equations. Lehtinen and Inan (2007) also made modifications to some of the reaction rates relative to Glukhov et al. (1992).

In this investigation, we present a Python implementation of the GPI5 D-region chemistry and ionization model, which we call pyGPI5. The goal is to produce an open-source, easy-to-use, fast, D-region chemistry and ionization model that is accurate and sufficient for many applications. pyGPI5 provides an implementation of ionization models developed by Fang et al. (2008) and Fang et al. (2010). In **Section 2**, we present the implementation of the pyGPI5 code, which includes a description of the Python classes. **Section 3** presents results which demonstrate various model outputs and basic results. A Jupyter notebook is included in the github repository that shows how the figures for this section were generated. **Section 4** presents some recent results that have utilized D-region chemistry codes, not specific to the Python implementation, and demonstrating some applications.

2 Methodology

2.1 Overview

We briefly discuss an overview of the code architecture with an emphasis on the production of electron density. There are two Python classes that compromise the code: the `Chemistry` class and the `Ionization` class. The `Chemistry` class contains the heart of the GPI5 code, in which, five first order stiff ordinary differential equations (ODEs) that describe the time evolution of the electrons, light positive and negative ions, and heavy positive and negative cluster ions are solved simultaneously. The `Ionization` class produces an altitude ionization profile that is necessary in the equations describing the time evolution of the ions and electrons.

As a forward model, the altitude resolved ionization profile is used as an input. However, in pyGPI5 the inverse problem is solved to produce the background ionization, i.e., given an observed or empirical electron density, the altitude ionization profile is determined. Estimating the differential number flux given the altitude ionization profile is not discussed in this paper, but there are several useful papers that present techniques to address this problem (Semeter and Kamalabadi, 2005; Simon Wedlund et al., 2013; Turunen et al., 2016; Sivadas et al., 2017; Kaeppler et al., 2020).

Before describing the code implementation in more detail, we elaborate on a few important assumptions. First, pyGPI5 is a 1-D model in the vertical component for a given longitude and latitude. The equations described below are solved at each altitude independently and there is no coupling between altitudes. Second, the time evolution of the ion and electron equation is quantified, but each time interval is quantified independently, depending on the ionization profile and the neutral atmospheric parameters. Third, the specification of the neutral atmosphere and corresponding neutral constituents is specified using the NRLMSIS00 model (Picone et al., 2002). pyGPI5 does not solve differential equations describing the time evolution of the neutral species. Fourth, transport effects associated with electric fields and neutral winds are not considered in the continuity equations presented below.

2.2 D-region chemistry class

The heart of the code is contained in the `Chemistry` class in which five coupled stiff first order differential equations are solved that describe the time evolution of the ions and electrons. The equations from Lehtinen and Inan (2007) that we solve are:

$$\frac{dn_e}{dt} = q + \gamma n^- + \gamma_x n_x^- - \beta n_e - \alpha_d n_e n^+ - \alpha_d^c n_e n_x^+ \quad (1a)$$

$$\frac{dn^-}{dt} = \beta n_e - \gamma n^- - \alpha_i n^- \left(n^+ + n_x^+\right) - \overline{X} n^- \quad (1b)$$

$$\frac{dn_x^-}{dt} = -\gamma_x n_x^- - \alpha_i n_x^- \left(n^+ + n_x^+\right) + \overline{X} n^- \quad (1c)$$

$$\frac{dn^+}{dt} = q - B n^+ - \alpha_d n_e n^+ - \alpha_i n^- n^+ - \alpha_i n_x^- n^+ \quad (1d)$$

$$\frac{dn_x^+}{dt} = -\alpha_d^c n_e n_x^+ + B n^+ - \alpha_i n^- n_x^+ - \alpha_i n_x^- n_x^+ \quad (1e)$$

where **Table 1** summarizes the terms found in Eq. 1. These equations are a set of stiff ordinary differential equations that we numerically solve using the Python Scipy `ode` package. We use the `vode` integration option with the backward differentiation formulas (bdf) to solve the stiff set of differential equations.

An observed electron density is composed of both a background electron density and the electron density enhancements caused by energetic particle precipitation or other sources (e.g., solar X-rays, galatic cosmic rays, etc). The background is required as an initial condition upon which the enhanced electron density overlays. We use a bounded bisection method (Press et al., 2007) to derive the ionization profile given an observed electron density or an electron density from a standard model, i.e., the 2016 International Reference Ionosphere (IRI) (Bilitza et al., 2017). At each altitude (z) an iterative process occurs in which, the ionization ($q(z)$) is varied, the ODEs are solved, and the model electron density is compared to the observations or an empirical specification. For the case of the background electron density, we use the corresponding IRI electron density profile, run the bisection method, and derive the altitude ionization profile. For energetic particle precipitation, the process is the same, but in this case observations are used. We assume that each altitude is independent of other altitudes when running the bisection method.

TABLE 1 Variables that are used in Eq. 1.

Symbol	Definition	Units
n_e	electron density	cm^{-3}
n^+	light positive ions	cm^{-3}
n^-	light negative ions	cm^{-3}
n_x^-	heavy negative cluster ions	cm^{-3}
n_x^+	heavy positive cluster ions	cm^{-3}
γ	effective electron detachment rate	s^{-1}
γ_x	effective electron detachment rate for heavy ions	s^{-1}
α_d	effective coefficient of dissociative recombination	$cm^{-3}s^{-1}$
α_d^c	effective recombination of electrons with positive cluster ions	$cm^{-3}s^{-1}$
α_i	ion-ion recombination rate	$cm^{-3}s^{-1}$
B	effective position ions into positive cluster ions	s^{-1}
$\overline{X}$	conversion rate from negative light ions to negative heavy ions	s^{-1}
β	effective electron attachment rate	s^{-1}
q	altitude ionization rate	$cm^{-3}s^{-1}$

The reaction rates used in Eq. 1 are found in Glukhov et al. (1992) and Lehtinen and Inan (2007). Many of these reaction rates are dependent on the neutral number density, N_m, as a function of altitude; we use the NRLMSIS00 neutral mass density model for atmospheric specification (Picone et al., 2002). We briefly summarize the reaction rates used, but more details regarding the origin of these coefficients can be found in Glukhov et al. (1992) and Lehtinen and Inan (2007). The effective recombination of electrons with positive cluster ions (n^+) we use is

$$\alpha_d^c = 10^{-5} \ \left[cm^3 s^{-1}\right] \quad (2)$$

(Glukhov et al., 1992; Florescu-Mitchell and Mitchell, 2006). The effective ion–ion recombination between positive ions with negative ions is

$$\alpha_i = 10^{-7} + N_m \times 10^{-24} \ \left[cm^3 s^{-1}\right], \quad (3)$$

which includes three-body mutual neutralization processes that are important below 40 km altitude (Lehtinen and Inan, 2007, and references therein). The effective electron attachment rate is

$$\beta = N_m^2 5.6 \times 10^{-32} \ \left[cm^3 s^{-1}\right] \quad (4)$$

which is an approximation for 3-body processes based on the temperature dependent formula found in Glukhov et al. (1992), but evaluated for a specific temperature $T = 200K$ and specified atmospheric concentrations. See Lehtinen and Inan (2007) for more details. The conversion rate from light positive ions, n^+, to heavy cluster ions n_x^+ is described by the term B,

$$B = N_m^2 \times 10^{-31} \ \left[s^{-1}\right] \quad (5)$$

The detachment rate of heavy negative ions, γ_x, is defined as $\gamma_x = 0$ at night and has a constant rate, $\gamma_x = 0.002$, during the day which is primarily due to photodetachment. We define $\overline{X}$ as,

$$\overline{X} = 0.2 \times 10^{-30} N_m^2 \ \left[s^{-1}\right] \quad (6)$$

which corresponds to the conversion rate from light (n^-) to heavy negative ions.

The effective electron detachment rate, γ, corresponds to the following reaction of O_2,

$$O_2^- + O_2 \rightarrow e^- + 2O_2 \tag{7}$$

and the default value in the code is specified as

$$\gamma = 3 \times 10^{-17} N_m \ [s^{-1}] \tag{8}$$

However, two other options are available. The first option is described by **Eq.7** in Lehtinen et al. (1999),

$$\gamma = 8.61 \times 10^{-10} N_m e^{-6030/T_n} \ [s^{-1}] \tag{9}$$

Where T_n is the neutral temperature. The second option is a slightly modified equation derived by (Kozlov et al., 1988)

$$\gamma = 2.7 \times 10^{-10} N_m e^{-5590/T_n} \sqrt{\left(\frac{T_n}{300}\right)} \ [s^{-1}] \tag{10}$$

The neutral temperature, T_n, needed in both calculations is provided by the NRLMSISE00 model.

pyGPI5 has been modified to include chemistry in the E-region of the ionosphere. In the E-region, the primary loss process is dissociative recombination (α_d) of the dominant molecular ions NO^+ and O_2^+ (Schunk and Nagy, 2004). The GPI5 code originally used a constant recombination rate for α_d. However for pyGPI5, we use the following recombination rates for NO^+ and O_2^+ (Schunk and Nagy, 2004), for $Te < 1200\ K$:

$$\begin{aligned} \alpha_{O_2^+} &= 1.95 \times 10^{-7} (300/T_e)^{0.70} \ [cm^3 s^{-1}] \\ \alpha_{NO^+} &= 3.50 \times 10^{-7} (300/T_e)^{0.69} \ [cm^3 s^{-1}] \end{aligned} \tag{11}$$

and for $T_e > 1200\ K$:

$$\begin{aligned} \alpha_{O_2^+} &= 1.95 \times 10^{-7} (300/T_e)^{0.56} \ [cm^3 s^{-1}] \\ \alpha_{NO^+} &= 3.02 \times 10^{-7} (300/T_e)^{0.56} \ [cm^3 s^{-1}] \end{aligned} \tag{12}$$

where T_e is the electron temperature from IRI2016. These two recombination rates are weighted using the relative concentrations of NO^+ and O_2^+:

$$\alpha(z) = \sum_i Ic_i(z)\, \alpha_i(z) \tag{13}$$

where Ic_i corresponds to the ion concentration that can be taken from IRI2016 which internally derives these values from the FLIP ion chemistry model (Richards et al., 2010).

There is one free parameter that is required to solve Eq. 1 which is the equation integration time, or the end time for the integration of the ODEs. The chemistry interactions are very rapid in the D-region, with steady-state conditions being achieved of the order of seconds or less; while in the E-region, 1–10 s of seconds is considered a pretty typical steady-state interval (Semeter and Kamalabadi, 2005). For example, Marshall et al. (2014) shows a 5 MeV 0.1A beam that has a peak electron density enhancement at 44 km, but the recovery time to ambient levels occurs of the order of 10–100 ms. These fundamental recovery time are the basis for the choice of the integration time, although the integration scheme should have time steps that are significantly shorter than the characteristic recovery time. However, in some cases, an appropriate integration time may depend on the application for which the code is being used or on another constraint, such as, the time interval associated with the observations. For the case of ISR, the collection time of the radar, i.e., the integration interval, is typically significantly longer and of the order of 10–300 s for most experiments. Therefore, it makes sense to use the longer integration interval for the equations when comparing to ISR observations. A recent experiment by Bernhardt et al. (2022) examined the decay of the electron density from stimulated VLF emissions using rocket engines, which were of the order of 15–240 s.

2.3 Ionization class

The second class in pyGPI5 is the `Ionization` class. Fang et al. (2008) and Fang et al. (2010) derived an easy-to-calculate electron transport model. Fang et al. (2008) derived the parameterization assuming that the precipitating electron energy flux was characterized by a Maxwellian distribution. Fang et al. (2010) expanded upon Fang et al. (2008) by relaxing the Maxwellian assumption and using monoenergetic beams thus enabling the use of other data sets (i.e., satellite data) with discrete energies. Effectively, Fang et al. (2008) and Fang et al. (2010) solved the following equations,

$$q(z) = \frac{Q_0}{2\Delta\epsilon} \frac{f(y, E_0)}{H(z)} \tag{14a}$$

$$f(y, E_0) = C_1 y^{C_2} \exp\left(-C_3 y^{C_4}\right) + C_5 y^{C_6} \exp\left(-C_7 y^{C_8}\right) \tag{14b}$$

$$y = \frac{1}{E_0} \left(\frac{\rho(z) H(z)}{\alpha_0} \right)^{\beta_0} \tag{14c}$$

$$C_i = \exp\left(\sum_j P_{ij} \ln(E_0)^j \right) \tag{14d}$$

where $\Delta\epsilon$ is the ion-pair production energy (i.e., 35 eV), Q_0 is the energy flux, $H(z) = k_B T_n(z)/m(z)g(z)$ is the atmospheric scale height, $\rho(z)$ is the atmospheric mass density given by NRLMSIS00, and α_0 is a constant. P_{ij} and β_0 are fit parameters that depend on whether the incident differential number flux is a Maxwellian distribution (Fang et al., 2008) or monoenergetic (Fang et al., 2010), and are not repeated here.

A key model which will be implemented in future versions of pyGPI5 is the Boulder Electron Radiation to Ionization

(BERI) model (Xu et al., 2020). BERI was developed as an easy-to-calculate ionization model that included pitch angle effects and is valid for energies of 3 keV–33 MeV. This model was specifically designed to address whether pitch angle effects had important consequences in D-region and mesosphere chemistry. In addition, other transport models can be implemented in pyGPI5. For a summary of other transport models, we refer the read to the monograph by Kaeppler et al. (2020).

2.4 Comparisons with Sodankylä ion chemistry

The GPI5 implementation has been compared with results from the SIC model in a few limited cases. **Figure 3** (not shown here) in Marshall et al. (2019) presents a comparison of the two models for the case of a 1 MeV electron beam delivering 1 kJ over 100 pulses every 5 ms. The comparison shows that the altitude of the peak electron density of the GPI5 model was 60 km, while the peak altitude for the SIC model was 61 km. Additionally, the SIC model predicted a peak electron density that was 50% larger than the GPI5 model. In spite of the differences the overall agreement between both models is favorable. An area of future effort could be more systematic comparisons between GPI5 and the SIC model.

3 Results

We present examples of different implementations within pyGPI5. The figures below can be generated in the `RunExamples.ipynb` Jupyter notebook. The results presented were collected with the Poker Flat Incoherent Scatter Radar (PFISR) (65.13°N and 147.47°W) for 08 May 2018 at 0500 UTC. This interval and location were chosen to be similar to an event of energetic particle precipitation presented in Sanchez et al. (2022). The geomagnetic conditions associated with this time interval were Dst = -30 nT and Kp = 2 (Sanchez et al., 2022). We present results from 60 to 150 km altitudes at 1 km altitude resolution.

Figure 1 presents reaction rates described in **Table 1** that are necessary to solve Eq. 1. The left column shows the recombination rates and the right column presents the other coefficients, both columns are separated by units. For the effective recombination rate, α_d, at approximately 75 km, the concentrations of NO^+ and O_2^+ both go to zero, thus resulting in the effective recombination rate going to zero at this altitude. Above 100 km, we see that the recombination rate has a nearly exponential decrease with respect to altitude, which is similar to previous investigations (Vickrey et al., 1982; Gledhill, 1986). The altitude profile of β, B, and $\overline{X}$ are similar to the NRLMSIS00 neutral mass density, as expected.

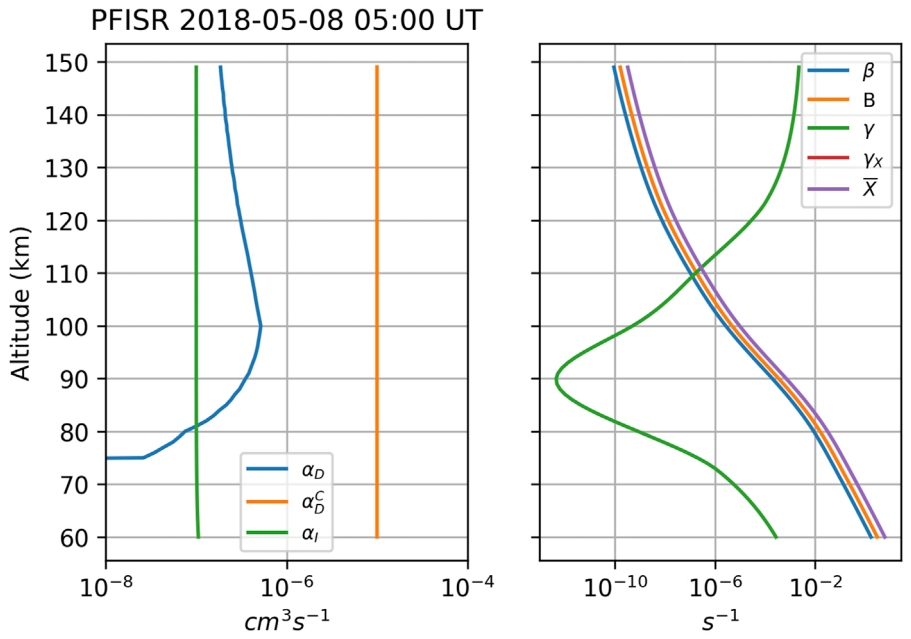


FIGURE 1
Altitude profiles of the coefficients used in Eq. 1. The left column shows the recombination rates with units of cm^3s^{-1} and the right column shows other reaction rates which have units of s^{-1}. The profiles are shown from 60 to 150 km altitudes at the location of the Poker Flat Incoherent Scatter Radar (PFISR) on 08 May 2018 at 0500 UTC.

Figure 2 shows the altitude ionization profiles for a monoenergetic flux and a Maxwellian energy distribution. The altitude ionization profiles caused by the monoenergetic energy flux described in Fang et al. (2010), while for the Maxwellian energy flux we used the Fang et al. (2008) implementation. In both cases, the same input energy flux of 1 mW/m^2 was used, but different monoenergetic beam energies or characteristic energy for the Maxwellian flux were used.

The peak altitude of the electron density for the Maxwellian flux distribution occurs at a lower altitude relative to the monoenergetic beam with the same energy. For example, the altitude of the peak electron density for the 2 keV monoenergetic beam is approximately 125 km, while the peak altitude for the 2 keV Maxwellian distribution is approximately 107 km. The Maxwellian distribution has a characteristic energy, E_0, which corresponds to the energy at which the peak flux occurs. However, the average energy $\langle E \rangle$ is used instead because it is a figure of merit that is independent of the spectral shape of the differential energy flux. For the a Maxwellian, $\langle E \rangle = 2E_0$ (Robinson et al., 1987), and the energy in the legend of **Figure 2** corresponds to the characteristic energy. If we compare the peak altitude of 2 keV Maxwellian flux with the 4 keV monoenergetic beam, the altitudes are much closer to each other, which is consistent with the difference between the average energy and the characteristic energy.

Figure 3 shows altitude profiles for the light and heavy positive and negative ions, and the electrons, respectively. The left column shows the altitude profiles for background conditions, which correspond to an IRI run on 08 May 2018 at 0500 UT. As stated in **Section 2.2**, Eq. 1 is solved in steady state to produce the light and heavy ion and electron profiles. The right column shows the response of the electron density caused by a 300 keV monoenergetic beam with a 1 mW/m^2 energy flux. The light positive ions, n^+, are the dominant ion species down to 85 km, and approximately match in magnitude with the electron density. The response we observe is what would be expected for chemistry with molecular ions in the E-region of the ionosphere. Below 85 km, for the background case the positive light ions, positive heavy ions, and negative heavy ions have similar magnitudes. These species work together to cause the electron density to be lower than simply the positive light ions contribution alone.

During the case of enhanced ionization from the energetic particle precipitation the electron density has a peak at ~ 75 km altitude. The peak altitude of the electron density is consistent with the altitude of the peak ionization shown in **Figure 2**. The electron density above 100 km altitude is unaffected by the energetic precipitation since the peak altitude of the ionization is below 80 km. The light positive ions approximately balance the electrons at all altitudes. This is partially a consequence of **Eq. 1a** and **Eq. 1d**), since both of these equations contain the altitude ionization profile represented by q. Although we note that a secondary peak in the negative light ions has developed at ~ 75 km altitude during the energetic precipitation, eventhough it is 1-2 orders of magnitude smaller than the positive light ions. The heavy positive and negative ions have the largest magnitude below 85 km, although there is no difference between

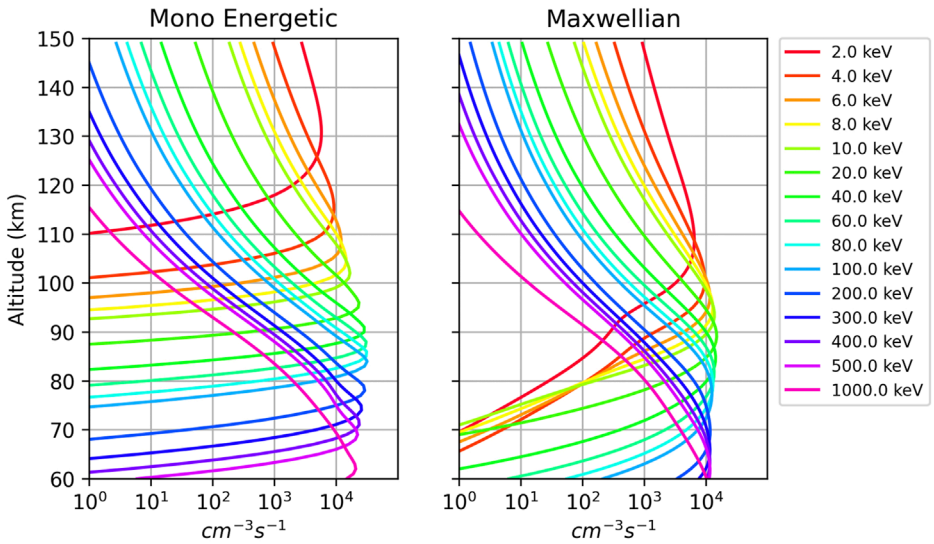


FIGURE 2
The altitude ionization rates for monoenergetic and Maxwellian flux distributions in the left and right columns, respectively. The monoenergetic beam energy and characteristic energy of the Maxwellian distribution are shown in the caption. Both of these cases use a 1 mW/m^2 energy flux.

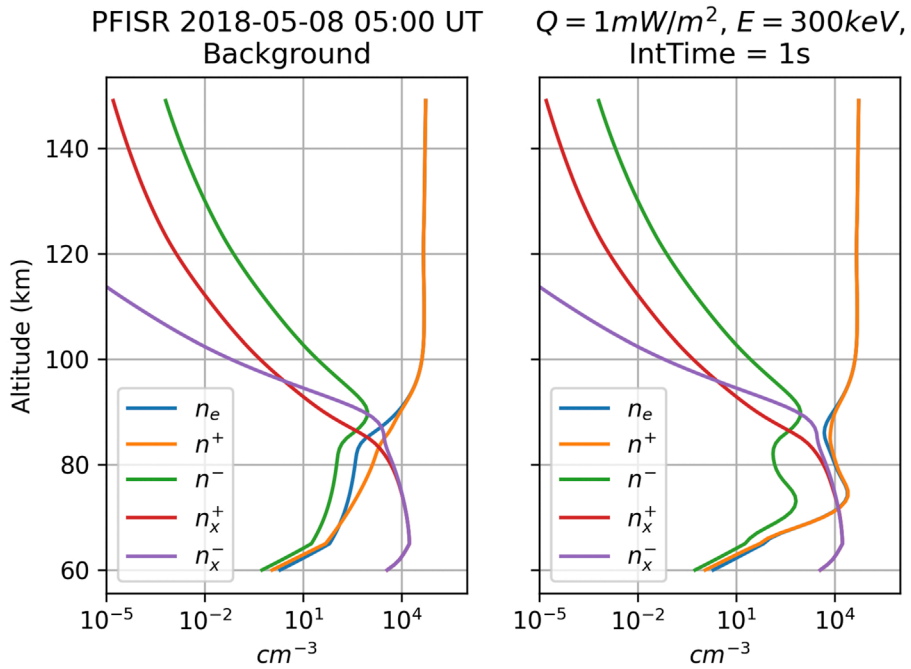


FIGURE 3
The altitude profiles of the positive and negative heavy and light ions, and the electrons. The left column shows the altitude profiles for the background, corresponding to IRI, at PFISR on 08 May 2018 at 0500 UT. The right panel shows the same time interval, but now including a monoenergetic beam with energy 300 keV and an energy flux of 1 mW/m^2. The equations are integrated for 1 s.

the background versus interval with particle precipitation. However, it is important to note that the cluster ions have effectively equal and opposite total charge, so these species do not impact the electron density or the light ions. The lack of difference is due to the coefficients being fixed by the empirical specification of the neutral mass density, which does not evolve in time.

Figure 4 shows how the electron density varies as a function of integration time (end time) for three monoenergetic beam energies. Each column shows a monoenergetic beam with an energy flux of 1 mW/m^2 but different energies corresponding to 300 keV, 50 keV, and 10 keV in the left, middle, and right columns, respectively. We chose a set of integration times to show the evolution and 300s (5 min) being a typical time interval associated with ISR experiments. For the 300 keV energy, the peak altitude moves up from 75 km to 80 km between 10 s and 60 s. We also find that 60 s of integration has reached a steady-state configuration, since there is little difference between the electron density profile at 60 s or 300 s. We find in all cases that going from 1 s to 5 s of integration seems to change the electron density profile significantly. For the 50 keV beam, steady-state has nearly been achieved between 5 and 10 s, and similarly for the 10 keV beam; one conclusion we can draw is that 10 s is nearly steady-state for energies above 100 keV. The peak for the 50 keV and 10 keV beams occurs at just below 90 km and near 100 km altitude, respectively.

4 Discussion and applications

We present recent examples that have used the GPI5 formulation. Marshall et al. (2014) and Marshall et al. (2019) quantify ground-based observables (i.e., electron density, photons, etc.) caused by a beam of energetic electrons driven by a linear accelerator. Both papers modeled optical emission, X-rays, and electron density enhancements that could be measured with ISR. Marshall and Cully (2020) also modeled subionospheric VLF signatures that would be modified by energetic particle precipitation. The GPI5 framework provides ion and electron densities which can be used to calculate light emission, VLF propagation, or used directly, as in the case of ISR.

Sanchez et al. (2022) presents an observational study of energetic particle precipitation observed with the Poker Flat Incoherent Scatter Radar for an event on 8 May 2018. During this event there were near conjunctions (< 500 km) between the Van Allen Probes satellites A and B and PFISR, thus enabling the electromagnetic wave fields to be observed nearly simultaneously relative to ground-based observations of the precipitation.

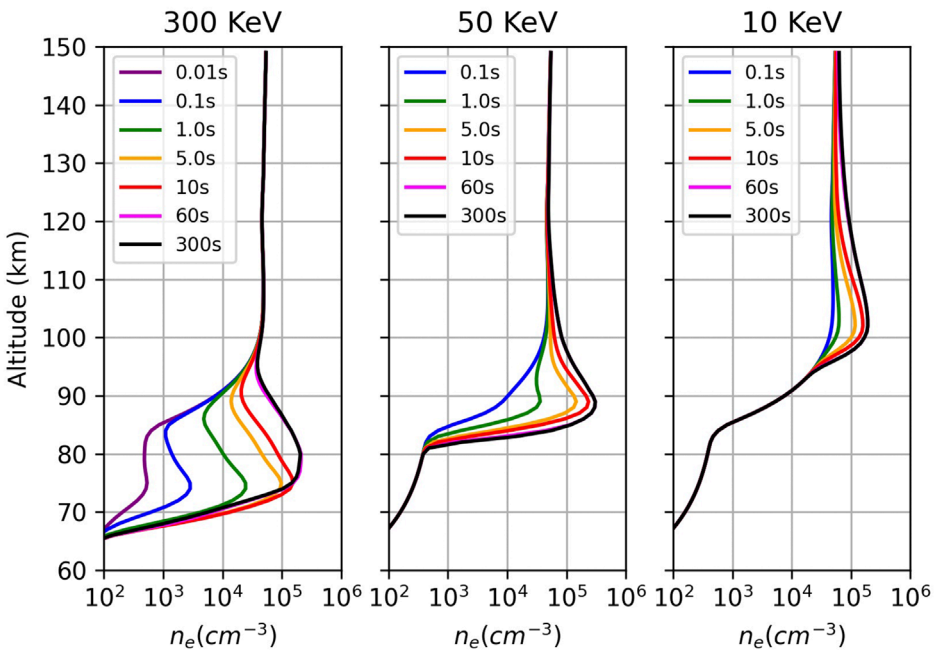


FIGURE 4
The electron density vs. altitude is plotted to show the relaxation time for a monoenergetic flux at 300 keV, 50 keV, and 10 keV shown as the left, middle, and right columns, respectively. All examples have an energy flux of 1 mW/m^2. We find that the 50 keV and 10 keV beams come into near steady state after 10 s, while the 300 keV beam takes 60 s.

The satellite electromagnetic wave data were predominantly chorus waves, which were used in quasi-linear diffusion codes to produce enhanced pitch angle scatter diffusion rates and the loss cone population. The BERI code was then used to produce observable signatures which were compared with the PFISR observations of electron density. Ma et al. (2022) presents similar results to Sanchez et al. (2022), but instead using near conjunctions between the Arase satellite and PFISR. In this case, Ma et al. (2022) found cases where plasmaspheric hiss was generating energetic precipitation that was observed with PFISR.

Bernhardt et al. (2022) demonstrated how rocket motors can impact the local plasma environment that can energize ions. The energized ions cause significant enhancement in VLF emission, which enhance wave particle interactions and thus generate energetic particle precipitation. Bernhardt et al. (2022) used the SIC model to model the D-region ionospheric response as a result of the energetic precipitation, which is detectable using ISR. They also modeled the impacts to subionospheric VLF propagation were also shown for a notional NLK transmitter to a receiver located in Dover DE.

Bernhardt et al. (2022) also calculated the impact that D-region electron density enhancements have on the absorption of high frequency radio wave propagation at 30 MHz, consistent with what one might expect to observe with a riometer. Recently, Zawdie et al. (2017) presented a review of methods for calculating the absorption rate of HF radio waves. **Eq. 3** from Zawdie et al. (2017) presents the complex part of the wavevector, $k = k_r + i\kappa$,

$$\kappa = \frac{e^2}{2\epsilon_0 mc}\frac{1}{\mu}\frac{n_e\nu_e}{\nu_e^2 + (2\pi f \pm \Omega_e cos\theta)^2} \tag{15}$$

where ν_e is the electron collision frequency, Ω_e is the electron cyclotron frequency, θ is the angle between k and the magnetic field B_0, and f is the frequency of the transmitter. The plus/minus sign corresponds to ordinary (O-mode) and extraordinary (X-mode) propagation, respectively. Deviate absorption corresponds to the case when $n_e\nu_{en}$ is large, and clearly this case is achieved during energetic particle precipitation (Davies, 1990). The pyGPI5 model provides a suitable means to model the D-region electron density enhancements.

5 Summary and conclusion

We present an open source python code base, pyGPI5, designed to quantify the vertical profile of light and heavy positive and negative ions and electrons in the D- and E-region of the ionosphere. An important distinction of this model is that the ions and electrons are treated as single fluids and the chemical reaction rates are represented by effective rates. The formulation in the pyGPI5 model is in contrast to the Sodankylä Ion Chemistry model, which tracks the time evolution of specific

ions. pyGPI5 solves five simultaneous coupled stiff ordinary differential equations, corresponding to heavy and light positive and negative ions, plus electrons. However, pyGPI5 does not solve differential equations describing the time evolution of the neutral species.

We present an overview of the code architecture, including the two dominant python classes that comprise the code. The `Chemistry` class solves the time evolution of the light and heavy ions and electrons. The reaction rates used in the code are presented. The `Ionization` class generates an altitude ionization profile which is a necessary input for the five differential equations and can change depending on the driving precipitation. For the ionization profiles, we implemented the Fang et al. (2008) and Fang et al. (2010) models for Maxwellian and monoenergetic beams, respectively. A future step will be to implement the Boulder Electron Radiation to Ionization model that describes the electron precipitation. Other ionospheric models, neutral atmospheric models, and transport codes could be implemented within the framework of pyGPI5.

We present results showing different model outputs. The figures presented in the results section are contained within a Jupyter notebook that is on the github page. We show how to instantiate the model, generate the altitude ion and electron profiles for background conditions, how to generate altitude ionization profiles, and how to generate the ion and electron profiles corresponding to energetic particle precipitation. We also show how the peak electron density changes for characteristic relaxation times. We present recent results that have used D-region chemistry models and including a few applications that require a D-region chemistry model, such as, absorption of high frequency radio waves.

In conclusion, we present the pyGPI5 model and the source code as a straight-forward model to generate D- and E-region ion and electron profiles as a function of altitude. We have additionally presented assumptions and caveats that the user should be aware of when running this code for a given application. We also emphasize once again that this model does have some significant simplifications and if the topics of interest require an understanding of the evolution of a specific ion species, other models are better suited to address those needs, such as the Sodankylä Ion Chemistry model.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: [https://github.com/srkaeppler/pyGPI5].

Author contributions

SK drafted the text and the figures presented in this investigation. RM and DJ provided the original GPI code which was translated to python and suggested code edits for the current version. ES was the PI of the NSF grant that partially funded this work and has provided useful input to the manuscript. AJ and RT have provided important code edits and useful input to the manuscript.

Funding

ES, RM, and SK were funded by NSF Collaborative Research Grant AGS–1732365. SK was additionally funded by Air Force Office of Scientific Research Grant FA9550-19-1-0130.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Bernhardt, P. A., Hua, M., Bortnik, J., Ma, Q., Verronen, P. T., McCarthy, M. P., et al. (2022). Active precipitation of radiation belt electrons using rocket exhaust driven amplification (reda) of man-made whistlers. *J. Geophys. Res. Space Phys.* 127, e2022JA030358. doi:10.1029/2022ja030358

Bilitza, D., Altadill, D., Truhlik, V., Shubin, V., Galkin, I., Reinisch, B., et al. (2017). International reference ionosphere 2016: From ionospheric climate to real-time weather predictions. *Space weather.* 15, 418–429. doi:10.1002/2016SW001593

Brekke, A. (2013). *The ionosphere.* Berlin, Heidelberg: Springer Berlin Heidelberg.

Cummer, S. A., Bell, T. F., Inan, U. S., and Chenette, D. L. (1997). Vlf remote sensing of high-energy auroral particle precipitation. *J. Geophys. Res.* 102, 7477–7484. doi:10.1029/96JA03721

Davies, K. (1990). *Ionospheric radio. Electromagnetic waves.* London, United Kingdom: The Institution of Engineering and Technology.

Fang, X., Randall, C. E., Lummerzheim, D., Solomon, S. C., Mills, M. J., Marsh, D. R., et al. (2008). Electron impact ionization: A new parameterization for 100 ev to 1 mev electrons. *J. Geophys. Res.* 113, n/a. doi:10.1029/2008JA013384. A09311

Fang, X., Randall, C. E., Lummerzheim, D., Wang, W., Lu, G., Solomon, S. C., et al. (2010). Parameterization of monoenergetic electron impact ionization. *Geophys. Res. Lett.* 37, n/a. doi:10.1029/2010gl045406

Florescu-Mitchell, A., and Mitchell, J. (2006). Dissociative recombination. *Phys. Rep.* 430, 277–374. doi:10.1016/j.physrep.2006.04.002

Gledhill, J. A. (1986). The effective recombination coefficient of electrons in the ionosphere between 50 and 150 km. *Radio Sci.* 21, 399–408. doi:10.1029/RS021i003p00399

Glukhov, V. S., Pasko, V. P., and Inan, U. S. (1992). Relaxation of transient lower ionospheric disturbances caused by lightning-whistler-induced electron precipitation bursts. *J. Geophys. Res.* 97, 16971–16979. doi:10.1029/92JA01596

Hendry, A. T., Clilverd, M. A., Rodger, C. J., and Engebretson, M. J. (2020). "Chapter 8 - ground-based very-low-frequency radio wave observations of energetic particle precipitation," in *The dynamic loss of earth's radiation belts*. Editors A. N. Jaynes, and M. E. Usanova (Elsevier), 257–277. doi:10.1016/B978-0-12-813371-2.00008-1

Kaeppler, S. R., Sanchez, E., Varney, R. H., Irvin, R. J., Marshall, R. A., Bortnik, J., et al. (2020). "Chapter 6 - incoherent scatter radar observations of 10–100kev precipitation: Review and outlook," in *The dynamic loss of earth's radiation belts*. Editors A. N. Jaynes, and M. E. Usanova (Elsevier). doi:10.1016/B978-0-12-813371-2.00006-8

Kovács, T., Plane, J. M. C., Feng, W., Nagy, T., Chipperfield, M. P., Verronen, P. T., et al. (2016). *D*-region ion–neutral coupled chemistry (Sodankylä Ion Chemistry, SIC) within the whole atmosphere community climate model (WACCM 4) – WACCM-SIC and WACCM-rSIC. *Geosci. Model Dev.* 9, 3123–3136. doi:10.5194/gmd-9-3123-2016

Kozlov, S. I., Vlaskov, V. A., and Smirnova, N. V. (1988). Speicalized aeronomic model for investigating artifical midification of the middle atmosphere and lower ionosphere. 1. requires for the model and basic concepts of its formation. *Cosm. Res. Engl. Transl.* 26, 635–642.

Lehtinen, N. G., and Inan, U. S. (2007). Possible persistent ionization caused by giant blue jets. *Geophys. Res. Lett.* 34, n/a. doi:10.1029/2006gl029051

Lehtinen, N. G., Bell, T. F., and Inan, U. S. (1999). Monte Carlo simulation of runaway MeV electron breakdown with application to red sprites and terrestrial gamma ray flashes. *J. Geophys. Res.* 104, 24699–24712. doi:10.1029/1999JA900335

Ma, Q., Xu, W., Sanchez, E. R., Marshall, R. A., Bortnik, J., Reyes, P. M., et al. (2022). A test of energetic particle precipitation models using simultaneous incoherent scatter radar and van allen probes observations. *J. Geophys. Res. Space Phys.* 127, e2021JA030179. e2022JA030545. doi:10.1029/2021JA030179

Marshall, R. A., and Cully, C. M. (2020). "Chapter 7 - atmospheric effects and signatures of high-energy electron precipitation," in *The dynamic loss of earth's radiation belts*. Editors A. N. Jaynes, and M. E. Usanova (Elsevier). doi:10.1016/B978-0-12-813371-2.00007-X

Marshall, R. A., Nicolls, M., Sanchez, E., Lehtinen, N. G., and Neilson, J. (2014). Diagnostics of an artificial relativistic electron beam interacting with the atmosphere. *J. Geophys. Res. Space Phys.* 119, 8560–8577. doi:10.1002/2014JA020427

Marshall, R. A., Xu, W., Kero, A., Kabirzadeh, R., and Sanchez, E. (2019). Atmospheric effects of a relativistic electron beam injected from above: Chemistry, electrodynamics, and radio scattering. *Front. Astron. Space Sci.* 6. doi:10.3389/fspas.2019.00006

Marshall, R. A., Xu, W., Woods, T., Cully, C., Jaynes, A., Randall, C., et al. (2020). The aepex mission: Imaging energetic particle precipitation in the atmosphere through its bremsstrahlung x-ray signatures. *Adv. Space Res.Advances Small Satell. Space Sci.* 66, 66–82. doi:10.1016/j.asr.2020.03.003

Miyoshi, Y., Oyama, S., Saito, S., Kurita, S., Fujiwara, H., Kataoka, R., et al. (2015). Energetic electron precipitation associated with pulsating aurora: Eiscat and van allen probe observations. *J. Geophys. Res. Space Phys.* 120, 2754–2766. doi:10.1002/2014ja020690

Narcisi, R. S., and Bailey, A. D. (1965). Mass spectrometric measurements of positive ions at altitudes from 64 to 112 kilometers. *J. Geophys. Res.* 70, 3687–3700. doi:10.1029/JZ070i015p03687

Picone, J. M., Hedin, A. E., Drob, D. P., and Aikin, A. C. (2002). NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *J. Geophys. Res.* 107, SIA 15-21–SIA 15-16. doi:10.1029/2002JA009430

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (2007). *Numerical recipes 3rd edition: The art of scientific computing. Numerical recipes: The art of scientific computing*. New York, NY: Cambridge University Press.

Richards, P. G., Bilitza, D., and Voglozin, D. (2010). Ion density calculator (idc): A new efficient model of ionospheric ion densities. *Radio Sci.* 45, n/a. doi:10.1029/2009rs004332

Rishbeth, H., and Garriott, O. K. (1969). *Introduction to ionospheric physics*. Academic Press: New York, NY.

Robinson, R. M., Vondrak, R. R., Miller, K., Dabbs, T., and Hardy, D. (1987). On calculating ionospheric conductances from the flux and energy of precipitating electrons. *J. Geophys. Res.* 92, 2565–2569. doi:10.1029/JA092iA03p02565

Sanchez, E. R., Ma, Q., Xu, W., Marshall, R. A., Bortnik, J., Reyes, P., et al. (2022). A test of energetic particle precipitation models using simultaneous incoherent scatter radar and van allen probes observations. *J. Geophys. Res. Space Phys.* 127, e2021JA030179. doi:10.1029/2021ja030179

Schunk, R. W., and Nagy, A. F. (2004). *Ionospheres*. New York, NY: Cambridge University Press.

Semeter, J., and Kamalabadi, F. (2005). Determination of primary electron spectra from incoherent scatter radar measurements of the auroralEregion. *Radio Sci.* 40, RS2006. doi:10.1029/2004RS003042

Simon Wedlund, C., Lamy, H., Gustavsson, B., Sergienko, T., and BräNdströM, U. (2013). Estimating energy spectra of electron precipitation above auroral arcs from ground-based observations with radar and optics. *J. Geophys. Res. Space Phys.* 118, 3672–3691. doi:10.1002/jgra.50347

Sivadas, N., Semeter, J., Nishimura, Y., and Kero, A. (2017). Simultaneous measurements of substorm-related electron energization in the ionosphere and the plasma sheet. *JGR. Space Phys.* 122, 547. doi:10.1002/2017ja023995

Turunen, E., Verronen, P. T., Seppälä, A., Rodger, C. J., Clilverd, M. A., Tamminen, J., et al. (2009). Impact of different energies of precipitating particles on nox generation in the middle and upper atmosphere during geomagnetic storms. *J. Atmos. Sol. Terr. Phys.High Speed Sol. Wind Streams Geosp. Interact.* 71, 1176–1189. doi:10.1016/j.jastp.2008.07.005

Turunen, E., Kero, A., Verronen, P. T., Miyoshi, Y., Oyama, S.-I., and Saito, S. (2016). Mesospheric ozone destruction by high-energy electron precipitation associated with pulsating aurora. *JGR. Atmos.* 121, 11. doi:10.1002/2016jd025015

Verronen, P. T., SeppäLä, A., Clilverd, M. A., Rodger, C. J., KyröLä, E., Enell, C.-F., et al. (2005). Diurnal variation of ozone depletion during the October-November 2003 solar proton events. *J. Geophys. Res.* 110, A09S32. doi:10.1029/2004JA010932

Verronen, P. T., Andersson, M. E., Marsh, D. R., Kovács, T., and Plane, J. M. C. (2016). Waccm-d—Whole atmosphere community climate model with d-region ion chemistry. *J. Adv. Model. Earth Syst.* 8, 954–975. doi:10.1002/2015MS000592

Vickrey, J. F., Vondrak, R. R., and Matthews, S. J. (1982). Energy deposition by precipitating particles and Joule dissipation in the auroral ionosphere. *J. Geophys. Res.* 87, 5184–5196. doi:10.1029/JA087iA07p05184

Xu, W., Marshall, R. A., Tyssøy, H. N., and Fang, X. (2020). A generalized method for calculating atmospheric ionization by energetic electron precipitation. *J. Geophys. Res. Space Phys.* 125, e2020JA028482. doi:10.1029/2020ja028482

Zawdie, K. A., Drob, D. P., Siskind, D. E., and Coker, C. (2017). Calculating the absorption of hf radio waves in the ionosphere. *Radio Sci.* 52, 767–783. doi:10.1002/2017RS006256

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 07 December 2022
DOI 10.3389/fspas.2022.1023163

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Reinaldo Roberto Rosa,
National Institute of Space Research (INPE), Brazil
Carl-Fredrik Enell,
European Incoherent Scatter Scientific Association, Sweden

*CORRESPONDENCE
Lei Cai,
lei.cai@oulu.fi

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 19 August 2022
ACCEPTED 14 November 2022
PUBLISHED 07 December 2022

CITATION
Cai L, Aikio A, Kullen A, Deng Y, Zhang Y, Zhang S-R, Virtanen I and Vanhamäki H (2022), GeospaceLAB: Python package for managing and visualizing data in space physics.
Front. Astron. Space Sci. 9:1023163.
doi: 10.3389/fspas.2022.1023163

COPYRIGHT
© 2022 Cai, Aikio, Kullen, Deng, Zhang, Zhang, Virtanen and Vanhamäki. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

GeospaceLAB: Python package for managing and visualizing data in space physics

Lei Cai[1]*, Anita Aikio[1], Anita Kullen[2], Yue Deng[3], Yongliang Zhang[4], Shun-Rong Zhang[5], Ilkka Virtanen[1] and Heikki Vanhamäki[1]

[1]Space Physics and Astronomy, University of Oulu, Oulu, Finland, [2]Space Plasma Physics, KTH Royal Institute of Technology, Stockholm, Sweden, [3]Department of Physics, University of Texas at Arlington, Arlington, TX, United States, [4]The Johns Hopkins University Applied Physics Laboratory, Laurel, MD, United States, [5]Haystack Observatory, Massachusetts Institute of Technology, Westford, MA, United States

In the space physics community, processing and combining observational and modeling data from various sources is a demanding task because they often have different formats and use different coordinate systems. The Python package GeospaceLAB has been developed to provide a unified, standardized framework to process data. The package is composed of six core modules, including *DataHub* as the data manager, *Visualization* for generating publication quality figures, *Express* for higher-level interfaces of *DataHub* and *Visualization*, *SpaceCoordinateSystem* for coordinate system transformations, *Toolbox* for various utilities, and *Configuration* for preferences. The core modules form a standardized framework for downloading, storing, post-processing and visualizing data in space physics. The object-oriented design makes the core modules of GeospaceLAB easy to modify and extend. So far, GeospaceLAB can process more than twenty kinds of data products from nine databases, and the number will increase in the future. The data sources include, e.g., measurements by EISCAT incoherent scatter radars, DMSP, SWARM, and Grace satellites, OMNI solar wind data, and GITM simulations. In addition, the package provides an interface for the users to add their own data products. Hence, researchers can easily collect, combine, and view multiple kinds of data for their work using GeospaceLAB. Combining data from different sources will lead to a better understanding of the physics of the studied phenomena and may lead to new discoveries. GeospaceLAB is an open source software, which is hosted on GitHub. We welcome everyone in the community to contribute to its future development.

KEYWORDS

Python, data management, space physics, space weather, aurora, solar wind, magnetosphere-ionosphere-thermosphere coupling

1 Introduction

A space physics research project is typically based on several kinds of measured and/or modeling data. Those data are often provided by different institutes or research groups. The data providers prepare and store their data in various file formats, such as ASCII, CSV, CDF, NetCDF, and HDF5. Even when using the same file format, different data can be documented with different data structures. The diversity of the data formats and data sources adds an unnecessary complexity to the data analysis in a research project. It often takes a lot of time for a researcher to collect and manage the data before the data are processed for a further analysis and interpretation. To improve the productivity in space physics research, we develop the Python package GeospaceLAB. We aim to establish a standardized process for data access, management, and visualization that connects the data provider and the space physics researchers. Using our package, researchers can promote their research in a quick manner and focus more on the data interpretation and research results.

Python has become the fastest-growing programming language in scientific research during the last 2 decades. The language is powerful, intuitive to learn, and easy to use. GeospaceLAB takes advantage of the programming language Python and its built-in and external open-source packages. In GeospaceLAB, we mainly adapt object-oriented programming (OOP) to construct the core modules. OOP is a programming paradigm that wraps associated properties and behaviors into individual objects (e.g., Luciano, 2015). For example, an object could represent a car with properties like a brand, model, and production year and behaviors such as starting, driving back and forth, as well as braking. Different cars have different properties and behaviors. With OOP, we build the core modules of GeospaceLAB from standard working classes. The use of OOP improves the code's reusability and quality. OOP also makes the package more extensible and lowers the maintenance costs. In addition to OOP, GeospaceLAB provides a collection of functions as utilities to support the development of the package. The classes and utility functions in the package are open for both developers and users.

The various Python packages make it possible to accomplish multiple tasks using Python alone, instead of using different programming languages or software. The development of GeospaceLAB is dependent on a number of open-source packages/scripts (so called dependencies). For example, GeospaceLAB utilizes the dependencies such as *Requests*[1], *Beautiful Soup 4*[2] and *Ftplib*[3] for pulling the data from online sources, *Pathlib*[4] for managing local file system, *Re*, *NetCDF4*[5], and *H5Py*[6] for file input or output (I/O), *NumPy* (Harris et al., 2020) for manipulating data arrays, *SciPy* (Virtanen et al., 2020) for scientific computing, and *Matplotlib* (Hunter, 2007; Caswell et al., 2022) and *Cartopy* (Met Office, 2010 - 2015) for data visualization and mapping. In **Figure 1** those dependencies are listed separately for each of the GeospaceLAB core modules.

In the space physics community, many Python packages have been developed during the last decade for data access and analysis (e.g., Burrell et al., 2018, for a recent review). GeospaceLAB uses several useful packages for building the core modules, including *CDFLIB*[7] for the reading/writing Computable Document Format (CDF) files, *MadrigalWeb*[8] and *SuperMAG Client* for accessing Madrigal and SuperMAG databases, respectively, *SSCWS*[9] for satellite orbit tracking, *AACGMV2*[10] (Shepherd, 2014), *GEOPACK*[11] (Tsyganenko et al., 2021), and *APEXPY*[12] (Richmond, 1995; Emmert et al., 2010) for coordinate system transformation.

Thanks to the existing packages, GeospaceLAB can complete multiple tasks that are required for a scientific data analysis. GeospaceLAB integrates the needed functionality of the dependencies, such that the users do not need to have a detailed knowledge of all dependencies. However, some knowledge about Python basics and the packages such as *Numpy*, *SciPy*, and *Matplotlib* will help the users to understand the data structure and workflows in the package.

In this paper, we present the current public release (v0.5.2) of GeospaceLAB. We introduce the design of GeospaceLAB's core modules and their functionality in **Section 2**. We present a few possible applications of the package in space physics research in **Section 3**. The current status, issues and future plans are described in **Section 4**. This paper is not intended to provide a detailed documentation of the package, but is rather intended to give an overview of the functionality and design. A full documentation of the package can be found online[13].

1 https://github.com/psf/requests.
2 https://www.crummy.com/software/BeautifulSoup/bs4/doc/.
3 https://docs.python.org/3/library/ftplib.html.
4 https://docs.python.org/3/library/pathlib.html.
5 https://github.com/Unidata/netcdf4-python.
6 https://www.h5py.org/.
7 https://github.com/MAVENSDC/cdflib.
8 http://cedar.openmadrigal.org/docs/name/rr_python.html.
9 https://sscweb.gsfc.nasa.gov/WebServices/REST/py/sscws/sscws.html.
10 https://github.com/aburrell/aacgmv2.
11 https://github.com/tsssss/geopack.
12 https://github.com/aburrell/apexpy.
13 https://geospacelab.readthedocs.io/en/latest/.

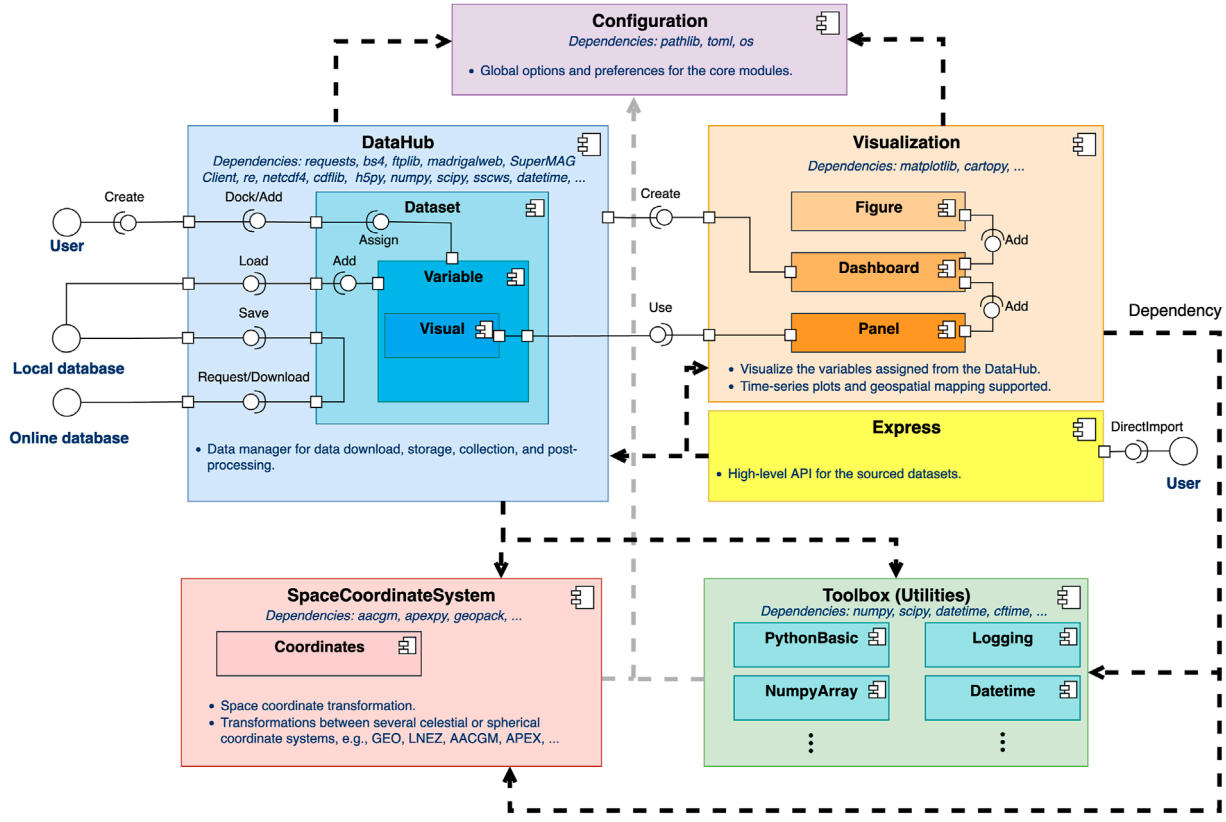


FIGURE 1
A UML component diagram showing the core modules of GeospaceLAB and their interdependence.

2 Software design

The modular design is applied in the development of GeospaceLAB. The package is composed of six core modules, including DataHub (`geospacelab.datahub`), Visualization (`geospacelab.visualization`), Express (`geospacelab.express`), SpaceCoordinateSystem (`geospacelab.cs`), Toolbox (`geospacelab.toolbox`), and *Configuration* (`geospacelab.config`). **Figure 2** shows the component diagram generated in Unified Modeling Language (UML) (Engels et al., 2000) for the six core modules. The diagram describes briefly the structure of the package and the relationship among the core modules. *DataHub* is the data manager in GeospaceLAB. This module is used to control the workflow when processing data and to manage the data and its properties from various data sources. *Visualization* controls the plotting process for the data managed by *DataHub*. As shown in **Figure 1**, users should first create a `DataHub` or `Dashboard` object for the research project. In case of a `DataHub` object, users can dock/add one or more dataset objects. A dataset object is a collection of variables which are loaded from data files. The dataset object also controls the procedure of downloading, storing, loading, and post-processing of a specific data product. Users can make publication-quality plots using the Dashboard object. *Express* contains several high-level interfaces based on *DataHub* and *Visualization*. With the high-level interfaces, users can directly obtain data and view quicklook plots from a specific data source. *SpaceCoordinateSystem* manages the coordinate transformation to commonly used space coordinate systems. *Toolbox* is a library of functions that supports other modules in GeospaceLAB. Both *SpaceCoordinateSystem* and *Toolbox* support the workflows in *DataHub*, *Visualization*, and *Express*. Finally, *Configuration* manages the preferences and global parameters used in the package. In the following subsections, we will introduce the core modules in more detail.

2.1 DataHub as the data manager (`geospacelab.datahub`)

DataHub is the most essential module in the package. The module is composed of several base classes and their subclasses. **Figure 2** shows the main structure of the modules *DataHub*,

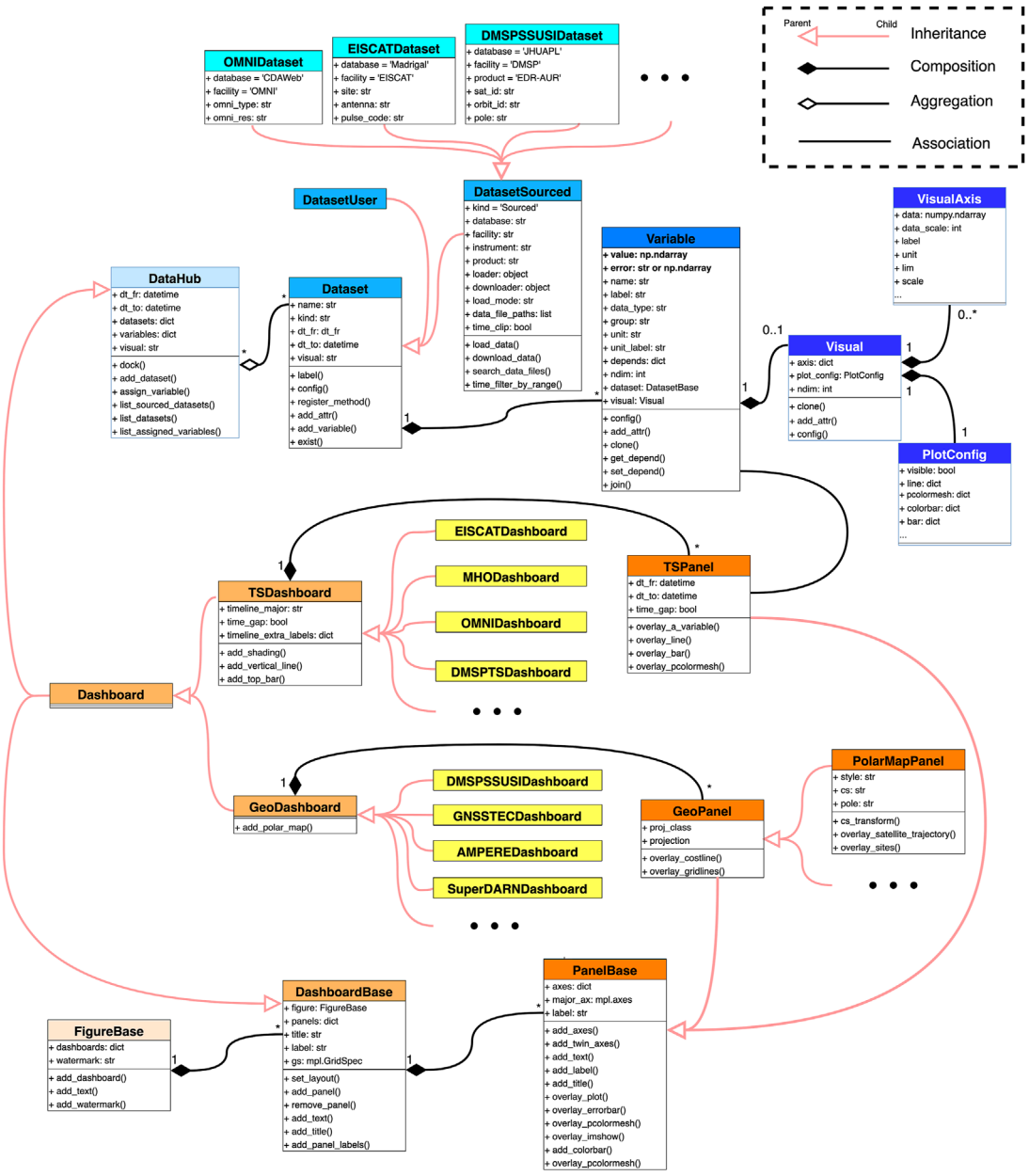


FIGURE 2
A UML class diagram that describes the main structure of the three core modules DataHub (blue color shades), Visualization (orange), and Express (yellow) in GeospaceLAB.

Visualization, and *Express* in a UML class diagram. The classes included in the module *DataHub* are marked by blue shades. Each class is illustrated with three compartments: the upper compartment shows the name of the class, the middle contains the attributes, and the lower contains the methods owned by the class. Note that the attributes and methods of a class are not fully listed in figure. A full description can be found in the online documentation.

In the family of DataHub classes, `DataHub` (with the same name as the module), `Dataset`, and `Variable` are the three core classes. The class `DataHub` is the top-level manager that governs the global properties and behavior of a project and the dataset objects that are included. The class Dataset is the middle-level manager that controls the workflow for a specific data product and manages the data variables in that data product. The class `Variable` is the base-level manager that

```
# Step 1: Import the required packages and modules
import datetime
from geospacelab.datahub import DataHub
# Step 2: Set the starting and stopping times
dt_fr = datetime.datetime(2015, 3, 16, 12)  # from
dt_to = datetime.datetime(2015, 3, 19, 12)  # to
# Step 3: Create a DataHub object
dh = DataHub(dt_fr=dt_fr, dt_to=dt_to)
# Step 4: Dock the sourced datasets
ds_omni = dh.dock(
    datasource_contents=['cdaweb', 'omni'],
    omni_type='OMNI2', omni_res='1min',
    load_mode='AUTO', allow_load=True)
# Step 5: Get the variable and
# extract the data array from the variable.
b_x = ds_omni['B_x_GSM']
print(b_x.value)
```

Listing 1. An example code with comments for retrieving OMNI data from 1200UT on 16 March 2015 to 1200 UT on 19 March 2015. The data array of the interplanetary magnetic field (IMF) B_x component is printed in the final.

manages a specific data variable and its own properties in a data product. A `DataHub` object can be compared with the core module of a modular space station. It can dock or add one or more `Dataset` objects like a space station core module docks different types of modules. Each `Dataset` object owns multiple `Variable` objects, just like various devices in a space station module.

Listing 1 shows a code example with comments to help users to understand the basic workflow in using a DataHub object. The code retrieves OMNI data using five steps by 1) importing the required module, 2) setting the time bounds (`dt_fr` and `dt_to`), 3) creating a `DataHub` object (assigned to `dh`), 4) docking the OMNI dataset with several inputs (assigned to `ds_omni`), and 5) getting a variable from the docked dataset (assigned to `B_x`) and extracting the data array from `B_x.value`. Only a few inputs (keyword arguments) are required in steps 3 and 4. The OMNI dataset object `ds_omni` is responsible for the query and retrieval of the OMNI data product. The OMNI data will be downloaded automatically, if the data files do not exist in the local directory of a user's computer. The above five steps are the standard procedures for users to obtain the data from one or more data sources. The code is simple, benefiting from the functionality of the three core classes and their inheritances.

2.1.1 Variable

To demonstrate the framework of the data management in *DataHub*, first we introduce the core class `Variable` which is the base-level manager. In space physics, a variable from observations or simulations is often expressed or recorded by a numeric array with associated properties. The variable properties include the name, unit, description, dependencies, and errors. The class `Variable` provides a data model to record both data array and common properties for a physical quantity. Meanwhile, additional properties can be added flexibly when needed.

The most commonly used attributes of a `Variable` object have been listed in **Figure 2**, including `value`, `ndim`, `error`, `name`, `label`, `data_type`, `group`, `unit`, `unit_label`, `depends`, and `visual`. The attribute value is used to record a data array in the format `numpy.ndarray`. A data array may have one or more dimensions. The attribute ndim returns the number of dimensions. The attributes `name`, `unit`, `data_type`, `group` record the basic properties of a variable as described in the attribute names. The attributes `label` and `unit_label` usually have the form of a Python raw string for the text rendering with mathematics expression. They are useful when shown with the plots.

Frequently, the measurement/calculation errors are provided with the data array in space sciences. The attribute `error` is used to record these error values. The attribute can be assigned with either the `numpy.ndarray` object or a string (Python `str` object). In case of the string, the string value points to the error variable stored in the associated dataset object.

Any data arrays in space sciences should depend on one or more support data along a specific axis. The support data usually indicates the time, spatial position or any quantity indirectly connected to the measurement/calculation. For example, the `Variable` object `B_x` in **Listing 1** is assigned by an 1-dimensional (1-D) array of the IMF B_x component with associated properties. The array depends on the universal time (UT) along the axis-0 (0 denotes the first dimensional axis, 1 the second, and 2 the third hereafter). Sometimes, a variable may have multiple dependencies along one axis. For instance, the electron density measured from a low-earth-orbit (LEO) satellite is a 1-D time-series variable, which depends on UT, geographic latitude and longitude, and geomagnetic latitude and longitude along the axis-0. The attribute `depends` manages the dependencies of the variable in multiple dimensions and multiple dependencies along one dimensional axis. A `Variable` object uses the methods `set_depend()` and `get_depend()` to set up and obtain the dependencies, respectively. Similar as `error`, the values of dependencies in depends can be assigned with a `numpy.ndarray` object or a string. In case of a string, the string value points to the supporting variable stored in the associated dataset object.

In case that a visualization is needed, the attribute `visual` will be assigned with a `Visual` object (see also **Figure 2**). Note that by default, the `DataHub` object sets its attribute visual with the `value` of `"off"`, meaning that the `Visual` objects will not be assigned to all variables. The class `Visual` manages the visualization properties for a variable with two compositions: `VisualAxis` and `PlotConfig`. A `VisualAxis` object controls data and options along a specific axis when plotting. Hence, one `Visual` object may have multiple `VisualAxis` objects, which are assigned to the `Visual` attribute `axis`. In addition, one `Visual` object owns one `PlotConfig` object, which controls the plotting type and the corresponding configuration.

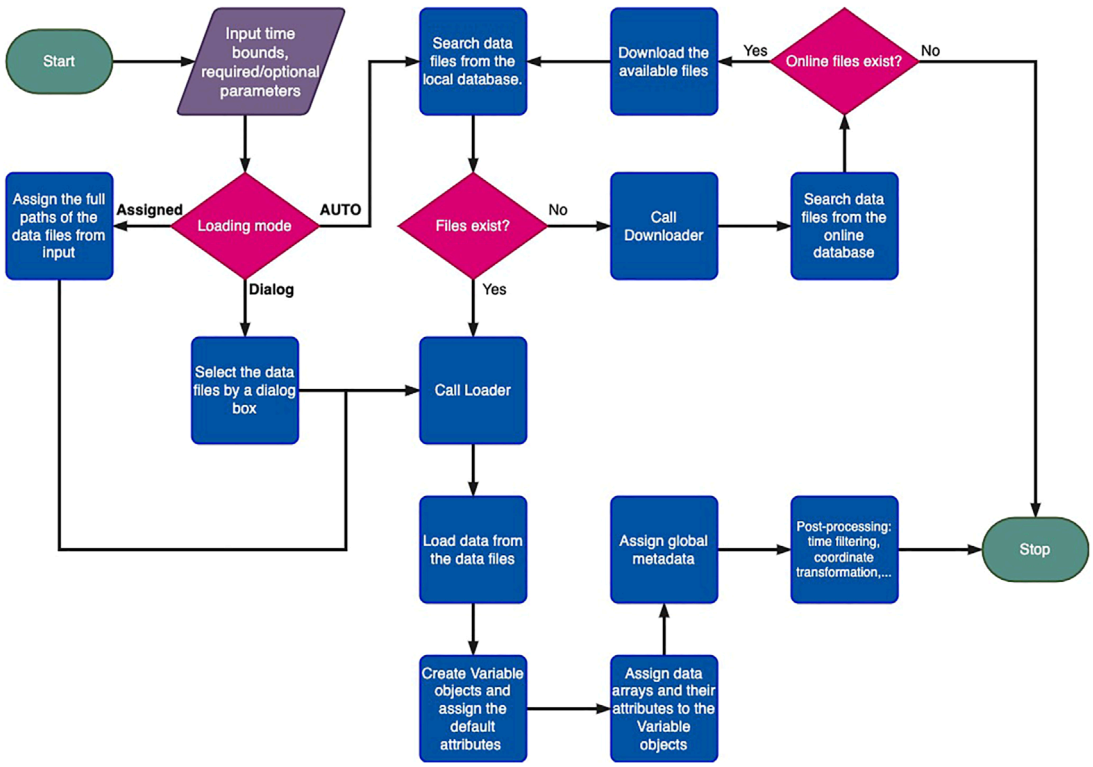


FIGURE 3
A flowchart showing the main process of downloading, storing, and loading data managed by a `DataSourced` object.

In practice, it is not necessary to set all the attributes when creating or using a `Variable` object. For a sourced dataset (see **Section 2.1.2.1**), the attributes have been assigned as default properties for a specific variable in the data product. On the other hand, the various attributes of `Variable` and `Visual` let users or developers customize as much as possible if the default settings do not suit their needs. The method `config ()` is used to set the multiple attributes that have been included in the classes, and `add_attr ()` to add additional attributes if needed. The same methods are also used for the `Dataset` objects as described next.

2.1.2 Dataset

The class `Dataset` is the middle-level manager in the module DataHub. It manages a collection of data variables and the global attributes in a data product. A `Dataset` object is a Python dictionary-like (`dict`) object, in which the keys are the variable name in the type of `str`, and the values are the correspond `Variable` objects. The `Dataset` object owns several basic attributes, such as `name` for the name of the dataset, kind for the type of the dataset (see also `DatasetSourced` and `DatasetUser` below), `dt_fr` for the starting time, `dt_to` for the stopping time, and `visual` for determining if the `Visual` objects are appended to the Variable objects. The `Dataset` also owns a series of methods, such as `add_variable ()` for adding a Variable object, `label ()` for automatically generating an identical label, `config ()` and `add_attr` for configuring the attributes, and `register_method ()` for adding an external method to the `Dataset` object.

The class `Dataset` has been extended into two subclasses (inheritances): `DatasetSourced` and `DatasetUser`. The inheritance means that the subclass owns the attributes and methods of its parent class. At the same time, the subclass has its private methods, which cannot be accessed by the parent class. The use of the inheritance is convenient for users and developers to reuse, extend, or to modify the attributes and behaviours in the parent class.

2.1.2.1 Managing a sourced dataset

The subclass `DatasetSourced` is designed to manage the data from a specific data source (so called sourced dataset) that have been included in the package. Besides the attributes of `Dataset`, `DatasetSourced` has its own attributes for identifying the properties of the specific data source, e.g., `database`, `facility`, `instrument`, and `product`. It also owns particular methods for managing the procedures of downloading, storing, and loading data, such as `load_data`

(), download_data (), and search_data_files () (see also Figure 2).

In DataHub, each sourced dataset has a corresponding subclass of DatasetSourced. As shown in Figure 2, the class OMNIDataset inherits from DatasetSourced with the assigned attributes database = "CDAWeb" and facility = "OMNI". Similarly, EISCATDataset, DMSPSSUSIDataset and many other subclasses have been developed for the sourced datasets. Each subclass uses the attributes and methods of DatasetSourced as the abstracts and has its own functionality.

Figure 3 shows the main process of querying, downloading, storing, and loading data by a DatasetSourced or its subclass object. The DatasetSourced object provides three modes to load a sourced dataset: "AUTO", "dialog", and "assigned". One of the values is assigned to the DatasetSourced attribute load_mode (see also Figure 2). For the mode "AUTO", the DatasetSourced object try to finish the entire process shown in Figure 3 automatically. First, it searches the associated data files that have been stored in the local directories. This procedure is managed by the method search_data_files (). The method provides a solution to collect all data files within the time bound between dt_fr and dt_to by identifying the string patterns in the file names. The rules for the string patterns are usually defined in the specific dataset class. As a result, the data arrays that are finally loaded in the dataset are independent of how many segmented files are included. If the associated data files are not found in the local directory, a downloading procedure will be activated. The downloading procedure is implemented by a Downloader class appended to DatasetSourced. The Downloader object will search the requested files in the online database. If the files exist, the object will download the files to the local directory/database. Again, the method search_data_files () will be called to collect those downloaded files and activate the loading procedure. Similar as the downloading procedure, the loading procedure is implemented by a Loader class appended to DataSourced. The Loader object will load data and metadata from the local files and pass them to the DatasetSourced object. The DatasetSourced object will first add the queried variable objects and assign the default attributes to those objects. Then the DatasetSourced object will assign the data arrays and associated metadata to the corresponding variable objects. Finally, several post-processing procedures may be done according to the input options, such as a time filter for clipping the data within the time bound, adding additional support data, making coordinate transformation, and controlling the data quality.

In GeospaceLAB, we aim to accomplish automatic downloading for most of the sourced datasets from online databases. For example, the package Requests is used for grabbing data from Hypertext Transfer Protocol (HTTP) web service, Ftplib for File Transfer Protocol (FTP) server, and several specialized packages/scripts provided by individual data services (e.g., MadrigalWeb and SuperMAG Client). We developed a family of Downloader classes to communicate with different online databases. Those sourced datasets with the same downloading mechanism can share one Downloader class. Alternatively, only a minor modification is applied by subclassing the parent Downloader class.

A similar arrangement for the Downloader classes is also applied for the family of Loader classes. So far, we have developed the Loader classes for loading data files with several kinds of formats, such as ASCII, CDF, NetCDF, HDF5, and binary files. Again, those Loader classes are highly reusable or extendable for various sourced datasets.

In case that the sourced dataset is not downloadable or only the local data files are available, users can utilize the loading modes "dialog" and "assigned' to select the local data files. When "dialog' is set, a dialog box will be activated. For the "assigned' mode, users can assign the full paths of the data files to the attribute data_file_paths of the DatasetSourced object.

So far, GeospaceLAB (current version: v0.5.2) supports more than twenty kinds of sourced datasets from nine databases. Table 1 lists those datasets with the associated properties. Most of the datasets can manage the procedure of automatically downloading ("Downloadable" is True). For a few datasets that are not downloadable, the users can download the data manually from the online services, e.g., the SuperDARN electric potential map data in the ASCII format and the AMEPRE Field-aligned current maps in the NetCDF format, or request from the data provider, e.g., the GITM model results from the research group at University of Texas, at Arlington (UTA). The data in all the listed datasets are be collected continuously within the input time bound. Even if the data is stored in separate files, the associated DatasetSourced object can detect and collect all associated data files automatically. Several datasets own their high-level dashboards for directly visualizing the data. Those dashboards are importable from the core module Express (see Section 2.3).

2.1.2.2 Managing a user-defined dataset

In addition to DatasetSourced, the subclass DatasetUser of Dataset is designed for a user-defined dataset that has not yet been supported by GeospaceLAB. With DatasetUser, users can add data that they have downloaded and loaded using their own scripts. Users can also build a subclass of DatasetUser to customize the attributes and methods. A DataHub object has the method add_dataset () to add a DatasetUser object. As a result, a user-defined dataset can be processed together with other datasets. This is useful when users want to use the functionality of GeospaceLAB to visualize or analyze the data that has not been supported by the package.

TABLE 1 A list of data sources that have been included in GeospaceLAB (Current version: v0.5.2).

Database	Facility, instrument & product	File format	Downloadable	Express	Notes & reference
CDAWeb	OMNI	CDF	True	OMNIDashboard	Papitashvili and King (2020)
WDC	Dst	ASCII (IAGA 2002)	True	OMNIDashboard	World Data Center for Geomagnetism, Kyoto
	ASY, SYM	ASCII (IAGA 2002)	True		
	AE	ASCII (IAGA 2002)	True		
GFZ	Kp, Ap	ASCII	True	OMNIDashboard	Matzka et al. (2021) and Yamazaki et al. (2022)
	Hp	ASCII	True	—	
	Sunspot number/F10.7	ASCII	True	—	
Madrigal	EISCAT	HDF5-EISCAT, HDF5-Mardrigal	True	EISCATDashboard	EISCAT incoherent scatter radar (ISR) measurements
	Millstone Hill ISR	HDF5	True	MillstoneHillDashboard	Millstone Hill ISR
	DMSP-e	HDF5	True	DMSTSDashboard	DMSP SSJ data
	DMSP-s1	HDF5	True	DMSPTSDashboard	DMSP SSM & SSIES
	DMSP-s4	HDF5	True	DMSPTSDashboard	DMSP SSIES
	GNSS-TECMAP	HDF5	True	—	GNSS TEC maps
JHUAPL	DMSP-SSUSI-EDR_AUR	NetCDF	True	DMSPSSUSIDashboard	Paxton et al. (2002), JHUAPL/, 2022
	DMSP-SSUSI-SDRDISK	NetCDF	True	—	
	AMPERE-Fitted	NetCDF	False	AMEPREDashboard	Anderson et al. (2002) and Waters et al. (2001)
SuperDARN	Potential map	ASCII	False	—	Greenwald et al. (1985)
ESA-EO	SWARM-EFI-LP_HM	NetCDF	True	—	Swarm Product Data Handbook
	SWARM-EFI-TCT02	NetCDF	True	—	
TUDelft	SWARM-DNS_POD	ASCII	True	—	Doornbos, (2012), Siemes et al. (2016), March et al. (2021), and van den IJssel et al. (2020)
	SWARM-DNS_ACC	NetCDF	True	—	
	GRACE-DNS_ACC	NetCDF	True	—	
	GOCE-WIND_ACC	NetCDF	True	—	
	CHAMP-WIND_ACC	NetCDF	True	—	
	CHAMP-DNS_ACC	NetCDF	True	—	
UTArlington	GITM-2DALL	binary, sav-IDL	False	—	GITM team at U. Texas Arlington, Deng and Ridley (2014)
	GITM-3DALL	binary, sav-IDL	False	—	

10.3389/fspas.2022.1023163

An example code[14] on how to add a user-defined dataset to a `DataHub` or `Dashboard` object can be found in the GitHub repository.

2.1.3 DataHub

The class `DataHub` is at the top-level in the three core classes. A `DataHub` object owns the attributes such as `dt_fr`, `dt_to`, and `visual` as the global settings. Those attributes are passed to the datasets being added. The `DataHub` object uses the method `dock ()` to add a `DatasetSourced` object (as shown in List 1), or the method `add_dataset ()` to add a `DatasetUser` object. For the method `dock ()`, the required keyword arguments can be checked by the `DataHub` method `list_sourced_datasets ()` or by the online documentation.

In summary, the module *DataHub* provides a framework of managing data from measurements or simulations. The module manages not only the data arrays but also the associated properties. The functionality of the core classes makes it easy to integrate multiple data products with different dependencies and properties in one project. Also based on *DataHub*, the following modules *Visualization* and *Express* can implement a quick integration of multiple data in one figure, which helps users to view space physics events in different aspects.

2.2 Visualization (`geospacelab.visualization`)

2.2.1 Base classes and inheritances

With the module *Visualization* we aim to make publication-quality figures using data and metadata provided from DataHub. Currently, the module supports generating static plots by wrapping the `matplotlib` objects. The module is composed of three base classes (`FigureBase`, `DashboardBase`, and `PanelBase`) and a series of subclasses (see the classes marked by orange shades in **Figure 2**). The class `FigureBase` is the top level container for all plotting elements. The class is an inheritance of `matplotlib.figure.Figure`. Thus, the same attributes and methods used in `matplotlib.figure.Figure` can be set and get from `FigureBase`. `FigureBase` contains the method `add_dashboard ()`, so that users can add one or more dashboards (defined below) to the figure object. In addition, users can add watermarks on the figure if needed.

The class `DashboardBase` is the second level container. A dashboard is composed of one or more panels. The method `set_layout ()` of a `DashboardBase` object is used to set the dashboard's position in a figure and to arrange the panels by rows and columns. `DashboardBase` owns the methods such as `add_text ()` and `add_title ()` to add text to the dashboard coordinates.

The class `PanelBase` is the base level container. The class wraps `matplotlib.axes.Axes` and adds additional functionality. A `PanelBase` object has one major ax for plotting. Several supporting axes can be added by the method `add_axes ()` for making the colorbar, legend box, and other purposes. Like `DashboardBase`, `PanelBase` owns the methods such as `add_text ()`, `add_label ()`, and `add_title ()` to add text in a panel coordinate instead of a dashboard coordinate.

From `DashboardBase` and `PanelBase`, a series of dashboards and panels have been developed (see also **Figure 2**). First, the class `Dashboard` inherits from both `DataHub` and `DashboardBase`. Hence, a `Dashboard` object can be used as a data manager (`DataHub` object) to add various datasets. Meanwhile, the class contains all functions from `DashboardBase`. Second, the module Visualization is currently focusing on two kinds of data: time-series data and geospatial data. Hence, two groups of dashboard and panel classes have been developed for making the time-series and geospatial mapping plots, respectively.

Using those classes, we can build a figure with multiple dashboards and panels for viewing various data simultaneously. **Figure 4** shows a schematic diagram as an example of a figure with a complex layout. The figure is equivalent to a `FigureBase` object. It contains three dashboards (`Dashboard` objects). The position of each dashboard is

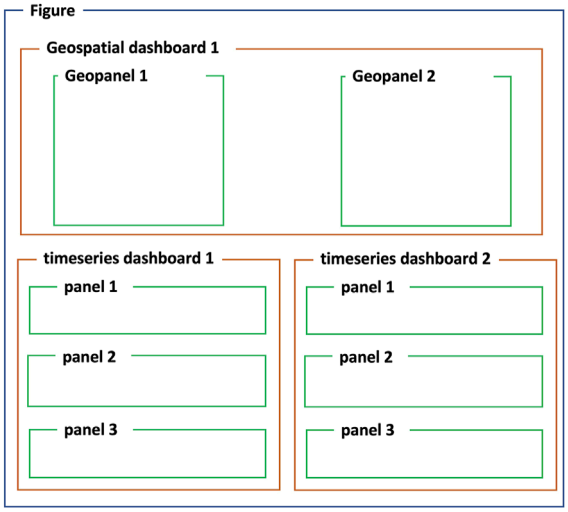


FIGURE 4
A schematic diagram as an example of a figure with a complex layout.

14 https://github.com/JouleCai/geospacelab/blob/master/examples/demo_user_defined_dataset.py.

controlled by the Dashboard method set_layout (). On the top, there is one geospatial dashboard (GeoDashboard object) with two geospatial panels (GeoPanel objects). On the bottom there are two time-series dashboards (TSDashboard objects) on the left and right sides, respectively. Each time-series dashboard contains several time-series panels (TSPanel objects). Using GeospaceLAB, users can create such figures with complex layouts in a simple and quick way.

2.2.2 Visualizing time-series data

In space physics, most of the observational and simulation data depends on time. Even an image or a map is typically associated with a time. To view the time-series data, we have developed the subclass TSDashboard from Dashboard. Due to the inheritance, TSDashboard can also be used as a data manager (DataHub) to dock/add datasets.

The code in **Listing 2** creates a TSDashboard object, which is used to retrieve and view OMNI data and additional geomagnetic indices instead of using the DataHub object as shown in **Listing 1**. The TSDashboard object generates a dashboard with multiple panels to show the time-series plots. The panel layouts, including the number of panels and the variables that will be plotted in one panel, are configured by calling the method set_layout () with the keyword argument panel_layouts. The plotting types, such as 1-D lines, points, error bars, and 2-D surfaces will be automatically detected, when the method draw () is called. Then the corresponding dashboard will be generated. For a sourced dataset, the plot property of a 1-D or a 2-D time-series variable have been preset. Hence, users only need to set the panel layouts by one-line command to make a free arrangement of panels and plots.

Figure 5 shows four time-series dashboards (a–d) corresponding to the code in **Listing 2**. Only the keyword argument panel_layouts is assigned with four different values, respectively. The assigned value is a nested Python list. The length of the outermost list indicates the number of time-series panels that will be shown in a dashboard. The element of the outermost list is a sub-list. Within the sub-list, the Variable objects that are included will be plotted in that panel. This one-line setting make it easy to add or remove panels and plots.

Additional functionalities of TSDashboard with TSPanel are listed as follows:

- Automatically adjust the time ticks and tick labels according to the time bounds assigned to dt_fr and dt_to.
- Support common 1-D and 2-D plot types like in matplotlib, including line plots, scatters, bars, pcolormesh, and image.
- Extend and customize plot types by registering new plotting function to the class TSDashboard.

```
# Step 1: Import the required packages and modules
import datetime
from geospacelab.visualization.mpl.dashboards import TSDashboard
# Step 2: Set the starting and stopping times
dt_fr = datetime.datetime(2015, 3, 16, 12)  # from
dt_to = datetime.datetime(2015, 3, 19, 12)  # to
# Step 3: Create a DataHub object
db = TSDashboard(dt_fr=dt_fr, dt_to=dt_to)
# Step 4: Dock the sourced datasets
ds_omni = db.dock(
    datasource_contents=['cdaweb', 'omni'],
    omni_type='OMNI2', omni_res='1min',
    load_mode='AUTO', allow_load=True)
ds_kp = db.dock(datasource_contents=['gfz', 'kpap'])
ds_sym = db.dock(datasource_contents=['wdc', 'asysym'])
# Step 5: Get the variables stored in their parent datasets.
b_x = ds_omni['B_x_GSM']
b_y = ds_omni['B_y_GSM']
b_z = ds_omni['B_z_GSM']
v_sw = ds_omni['v_sw']
n_p = ds_omni['n_p']
p_dyn = ds_omni['p_dyn']
kp = ds_kp['Kp']
sym_h = ds_sym['SYM_H']
# Step 6: Assign and set the panel layouts
# Figure 5a
# db.set_layout(panel_layouts=[[b_x], [b_y], [b_z]])
# Figure 5b
# db.set_layout(panel_layouts=[[b_x, b_y, b_z]])
# Figure 5c
# panel_layouts = [[b_x, b_y, b_z], [v_sw]]
# Figure 5d
panel_layouts = [[b_x, b_y, b_z], [v_sw], [p_dyn], [kp, [sym_h]]]
# Step 7: Make the plots
db.draw()
# Step 8: Show the plots
db.show()
# Step 9: Add additional marking tools as shown in Figure 5d
# Add vertical lines
dt_1 = datetime.datetime(2015, 3, 17, 4, 45)
dt_2 = datetime.datetime(2015, 3, 17, 6, 40)
dt_3 = datetime.datetime(2015, 3, 17, 22, 50)
dt_4 = datetime.datetime(2015, 3, 18, 18, 0)
db.add_vertical_line(dt_1, color='r')
db.add_vertical_line(dt_2)
db.add_vertical_line(dt_3)
db.add_vertical_line(dt_4)
# Add top bars
db.add_top_bar(dt_1, dt_2, bottom=0., top=0.02, color='y', label='SSC')
db.add_top_bar(dt_2, dt_3, bottom=0., top=0.02, color='r', label='MP')
db.add_top_bar(dt_3, dt_4, bottom=0., top=0.02, color='g', label='RP')
# Add shadings
db.add_shading(dt_1, dt_2, bottom_extend=0, color='y', alpha=0.2)
db.add_shading(dt_2, dt_3, bottom_extend=0, color='r', alpha=0.2)
db.add_shading(dt_3, dt_4, bottom_extend=0, color='g', alpha=0.2)
```

Listing 2. A code example for obtaining and visualizing OMNI and geomagnetic index data from three sourced datasets. The dashboard generated by this code is shown in **Figure 5D**.

- Detect and show the time gaps for regularly measured data, e.g., the blanks between some data points shown in **Figure 5D** top panel.
- Provide several marking tools, such as vertical lines across panels, horizontal bars, and rectangular shadings. The marking tools are often used to indicate interesting time periods for a event analysis, as shown in **Figure 5D**.
- Support to generate multiple lines of tick labels that indicate different support data, respectively. For example, this option is useful for viewing the satellite data depending not only on UT but also on geospatial locations simultaneously (see also **Figure 9** bottom).

2.2.3 Visualizing geospatial data

Based on Dashboard, the subclass GeoDashboard is used for viewing the geospatial data. A GeoDashboard object uses the method set_layout () to arrange the GeoPanels (GeoPanel objects) in multiple rows and columns. The class GeoPanel inherits from PanelBase and wraps

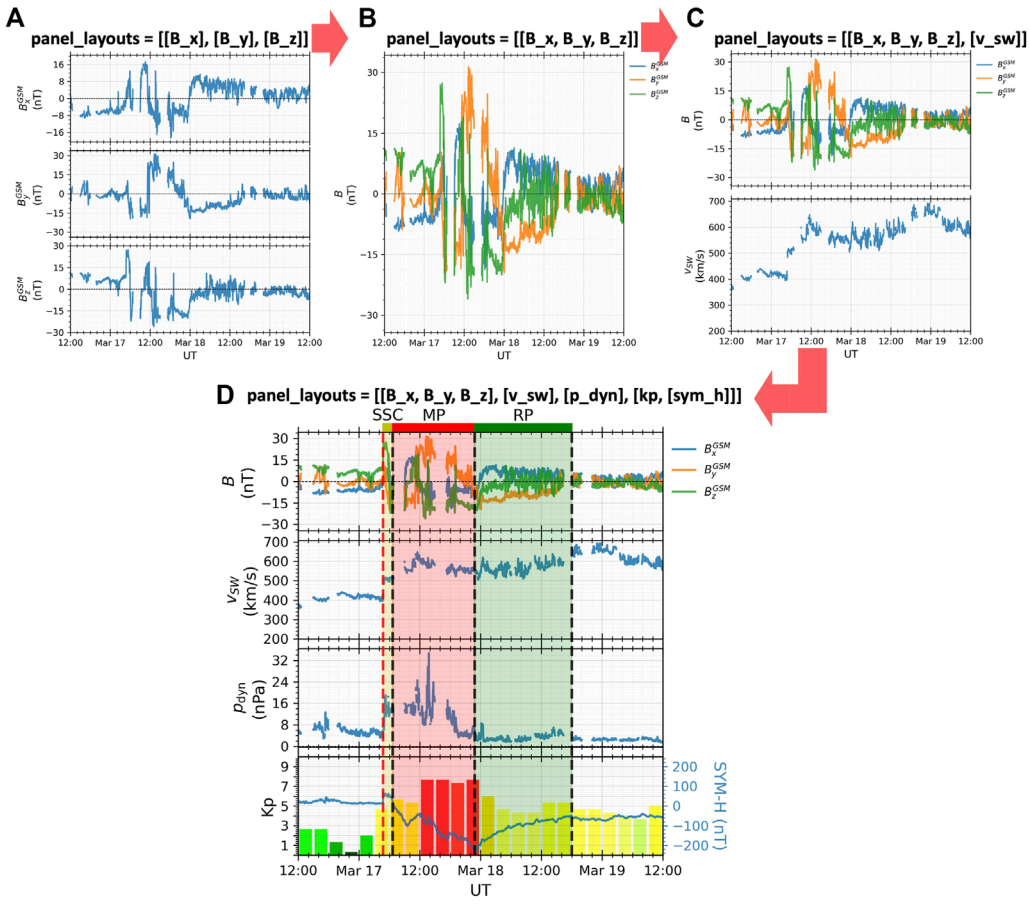


FIGURE 5
An example of using the keyword argument `panel_layouts` to set the panel layouts in a time-series dashboard. The keyword value determine the panel layout of a dashboard as shown in panels **(A–D)**. In panel **(D)**, from top to bottom are the IMF B_x, B_y and B_z components, the solar wind speed, the solar wind dynamic pressure, and Kp and SYM-H indices during the 2015 St. Patrick's Day storm. The dashed vertical lines, shadings, top bars indicate the three phases of the storm, including the initial phase (or SSC), main phase (MP), and recovery phase (RP)[15].

the "GeoAxes" of the package `cartopy`. As a result, users can make most types of map projections in `cartopy` by adding the `GeoPanel` object to the `GeoDashboard` object.

The map projections in `cartopy` use the geographic (GEO) coordinate system. However, in many space physics studies, especially when examining the magnetosphere-ionosphere-thermosphere (M-I-T) coupling, a geomagnetic coordinate system is used, because the geomagnetic field plays an important role in governing the dynamics and electrodynamics in the M-I-T system. To solve the mapping projections in a geomagnetic coordinate system, a subclass of `GeoPanel` has been developed, so called `PolarMapPanel`.

Using `PolarMapPanel`, users can project the 1-D or 2-D geospatial data on the polar map in three view styles: 1) polar projection with the geographic latitude/longitude (GLON-fixed), 2) polar projection with geographic latitude and local solar time (LST-fixed), and 3) polar projection with geomagnetic latitude and magnetic local time (MLT-fixed). The polar projection is based on the stereographic projection method. The class `PolarMapPanel` supports mapping in several geomagnetic coordinate systems, such as Altitude Adjusted Corrected Geogmagnetic Coordinates (AACGM), Magnetic Apex Coordinates (APEX), and Quasi-Dipole coordinates (QD) (see also Laundal and Richmond, 2017). The coordinate transformations are supported by the module *SpaceCoordinateSystem*.

Figure 6 shows an example of the GNSS TEC maps in the three view types at 0620 UT on 17 March 2015 during the St. Patrick's Day storm. The coastlines, grid lines and labels can optionally be added in all three views, which helps users to identify and pinpoint interesting structures and their locations in the polar maps (see the discussion in **Section 3.2**).

15 Source code: https://github.com/JouleCai/geospacelab/blob/master/examples/manuscript_example_2_%20omni.py.

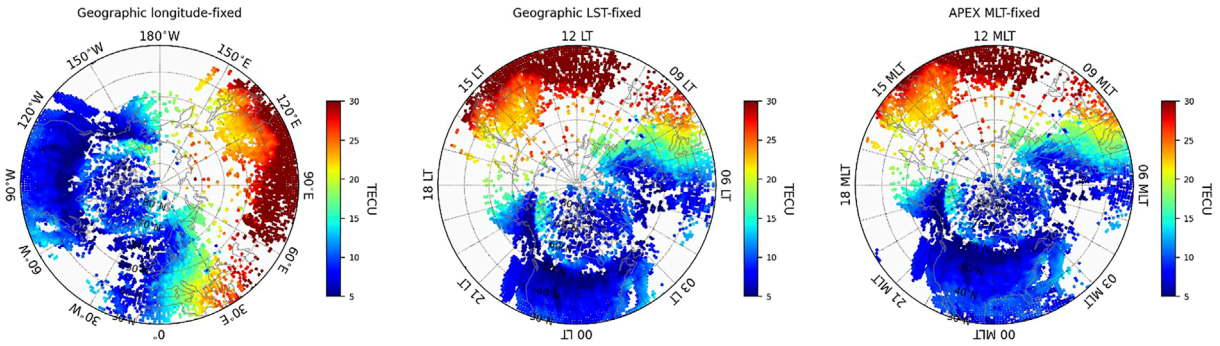


FIGURE 6
An example of the polar maps showing the GNSS/TEC in the northern hemisphere at 0620 UT on 17 March 2015 during the SSC of the 2015 St. Patrick's Day storm in three view modes: left—fixed at the geographic longitude of 0°, middle—fixed at the solar local time (SLT) of 0 o'clock, and right—fixed at 0 MLT in the APEX coordinate system[16].

In summary, a `PolarMapPanel` can be used to:

- Show polar maps in three view styles: GLON-fixed, LST-fixed, and MLT-fixed.
- Utilize optional geomagnetic coordinate systems, so far AACGM, APEX, and QD are included.
- Add coastlines in either a geographic or a geomagnetic coordinate system.
- Add latitude and longitude grid lines and their labels.
- Support basic 1-D or 2-D plots provided by `matplotlib` and `cartopy`.
- Overlay satellite trajectories with time ticks and tick labels (see nine top).
- Overlay satellite cross-track vectors along the satellite trajectory (see nine top).
- Mark ground-based sites on the polar map (see nine top).
- Re-sample or re-grid data for proper mapping.

In addition to `PolarMapPanel`, other subclasses of `GeoPanel` are currently under development. Several more types of map projections will be added for both global and regional mapping in the future.

2.3 Express (`geospacelab.express`)

The module *Express* contains a number of high-level dashboards developed from `TSDashboard` or `GeoDashboard`. Those dashboard classes are shown in **Figure 2** with yellow shades. A set of classes inherit from `TSDashboard`, including `EISCATDashboard`, `MHODashboard`, `OMNIDashboard`, and `DMSPDashboard`. The class names have indicated the purposes. They are used for viewing time-series plots for the variables from specific datasets (EISCAT, Millstone Hill Observatory, OMNI, DMSP, respectively). Users only need a few input to get the data and make a quicklook figure.

The code in **Listing 3** is used to generate a quicklook dashboard for the ionospheric parameters measured by the Millstone Hill incoherent scatter radars during the 2015 St. Patrick's Day storm from 12 UT on 17 March 2015 to 12 UT on 19 March 2015. By importing the high-level dashboard `MillStoneHillISRDashboard`, only a few lines of code can generate a quicklook dashboard. The output dashboard is shown in **Figure 7A**. The dashboard contains five panels. From top to bottom are altitude *versus* UT variations for 1) electron density, 2) electron temperature, 3) ion temperature, 4) line-of-sight ion velocity, as well as 5) the radar parameters.

The dashboards in Express are also used to add customized functionality for a specific dataset. For example, the dashboards like `EISCATDashboard` and `MillstoneHillDashboard` provide methods to check the beam directions of a radar antenna. In case of a steerable antenna, the radar experiment is often made with a multi-beam scanning mode. To check and select beams, several particular methods have been developed. As shown in **Figure 7B**, the dashboard shows five panels in the same format as on the left, but with the selected beam with az = 0° and el = 45°. To generate this quicklook dashboard, users only need to uncomment the line with `dashboard.select_beams()` in the code in List 3.

In addition, **Figure 7B** shows the ionospheric parameters as a function of the AACGM MLAT instead of the altitude along the *y*-axis. The configuration is implemented by making a minor modification of the attributes in the `Visual` object appended to the corresponding `Variable` object.

16 Source code: https://github.com/JouleCai/geospacelab/blob/master/examples/manuscript_example_4_GNSS.py.

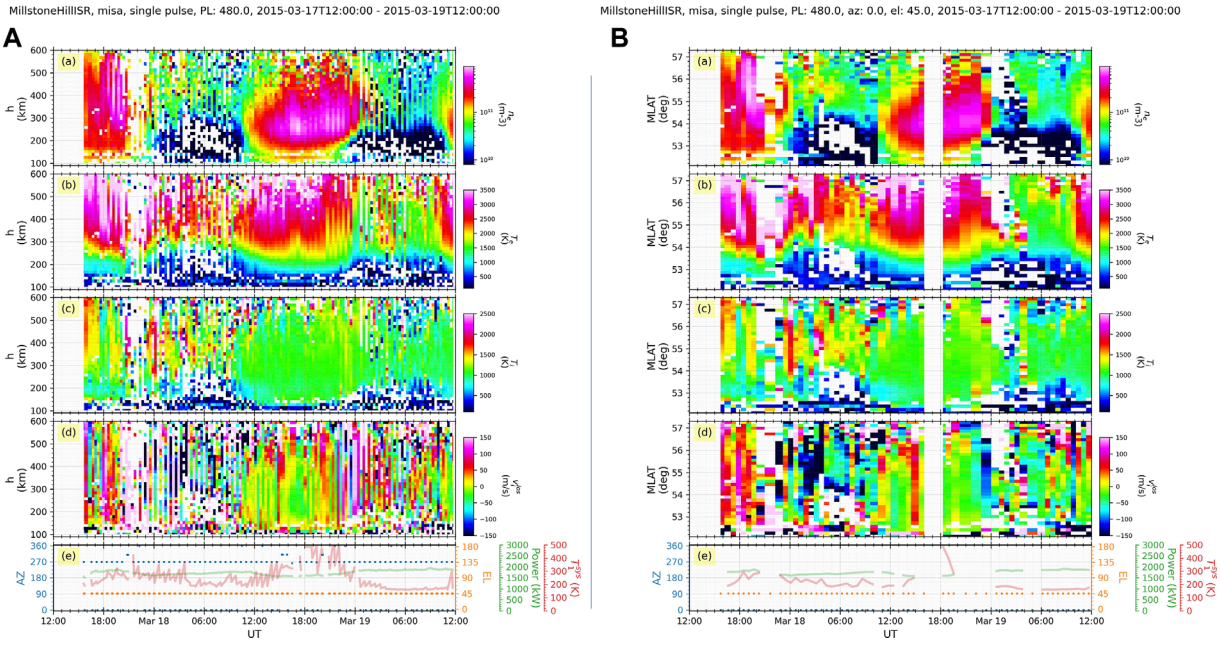


FIGURE 7
An example of two quicklook figures generated from `MillstonHillDashboard` in Express, which shows the Millstone Hill incoherent scatter radar measurements during the 2015 St. Patrick's Day storm. **(A)**: Measurements from the steerable antenna MISA with multiple scanning beams. **(B)**: Measurements from the same radar, but only one beam selected with the az = 0° (azimuth angle) and el = 45° (elevation angle)[17].

```
import datetime
from geospacelab.express.millstonehill_dashboard
    import MillstoneHillISRDashboard
# Set the starting and stopping times
dt_fr = datetime.datetime.strptime('20150317' + '1200', '%Y%m%d%H%M')
dt_to = datetime.datetime.strptime('20150319' + '1200', '%Y%m%d%H%M')
# Set the radar parameters
antenna = 'misa'     # antenna name
pulse_code = 'single pulse'      # pulse code
pulse_length = 480               # pulse length
exp_name_pattern = ''
exp_check = False                # If True, list all the experiments.
load_mode = 'AUTO'               # default loading mode.
# Create a dashboard object.
dashboard = MillstoneHillISRDashboard(
        dt_fr, dt_to,
        antenna=antenna,
        pulse_code=pulse_code, pulse_length=pulse_length,
        exp_name_pattern=exp_name_pattern, exp_check=exp_check,
        load_mode=load_mode, figure_config={'figsize': (10, 15)})
# Select one or multiple beams shown in the quicklook plot.
# if commented, all beams will shown following the time sequence.
# dashboard.select_beams(az_el_pairs=[[0, 45]])
# Make quicklook plot
dashboard.quicklook(depend_MLAT=False)
# Display the figure in the screen
dashboard.show()
```

Listing 3. A code example for generating a quicklook dashboard of Mill Stone Hill measurements during the 2015 St. Patrick's Day storm from 12 UT on 17 March 2015 to 12 UT on 19 March 2015.

2.4 SpaceCoordinateSystem (`geospacelab.cs`)

In space physics research, establishing a proper coordinate system and frame of reference helps us to simplify a given problem and to better understand the underlying physical processes. A variety of coordinate systems and frame of references exists which are widely used (see the reviews by Hapgood (1992), Laundal and Richmond (2017); Shi et al. (2019)). Several python packages have been developed for coordinate system transformations, e.g., *AACGMV2*, *APEXPY*, *GEOPACK*, and *IGRF* (Michael, 2021). Those packages are widely used, however, the functions from different packages are called in different ways and the outputs have different data structures. The module *SpaceCoordinateSystem* aims to provide a unified interface for the coordinate transformation among different coordinate systems. The module wraps those popular packages and develops coordinate system classes based on two base classes: `SpaceCSBase` and `CoordinateBase`.

Figure 8 shows the UML class diagram of the module *SpaceCoor*dinateSystem. The base class `CoordinateBase` manages a collection of coordinates in a specific coordinate system. `SpaceCSBase` abstracts the functions for the coordinate transformation. A `SpaceCSBase` object represents a coordinate system, which is appended with one `CoordinateBase` object recording the coordinate data.

17 Source code: https://github.com/JouleCai/geospacelab/blob/master/examples/manuscript_example_3_%20isr.py.

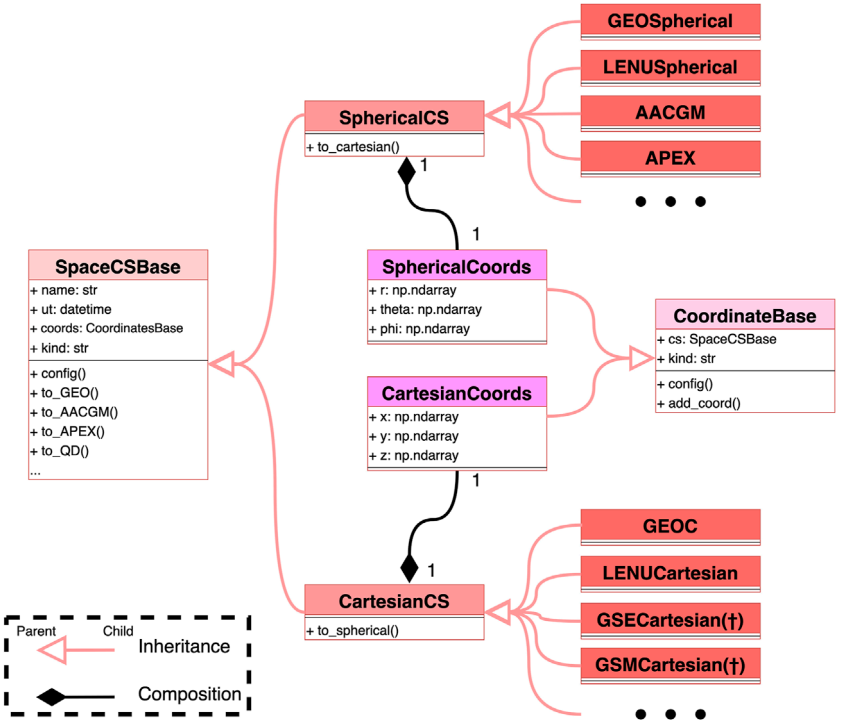


FIGURE 8
A UML class diagram that describes the main structure of the module SpaceCoordinateSystem in the same format as in **Figure 2**. The classes marked by * are under development.

Based on those two classes, two groups of classes have been developed: one group for spherical coordinate systems and the other for Cartesian coordinate systems. So far, the classes for spherical coordinate systems include `GEOSpherical`, `LENUSpherical`, `AACGM`, and `APEX`. The classes for Cartesian coordinate systems include `GEOC`, `LENUCartesian`, `GSECartesian`, and `GSMCartesian`. The name LENU refers to the local east-north-upward coordinate system fixed at the Earth's surface. Those classes use unified commands to achieve the coordinate transformation. For example

```
cs_LENUC = cs_LENUS.to_cartesian ()
cs_GEO = cs_LENUC.to_GEO ()
```

where the arguments `cs_LENUC`, `cs_LENUS` and `cs_GEO` are the objects of `LENUCartesian`, `LENUSpherical`, and `GEOSpherical`, respectively. The coordinate transformations are completed by calling the functions with the same structure. The coordinate data stored in those objects also have the same data structure.

The module SpaceCoordinateSystem supports the modules such as DataHub, Visualization, and Express in case a coordinate transformation is needed. On the other hand, users can use the functionality provided by SpaceCoordinateSystem for their own codes. For this purpose, an example code[19] is provided in the GitHub repository.

2.5 Toolbox (`geospacelab.toolbox`)

The module Toolbox is built mainly with a function-oriented design. The module is composed of several sub-modules. Each sub-module contains a series of functions. **Figure 1** lists four commonly used sub-modules in *Toolbox*, including:

- *PythonBasic*: provides the utility functions associated with the basic data types in Python, such as numerical number, `list`, `str`, and `dict`.
- *Logging*: provides a customized logging system for tracking the events that occur when the package runs.
- *NumpyArray*: provides the utility functions associated with `numpy` arrays.

19 https://github.com/JouleCai/geospacelab/blob/master/examples/demo_cs_transform.py.

- *Datetime*: provides the utilitarian functions for the date and time, including convert time between different formats (such as unix_time, datetime, and MATLAB datenum) and supplementary functions for `datetime`.

2.6 Configuration (geospacelab.config)

The module configuration contains the class `Preferences` to manage the global settings in the package. Users can customize the global settings either temporally or permanently, based on the method `set_user_config ()` of `Preferences`. For example, a basic configuration is needed, when the GeospaceLAB package is imported for the first time after installation. The configuration is to set the default root directory that stores the data downloaded by the sourced dataset classes. In addition, when accessing the Madrigal online database, it will ask for inputs of the user's full name, email, and affiliation as cookies. The module Configuration also manages the cookies, so that users do not need to input those cookies every time when accessing the Madrigal database.

A configuration file in the `toml` format is used to record the users' default settings. The file can be found at " [USER_HOME_DIRECTORY]/.geospacelab/config.toml". Users can modify the default settings recorded in the file. The new settings will be automatically loaded when the package is imported the next time.

In the current version (v0.5.2), the `Preferences` object is used only to set several key parameters. However, with increasing functionality of the package, users may need to control more global settings for the modules and functions. For this purpose, the class `Preferences` provides a frame to configure the global settings and it can be easily extended in future.

3 Applications

The core modules make GeospaceLAB suitable for managing and visualizing various data sets in space physics. Users can directly use the sourced datasets listed in **Table 1** for their research projects. The number of the sourced datasets will increase in the future.

Currently, the sourced datasets include data from ground-based measurements (e.g., EISCAT, Millstone Hill incoherent scatter radars and SuperDARN), satellite measurements (e.g., DMSP, SWARM, and Grace), simulations (GITM), and various solar and geomagnetic activity indices (e.g., F10.7, Kp, Ap, Dst, ASY/SYM, and AE). Those datasets are typically used in magnetosphere-ionosphere-thermosphere coupling studies. A few examples are given in the following subsection. These are chosen to illustrate the capabilities of GeospaceLAB, but do not aim at a detailed scientific analysis of the events.

3.1 Space weather events

Users can search and study space weather events by combining solar wind and geomagnetic activity index data in a dashboard. For example, **Figure 5D** (see also **Section 2.1.2.1**) shows a space weather event during the St. Patrick's Day in 2015. This famous St. Patrick's Day storm has been studied in a number of papers (e.g., Astafyeva et al., 2015; Wu et al., 2016; Zhang et al., 2017, and references therein). **Figure 5D**, shows an interplanetary coronal mass ejection (ICME) arriving at the Earth's magnetospause at about 0445 UT on 17 March 2015, which is associated with an enhanced magnetic field strength in the top panel, a sudden increase in the solar wind velocity (second panel) and an enhancement in the dynamic pressure (third panel). The interaction between the ICME and the Earth's magnetosphere causes a strong geomagnetic storm. The SYM-H index shown in the bottom panel can be used to identify the storm phases. The sudden storm commencement (SSC) (also referred as the initial phase) is the time period with positive SYM-H (in this case between 0445 and 0640 UT on 17 March 2015) the main phase (MP) starts when the SYM-H index turns negative (in this case at 0640 UT) and the main phase ends and the recovery phase (RP) begins, when the SYM-H index has reached its minimum value (in this case at 2250 UT). Finally, when the SYM-H index recovers, the recovery phase ends (in this case at 1800 UT on 18 March 2015). Users can easily mark the storm phases using the marking methods provided by the class `TSDashboard`, including vertical lines, shadings, and topbars.

3.2 Global view of the key parameters in the ionosphere-thermosphere system

GeospaceLAB contains classes such as `GeoDashboard` and `GeoPanel` for visualizing geospatial data. To understand the physics behind ionospheric features, it is often necessary to study 2D-maps in geomagnetic rather than geographic coordinates, as the ionospheric coupling with the solar wind is guided by the geomagnetic field. A geographic view is on the other hand often necessary, e.g., to be able to examine whether a ground-based station is located in the proximity of an interesting ionospheric feature. However, online datasets often provide data only in geographic coordinates. The class `PolarMapPanel` inheriting from `GeoPanel` can be used for mapping 1-D or 2-D data in a polar map with either geographic or geomagnetic coordinates.

Figure 6 (discussed in **Section 2.2.3**), shows the global distribution of the total electron content (TEC) derived from the GNSS network at 0620 UT during the SSC of the 2015 St. Patrick's Day storm. With the help of the global maps, the researchers can identify both large- and meso-scale structures in a 2-D

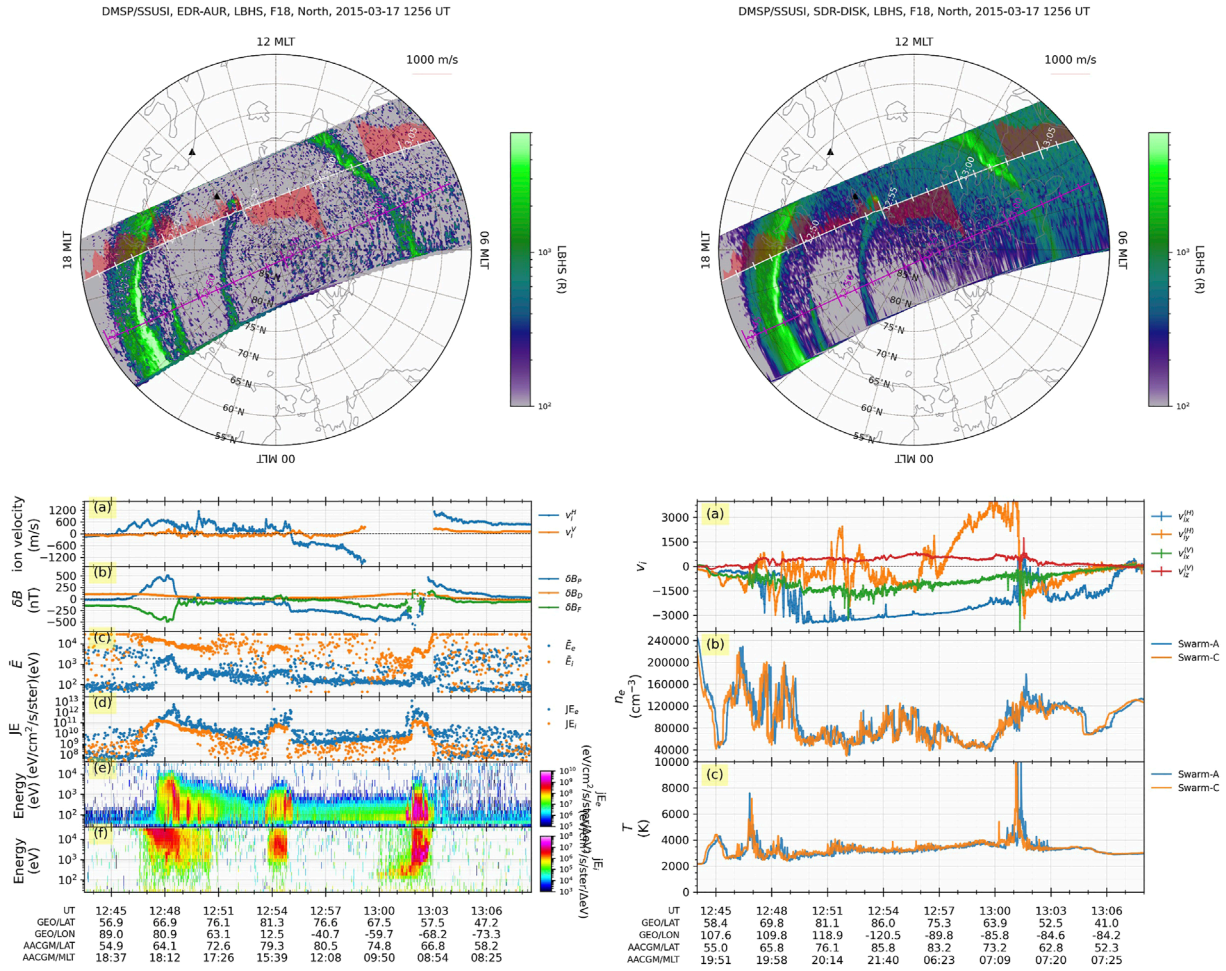


FIGURE 9
Measurements from the DMSP and SWARM satellites around 1256 UT on 17 March 2015. Top-left: DMSP/SSUSI EDR-AUR auroral image in the LBHS band overlaid with both DMSP (white) and SWARM-A/C (magenta) trajectories, top-right: DMSP/SSUSI EDR-DISK image in the same format as the top-left, bottom-left: DMSP SSIES, SSJ, and SSM measurements along the trajectory shown in top panels, and bottom-right: SWARM A and C measurements of electron density, electron temperature, and cross-track and ram velocities along the trajectory shown in the top panels. This figure shows a complex layout corresponding to **Figure 4**.[18]

TEC map. For example, the maps show the typical large-scale distribution of the TEC from dayside to the nightside due to solar radiation. In addition, a mid-latitude trough is seen over North America, extending from the evening sector to the post mid-night sector. Using GeospaceLAB, it is also feasible to overlay additional layers on the top of the TEC maps for a comparison across multiple types of data, e.g., a contour plot of the electric potential measured by SuperDARN or cross-track vectors along a satellite trajectory.

3.3 Regional or local responses

The sourced datasets include local measurements from an individual ground-based instrument or *in situ* measurements along a satellite trajectory. To view the local ionospheric/thermospheric responses, the `TSDashboard` and `TSPanel` described in **Section 2.2.2** are often used.

As shown in **Figure 7**, the Millstone Hill incoherent radar measured several basic plasma parameters in the ionosphere during the MP and RP of the 2015 St. Patrick's Day storm. The measurements show a strong enhancement of the electron density in the F-region at around 1800 UT on 17 March 2015 during the MP. This enhancement is identical and associated with the storm-enhanced density (SED) in the mid-latitude ionosphere (Zhang et al., 2017).

18 Source code: https://github.com/JouleCai/geospacelab/blob/master/examples/manuscript_example_5_dmsp_swarm.py.

The two dashboards shown in **Figure 7** are generated by the high-level dashboard object in the module Express for a quicklook view. Again, users can make more customized plots by using the lower-level `TSDashboard` class.

3.4 Conjugate satellite observations

The module *Visualization* gives the flexibility to arrange multiple dashboards and panels in one figure (see e.g., **Figure 4**), and to combine conjugate satellite observations and integrate these in a single figure.

Figure 9 shows an event with near-simultaneous measurements from the DMSP and SWARM A/C satellites above the northern auroral oval. The layout of the figure corresponds to that shown in **Figure 4**. On the top is the geospatial dashboard composed of two polar maps. The polar maps show the DMSP SSUSI auroral images in the LBHS wavelength band, overlaid with the trajectories of the two satellites and the cross-track velocities measured by the DMSP F18 satellite. In addition, two EISCAT incoherent scatter radar sites in Tromsø and on Svalbard, Norway are marked with black triangles on the maps.

This example shows an interesting event of a large-scale transpolar arc (TPA) (Kullen, 2012; Hosokawa et al., 2020) appearing in the middle of the polar cap during the main phase of the 2015 St. Patrick's Day storm. Both, the DMSP F18 and the SWARM A/C satellite crossed the northern hemisphere from the duskside to the dawnside almost at the same time. The two satellites measured the ionospheric parameters of the TPA at two locations: DMSP F18 was close to the dayside tip of the TPA and SWARM A/C across the main part of the TPA.

The *in situ* measurements of the DMSP F18 satellite are shown in the time-series dashboard on the bottom-left side, and the Swarm A/C measurements are shown on the bottom-right side. On the bottom, time tick labels are shown with the corresponding geospatial information, including GEO/LAT, GEO/LON, AACGM/LAT, and AACGM/MLT. The additional labels help us to identify the location of the key features shown in the time-series dashboards and make it easier to compare those with the auroral structures shown in the geo-dashboard on the topside of the figure. The entire figure including image, time series plots and all labels was produced with the *visualization* module. Among other details, the time series reveals that the TPA lies on sunward plasma flow (top panel at 1554 UT in DMSP time-series plots and top panel at 1551 UT in SWARM time-series plots). The bottom panel of the DMSP measurements shows that the TPA is populated by plasma sheet ions, hinting on a source region on closed magnetic field lines (Kullen, 2012).

The two polar maps in **Figure 9** show the same auroral structures, however, the appearances are slightly different. The left side shows SSUSI EDR-AUR data and the right side shows SSUSI SDR-DISK data. Both data products are listed in **Table 1**. The SDR-DISK image is a lower-level data product and the daylight dilution of the image has not been removed. The key issue in the two maps is that the EDR-AUR image on the left is slightly tilted counter-clockwise with respect to the satellite trajectory. The SDR-DISK image position is normal, parallel to the satellite trajectory as expected. The difference is plausibly caused by using a different version of the AACGM model by the data provider and GeospaceLAB. The SDR-DISK product provides image grids in the geographic coordinate system. The geographic coordinates are transformed to the AACGM coordinates by GeospaceLAB when the SDR-DISK image is mapped. However, the EDR-AUR product only includes the AACGM coordinates for the image grids. GeospaceLAB uses that AACGM grids directly to map the image. Since the former mapping with the SDR-DISK data is consistent with the mapping of the satellite trajectory, for which the coordinates are also transformed from the geographic coordinates by the module *SpaceCoordinateSystem*, we suggest that the version of AACGM model used by the SSUSI Team may be different from the AACGMV2 python package used in GeospaceLAB.

4 Current status and future

The GeospaceLAB project was started in 2021. The initial establishment was based on the proprietary MATLAB and Python libraries used in the authors' own research works (e.g. Cai et al., 2021). After one and a half years of systematic development, GeospaceLAB has become stable for most of the modules and functions. The package is open-sourced with the BSD-3-Clause license and available in a GitHub repository (JouleCai/geospacelab version v0.5.2). The package is also uploaded to PyPI and can be easily installed *via* "pip".

4.1 Issues and solutions

The development of GeospaceLAB is ongoing. The main structures of the core modules in the package are well established. Still, there are several known issues regarding the package itself, the package dependencies, and the data sources that would benefit from improvement or expansion. These are listed below:

- So far, the package has been developed with focus on the demands by the authors. It does not yet cover several data products, specifically data from the Earth's magnetosphere or heliosphere.

Due to the well-designed core framework, it is feasible to include more datasets in *DataHub*, add more plotting routines

in *Visualization*, and expand *SpaceCoordinateSystem* with more coordinate system transformation methods. To do so, additional workflows and maintenance are required. Hence, we would like to invite contributors to join this project by submitting new data products and adding new functionalities. Guidelines for advanced users and developers are available in the online documentation[20].

- The first priority for GeospaceLAB is to include high-level data products from data providers. The term "high-level data" means that the data product is calibrated, qualified, or can be directly used in research studies. However, issues with the data may still occur. For example, the mapping difference between two kinds of DMSP SSUSI products is discussed in **Section 3.4**. We highly recommend that users should always familiarize themselves with the data definitions, formats, and usage policy from the data providers. The `DatasetSourced` object in GeospaceLAB has the attributes, such as `database`, `facility`, and `product`, which record the URL and notes of data usage for individual data products. The users can extract the information and contact the principal investigators of the data product if needed.
- The package is compatible with most of the dependent packages. However, sometimes users have reported that the package *AACGMV2* returns errors when it is imported from GeospaceLAB. This happened mainly when a user used a specific integrated development environment (IDE), such as "PyCharm" or "VS. Code". The issue can be solved, if the IDE "Spyder" is used. However, a more general solution should be implemented in future. We have reported this issue to the GitHub repository of *AACGMV2*.
- The package has mostly been tested in the operating systems (OS) of Ubuntu and macOS. Its compatibility with the Windows OS and other Linux distributions is not guaranteed.

We thank the users who have contributed to the GeospaceLAB project by pointing out open issues and providing feedback on GitHub. The activities have helped to improve the performance of the package.

4.2 Proposed features

Since the core modules of GeospaceLAB are easily extendable, many new features can be added to them in future releases. The next major release will be version 1.0, scheduled for the end of 2022. In version 1.0, the package will become more stable, and will include the following features:

- Extension of the sourced datasets with several commonly used data products in the community.
- A well-structured online documentation.
- Support of several new plotting styles in the *Visualization* module for the 1-D or 2-D data.
- Support of more coordinate system transformations in the module *SpaceCoordinateSystem*.
- Enhanced testing of the package to make it more robust using the Python package *PyTest*.

5 Summary

GeospaceLAB is an easy-to-use, extendable Python package for managing and visualizing observational and modeling data that is used in the space physics community. The package has been applied to research topics regarding the study of M-I-T coupling in the authors' research groups. So far, the package has been downloaded from GitHub or PyPI with cumulative number of more than 28,000 times. In each released version, the number of users has been more than 50 (See also the online statistics[21]).

GeospaceLAB bridges data providers and scientists. It provides systematic solutions in data management, visualization, and space coordinate transformation.

- GeospaceLAB can be used as a data manager for various observational and modeling data in space physics. The data products can be obtained from external providers, e.g., an online database, or from the users' local database. For online data, GeospaceLAB aims to achieve automatic downloading, storing, and loading of data.
- GeospaceLAB can also generate high-qualify publication-ready figures. With specialized functions and tools, the package is particularly suitable for viewing time-series and geospatial data in space physics. Users can easily combine multiple kinds of data and integrate them into one figure.
- GeospaceLAB provides a simple, unified interface for transforming from one space coordinate system to another. Currently, it can be used to switch between the geographic (GEO), geocentric (GEOC), AACGM, APEX, QD, and local east-north-upward (LENU) coordinate systems. More transformations will be added in the future.
- The functionality of GeospaceLAB described in previous sections makes it possible to advance quickly with the data analysis itself. Since a `Variable` object stores

20 https://geospacelab.readthedocs.io/en/latest/dev/guidance.html.

21 https://pepy.tech/project/geospacelab.

the data in the form of *NumPy* array, GeospaceLAB is compatible with most popular packages in data analysis, such as *NumPy*, *SciPy*, and *Pandas*. In addition, the data arrays collected by GeospaceLAB are independent of file segments. That could potentially be used for analyzing big data in collaboration with other popular Python libraries.

Since the framework in the core modules is well structured, the package is highly extendable for multiple purposes in space research. We invite members of the community to contribute to GeospaceLAB, by using the package for their research, reporting issues, proposing new functions, and joining its further development.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

LC is the principal developer of the GeospaceLAB package and wrote the manuscript. AA is the principal investigator of LC's postdoc project and suggested the functionality of the package. AK and YD were the supervisors for LC's former projects and provided comments on the package development. S-RZ commented on the usage of Millstone Hill ISR and GNSS TEC data. YZ commented on the usage of DMSP SSUSI data. IV and HV helped the development of the package. All co-authors contributed to the discussions and provided editorial comments on the manuscript.

Funding

The development of the GeospaceLAB package is supported by the Kvantum Institute at University of Oulu. AK and LC acknowledge postdoc grant DNR-155A/17 funded by Swedish National Space Agency. YD was supported by AFOSR through award FA9559-16-1-0364 and NASA grants 80NSSC20K0195, 80NSSC20K1786 and 80NSSC22K0061. S-RZ acknowledges MURI grant ONR15-FOA-0011 and NSF grant AGS-2033787. HV was supported by the Academy of Finland project 314664. The authors acknowledge all the developers who developed the dependencies used in GeospaceLAB. The authors also acknowledge the data providers listed in **Table 1** to support the open access of data. Millstone Hill ISR observation, GNSS TEC data processing, and Madrigal database system are provided to the community by MIT under the US NSF grant AGS-1952737 support. We acknowledge use of NASA/GSFC's Space Physics Data Facility's OMNIWeb (or CDAWeb or ftp) service, and OMNI data.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Anderson, B., Takahashi, K., Kamei, T., Waters, C., and Toth, B. (2002). Birkeland current system key parameters derived from Iridium observations: Method and initial validation results. *J. Geophys. Res.* 107, 1079. SMP–11. doi:10.1029/2001ja000080

Astafyeva, E., Zakharenkova, I., and Förster, M. (2015). Ionospheric response to the 2015 St. Patrick's day storm: A global multi-instrumental overview. *J. Geophys. Res. Space Phys.* 120, 9023–9037. doi:10.1002/2015ja021629

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R., Morley, S. K., Annex, A., et al. (2018). Snakes on a spaceship—an overview of Python in heliophysics. *J. Geophys. Res. Space Phys.* 123, 10–384. doi:10.1029/2018ja025877

Cai, L., Kullen, A., Zhang, Y., Karlsson, T., and Vaivads, A. (2021). DMSP observations of high-latitude dayside aurora (HiLDA). *JGR. Space Phys.* 126, e2020JA028808. doi:10.1029/2020ja028808

Caswell, T. A., Droettboom, M., Lee, A., de Andrade, E. S., Hoffmann, T., Klymak, J., et al. (2022). Matplotlib/matplotlib: Rel: 3.5. doi:10.5281/zenodo.6513224

[Dataset] Michael, (2021). space-physics/igrf: IGRF model. doi:10.5281/zenodo.5560949

Deng, Y., and Ridley, A. J. (2014). The global ionosphere-thermosphere model and the nonhydrostatic processes. *Model. Ionosphere–Thermosphere Syst.*, 85–100. doi:10.1002/9781118704417.ch8

Doornbos, E. (2012). *Thermospheric density and wind determination from satellite dynamics*. Springer Science & Business Media.

Emmert, J., Richmond, A., and Drob, D. (2010). A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors. *J. Geophys. Res.* 115. doi:10.1029/2010JA015326

Engels, G., Heckel, R., and Sauer, S. (2000). "UML—A universal Modeling Language?" in *International conference on application and theory of petri nets* (Springer), 24–38.

Greenwald, R., Baker, K., Hutchins, R., and Hanuise, C. (1985). An HF phased-array radar for studying small-scale structure in the high-latitude ionosphere. *Radio Sci.* 20, 63–79. doi:10.1029/rs020i001p00063

Hapgood, M. (1992). Space physics coordinate transformations: A user guide. *Planet. Space Sci.* 40, 711–717. doi:10.1016/0032-0633(92)90012-d

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Hosokawa, K., Kullen, A., Milan, S., Reidy, J., Zou, Y., Frey, H. U., et al. (2020). Aurora in the polar cap: A review. *Space Sci. Rev.* 216, 15–44. doi:10.1007/s11214-020-0637-3

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Jhuapl/, S. S. U. S. I. (2022). DMSP SSUSI data products. Available at: https://ssusi.jhuapl.edu/(Accessed August 19, 2022).

Kullen, A. (2012). Transpolar arcs: Summary and recent results. *Auror. Phenomenology Magnetos. Process. Earth Other Planets, Geophys. Monogr. Ser* 197, 69–80. doi:10.1029/2011GM001183

Laundal, K. M., and Richmond, A. D. (2017). Magnetic coordinate systems. *Space Sci. Rev.* 206, 27–59. doi:10.1007/s11214-016-0275-y

Luciano, R. (2015). *Fluent Python: Clear, concise, and effective programming*. Sebastopol, CA, United States: OŔeilly Media.

March, G., Van Den Ijssel, J., Siemes, C., Visser, P. N., Doornbos, E. N., and Pilinski, M. (2021). Gas-surface interactions modelling influence on satellite aerodynamics and thermosphere mass density. *J. Space Weather Space Clim.* 11, 54. doi:10.1051/swsc/2021035

Matzka, J., Stolle, C., Yamazaki, Y., Bronkalla, O., and Morschhauser, A. (2021). The geomagnetic K_p index and derived indices of geomagnetic activity. *Space weather.* 19, e2020SW002641. doi:10.1029/2020sw002641

Met Office (20102015). Cartopy: A cartographic python library with a matplotlib interface. *Exeter, Devon*.

Papitashvili, N. E., and King, J. H. (2020). *OMNI combined heliospheric observations (COHO), merged magnetic field, plasma and ephemeris, definitive hourly data*. doi:10.48322/6ffx-3441

Paxton, L. J., Morrison, D., Zhang, Y., Kil, H., Wolven, B., Ogorzalek, B. S., et al. (2002). Validation of remote sensing products produced by the special sensor ultraviolet scanning imager (SSUSI): A far uv-imaging spectrograph on dmsp f-16. *Opt. Spectrosc. Tech. remote Sens. Instrum. Atmos. space Res. IV* 4485, 338–348. doi:10.1117/12.454268

Richmond, A. (1995). Ionospheric electrodynamics using magnetic Apex coordinates. *J. Geomagn. Geoelec.* 47, 191–212. doi:10.5636/jgg.47.191

Shepherd, S. (2014). Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations. *J. Geophys. Res. Space Phys.* 119, 7501–7521. doi:10.1002/2014ja020264

Shi, Q., Tian, A., Bai, S., Hasegawa, H., Degeling, A., Pu, Z., et al. (2019). Dimensionality, coordinate system and reference frame for analysis of *in-situ* space plasma and field data. *Space Sci. Rev.* 215, 35–54. doi:10.1007/s11214-019-0601-2

Siemes, C., de Teixeira da Encarnação, J., Doornbos, E., Van Den Ijssel, J., Kraus, J., Pereštỳ, R., et al. (2016). Swarm accelerometer data processing from raw accelerations to thermospheric neutral densities. *Earth Planets Space* 68, 92–16. doi:10.1186/s40623-016-0474-5

Tsyganenko, N., Andreeva, V., Kubyshkina, M., Sitnov, M., and Stephens, G. (2021). Data-based modeling of the Earth's magnetic field. *Magnetos. Sol. Syst.*, 617–635. doi:10.1002/9781119815624.ch39

van den Ijssel, J., Doornbos, E., Iorfida, E., March, G., Siemes, C., and Montenbruck, O. (2020). Thermosphere densities derived from Swarm GPS observations. *Adv. Space Res.* 65, 1758–1771. doi:10.1016/j.asr.2020.01.004

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2

Waters, C., Anderson, B., and Liou, K. (2001). Estimation of global field aligned currents using the Iridium® system magnetometer data. *Geophys. Res. Lett.* 28, 2165–2168. doi:10.1029/2000gl012725

Wu, C.-C., Liou, K., Lepping, R. P., Hutting, L., Plunkett, S., Howard, R. A., et al. (2016). The first super geomagnetic storm of solar cycle 24:"The St. Patrick's day event (17 March 2015)". *Earth Planets Space* 68, 151. doi:10.1186/s40623-016-0525-y

Yamazaki, Y., Matzka, J., Stolle, C., Kervalishvili, G., Rauberg, J., Bronkalla, O., et al. (2022). *Geomagnetic activity index Hpo*. Geophysical Research Letters. e2022GL098860.

Zhang, S.-R., Zhang, Y., Wang, W., and Verkhoglyadova, O. P. (2017). Geospace system responses to the St. Patrick's Day storms in 2013 and 2015. *JGR. Space Phys.* 122, 6901–6906. doi:10.1002/2017ja024232

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 14 December 2022
DOI 10.3389/fspas.2022.1025823

Check for updates

OPEN ACCESS

EDITED BY
Leslie Lamarche,
SRI International, United States

REVIEWED BY
Octav Marghitu,
Space Science Institute, Romania
Ashley Smith,
University of Edinburgh, United Kingdom
Magnar G. Johnsen,
UiT The Arctic University of Norway, Norway

*CORRESPONDENCE
K. M. Laundal,
karl.laundal@uib.no

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 23 August 2022
ACCEPTED 30 November 2022
PUBLISHED 14 December 2022

CITATION
Hovland AØ, Laundal KM, Reistad JP, Hatch SM, Walker SJ, Madelaire M and Ohma A (2022), The Lompe code: A Python toolbox for ionospheric data analysis.
Front. Astron. Space Sci. 9:1025823.
doi: 10.3389/fspas.2022.1025823

COPYRIGHT
© 2022 Hovland, Laundal, Reistad, Hatch, Walker, Madelaire and Ohma. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

The Lompe code: A Python toolbox for ionospheric data analysis

A. Ø. Hovland, K. M. Laundal*, J. P. Reistad, S. M. Hatch, S. J. Walker, M. Madelaire and A. Ohma

Department of Physics and Technology, Birkeland Centre for Space Science, University of Bergen, Bergen, Norway

A recent paper by Laundal et al. (2022c) presented a new technique to combine all available measurements of polar ionospheric electrodynamics; magnetic field measurements from ground and space, ionospheric convection data from radars and satellites, and conductance measurements; to a full 2D map within analysis regions of arbitrary resolution and extent. The technique, called Local Mapping of Polar Ionospheric Electrodynamics (Lompe), is implemented in Python (Laundal et al., 2022a). The Lompe technique combines spherical elementary current system analysis, finite element analysis on a cubed-sphere projection, the use of empirical models like the International Geomagnetic Reference Field, and visualization tools. In this paper, we go through these different components of the Lompe code and show how they are useful on their own, for example in the analysis of ground magnetometer data or data from the upcoming Electrojet Zeeman Imaging Explorer mission. We also demonstrate how to use the Lompe code to produce a coherent picture of ionospheric electrodynamics.

KEYWORDS

ionospheric electrodynamics, Python, ionospheric data analysis, data assimilation, spherical elementary current systems, cubed-sphere coordinates

1 Introduction

The Local mapping of polar ionospheric electrodynamics (Lompe) technique (Laundal et al., 2022c) combines all relevant data points in a region of interest to produce a 2D map of ionospheric electrodynamics. Given a map of the ionospheric conductance, Lompe can be fed magnetic field measurements from ground and/or space, and any type of electric field or F-region ionospheric convection measurement and output a continuous map of ionospheric electrodynamics: The electric field, plasma flow, ionospheric horizontal and field-aligned current (FAC), and associated magnetic field perturbations on ground and in space. It is conceptually similar to the Assimilative Mapping of Ionospheric Electrodynamics (AMIE) technique (Richmond and Kamide, 1988; AMGeO Collaboration, 2019), but it uses a different set of basis functions: While AMIE uses spherical cap harmonics that span the entire region poleward of some latitude (usually set to 50°), Lompe represents ionospheric electrodynamics with

spherical elementary current systems (SECS; Amm, 1997), which in principle allows more flexibility in choosing the spatial resolution and extent of the analysis region. The Lompe technique is open-source (Laundal et al., 2022a). It is implemented entirely in Python and optimized by using NumPy array operations instead of Python loops where possible. This makes the code portable, user-friendly, and easy to interface with extensive scientific Python modules like NumPy, SciPy, Matplotlib, and Pandas, without a big loss in performance compared to compiled languages.

The Lompe Python package includes several submodules which are required to carry out the Lompe inversion, but are also useful on their own in many aspects of ionospheric data analysis. In this paper, we go through the main components of the Lompe code and present examples of how they can be used. Except for **Figure 6**, all figures in this paper are outputs from Jupyter notebooks published as part of the Lompe code repository (Laundal et al., 2022a).

A core part of the Lompe code is contained in a module called `secsy` (Laundal and Reistad, 2022), included as a submodule, which handles both the analysis grid and the use of spherical elementary current systems (SECS). The Lompe grid is defined in a cubed-sphere projection (Ronchi et al., 1996), which projects points on the sphere to the face of a circumscribed cube aligned with the center of the grid. The `secsy` module includes a projection class to convert between global and "cube" coordinates and vector components, and a grid class. The grid class includes functions to calculate finite difference matrices. In Lompe, these matrices are used to evaluate gradients of scalar fields and the divergence of vector fields defined on the grid. In **Section 2**, we present an example where the differentiation matrices are used to calculate electric field components and electric charge density from a Weimer (2005a), Weimer (2005b) electric potential. The matrices can also be used to solve partial differential equations on a section of a spherical shell. In **Section 2.2**, we demonstrate this capability by solving the 2D continuity equation to explain a typical distribution of F-region plasma density.

In the Lompe technique, spherical elementary currents (Amm, 1997) are used as basis functions to represent the electric field and to relate ionospheric currents and magnetic fields. SECS are local basis functions whose weighted sum can describe any well-behaved vector field on a spherical shell (Vanhamäki and Amm, 2011). The secsy module includes code to calculate matrices that relate the weights of the basis functions (also referred to as amplitudes) to corresponding electromagnetic fields. Using SECS in combination with the cubed-sphere grid and associated differentiation matrices offers at least two significant advantages: (i) It gives a convenient way to introduce prior information about spatial structures when regularizing inverse problems to find a set of SECS amplitudes (Laundal et al., 2021), and (ii) it allows us to relate different quantities like electric fields and currents, through the ionospheric Ohm's law. In **Section 3.4**, we present an example where we use this property, which is fundamental in the Lompe technique, to calculate ground magnetic field perturbations associated with a Weimer (2005a), Weimer (2005b) electric potential assuming uniform conductivity.

In the Lompe technique, magnetic and electric fields are related *via* the ionospheric Ohm's law. Use of this equation requires that the electric field is given in the reference frame of the neutrals. In all the examples in this paper and in Laundal et al. (2022c), the neutral wind is assumed to be zero in an Earth-fixed frame. The ionospheric Ohm's law equation also involves the ionospheric conductance and the main magnetic field of the Earth. In **Section 4**, we discuss Python implementations of empirical models that can be used to specify these quantities. In particular, a new method has been developed to calculate sunlight-produced conductance that avoids infinite gradients at the sunlight terminator, yet scales with frequently employed empirical relationships valid at smaller solar zenith angles (Moen and Brekke, 1993).

The Lompe code also contains tools for visualizing ionospheric electrodynamics on both cubed-sphere projections and in polar coordinates. In **Section 5**, we discuss the polar coordinate visualization tool, which is essentially a wrapper for many Matplotlib functions, where Cartesian coordinates are replaced with latitude and local time.

Throughout the paper we do refer to specific variables (classes, functions, *etc.*,) to make them easy to find, but the focus is on the concepts rather than syntax. For details we refer to the doc strings and the extensive example notebook available in the code repository (Laundal et al., 2022a). While the code was designed with real data in mind, we use the same Weimer (2005a), Weimer (2005b) electric potential as the only input in most examples in this paper. We believe this gives a logical progression from the cubed-sphere projection and grid, and associated differentiation matrices (**Section 2**); to the combination with SECS analysis (**Section 3**); and finally, to the full Lompe technique, demonstrated in **Section 6**. **Table 1** gives an overview of the sections describing the various parts of the Lompe code.

2 Cubed-sphere projection and grid

The basis of the numerical implementations in Lompe is a grid in the cubed-sphere projection (Ronchi et al., 1996). The ionosphere is modeled as a two-dimensional spherical shell at radius R_I (in this paper we use $R_E + 110$ km, where R_E is the mean Earth radius), and the cubed-sphere projection maps every point of the sphere onto a circumscribed cube by extending the line that connects the center of the Earth and the position on the sphere until it intersects the cube. A significant advantage of an analysis grid in the cubed-sphere projection is that regular grids become free of any singularity, avoiding the numerical difficulties of the poles that are present when using spherical polar coordinates

TABLE 1 Overview of the sections describing the Lompe code.

Section number	Section title
2	Cubed-sphere projection and grid
2.1	Numerical differentiation on a cubed-sphere grid
2.2	Example: Solving the 2D continuity equation
3	Spherical Elementary Current System analysis with Python
3.1	SECS analysis of ground magnetometer data
3.2	SECS analysis of satellite magnetometer data
3.3	Other features of the `secsy` SECS functions
3.4	Example: SECS analysis on cubed-sphere grid
4	Empirical models
4.1	Main magnetic field
4.2	Conductance models
5	Polplot: visualization in polar coordinates
6	Lompe

(often referred to as the "pole problem"). In addition, regular grids are almost equal area in the cubed-sphere projection.

Cubed-sphere projections and grids are handled in the `cubedsphere` script, a part of the `secsy` module. The purpose of the module is to facilitate regional data analyses such as Lompe, and, as we will demonstrate, it can also be used for solving certain partial differential equations on a sphere. The module implements a cubed-sphere projection (`CSprojection` class) on one face of a cube that has an orientation with respect to the sphere that the user specifies. A regular grid (`CSgrid` class), centered at the touch point between the surface and the sphere, can be set up to cover a region of interest. The grid resolution can be specified in each direction. With the current implementation, the grid is not intended for global analyses since only one cube face is used. The projection is illustrated in **Figure 1A**. The figure shows a cross-section of the cube face intersecting a sphere with radius R_I. The grid is equally spaced in angular coordinates (ξ and η, following the notation in Ronchi et al., 1996). Regular grid cells in ξ, η coordinates are projected to increasingly larger cells on the cube face (**Figure 1A**) and increasingly smaller cells on the sphere as the distance from the intersection point between the cube face and the sphere increases. However, as seen in **Figure 1B**, the variation in grid cell area on the sphere is small compared to a regular grid in spherical coordinates. The grid projected on the sphere is non-orthogonal. The non-orthogonality is taken into account in all conversions and calculations performed with `cubedsphere`.

The flexibility in choosing the cube's orientation makes it easy to set up rectangular grids that cover a specific region. For example, in the Observing System Simulation Experiment (OSSE) presented by Laundal et al. (2021), cubed-sphere grids were aligned with simulated satellite tracks for the upcoming Electrojet Zeeman Imaging Explorer (EZIE) mission (Yee et al., 2021). The three EZIE satellites will use the Zeeman effect to give multi-point magnetic field measurements at ~85 km altitude. These measurements will be interpolated to a 2D image of the electrojet by using spherical elementary currents on a cubed-sphere grid. In **Section 3**, we discuss how to use the `secsy` module for such analyses.

2.1 Numerical differentiation on a cubed-sphere grid

The `CSgrid` class facilitates numerical differentiation of functions that are defined on the grid. The `CSgrid.get_Le_Ln()` function returns two $N \times N$ matrices, $\mathbb{D}_{\mathbf{e}\cdot\nabla}$ and $\mathbb{D}_{\mathbf{n}\cdot\nabla}$, where N is the number of grid cells. The matrices yield the eastward and northward components of the gradient of a scalar field that is defined on the N cells of the `CSgrid` object, respectively. That is, the matrix elements are defined by the combination of the finite central difference scheme for differentiation in ξ, η coordinates, the conversion between cubed-sphere coordinates and spherical coordinates, and the orientation and position of the cube face with respect to the underlying global coordinate system (for the example in **Figure 1**, we use centered dipole coordinates). The conversions between cubed-sphere and spherical coordinates are given by Ronchi et al. (1996) and take the non-orthogonality of the projected coordinates into account. Forward/backward difference schemes are used near the edges, and the size of the stencil used to calculate the elements of the differentiation matrices can be chosen by the user. These differentiation matrices can, for example, be applied to a steady-state electric potential Φ to yield the ionospheric electric field, $\mathbf{E} = -\nabla\Phi$. To illustrate this capability, we use an electric potential from the empirical Weimer (2005a), Weimer (2005b) model, with solar wind velocity 350 km/s, IMF B_y = 0 nT, B_z = −4 nT, solar wind density 2 particles per cm^3, and dipole tilt angle 25° (referred to as the "Weimer potential" for the rest of the paper). With the Weimer potential interpolated to a cubed-sphere grid, the electric field can be calculated as

$$\begin{aligned} \underline{E}_e &= -\mathbb{D}_{\mathbf{e}\cdot\nabla}\underline{\Phi}, \\ \underline{E}_n &= -\mathbb{D}_{\mathbf{n}\cdot\nabla}\underline{\Phi}, \end{aligned} \tag{1}$$

where $\underline{\Phi}$, $\underline{E}_e$, and $\underline{E}_n$ are column vectors containing elements corresponding to the N cells in the `CSgrid`; the Weimer potential, the eastward component of the electric field, and the northward component of the electric field, respectively. We use the underline to indicate column vectors that represent a set of scalar field values throughout the paper. **Figure 1C** shows the Weimer potential in the grid projection, while the derived

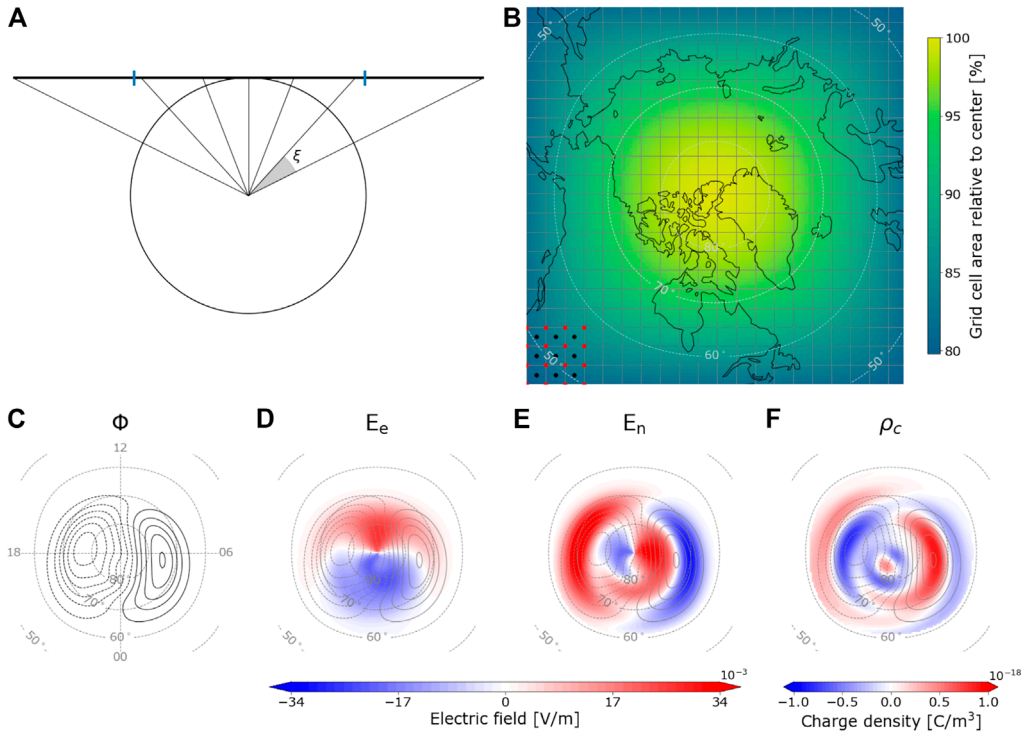


FIGURE 1
Example use of `cubedsphere`. **(A)** Cross-section of a cube face (bold black line) tangent to a sphere with radius R_I. In this plane, ξ appears as a polar angle and η as azimuth angle; in the perpendicular plane the roles are reversed. Blue markers show the extent of the example grid in panel **(B)**. **(B)** An example cubed-sphere grid centered at the north geomagnetic pole. The black dots in the lower left corner represent grid cell center points (the `lat`, `lon`, `xi`, and `eta` variables in the `CSgrid` class), and the red dots represent grid cell corners (specified in class variables `lat_mesh`, `lon_mesh`, *etc.*,). The color shading shows the relative area of the grid cells. Coastlines are plotted in the cubed-sphere projection. **(C)** Electric potential contours from the Weimer (2005a,b) model interpolated on the cubed-sphere grid. **(D)** Colors show the eastward electric field described by the Weimer potential (gray contour). **(E)** Same as panel D, but for the northward electric field. **(F)** Charge density calculated from the divergence of the electric field shown in panels **(D,E)**. The quantities in panels **(D–F)** are calculated with the finite element differentiation matrices of `CSgrid`. Note that the differentiation is performed with a higher resolution grid than what is shown in panels **(A,B)**, which is down-scaled for illustration purposes. Panels **(C–F)** use cubed-sphere projections, oriented such that noon magnetic local time is on top and midnight at bottom [indicated in panel **(C)**].

eastward and northward components of the electric field are shown in panels D and E.

The divergence of a vector field defined on the N cells in the `CSgrid` can be found using the $N \times 2N$ matrix $\mathbb{D}_\nabla$. returned from `CSgrid.divergence()`. This divergence matrix is also implemented using a finite difference scheme. It operates on a $2N$ element column vector comprised of the eastward and northward components of a vector field defined on the grid and stacked on top of each other. For example, Gauss' law says that the electric charge is $\rho_c = \epsilon_0 \nabla \cdot \mathbf{E}$, which means that we can use the set of vector components found above, $\underline{E}_e$ and $\underline{E}_n$, to calculate the electric charge density associated with the Weimer potential in every grid cell

$$\underline{\rho}_c = \epsilon_0 \mathbb{D}_\nabla \cdot \begin{pmatrix} \underline{E}_e \\ \underline{E}_n \end{pmatrix}, \tag{2}$$

Assuming that the radial derivative of the electric field is zero. ϵ_0 is the vacuum permittivity. The charge density is shown in Figure 1F.

There are also methods in the `CSgrid` class that can be useful for working with observational data on cubed-sphere grids. Given the observation locations, `ingrid()` can be used to check if observations are located within the grid. The indices of the grid cells in which the observations are located can be found using `bin_number()`. There is also a possibility to use `count()` to obtain the total number of observations within each grid cell.

2.2 Example: Solving the 2D continuity equation

The differentiation matrices in `CSgrid` can be used to solve certain partial differential equations. In this section, we show

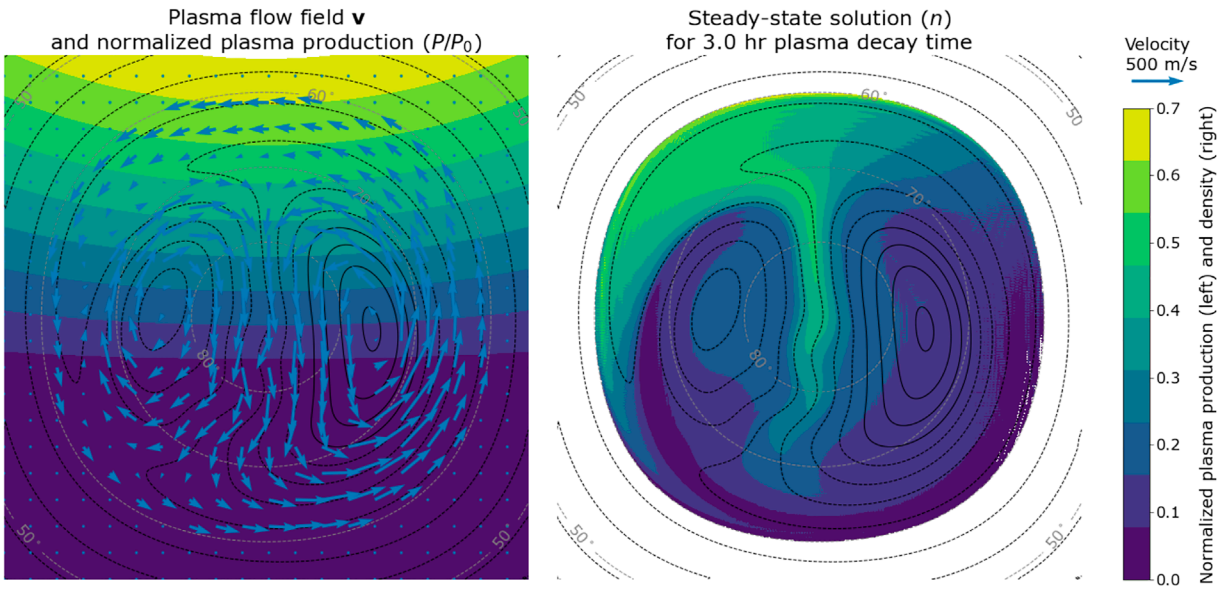


FIGURE 2
Left: Normalized plasma production P (colors), electric potential (black contours), and resulting velocity field **v** (above 60° latitude, blue arrows). Right: Normalized steady-state solution for the plasma density n (colors) assuming a 3-h plasma decay time, and electric potential (black contours).

an example where they are used to solve the 2D steady-state continuity equation on a domain covered by a `CSgrid` object. We want to find the resulting distribution of plasma density, n, given a plasma production function, P, a 2D velocity field specified by v_e and v_n, and a plasma decay factor β. For an incompressible plasma, we have that

$$\mathbf{v} \cdot \nabla n = P - \beta n. \tag{3}$$

With P, v_e, v_n and n defined on each cell of a `CSgrid`, Eq. 3 can be written as a matrix equation,

$$\left[\mathrm{diag}\left(\underline{v}_e\right)\mathbb{D}_{\mathbf{e}\cdot\nabla} + \mathrm{diag}\left(\underline{v}_n\right)\mathbb{D}_{\mathbf{n}\cdot\nabla} + \beta\mathbb{I}\right]\underline{n} = \underline{P}. \tag{4}$$

The differentiation matrices $\mathbb{D}_{\mathbf{e}\cdot\nabla}$ and $\mathbb{D}_{\mathbf{n}\cdot\nabla}$ can be returned as `scipy.sparse` matrices, which is useful in applications like this, since most of the elements are zero. $\mathbb{I}$ is the $N \times N$ identity matrix. Solving the continuity equation is now only a matter of inverting the matrix in square brackets in Eq. 4.

An example production function P and velocity field **v** are shown in the left panel of Figure 2. In this example P represents solar EUV ionization in proportion to $\cos(\chi)$ (Ieda et al., 2014), where χ is the solar zenith angle, and the subsolar point is located at 10° latitude and noon local time (up in the figure). In Figure 2 the production function is normalized as P/P_0, where P_0 is the plasma production rate at the subsolar point. The velocity field is based on the Weimer potential. It is shown as vectors and as contours of constant electric potential, Φ, where $-\nabla\Phi = -\mathbf{v} \times \mathbf{B}$. We assume a dipole magnetic field with mean magnetic field $B_0 = 30\ \mu$T. In this example, we have transformed the electric potential and velocity field to an inertial frame by adding co-rotation (Laundal et al., 2022b). The decay factor, β, is set to 1/(3 h), i.e., the plasma decay factor is such that it takes 3 h for the plasma to decay by factor $1/e$.

The solution plasma density, n, is shown to the right in Figure 2. Like the production function, the density is also normalized, by dividing by P_0/β, which is the solution to Eq. 3 at the subsolar point, where $\nabla n = 0$. The inversion of the matrix in square brackets in Eq. 4 gives meaningful results only within closed convection contours, within which the boundary value problem is well defined. We therefore mask densities equatorward of 60° latitude, where co-rotation dominates. We see that poleward of 60°, the density pattern has features that are well known from studies of the long-lived F-region plasma: A tongue of ionization in the central region, due to anti-sunward transport of plasma produced at lower latitudes, and a mid-latitude trough in the dusk return flow region (Kelley, 2009).

3 Spherical elementary current system analysis with Python

The `secsy` module contains functions which facilitate SECS analyses. SECS are basis functions that were originally used for regional analyses of ionospheric current systems (Amm, 1997). They can be used as alternatives to spherical harmonics when the focus is on localized regions rather than global patterns.

There are two types of SECS, describing divergence-free and curl-free vector fields on a spherical shell. Both divergence-free and curl-free basis functions describe a global 2D vector field that decreases in amplitude as $1/\tan(\theta/2)$, where θ is the polar angular distance from the basis function's "pole". This functional form implies that the amplitudes fall off rapidly; each basis function has a short range even though they are, in principle, global. According to Vanhamäki and Amm (2011) and the Helmholtz theorem, by placing the SECS basis functions sufficiently close and choosing their amplitudes appropriately, their sum can represent any well-behaved 2D vector field on a spherical shell.

3.1 SECS analysis of ground magnetometer data

So far, SECS have primarily been used for analyses of ground magnetometer data. Given a set of simultaneous measurements from ground magnetometer stations, divergence-free SECS can be used to estimate an equivalent overhead current sheet density, and a corresponding magnetic field everywhere within the analysis region. The divergence-free equivalent current $\mathbf{J}^\circ$ at radius R_I, is represented with SECS as

$$\mathbf{J}^\circ(\mathbf{r}) = \sum_i \frac{S_i^\circ}{4\pi R_I} \cot\left(\frac{\theta_i}{2}\right) \hat{\mathbf{e}}_i, \tag{5}$$

where θ_i is the colatitude of the location $\mathbf{r}$ in a coordinate system where the location of the ith SECS basis function defines the north pole, and $\hat{\mathbf{e}}_i$ is an eastward unit vector in that coordinate system. S_i° represents the amplitude of the ith SECS basis function. As in Laundal et al. (2022c), the superscript ° signifies "divergence-free".

The `secsy` module contains functions that calculate matrices that relate a set of divergence-free SECS amplitudes, S_i°, to the corresponding divergence-free current at any given set of coordinates on the spherical shell (`get_SECS_J_G_matrices()`). In addition, `get_SECS_B_G_matrices()` returns matrices that relate SECS amplitudes to corresponding magnetic field perturbations at any given set of coordinates at any altitude. This function uses the analytical expressions derived by Amm and Viljanen (1999). Together, these two functions can be used to calculate a map of the equivalent current through the following steps:

1. Get design matrix $\mathbb{H}_g^\circ$ that relates a set of measured magnetic field components $\Delta\underline{B}_g$ to a set of divergence-free SECS amplitudes $\underline{S}^\circ$ (containing the amplitudes S_i°) from `get_SECS_B_G_matrices()`.
2. Solve the inverse problem $\Delta\underline{B}_g = \mathbb{H}_g^\circ \underline{S}^\circ$ for $\underline{S}^\circ$
3. Get design matrix $\mathbb{J}^\circ$ that relates a set of divergence-free current densities, $\underline{J}^\circ$, to $\underline{S}^\circ$ from `get_SECS_J_G_matrices()`.
4. Calculate the current densities as $\underline{J}^\circ = \mathbb{J}^\circ \underline{S}^\circ$.

If the task is to interpolate magnetometer measurements, $\mathbb{J}^\circ$ in the last two steps can be replaced with a different $\mathbb{H}_g^\circ$ matrix, that relates the amplitudes $\underline{S}^\circ$ to the magnetic field at the interpolation points.

The above procedure focuses on ground magnetometers, but the `get_SECS_B_matrices()` function accepts evaluation points at any radius. It can thus also be used in analyses of magnetometer data from higher altitudes; below the ionospheric current layer at R_I, for example with data from the upcoming EZIE mission (Laundal et al., 2021), or above the current layer, for example with data from low-flying satellites like *Swarm* or CHAMP (Laundal et al., 2016).

Note that `get_SECS_B_matrices()` and `get_SECS_J_matrices()` return multiple matrices, one for each vector component, instead of the single matrix in this example. The component matrices can be stacked vertically to form a single composite matrix that calculates all the desired vector components. Note also that the functions accept NumPy arrays as input and that all calculations are vectorized and therefore fast.

3.2 SECS analysis of satellite magnetometer data

The above example assumes that only the divergence-free part of the horizontal ionospheric current contributes to the observed magnetic field. This is true for ground observations; according to the Fukushima theorem, the magnetic fields of field-aligned currents and associated horizontal curl-free currents cancel below the ionosphere in polar regions where the main magnetic field is approximately radial (Fukushima, 1994). In space, above the horizontal current, we must include the curl-free current system in the analysis. The curl-free current can be represented with SECS as

$$\mathbf{J}^\star(\mathbf{r}) = \sum_i \frac{-S_i^\star}{4\pi R_I} \cot\left(\frac{\theta_i}{2}\right) \hat{\mathbf{n}}_i, \tag{6}$$

where the superscript * signifies "curl-free", and $\hat{\mathbf{n}}_i$ is a northward unit vector in a coordinate system that has the ith SECS basis function in the north pole. The `get_SECS_J_G_matrices()` and `get_SECS_B_G_matrices()` functions can be used with curl-free currents in the same way as for divergence-free currents, except that the `current_type` keyword should be set to "`curl_free`" instead of "`divergence_free`". The curl-free amplitudes, $S_i^\star$, can be thought of as electric line currents that flow down to or up from the spherical shell at radius R_I, extending to infinity. Therefore, its unit is Ampere. The corresponding horizontal curl-free current [Ampere per meter] distributes this current across the globe, with a uniform radial divergence with the opposite sign as $S_i^\star$, ensuring current continuity (Amm, 1997).

3.3 Other features of the secsy SECS functions

The SECS functions in `secsy` also support a number of features that are often useful in SECS analysis.

The SECS basis functions are infinite at $\theta_i = 0$, which can cause numerical problems. To avoid this, a modification is often applied poleward of some limit θ_0. This modification, described in detail by Vanhamäki and Juusola (2020), can be applied with the SECS functions by specifying θ_0 with the `singularity_limit` keyword. Note that the singularity modification is applied to both types of currents but not to the magnetic field of the divergence-free current. The reason is that the magnetic field of the modified $\mathbf{J}^\circ$ likely does not have an analytic expression, and that the modification would be minimal since the ground magnetic field is usually evaluated at radii where the effect of the singularity is greatly reduced (Vanhamäki and Juusola, 2020).

Magnetic disturbances observed with ground magnetometers are not only associated with currents in space, but also with induced currents in the conducting Earth. The magnetic field of ground-induced currents can be taken into account in SECS analyses in at least two ways: (i) They can be modeled directly, in the same way as ionospheric currents, by placing divergence-free SECS poles at some radius below ground, or (ii) they can be modeled as so-called *image currents*. The image current method assumes that there is a super-conducting layer in the Earth's interior that exactly cancels the radial magnetic field of the ionospheric currents at some radius R_C. Juusola et al. (2016) showed how the magnitudes of the image currents relate to the corresponding ionospheric currents. The effect of the image current is to change the radial dependence of the magnetic field, and this can be included in secsy SECS analyses by specifying R_C through the `induction_nullification_radius` in calls to `get_SECS_B_G_matrices()`. An advantage of the image current method is that it does not add any degrees of freedom to the SECS model since the image current amplitudes are given by the ionospheric current amplitudes. A disadvantage is that it does not account for the effects of finite and non-uniform ground conductivities.

The curl-free SECS basis functions can–since they have zero curl–be written as gradients of scalar fields (potentials). That is, Eq. **6** can be written as

$$\begin{aligned}\mathbf{J}^\star &= \sum_i \frac{-S_i^\star}{4\pi R_I}\cot\left(\frac{\theta_i}{2}\right)\hat{\mathbf{n}}_i = \nabla V \\ &= \nabla\left[\sum_i \frac{-S_i^\star}{2\pi}\ln\left(\sin\left(\frac{\theta_i}{2}\right)\right)\right].\end{aligned} \tag{7}$$

The scalar potential representation can be useful in studies of ionospheric convection electric fields, assuming that Faraday's law can be set equal to zero. Reistad et al. (2019) used SuperDARN (Chisham et al., 2007) line-of-sight measurements of ionospheric convection to constrain curl-free SECS representations of convection electric fields and visualized the result by plotting equipotential contours. The same approach is used in the Lompe technique (Laundal et al., 2022c). The corresponding curl-free SECS amplitudes can be interpreted in terms of electric charges: Each basis function represents a line that extends from the base of the ionosphere to infinity, and the amplitude is equal to the electric line charge density. The potential representation on the right hand side of Eq. **7** can be calculated with the matrix returned by `get_SECS_J_G_matrices()` with the *current_type* keyword set to "`potential`".

3.4 Example: SECS analysis on cubed-sphere grid

In this section, we show an example where `secsy` design matrix functions and cubed-sphere grids are used to estimate magnetic perturbations on ground given the Weimer potential, and an assumption of constant ionospheric conductances. Without gradients in the Pedersen or Hall conductances, the divergence of the height-integrated ionospheric Ohm's law reduces to

$$j_\parallel = \Sigma_P \nabla \cdot \mathbf{E}, \tag{8}$$

where Σ_P is the Pedersen conductance, $j_\parallel$ is the field-aligned current density, and $\mathbf{E}$ is the ionospheric electric field.

Consider a set of curl-free SECS poles in the center of the cells of the cubed-sphere grid in **Figure 1B**. The curl-free amplitude in the ith grid cell, $S_i^\star$, represents an electric line current flowing into or out of the spherical shell at radius R_I, i.e., the total field-aligned current integrated over the cell. In **Section 2**, we obtained the divergence of the Weimer ionospheric electric field defined on the grid cells (**Figure 1F** shows this quantity multiplied by ϵ_0). The corresponding FAC densities can be found through Eq. **8** if Σ_P is known. Then $S_i^\star$ can be found directly through multiplication with the grid-cell area, A_i,

$$S_i^\star = j_{\parallel,i} A_i. \tag{9}$$

Recall that only the divergence-free part of the horizontal ionospheric current, $\mathbf{J}^\circ$, contributes to the observed magnetic field on ground, and the divergence-free SECS amplitudes, S_i°, that scale $\mathbf{J}^\circ$ must be found. With no conductance gradients and a constant Hall-to-Pedersen conductance ratio $\alpha = \Sigma_H/\Sigma_P = 1$, the curl-free and divergence-free SECS amplitudes are related as (Amm et al., 2002; Juusola et al., 2009)

$$S_i^\circ = -S_i^\star. \tag{10}$$

By assuming that both the Hall and Pedersen conductances are 10 mho across the grid, we get a vector $\underline{S}^\circ$ containing the divergence-free SECS amplitudes for all cells on the grid. The

corresponding magnetic perturbations for a set of locations on ground, $\Delta\underline{B}_g$, can be calculated as

$$\Delta\underline{B}_g = \mathbb{H}_g^{\circ}\underline{S}^{\circ}, \tag{11}$$

where $\mathbb{H}_g^{\circ}$ is the matrix that relates the SECS amplitudes ($S_i{}^{\circ}$, contained in $\underline{S}^{\circ}$) to ground magnetic field vector components. The function `get_SECS_B_G_matrices()` calculates $\mathbb{H}_g^{\circ}$ given the coordinates and radii of the evaluation locations, and the coordinates of the SECS poles, in this case the centers of the cells in our cubed-sphere grid. In this example, the evaluation locations are at radius R_E and are set to the corners of the grid cells. The magnetic disturbance on ground is shown in **Figure 3**. The horizontal magnetic field is shown as black arrows, while the upward magnetic field is shown with colors. The gray contours represent the Weimer potential.

The Lompe technique uses the same approach as above except that the conductances are not assumed to be uniform. In Lompe, the full expressions for the divergence and curl of the ionospheric Ohm's law are taken into account by using the `CSgrid` differentiation matrices that were introduced in **Section 2**.

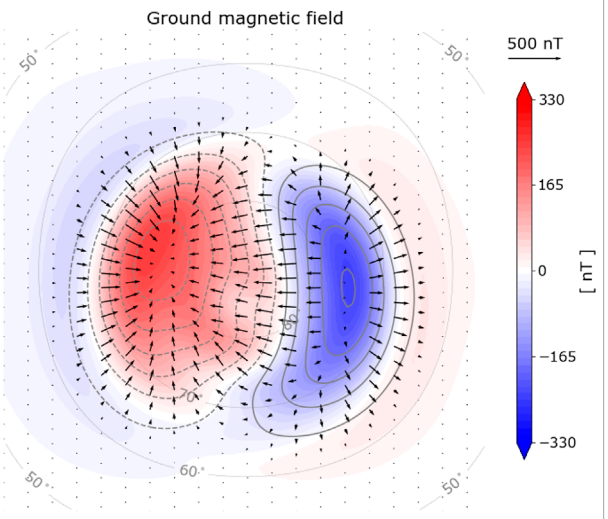


FIGURE 3
Example of how the `secsy` module can be used to relate an ionospheric electric potential to magnetic field disturbances on ground in the polar region, assuming uniform ionospheric conductance. Results are shown in the cubed-sphere projection. The black arrows represent the horizontal components of the magnetic field disturbance, and color contours represent the upward magnetic field disturbance. The gray contours represent the Weimer ionospheric electric potential used throughout this paper.

4 Empirical models

In the Lompe technique, magnetic and electric fields are related through the ionospheric Ohm's law, and the electric field and F-region ion velocity are related through the generalized Ohm's law. The ionospheric Ohm's law involves ionospheric conductances, and both equations involve the main magnetic field of the Earth. The Lompe code includes modules to estimate these quantities, discussed briefly in this section.

4.1 Main magnetic field

The International Geomagnetic Reference Field (IGRF) is a standard model of the Earth's magnetic field, maintained by the International Association of Geomagnetism and Aeronomy (IAGA). The IGRF model represents the magnetic field as a set of spherical harmonics, with a new set of coefficients every 5 years to account for temporal changes. Linear interpolation of the model coefficients is used between versions. The most recent version was presented by Alken et al. (2021). The Lompe technique requires magnetic field values on every grid point. When the grid is defined in geographic coordinates (default), IGRF magnetic field values are calculated with the `ppigrf` module (Laundal, 2022), which is a pure-Python implementation of the IGRF that gives IGRF model predictions given position and date. The position can be specified in either geodetic or geocentric coordinates. While other Python modules that calculate IGRF values exist, many of them are wrappers of Fortran code, which can be tricky to compile. Despite being a pure-Python implementation, the IGRF calculations are quite fast since `ppigrf` is fully vectorized.

For some applications IGRF is not the appropriate model. For example, in this paper our examples are based on the statistical Weimer (2005a), Weimer (2005b) model of electric potential, which is given in magnetic latitude and local time. Since longitude information is missing, it is more appropriate to use a dipole magnetic field, since it is symmetric about the dipole axis. To accommodate such cases, the Lompe code includes an option to use dipole coordinates. This is accomplished using another submodule to Lompe, `dipole`, which contains functions to calculate dipole magnetic field values, and to convert between geocentric and dipole coordinates. The `dipole` module also contains functions to convert between magnetic local time and magnetic longitude using Equation (93) of Laundal and Richmond (2017). The `dipole` module uses the `ppigrf` module to extract the first three spherical harmonic coefficients, which defines the centered dipole, for any given epoch covered by the IGRF.

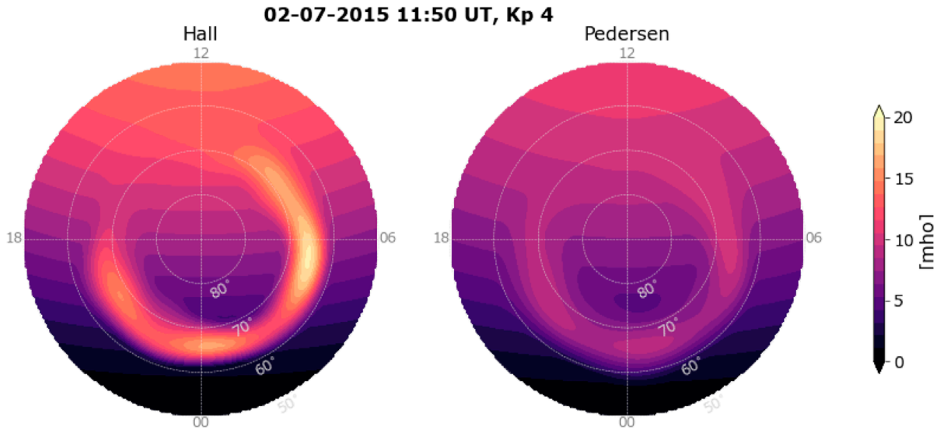


FIGURE 4
Example Hall and Pedersen conductances where solar conductances are like on 2 July, and auroral conductances are Hardy et al. (1987) model patterns for Kp 4. The axes have magnetic latitude and local time (centered dipole) coordinates, and show a region poleward of 50° magnetic latitude. Magnetic noon is up.

4.2 Conductance models

The interpretation of ground magnetometer data in terms of high latitude ionospheric convection requires knowledge about ionospheric conductances, the height-integrated conductivities (e.g., Kamide et al., 1981). The ionospheric conductance depends primarily on the ionization by solar EUV radiation and the contribution to ionization from precipitating particles (auroral conductance). The Lompe code also contains functions for calculating the solar EUV conductances and the auroral conductances from the Hardy et al. (1987) empirical model.

A novel method for calculating the solar EUV conductance, Σ_{EUV}, is implemented in the `EUV_conductance()` function. The method uses a modified version of the empirical model from Moen and Brekke (1993) where $\cos(\chi)$ is replaced with a function $q'(\chi)$ that specifies the relative maximum production due to solar EUV assuming a radially stratified atmosphere, with χ the solar zenith angle. The full technique is explained in Section 2.4 in Laundal et al. (2022c). This adjustment gives EUV conductances without infinite gradients at the sunlight terminator. The Hall and Pedersen conductances are calculated given a solar radio flux index, $F_{10.7}$, and a set of solar zenith angles corresponding to the locations of interest. Functions in the `sunlight` module can be used to calculate χ. Σ_{EUV} is by default scaled to coincide with the empirical model by Moen and Brekke (1993) at low solar zenith angles (other empirical models can be chosen using the `calibration` keyword).

For auroral conductances, $\Sigma_{auroral}$, the `hardy()` function is an implementation of the Hardy et al. (1987) model, which is based on satellite observations of precipitating particles. Given a Kp index and coordinates in magnetic latitude and local time, the method returns empirical Hall and Pedersen conductances. It is difficult to know the auroral conductances precisely, mainly because of the high variability in the auroral precipitation. The Hardy et al. (1987) model function is only meant to be a rough estimate in applications of the Lompe technique. If possible, better auroral conductance estimates should be obtained from observations such as auroral images.

The `hardy_EUV()` function combines the implementations for solar and auroral conductance contributions and returns the total conductances given a set of coordinates. In the latest version of the Lompe code (v1.1), the total conductances are calculated using the vector sum of the solar EUV and auroral contributions, $\Sigma_{total} = \sqrt{\Sigma^2_{EUV} + \Sigma^2_{auroral}}$ (Robinson et al., 2020). Example output from the conductance function is shown in **Figure 4**, where the Hall conductance (left) and Pedersen conductance (right) are plotted for a region poleward of 50° magnetic latitude. The solar zenith angles are calculated for 2 July, and the auroral conductances are patterns for Kp 4.

5 Polplot: Visualization in polar coordinates

The `polplot` module, which is included as a submodule to Lompe, is useful for visualizing data in a polar coordinate system, specifically a latitude and local time grid. Given a Matplotlib axis object, an object of the `Polarplot` class returns a polar axis centered at the pole, where noon is at the top and dusk to the left. Most `Polarplot` plotting functions are equivalent to the corresponding Matplotlib function, and keyword arguments accepted by `pyplot` functions (such as color, linewidth, zorder, *etc.*,) can also be given to the `Polarplot` functions.

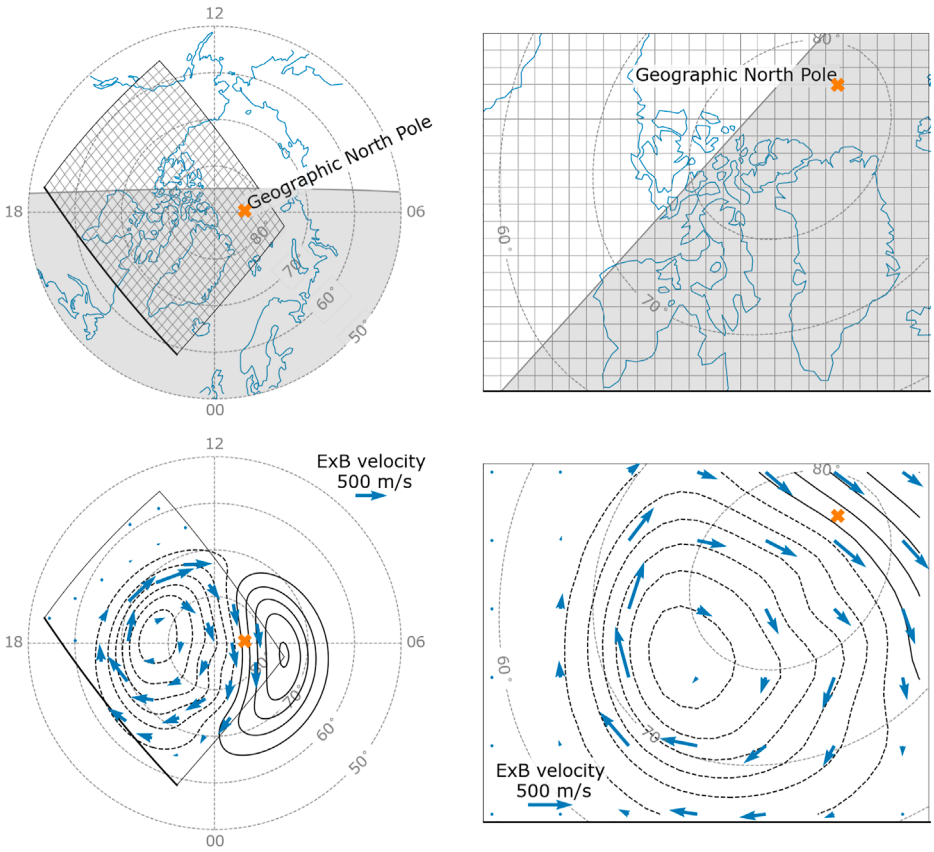


FIGURE 5
Example of different representations of data on `Polarplot` polar axes (left), and Cartesian axes (right). The left panels have coordinates in magnetic latitude and local time, and show a region poleward of 50° magnetic latitude. The black frames show the extent of the grid in the right panels. The bold edge corresponds to the lower edge of the grid projections. The right panels show data in the cubed-sphere projection. The cubed-sphere grid is 7,000 × 5,000 km defined on a cube face centered at 88°W and 72°N and rotated 45°. The gray mesh in the top panels is the grid, where each cell is 200 × 200 km. In the top panels, coastlines are shown in blue, and regions where the solar zenith angle is ≥90° (sunlight terminator) are shaded gray. The black contours in the bottom panels represent the ionospheric electric potential. The blue arrows represent the $\mathbf{E} \times \mathbf{B}$ convection velocities. An orange X represents the Geographic North Pole in all panels.

For example, `polplot` was used when making **Figure 4**, where `Polarplot.contourf()` made the filled contours representing the ionospheric conductances in a magnetic latitude and local time system.

Figure 5 shows examples of data visualization in polar coordinates and in a cubed-sphere projection. Coordinates are given in magnetic latitude and local time for the polar axes (left). In the plot to the right, the grid ξ, η coordinates are treated as Cartesian coordinates on the Matplotlib axes. An orange X marks the Geographic North Pole. Black rectangles on the polar axes show the extent of a cubed-sphere grid covering much of North America and Greenland. The bold grid edge corresponds to the lower edge of the grid. The pairs of panels in the two rows show the same grids. The gray meshes in the top row panels represent the grid cells. Coastlines, converted to magnetic apex coordinates, are added to the polar axis using the `coastlines()` function. The cubed-sphere projection class has a function called `get_projected_coastlines()` that returns coastlines projected to cubed-sphere ξ, η coordinates. The gray line in both panels marks the sunlight terminator as it is on 10 March at midnight UT. Regions where the solar zenith angle is more than 90° are shaded gray using the `plot_terminator()` function, which also adds the location of the terminator to the polar axis. The bottom row panels show the Weimer potential as black contours. Blue arrows show the $\mathbf{E} \times \mathbf{B}$ convection velocity resulting from the electric field described by the Weimer potential in a co-rotating frame.

6 Lompe

In this section we demonstrate the full Lompe technique, which combines the modules from the previous sections. The Lompe technique (Laundal et al., 2022c) is implemented in

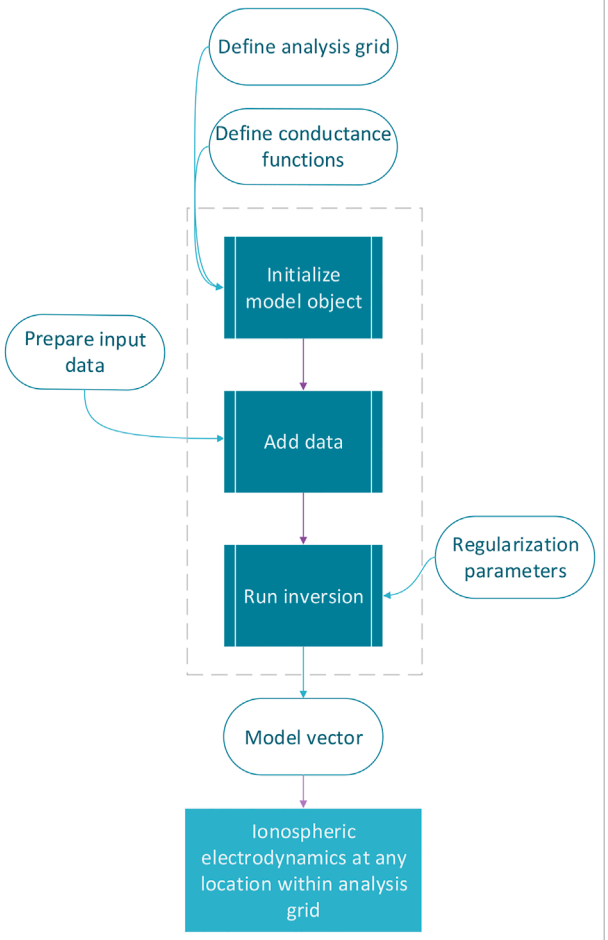


FIGURE 6
Flow chart showing the basics steps in the Lompe workflow.

the model module. The technique relates vector components of ionospheric convection electric fields, $\underline{E}$, F-region plasma convection velocities, $\underline{v}$, ground magnetic field disturbances, $\underline{B}_g$, and space magnetic field disturbances, $\underline{B}_s$, to a model vector $\underline{m}$ through a set of linear equations

$$\begin{pmatrix} \underline{E} \\ \underline{v} \\ \underline{B}_g \\ \underline{B}_s \end{pmatrix} = \underline{y} = \begin{pmatrix} \mathbb{E} \\ \mathbb{V} \\ \mathbb{B}_g \\ \mathbb{B}_s \end{pmatrix} \underline{m} = \mathbb{G}\underline{m}. \tag{12}$$

In Lompe, the model vector, $\underline{m}$, contains the amplitudes of the curl-free SECS basis functions representing the convection electric field. The different block matrices relate different quantities to the model vector $\underline{m}$: $\mathbb{E}$ relates electric field vector components to $\underline{m}$ through Eq. 6, $\mathbb{V}$ relates F-region plasma velocity components to $\underline{m}$ by assuming that the electric field and plasma are frozen-in, $\mathbb{B}_g$ and $\mathbb{B}_s$ relate magnetic field vector components below (subscript g) and above (subscript s) the ionosphere to $\underline{m}$ by using the ionospheric Ohm's law. Use of the ionospheric Ohm's law requires that conductances are known. The conductance determines both a scale factor–how strong the electric fields have to be in order to explain the observed magnetic field–and the distribution of the electric field. If magnetic field measurements are used in the inversion, it is therefore highly recommended to use Lompe in cases when the conductances are precisely known. The Lompe conductance module (**Section 4.2**) includes functions that yield precise estimates of solar EUV-induced conductance, but auroral conductance should preferably come from measurements. The total design matrix $\mathbb{G}$ thus depends on ionospheric conductances, the choice of grid, and the coordinates of the model predictions, $\underline{y}$. For a thorough description of how $\mathbb{G}$ is calculated, see Section 3.2 in Laundal et al. (2022c).

Figure 6 gives an overview of the various steps to carry out a Lompe inversion. The first step is setting up a cubed-sphere grid that covers the region we want to model. The location, orientation, size, and resolution of the grid should be adapted to the input data coverage. In addition, functions for calculating the Hall and Pedersen conductances are required. The Lompe `Emodel` is then initialized given the grid and conductance functions. `Emodel` assumes all input in geographic coordinates by default, but the `dipole` keyword can be used to make all calculations in centered dipole coordinates, and with a centered dipole magnetic field instead of the IGRF (see **Section 4.1**).

The next step is to add the input data ($\underline{y}$ in the equation above). The input data to a Lompe `Emodel` must be an object of the `Data` class, which contains the data values (vector components) and coordinates. To help set up a `Data` object, the Lompe package includes a stand-alone module called `dataloader`. It contains functions that work with convection data from DMSP satellites (Rich, 1994) (from the CEDAR Madrigal database), the SuperMAG network of magnetometers (Gjerloev, 2012), the network of SuperDARN radars (Chisham et al., 2007), and AMPERE's Iridium magnetometer data (Waters et al., 2020).

When initializing `Data` objects, the measurements go through sanity checks that ensure the input data is of the correct shape and with valid values, and NaNs are removed from the data set. All data should be given in SI-units. Coordinates and components should be given in geographic coordinates, unless the `Emodel` object is initialized with the `dipole` keyword. If not all vector components are known, the `components` parameter can be used to indicate which of the eastward, northward, and upward components are included in the data set. For convection and electric field data, a line of sight can be specified.

The `Data` initialization requires specification of the type of measurement. The `datatype` categories are: magnetometer

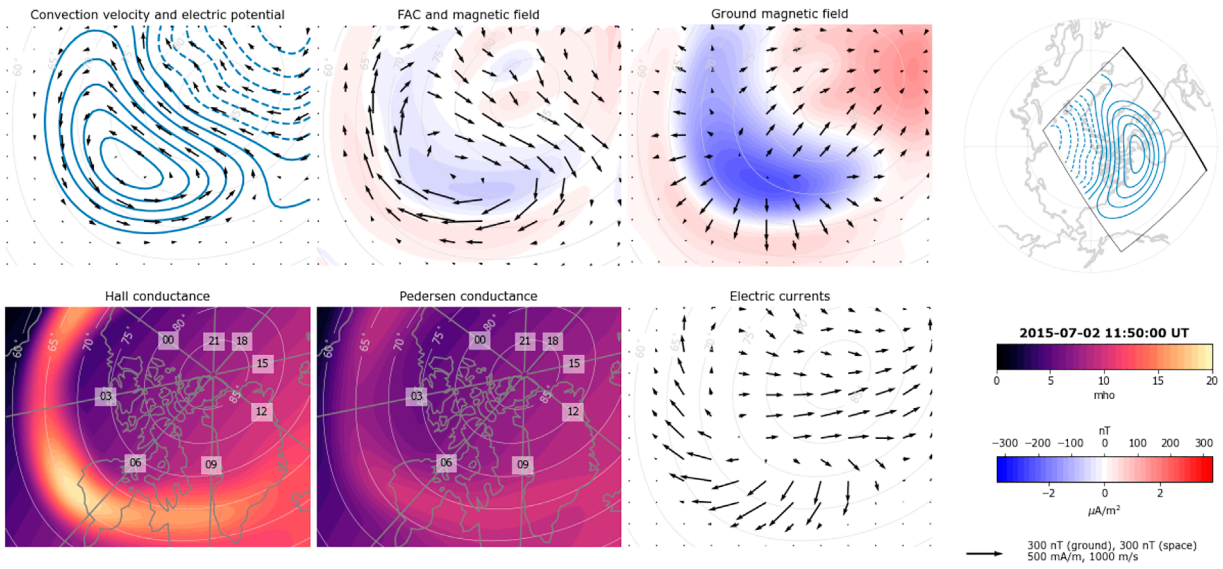


FIGURE 7
Lompe output given synthetic electric field data as input. The model conductances are for 2 July 2015 at 11:50 UT and Kp 4. The top row shows, from left to right: Convection flow field and electric potential contours; horizontal magnetic field disturbances 110 km above the ionosphere as black arrows and radial current density as color contours; horizontal ground magnetic field perturbations as black arrows and radial magnetic field perturbations as color contours; and a map that shows the grid's position and orientation with respect to apex magnetic latitude and local time. The bold grid edge corresponds to the lower edge of the projections shown in the other plots. The bottom row shows, from left to right: Pedersen conductance; Hall conductance; horizontal height-integrated ionospheric currents based on Lompe output; and color scale/vector scales.

observations from ground ("`ground_mag`"), magnetometer observations from space associated only with field-aligned currents ("`space_mag_fac`"), "full" magnetometer observations from space ("`space_mag_full`"), "`convection`" data, and ionospheric convection electric field ("`Efield`"). In addition, there is an option to use field-aligned current density ("`fac`") as input data, which can be useful for studies of, e.g., magnetosphere-ionosphere (M-I) coupling.

There are two categories of space magnetometer observations due to different heights and magnetometer precision of satellites that measure magnetic disturbances from space. For example, Iridium magnetic data is dominated by FACs since it is taken at around 800 km altitude and by magnetometers that do not have the precision of science-mission instruments. Low-flying, precise magnetometers (e.g., *Swarm* and CHAMP) will measure perturbations associated with both field-aligned currents and the horizontal divergence-free currents below the satellite, i.e., the "full" disturbance (Laundal et al., 2016).

The `Data` objects must contain the typical `scale` of the measurements. For example, convection data could have typical scales of 100 m/s, while ground magnetic field disturbances are typically on the scale of $100 \cdot 10^{-9}$ T. The scales contribute to the data covariance matrix (for details, see **Section 3.3** in Laundal et al., 2022c), and therefore partly determine the relative weight of the data set in the inversion; by increasing the scale of one dataset while keeping the scale of other datasets fixed, its relative weight in the inversion decreases. Default values are used if the scale is not specified. In addition, the data covariance matrix depends on the `error` parameter, which specifies the measurement error. While `scale` is a single value for a whole dataset, `error` can be specified on a point by point basis. If the error is not given, it is set to zero.

The finished `Data` objects can be passed to the Lompe `Emodel` object using the `add_data()` method. Once all input data is added to the `Emodel`, the `run_inversion()` method can be called. This function automatically creates the appropriate design matrix and solves Eq. **12** for $\underline{m}$ using regularized least-squares. Two regularization parameters can be specified in calls to `run_inversion()`. The regularization encourages relatively smooth solutions, with stronger gradients in the Quasi-Dipole (Richmond, 1995) north-south direction compared to east-west. Therefore this step depends on the `apexpy` Python module (Emmert et al., 2010; van der Meeren et al., 2021). The regularization represents prior assumptions about the model, and the regularization parameters determine how much weight these assumptions should have relative to the model's ability to fit the data. Choosing the optimal set of regularization parameters is therefore an important task, which requires some experimentation. Automated methods for choosing regularization parameters, for example based on L-curve analysis

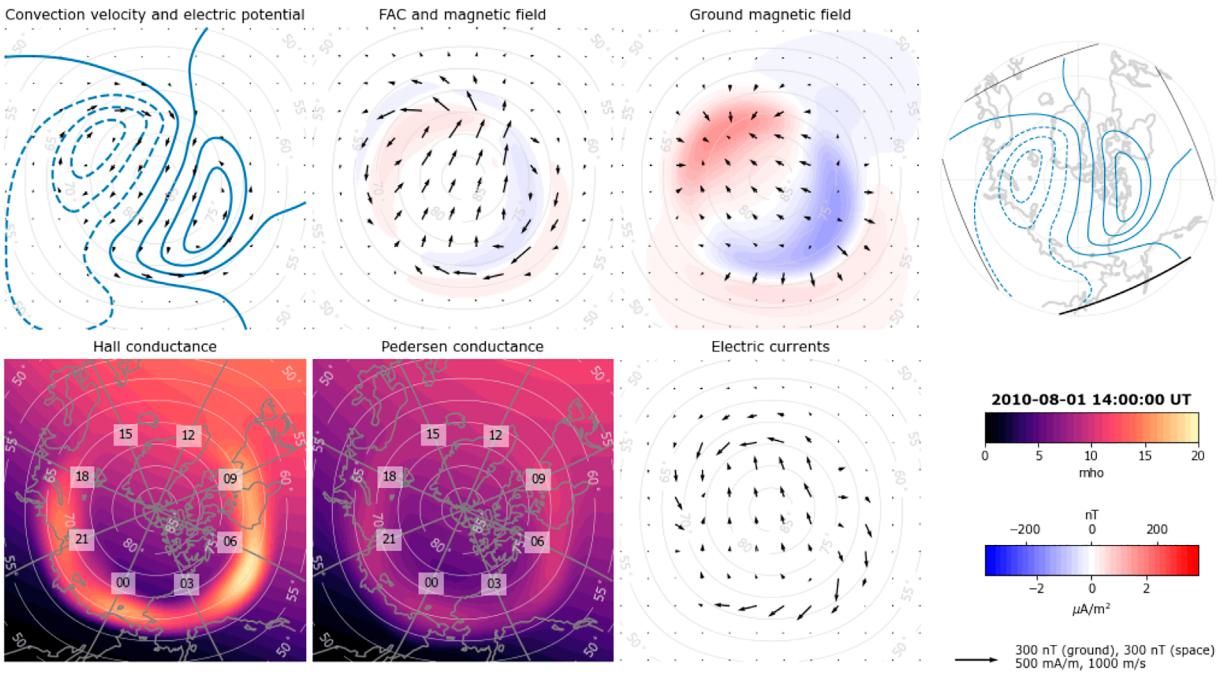


FIGURE 8
Lompe inversion results using synthetic field-aligned current data from the AMPS model as input. Model ionospheric conductances are for sunlight as on 1 August and empirical auroral patterns for Kp 4. The format of this figure is the same as **Figure 7**.

(Hansen, 1992), could be included in future updates of the code. The details of the inversion is described in **Section 3.3** in Laundal et al. (2022c). The `perimeter_width` parameter can be used to specify the perimeter around the grid from which observations are included in the inversion. The default is 10, which means observations within a 10 cell wide perimeter around the grid will be included. Measurements from outside the perimeter are ignored. If the Lompe inversion is to be performed for multiple time steps but with the same grid setup, the `clear_model()` function will reset the `Emodel` so that the inversion can be performed using new input data.

After the inversion, $\underline{m}$ is known, and ionospheric parameters can be estimated at any location within the analysis region. `Emodel` methods can return model predictions for the following parameters: convection electric field (`E()`), electric potential (`E_pot()`), convection velocity (`v()`), ground magnetic field disturbances (`B_ground()`), space magnetic field disturbances (`B_space()` for "full" disturbance, or `B_space_FAC()` for disturbance due to FACs only), ionospheric horizontal height-integrated currents (`j()`), and FACs (`FAC()`). The predicted ionospheric quantities can also be visualized using the lompeplot() function.

Figure 7 shows an example of a Lompe output, visualized with `lompeplot`. The analysis grid covers the same area as in **Figure 5**, and the cell dimension is 100 × 100 km. We use the same conductance maps as in **Figure 4**, shown in the bottom left panels of **Figure 7**. The input data is the ionospheric electric field derived from the Weimer potential (**Figures 1D, E**). We set the regularization parameters in the inversion to $\lambda_1 = 0.02$ and $\lambda_2 = 0.01$. These values are low since the Weimer electric field can be evaluated everywhere on the grid, with near zero error. With realistic data distributions and uncertainties, the inverse problem is ill-conditioned, and stronger regularization is required. The top left panel shows the predicted convection pattern and electric potential. The magnetic field in space, evaluated 110 km above R_I, is shown as black arrows in the next panel, and the color contours show the vertical current density (FACs). The third panel from the left shows the magnetic field disturbances on ground in the same format as in **Figure 3**. The panel below shows the horizontal height-integrated ionospheric currents.

The Lompe technique also offers an alternative way to solve the current continuity equation used in global magnetohydrodynamic (MHD) simulations to couple the magnetosphere and the ionosphere (e.g., Wiltberger et al., 2004; Merkin and Lyon, 2010). The ionospheric boundary condition in MHD models is specified by using the ionospheric Ohm's law to solve for the electric field given a map of field-aligned currents. This can be done in a Lompe inversion by using field aligned currents as data input. Currently fac data points must be given at all grid locations, in contrast to all the other data types which can

be specified at arbitrary locations. **Figure 8** shows an example of the Lompe technique applied with (synthetic) FAC data obtained from the Average Magnetic field and Polar current System (AMPS) model (Laundal et al., 2018). The AMPS output is for solar wind velocity 350 km/s, IMF $B_y = 0$ nT, $B_z = -4$ nT, and dipole tilt angle 25°. Model ionospheric conductances are for 1 August and Kp 4. The analysis region covers a large portion of the high-latitude northern hemisphere, and the grid cell dimension is 80 × 80 km. In this inversion, the regularization parameters are $\lambda_1 = 0.02$ and $\lambda_2 = 0.01$. This is an example of how Lompe can be used to calculate the electric field, ionospheric convection, horizontal ionospheric currents, and ground magnetic field perturbations implied by given patterns of field-aligned current and ionospheric conductance.

7 Concluding remarks

The Lompe Python package is available through Zenodo (Laundal et al., 2022a), but we recommend getting the latest version from the *stable* branch at https://github.com/klaundal/lompe. It depends on the usual scientific Python modules (NumPy, SciPy, Pandas, Matplotlib) and on two geospace-specific modules (`apexpy` and `ppigrf`) that can be installed with `pip`. The optional `dataloader` helper module also has some other dependencies, depending on which dataset it is used for. The Lompe package itself does currently not include an install script, but `lompe` can be imported as a module if the repository is placed in the user's Python module search path. The `lompe` namespace includes the `secsy`, `dipole`, and `polplot` modules, and the `Data` and `Emodel` classes. We recommend running some of the repository's example notebooks to test that it is set up correctly. In many cases, the example notebooks should be sufficient to use the Lompe technique: One needs only to adapt the grid setup parameters to the region of interest, change the conductance functions and `Data` objects, and experiment with the regularization.

As demonstrated in this paper, the Lompe Python package is modular, and the different modules can be useful independently of the Lompe technique. Some modules replicate the functionality of already existing Python packages (Burrell et al., 2018), but with some features that we believe are distinctive. `ppigrf`, for example, is different from most published IGRF Python packages since it is pure Python. `polplot` is another example; given the prevalence of polar plots in the literature, similar codes must have been implemented numerous times by several researchers, but we are not aware of any other open-source Python module for plotting data on polar local time/latitude grids. Until recently, there were no public code for SECS analysis; hopefully, the `secsy` module will help to make this technique more accessible. The Lompe Python package is open-source, and we welcome community participation in continuing its development.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/klaundal/lompe.

Author contributions

AH, KL, and JR planned the paper. AH led the work, wrote most of the paper, produced all but one figure, and wrote the accompanying Jupyter notebooks. KL led the Lompe development, wrote parts of the paper, and made **Figure 2**. All authors contributed to the Lompe code, read and commented on the manuscript.

Funding

The development of the Lompe code was also helped by synergies with the AMPS model development, funded by ESA through the Swarm Data Innovation and Science Cluster (Swarm DISC) within the reference frame of ESA contract 000109587/13/I-NB.

Acknowledgments

We are grateful to Daniel R. Weimer, Virginia Tech, for developing the Weimer 2005 model and for making it available via CCMC. We thank the science team of the NASA Electrojet Zeeman Imaging Explorer (EZIE) mission for fruitful discussions. EZIE is the Heliophysics Science Mission which was selected to study electric currents in Earth's atmosphere linking aurora to the Earth's magnetosphere. EZIE will launch no earlier than September 2024. The principal investigator for the mission is Jeng-Hwa (Sam) Yee at the Johns Hopkins University Applied Physics Laboratory in Laurel, Maryland. We also thank the International Space Science Institute in Bern, Switzerland, for hosting the team "Understanding Mesoscale Ionospheric Electrodynamics Using Regional Data Assimilation".

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Alken, P., Thébault, E., Beggan, C. D., Amit, H., Aubert, J., Baerenzung, J., et al. (2021). International geomagnetic reference field: The thirteenth generation. *Earth Planets Space* 73, 49. doi:10.1186/s40623-020-01288-x

AMGeO Collaboration (2019). *A collaborative data science platform for the geospace community: Assimilative mapping of geospace observations (AMGeO) v1*, 0.0. doi:10.5281/zenodo.3564914*Zenodo*

Amm, O., Engebretson, M. J., Hughes, T., Newitt, L., Viljanen, A., and Watermann, J. (2002). A traveling convection vortex event study: Instantaneous ionospheric equivalent currents, estimation of field-aligned currents, and the role of induced currents. *J. Geophys. Res.* 107, 1334. doi:10.1029/2002JA009472

Amm, O. (1997). Ionospheric elementary current systems in spherical coordinates and their application. *J. Geomagn. Geoelec.* 49, 947–955. doi:10.5636/jgg.49.947

Amm, O., and Viljanen, A. (1999). Ionospheric disturbance magnetic field continuation from the ground to the ionosphere using spherical elementary current systems. *Earth Planets Space* 51, 431–440. doi:10.1186/BF03352247

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship – An overview of Python in heliophysics. *J. Geophys. Res. Space Phys.* 123, 10384–10402. doi:10.1029/2018JA025877

Chisham, G., Lester, M., Milan, S. E., Freeman, M. P., Bristow, W. A., Grocott, A., et al. (2007). A decade of the super dual auroral radar network (SuperDARN): Scientific achievements, new techniques and future directions. *Surv. Geophys.* 28, 33–109. doi:10.1007/s10712-007-9017-8

Emmert, J. T., Richmond, A. D., and Drob, D. P. (2010). A computationally compact representation of Magnetic-Apex and Quasi-Dipole coordinates with smooth base vectors. *J. Geophys. Res.* 115, A08322. doi:10.1029/2010JA015326

Fukushima, N. (1994). Some topics and historical episodes in geomagnetism and aeronomy. *J. Geophys. Res.* 99, 19113–19142. doi:10.1029/94JA00102

Gjerloev, J. W. (2012). The SuperMAG data processing technique. *J. Geophys. Res.* 117, A09213. doi:10.1029/2012JA017683

Hansen, P. C. (1992). Analysis of discrete ill-posed problems by means of the L-curve. *SIAM Rev. Soc. Ind. Appl. Math.* 34, 561–580. doi:10.1137/1034115

Hardy, D. A., Gussenhoven, M. S., Raistrick, R., and McNeil, W. J. (1987). Statistical and functional representations of the pattern of auroral energy flux, number flux, and conductivity. *J. Geophys. Res.* 92, 12275–12294. doi:10.1029/JA092iA11p12275

Ieda, A., Oyama, S., Vanhamäki, H., Fujii, R., Nakamizo, A., Amm, O., et al. (2014). Approximate forms of daytime ionospheric conductance. *J. Geophys. Res. Space Phys.* 119, 10397–10415. doi:10.1002/2014JA020665

Juusola, L., Kauristie, K., Vanhamäki, H., Aikio, A., and van de Kamp, M. (2016). Comparison of auroral ionospheric and field-aligned currents derived from Swarm and ground magnetic field measurements. *J. Geophys. Res. Space Phys.* 121, 9256–9283. doi:10.1002/2016JA022961

Juusola, L., Nakamura, R., Amm, O., and Kauristie, K. (2009). Conjugate ionospheric equivalent currents during bursty bulk flows. *J. Geophys. Res.* 114, A04313. doi:10.1029/2008JA013908

Kamide, Y., Richmond, A. D., and Matsushita, S. (1981). Estimation of ionospheric electric fields, ionospheric currents, and field-aligned currents from ground magnetic records. *J. Geophys. Res.* 86, 801–813. doi:10.1029/JA086IA02P00801

Kelley, M. C. (2009). *The Earth's ionosphere: Plasma Physics and electrodynamics*. second edn. Burlington, MA, USA: Academic Press.

Laundal, K. M., Finlay, C. C., Olsen, N., and Reistad, J. P. (2018). Solar wind and seasonal influence on ionospheric currents from Swarm and CHAMP measurements. *J. Geophys. Res. Space Phys.* 123, 4402–4429. doi:10.1029/2018JA025387

Laundal, K. M., Finlay, C. C., and Olsen, N. (2016). Sunlight effects on the 3D polar current system determined from low Earth orbit measurements. *Earth Planets Space* 68, 142. doi:10.1186/s40623-016-0518-x

Laundal, K. M., Hovland, A. Ø., Hatch, S. M., Reistad, J. P., Walker, S. J., and Madelaire, M. (2022a). Local mapping of polar ionospheric electrodynamics (lompe). *Zenodo*. doi:10.5281/zenodo.5973739

Laundal, K. M. (2022). klaundal/ppigrf: ppigrf first release. *Zenodo* 0, 1. doi:10.5281/zenodo.5962661

Laundal, K. M., Madelaire, M., Ohma, A., Reistad, J., and Hatch, S. M. (2022b). The relationship between interhemispheric asymmetries in polar ionospheric convection and the magnetic field line footpoint displacement field. *Front. Astron. Space Sci.* 9, 957223. doi:10.3389/FSPAS.2022.957223

Laundal, K. M., Reistad, J. P., Hatch, S. M., Madelaire, M., Walker, S. J., Hovland, A. Ø., et al. (2022c). Local mapping of polar ionospheric electrodynamics. *J. Geophys. Res. Space Phys.* 127, e2022JA030356. doi:10.1029/2022JA030356

Laundal, K. M., and Reistad, J. P. (2022). klaundal/secsy: secsy (v1.0.0). *Zenodo*. doi:10.5281/zenodo.5962561

Laundal, K. M., and Richmond, A. D. (2017). Magnetic coordinate systems. *Space Sci. Rev.* 206, 27–59. doi:10.1007/s11214-016-0275-y

Laundal, K. M., Yee, J. H., Merkin, V. G., Gjerloev, J. W., Vanhamäki, H., Reistad, J. P., et al. (2021). Electrojet estimates from mesospheric magnetic field measurements. *JGR. Space Phys.* 126, e2020JA028644. doi:10.1029/2020ja028644

Merkin, V. G., and Lyon, J. G. (2010). Effects of the low-latitude ionospheric boundary condition on the global magnetosphere. *J. Geophys. Res.* 115, A10202. doi:10.1029/2010JA015461

Moen, J., and Brekke, A. (1993). The solar flux influence on quiet time conductances in the auroral ionosphere. *Geophys. Res. Lett.* 20, 971–974. doi:10.1029/92GL02109

Reistad, J. P., Laundal, K. M., Østgaard, N., Ohma, A., Haaland, S., Oksavik, K., et al. (2019). Separation and quantification of ionospheric convection sources: 1. A new technique. *JGR. Space Phys.* 124, 6343–6357. doi:10.1029/2019JA026634

Rich, F. J. (1994). *Users Guide for the Topside Ionospheric Plasma Monitor (SSIES, SSIES-2 and SSIES-3) on Spacecraft of the Defense Meteorological Satellite Program*, Vol. 1. Hanscom Air Force Base, Massachusetts: PHILLIPS LAB HANSCOM AFB MA. Technical description.

Richmond, A. D. (1995). Ionospheric electrodynamics using magnetic apex coordinates. *J. Geomagn. Geoelec.* 47, 191–212. doi:10.5636/jgg.47.191

Richmond, A. D., and Kamide, Y. (1988). Mapping electrodynamic features of the high-latitude ionosphere from localized observations: Technique. *J. Geophys. Res.* 93, 5741–5759. doi:10.1029/JA093iA06p05741

Robinson, R. M., Kaeppler, S. R., Zanetti, L., Anderson, B., Vines, S. K., Korth, H., et al. (2020). Statistical relations between auroral electrical conductances and field-aligned currents at high latitudes. *JGR. Space Phys.* 125, e2020JA028008. doi:10.1029/2020JA028008

Ronchi, C., Iacono, R., and Paolucci, P. S. (1996). The "cubed sphere": A new method for the solution of partial differential equations in spherical geometry. *J. Comput. Phys.* 124, 93–114. doi:10.1006/jcph.1996.0047

van der Meeren, C., Laundal, K. M., Burrell, A. G., Starr, G., Reimer, A. S., and Morschhauser, A. (2021). aburrell/apexpy: ApexPy Version. *Zenodo* 0, 1. doi:10.5281/zenodo.4585641

Vanhamäki, H., and Amm, O. (2011). Analysis of ionospheric electrodynamic parameters on mesoscales – A review of selected techniques using data from ground-based observation networks and satellites. *Ann. Geophys.* 29, 467–491. doi:10.5194/angeo-29-467-2011

Vanhamäki, H., and Juusola, L. (2020). "Introduction to spherical elementary current systems,". *Ionospheric multi-spacecraft analysis tools: Approaches for deriving ionospheric parameters*. Editors M. W. Dunlop, and H. Lühr (Springer International Publishing), 175. doi:10.1007/978-3-030-26732-2_2

Waters, C. L., Anderson, B., Green, D., Korth, H., Barnes, R., and Vanhamäki, H. (2020). "Science data products for AMPERE," in *Ionospheric multi-spacecraft analysis tools* (Cham: Springer), 141–165.

Weimer, D. R. (2005a). Improved ionospheric electrodynamic models and application to calculating Joule heating rates. *J. Geophys. Res.* 110, A05306. doi:10.1029/2004JA010884

Weimer, D. R. (2005b). Predicting surface geomagnetic variations using ionospheric electrodynamic models. *J. Geophys. Res.* 110, A12307. doi:10.1029/2005JA011270

Wiltberger, M., Wang, W., Burns, A. G., Solomon, S. C., Lyon, J. G., and Goodrich, C. C. (2004). Initial results from the coupled magnetosphere ionosphere thermosphere model: Magnetospheric and ionospheric responses. *J. Atmos. Solar-Terrestrial Phys.* 66, 1411–1423. doi:10.1016/J.JASTP.2004.03.026

Yee, J.-H., Gjerloev, J., and Wu, D. (2021). "Remote sensing of magnetic fields induced by electrojets from space," in *Upper atmosphere dynamics and energetics*. Editors W. Wang, Y. Zhang, and L. J. Paxton (American Geophysical Union AGU). chap. 21. 451–468. doi:10.1002/9781119815631.ch21

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 15 December 2022
DOI 10.3389/fspas.2022.1073578

Check for updates

OPEN ACCESS

EDITED BY
John Coxon,
Northumbria University, United Kingdom

REVIEWED BY
Frederic Efffenberger,
Ruhr University Bochum, Germany
Ingmar Sandberg,
Space Applications & Research Consultancy (SPARC), Greece

*CORRESPONDENCE
Christian Palmroos,
christian.o.palmroos@utu.fi

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 18 October 2022
ACCEPTED 18 November 2022
PUBLISHED 15 December 2022

CITATION
Palmroos C, Gieseler J, Dresing N, Morosan DE, Asvestari E, Yli-Laurila A, Price DJ, Valkila S and Vainio R (2022), Solar energetic particle time series analysis with Python.
Front. Astron. Space Sci. 9:1073578.
doi: 10.3389/fspas.2022.1073578

COPYRIGHT
© 2022 Palmroos, Gieseler, Dresing, Morosan, Asvestari, Yli-Laurila, Price, Valkila and Vainio. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Solar energetic particle time series analysis with Python

Christian Palmroos[1]*, Jan Gieseler[1], Nina Dresing[1], Diana E. Morosan[2], Eleanna Asvestari[2], Aleksi Yli-Laurila[1], Daniel J. Price[2], Saku Valkila[1] and Rami Vainio[1]

[1]Space Research Laboratory, Department of Physics and Astronomy, University of Turku, Turku, Finland, [2]Department of Physics, University of Helsinki, Helsinki, Finland

Solar Energetic Particles (SEPs) are charged particles accelerated within the solar atmosphere or the interplanetary space by explosive phenomena such as solar flares or Coronal Mass Ejections (CMEs). Once injected into the interplanetary space, they can propagate towards Earth, causing space weather related phenomena. For their analysis, interplanetary *in situ* measurements of charged particles are key. The recently expanded spacecraft fleet in the heliosphere not only provides much-needed additional vantage points, but also increases the variety of missions and instruments for which data loading and processing tools are needed. This manuscript introduces a series of Python functions that will enable the scientific community to download, load, and visualize charged particle measurements of the current space missions that are especially relevant to particle research as time series or dynamic spectra. In addition, further analytical functionality is provided that allows the determination of SEP onset times as well as their inferred injection times. The full workflow, which is intended to be run within Jupyter Notebooks and can also be approachable for Python laymen, will be presented with scientific examples. All functions are written in Python, with the source code publicly available at GitHub under a permissive license. Where appropriate, available Python libraries are used, and their application is described.

KEYWORDS

python, software package, solar energetic particle (SEP), coronal mass ejection (CME), spacecraft, heliosphere, data, onset time

1 Introduction

Solar Energetic Particle (SEP) events, as observed at a spacecraft, are determined by a combination of physical processes such as their acceleration mechanism, the particle injection into the interplanetary magnetic field, and their transport through this medium (see, e.g., Desai and Giacalone, 2016; Reames, 2021, and reference therein for a review). All of these processes can vary, and they are not yet fully understood. Using only measurements of a single spacecraft, it can be very difficult to disentangle these processes, making multi-spacecraft observations indispensable. A careful and comprehensive analysis is therefore needed. It should focus not only on the energetic particle observations but also on *in situ* measurements of the magnetic field and solar

wind plasma, as well as on remote-sensing observations of the solar counterpart of the event, such as a solar flare or a Coronal Mass Ejection (CME), both being sites of potential particle acceleration.

A powerful method to study SEP events is to use multi-spacecraft observations of the same event that are situated at various locations with respect to the source region. This can allow, for example, to disentangle source properties from transport effects. The tools presented in this manuscript, which enable such multi-spacecraft analyses, have been developed within the Solar energetic particle analysis platform for the inner heliosphere (SERPENTINE), funded by the European Union's Horizon 2020 framework programme (see, Gieseler et al., 2022a for an introduction). They focus mainly on the study of an SEP event, which requires a careful characterization of its various features. These include the shape of the energetic particle time profiles at various energies and for different particle species, the maximum observed energies, as well as peak intensities, differences in directional measurements that allow determining anisotropies, and the onset time of the event. The latter is defined by the first arriving particles at the spacecraft, and it is one of the most important parameters for linking the energetic particle measurements with remote-sensing observations of the solar phenomena associated to the SEP event. As the kinetic energy, and hence the speed, of the detected particles is usually known, one can infer their injection time at the Sun when assuming a certain path length they had to travel to reach the spacecraft. This allows then the comparison with the times of various solar phenomena to figure out their importance for the SEP event. Considering that higher-energy particles travel faster than lower-energy ones, the onset of an SEP event often shows a velocity dispersion, i.e., higher-energy particles arriving earlier than those with lower energies. An analysis of this velocity dispersion, together with the assumption that all particles were injected at the same time and travelled the same path length, allows inferring not only a common injection time but also the path length itself.

The tools presented here constitute a comprehensive toolkit that enables the user to perform the aforementioned energetic particle characterization using multiple spacecraft, such as Solar Orbiter (Müller et al., 2020), Parker Solar Probe (Fox et al., 2016), Solar Terrestrial Relations Observatory (STEREO, Kaiser, 2005; Kaiser et al., 2008), SOlar and Heliospheric Observatory (SOHO, Domingo et al., 1995), and Wind (Harten and Clark, 1995). The basis of the toolkit lies in the functionality that allows the user to automatically download data files from online repositories and to load them into powerful Pandas DataFrame objects (McKinney, 2010) in Python. All the data loaders (*cf.* **Section 2.1**) are made available through a Python package that can be installed with a single command. In addition, a Jupyter Notebook provides visualization examples and additional functionality to explore the content of the loaded data files, for example various energy ranges, particle species and further metadata. The rest of the tools presented in this paper use the data loaders in the background and provide more advanced analysis methods, such as the onset determination tool (**Section 2.2**) or the visual time shift analysis tool (**Section 2.4**). Finally, we also provide a tool that allows a first comparison of another phenomenon of solar activity, namely radio bursts, with the energetic particle observations, the spectrogram tool (**Section 2.3**). It can be used to plot dynamic spectrograms of energetic particles as well as radio observations.

2 Methods

All the tools described in this paper are open-source Python software released under the BSD-3-Clause license. In general, they are all provided as self-explanatory Jupyter Notebooks (Kluyver et al., 2016; see jupyter.org for more details) that give detailed instructions and examples (see **Figures 1**, **5** for examples). Based on these Notebooks, the user should be able to modify the code according to their needs. The actual software code of the tools is delivered as a Python package called `SEPpy` (Gieseler et al., 2022c). It is listed on PyPI so that it can be installed using the widely-used `pip` command. By delivering the package in this manner, all other required packages are automatically installed. In the Notebooks itself, all necessary functions are imported from this package. Thus, the user only needs to modify some parameters to their needs, such as selecting the date or instrument of interest. Depending on this selection, all required data files are dynamically loaded from online sources or local files using the data loaders presented in **Section 2.1**. The Jupyter Notebooks are distributed through the GitHub repository serpentine-h2020/serpentine (Gieseler et al., 2022b), which is archived at Zenodo (European Organization For Nuclear Research and OpenAIRE, 2013). The repository contains detailed installation instructions, and the tools will be continuously updated through it.

While running the tools as Jupyter Notebooks requires only a very basic level of Python knowledge, the user still needs to follow an installation procedure in order to use the tools. As an alternative, the SERPENTINE project provides a JupyterHub server that allows users to run the Notebooks described in this manuscript completely online, without any requirements except for a web browser and a GitHub account for authentication. More details about the JupyterHub server can be found on the SERPENTINE website and will be presented in an upcoming publication.

2.1 Data loaders

The Data loaders Notebook is a collection of functions that drastically simplifies obtaining (i.e., automatically downloading and loading into Python structures) energetic

STEREO (stereo-loader)

- GitHub: https://github.com/jgieseler/seppy

Python data loader for some STEREO instruments (i.e., magnetic field and charged particles). At the moment provides released data obtained by SunPy through CDF files from CDAWeb for the following datasets:

- `'HET'` : STEREO IMPACT/HET Level 1 Data (Info)
- `'LET'` : STEREO IMPACT/LET Level 1 Data (Info)
- `'MAG'` : STEREO IMPACT/MAG Magnetic Field Vectors (RTN or SC) (Info RTN, Info SC)
- `'MAGB'` : STEREO IMPACT/MAG Burst Mode (~0.03 sec) Magnetic Field Vectors (RTN or SC) (Info RTN, Info SC)
- `'SEPT'` : STEREO IMPACT/SEPT Level 2 Data (Info 1, Info 2) [1]

[1] STEREO IMPACT/SEPT Level 2 data is directly obtained through ASCII files from the server of the instrument team, not via SunPy/CDAWeb.

Imports

In [42]:
```
from seppy.loader.stereo import stereo_load
import datetime as dt
```

Define options

- `instrument` : `'HET'` , `'LET'` , `'MAG'` , `'MAGB'` , or `'SEPT'` . See above for explanation.
- `startdate` , `enddate` : datetime object or "standard" datetime string
- `spacecraft` : String, optional. Name of STEREO spacecraft: `'ahead'` or `'behind'` , by default `'ahead'` .
- `mag_coord` : String, optional. Coordinate system for MAG: `'RTN'` or `'SC'` , by default `'RTN'` .
- `sept_species` : String, optional. Particle species for SEPT: `'e'` for electrons or `'p'` for protons (resp. ions), by default `'e'` .
- `sept_viewing` : String, optional. Viewing direction for SEPT: `'sun'` , `'asun'` , `'north'` , or `'south'` , by default `'sun'` .
- `path` : String, optional. Local path for storing downloaded data, e.g. `path='data/stereo/'` . By default `None` . Default setting saves data according to sunpy's Fido standards.
- `resample` : Pandas frequency (e.g., `'1min'` or `'1h'`), or `None` , optional. Frequency to which the original data is resamepled. By default `None` .
- `pos_timestamp` : String, optional. Change the position of the timestamp: `'center'` or `'start'` of the accumulation interval, by default `None` .
- `max_conn` : Integer, optional. The number of parallel download slots used by `Fido.fetch` , by default `5` .

In [24]:
```
instrument = "HET"
startdate = dt.datetime(2021, 10, 28)
enddate = "2021/10/30"
path = '/home/gieseler/uni/stereo/data/'
resample = "1min"
pos_timestamp = None
```

Load data

In [25]:
```
df, meta = stereo_load(instrument=instrument, startdate=startdate, enddate=enddate,
                       path=path, resample=resample, pos_timestamp=pos_timestamp)
```

In [28]:
```
df.Electron_Flux_1.plot(logy=True, ylabel=meta['Electron_Flux_UNITS'], label=meta['Electron_Bins_Text'][1][0],
                        title='STEREO/HET electrons')
plt.legend()
```

Out[28]: `<matplotlib.legend.Legend at 0x7f341a792dc0>`

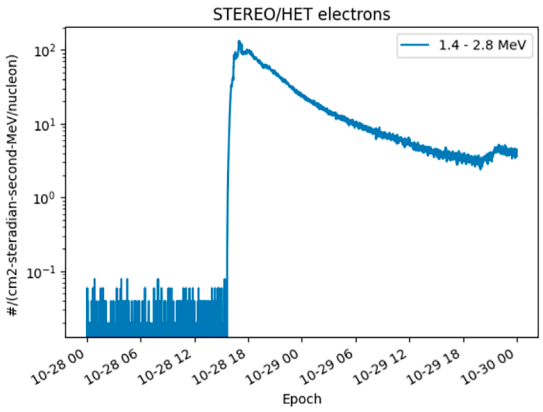


FIGURE 1
Example workflow within the Data loaders Notebook showing how to load and visualize STEREO/HET observations.

TABLE 1 All instruments supported by the EnergeticParticle Data Loaders Notebook.

Spacecraft	Instrument
Parker Solar Probe	IS⊙IS/EPI-Hi HET IS⊙IS/EPI-Lo PE COSTEP/EPHIN
SOHO	ERNE-HED ERNE-LED
Solar Orbiter	EPD/EPT EPD/HET EPD/STEP[a]
STEREO-A/B[b]	HET LET SEPT
Wind	3DP

[a]so far only until October 2021 due to a change in data product.
[b]STEREO-B, only until October 2014 due to spacecraft contact loss.

charged particle data sets of *in situ* measurements by the current heliospheric spacecraft fleet. This is especially valuable because the data products of the newest spacecraft like Solar Orbiter or Parker Solar Probe are only released in the binary Common Data Format (CDF) that requires advanced programming skills in order to read just the basic data structures.

Table 1 gives an overview of the currently supported instruments. For most of the missions, the data is downloaded as CDF data files from the Coordinated Data Analysis Web (CDAWeb) service provided by NASA's Space Physics Data Facility at Goddard Space Flight Center. The data loading is in these cases utilized through version 4.0.5 (Mumford et al., 2022) of the `sunpy` open-source software package (The SunPy Community et al., 2020). Some data sets are directly obtained from the web servers of the responsible institutes, primarily due to not being available through CDAWeb. This applies to Wind/3DP data, STEREO/SEPT data, and electron measurements of SOHO/EPHIN. The latter two data sets are obtained as ASCII files, and not in the CDF format. In general, level 2 data products or higher are used. This is calibrated and validated data that is ready for scientific use (e.g., McComas et al., 2016; Rodríguez-Pacheco et al., 2020). However, in some cases, when for example only level 1 data is available, lower-level data products are chosen. All data files from the Energetic Particle Detector (EPD) instrument suite of Solar Orbiter are directly obtained as CDF files from ESA's official Solar Orbiter Archive (SOAR), using its own Application Programming Interface (API). For this, the already established version 0.1.10 (Gieseler et al., 2022d) of the `solo-epd-loader` Python package is used, which in addition to level 2 data also supports the so-called low-latency (LL) data. This data product provides the latest observations, which are very useful in terms of quick-look purposes. However, because the data set is not yet verified it shall not be used for scientific publications.

For most data sets, the data files are loaded into Pandas DataFrames (McKinney, 2010) in Python using SunPy's TimeSeries functionality. In some cases this is not possible, namely when the data is multidimensional, that is, if multiple dependencies are explicitly defined in the corresponding CDF file. This can for example be the case if a measurement is not only related to time, but also to energy and viewing direction, and depends on the way the CDF file has been structured by the instrument team. In these cases, manual loading functions adopted from the function `cdf2df` of version 0.15.4 of the discontinued Python package `heliopy` (Stansby et al., 2021) are used. These functions exploit the Python package `cdflib` (Stansby et al., 2022).

Figure 1 shows an example workflow within the Data loaders Notebook that illustrates how to load data from the STEREO mission. Next to a direct example, the Notebook also gives information on the other available options. In addition, the Notebook provides different examples on how to visualize the data with Python. For example, how to combine observations of multiple energy channels of different species (see **Figure 2**) or different instruments into a multi-panel plot (**Figure 3**), or how to add multiple viewing directions into a single plot (**Figure 4**). Versed Python users can then easily build their own analyses based on this. With these functions, they can skip the inconvenient start of setting up corresponding routines, and can directly start their scientific analysis. This Jupyter Notebook shows only a selection of data loading examples for some of the data loaders, and will be continuously expanded with further missions and instruments. The data loading functionality of this Notebook is employed also in the following Notebooks presented in **Sections 2.2–2.4**.

Next to the loading of charged particle observations, in SEPpy partial support is given for magnetic field and radio measurements. For these, the download and loading of the data is done in the same way as described above. At the moment, the data obtained by the MAG instruments onboard STEREO and Solar Orbiter as well as radio observations by the STEREO/SWAVES instrument are included. The radio data is especially important for the dynamic spectrum tool described in **Section 2.3**.

2.2 SEP event onset determination

The onset time of an SEP event, that is, the initially measured intensity increase at a spacecraft, is a key information for the event analysis. For example, it is necessary for inferring the injection time and connecting remote-sensing observations to *in situ* measurements. Furthermore, by knowing the onset times of particles of varying energies, one can get a better estimation of the path that the particles travelled. As the particles travel at different speeds corresponding to their different kinetic energies, they are expected to show different onset times and profiles for different energy channels if one assumes that all particles were injected at the same time at the Sun. This effect is known as

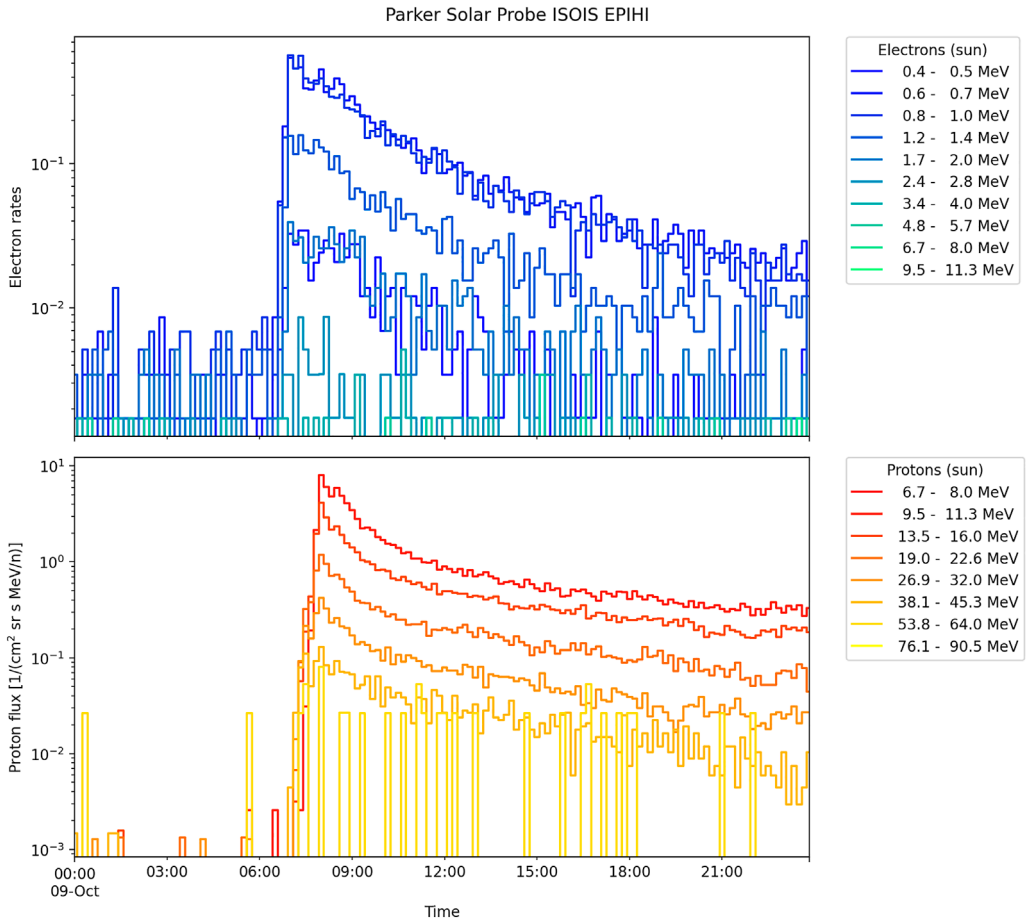


FIGURE 2
Example output of plotting Parker Solar Probe data in the SEP Data Loaders Notebook. Shown here are electron and proton measurements of the ISʘIS/EPIHI instrument of Parker Solar Probe for the SEP event on 9 October 2021. Note that the electron measurements are count rates and not fluxes.

velocity dispersion, and it is the basis for a Velocity Dispersion Analysis (VDA, e.g., Lintunen and Vainio, 2004), in which one can derive the initial injection of the SEPs at the Sun. There are different methods to determine onset times, such as the widely used 3-sigma method, in which the event is defined by the time when a certain number of data points increase by three sigma above the predefined background, or by applying a fit to the rising phase of the event in order to extrapolate backwards (e.g., Dresing et al., 2012). The tool presented in this paper employs the Poisson-CUSUM method (Lucas, 1985; Huttunen-Heikinmaa et al., 2005) to find the onsets of an SEP event in different energy channels. Cumulative Sum (CUSUM) methods are a set of similar statistical methods that are employed in many industries as quality control schemes (Page, 1954). CUSUM methods accumulate the difference between consecutive entries of time series data. The first entry of the CUSUM function is usually set to zero. In traditional CUSUM methods, the quality of an industrial product is monitored as a function of time, and this quality may be some numerical value assigned to the product. CUSUM methods are designed to give an early warning in the case this numerical value rapidly deviates from a default range of values, and as such they are widely used in SEP onset determinations (e.g., Paassilta et al., 2018).

The traditional CUSUM method operates under the assumption that the variable under monitoring is a random variable that is normally distributed. Hence, mean and standard deviation of the distribution are the two critical parameters that dictate if and when CUSUM gives a warning of the process getting out of control. We consider the intensity time series measurements as analogous to the quality of an industrially manufactured product, and as such find the onset of an SEP event at the moment of time when intensity measurements start to noticeably deviate from the pre-event background intensity. In the classic approach of the CUSUM method, the monitored variable is assumed to follow a normal distribution. Because measurements of SEPs are expected to be Poisson-distributed,

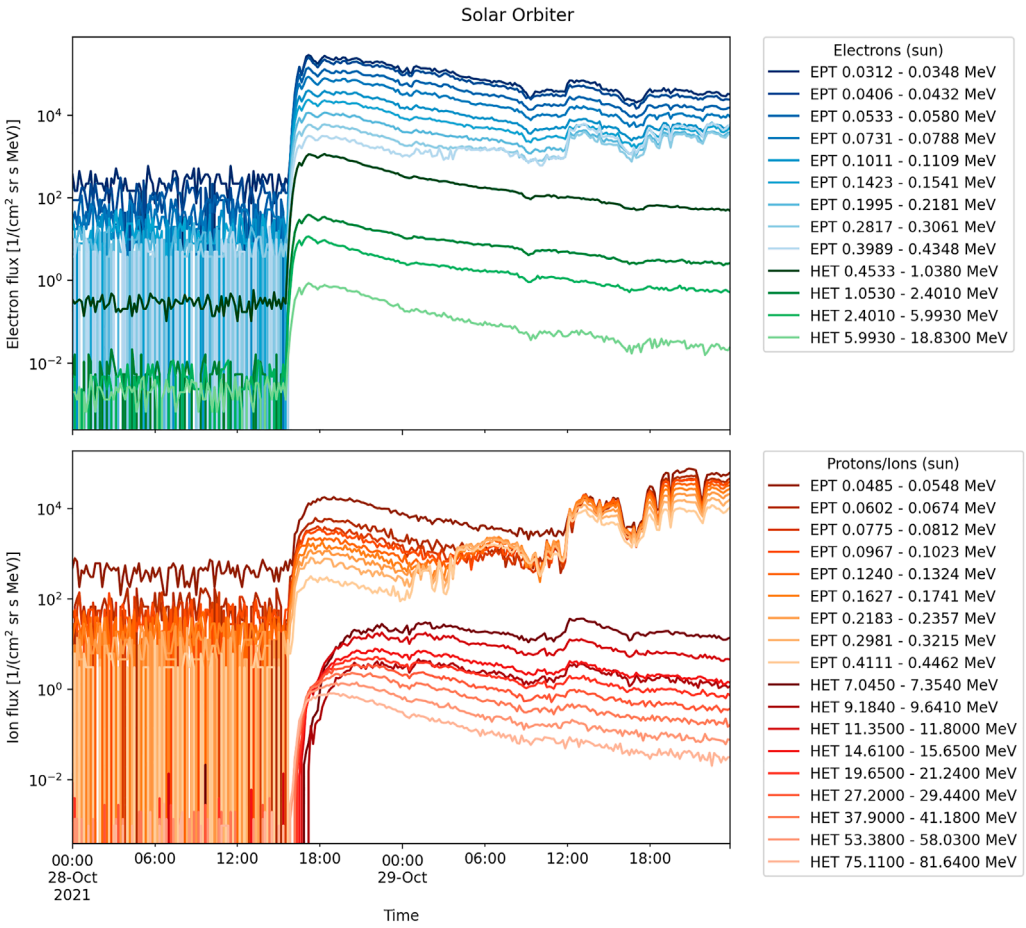


FIGURE 3
Example output of how to combine different instrument observations into a multi-panel plot in the SEP Data Loaders Notebook. Shown here are electron and ion measurements of the EPT and HET sensors of Solar Orbiter's EPD instrument suite for the SEP event on 29 October 2021. Note that the increases in ion fluxes observed by EPT around 18:00 are due to contamination caused by electrons.

we use the modified Poisson-CUSUM method. To find the onset of an SEP event, it is important to first accurately define the pre-event background of the observations by its mean intensity μ and standard deviation σ. These parameters define the control parameter k that restricts the growth of the CUSUM function:

$$\mu_d = \mu + n\sigma \tag{1}$$

$$k = \frac{\mu_d - \mu}{\sigma\left(\ln\left(\mu_d\right) - \ln\left(\mu\right)\right)}, \tag{2}$$

Where μ_d is an uncertainty limit for the background variations and n is a natural number that is usually set to $n = 2$. Finally, the dimensionless CUSUM function is calculated with normalized intensity measurements as follows:

$$I_n = \frac{I - \mu}{\sigma} \tag{3}$$

$$C[0] = 0 \tag{4}$$

$$C[i] = \max\left(0, I_n - k + C[i-1]\right), \tag{5}$$

Where I is the measured intensity, I_n is the normalized deviation from the mean intensity of the pre-event background, and C is the CUSUM function. One can see from the definition of C that it accumulates the difference of I_n and k as long as I_n stays larger than k. When the values of C exceed a defined hastiness threshold h, the method starts counting consecutive entries of C until a predefined value of consecutive $C > h$ entries is reached. At that point, the method backtracks the same amount of data points, and marks the found data point as the onset of an event. The amount of consecutive $C > h$ entries that the method counts is a freely changeable parameter called `cusum_window`. It is important to note that this parameter corresponds to the number of data points, and not to the time that has passed since C exceeded h. The default value of `cusum_window` = 30 is chosen so that it corresponds to 30 min of continuous threshold-exceeding

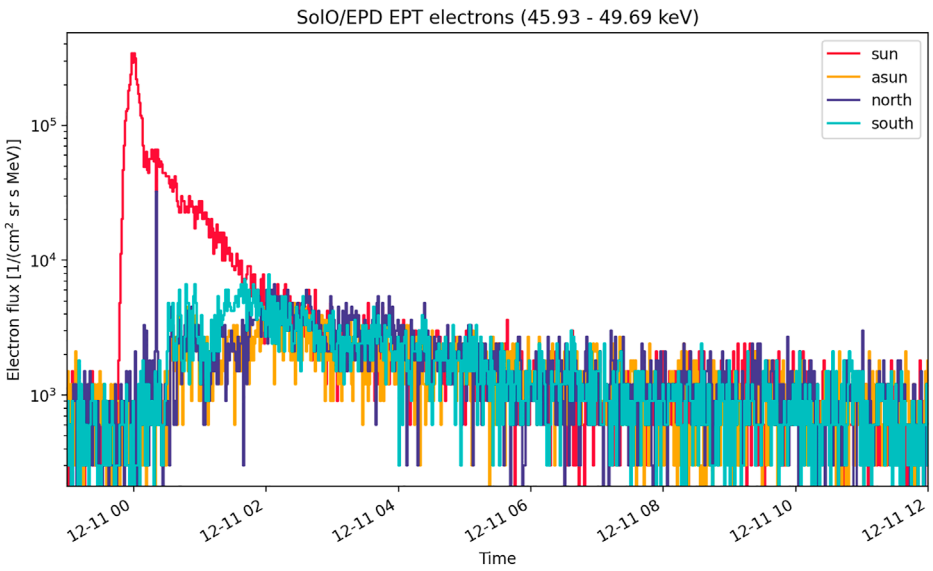


FIGURE 4
Example output of how to add multiple viewing directions into a single plot in the Data loaders Notebook. Shown here are the four viewing directions of the EPT sensor of Solar Orbiter's EPD instrument suite.

intensity measurements for a typical 1-min time resolution. Following Huttunen-Heikinmaa et al. (2005), we define h such that it is by default set to $h = 1$, but if k is large, which in the context of Poisson-CUSUM means that $k \geq 1$, then it is set to $h = 2$. Choosing h to be sufficiently small is imperative for identifying the onset of the event as early as possible.

The onset time determination tool is built in Python, and the user interface is implemented in a Jupyter Notebook for easy usage. After importing the necessary libraries, the user needs to choose the observing spacecraft, measuring sensor, viewing direction (if the instrument provides multiple ones), and the particle species from drop-down menus (shown in **Figure 5**). After that, a data path can be defined to which data files should be saved to or loaded from. By default, this is set to the directory where the notebook is located. Finally, the user needs to specify the start and end of the time range that should be loaded. With these defined parameters, an `Event` object is initialized that contains the corresponding data and provides different functions that can be applied to it. Creating the object also automatically downloads the necessary data using the data loader functions presented in **Section 2.1** in case it is not found in the designated data directory.

Once the object is created, the onset analysis can be run by defining four different parameters: `averaging`, `background_range`, `channels`, and `plot_range`. `averaging` is a parameter that defines the time-averaging of the intensity data. It accepts a Pandas-compatible time string, e.g., "1min" or "30s" for 1-min or 30-s time resolution, respectively. It can also be set to `None`, in which case the intensity data is not time-averaged at all. It is noteworthy that data cannot be averaged to a finer resolution than its original measurement. `background_range` defines the time window that is considered to be the pre-event background, and `plot_range` defines the limits of the time-axis of the entire plot. The analysis can be run on a combination of multiple channels by giving the start and end indices of the channels as an input. This way, all channels in between are combined to a single energy channel, according to their respective energies and intensity measurements. It is of course also possible to run the onset analysis on only a single channel by just providing a single index as the input. The list of available energy channels for the chosen instrument can be obtained by running the function `Event.print_energies()`. The output information and figure that are produced with the input parameters defined in **Figure 5** are displayed in **Figure 6**.

The CUSUM method is very effective in finding the time when the monitored variable suddenly changes drastically. While it still performs better than the 3-sigma method for gradual SEP rise phases, it can be challenging to find a correct onset time in such cases. Usually, a longer time averaging is recommendable then. Furthermore, a rational choice of the background interval is imperative for the method to work correctly, so the user needs to be careful in assessing whether to include transient structures that do not necessarily represent the actual background into `background_range`.

First import the necessary library

In [1]:
```
from seppy.tools import Event
import seppy.tools.widgets as w
import datetime, os
```

Choose the spacecraft, sensor, view direction and particle species:

In [2]:
```
display(w.spacecraft_drop, w.sensor_drop, w.view_drop, w.species_drop)
```

Spacecraft: Solar Orbiter
Sensor: EPT
Viewing: sun
Species: electrons

Set the path to your data folder:

In [3]:
```
# Path for the downloaded data (by default the current directory)
data_path = f"{os.getcwd()}/data/"
```

Choose time range for data loading, and create the Event object:

In [4]:
```
# Format of date: year, month, day
startdate = datetime.date(2021, 10, 28)
enddate = datetime.date(2021, 10, 29)

# Get event data:
Event_onset = Event(spacecraft=w.spacecraft_drop.value, sensor=w.sensor_drop.value, data_level='l2',
                    species = w.species_drop.value, start_date=startdate, end_date=enddate, data_path=data_path)
```

Print out the energy channel keys and corresponding energy ranges

In [5]:
```
Event_onset.print_energies()
```

```
solo, ept:

Channel number | Energy range
 0  :  0.0312 - 0.0348 MeV
 1  :  0.0330 - 0.0369 MeV
 2  :  0.0348 - 0.0380 MeV
 3  :  0.0380 - 0.0406 MeV
 4  :  0.0406 - 0.0432 MeV
 5  :  0.0432 - 0.0459 MeV
 6  :  0.0459 - 0.0497 MeV
 7  :  0.0497 - 0.0533 MeV
 8  :  0.0533 - 0.0580 MeV
 9  :  0.0580 - 0.0627 MeV
 10  :  0.0627 - 0.0673 MeV
 11  :  0.0673 - 0.0731 MeV
 12  :  0.0731 - 0.0788 MeV
 13  :  0.0788 - 0.0856 MeV
 14  :  0.0856 - 0.0934 MeV
 15  :  0.0934 - 0.1011 MeV
```

Set the averaging period, background_range, channels to plot and the time range of the plot:

In [6]:
```
# Use pandas-compatible resampling period string, e.g. '10min' or '30s'. Set to None to not resample data
averaging = "3min"

# Set the time range for what is considered the background preceding the event:
# Format of datetime: year, month, day, hour, minute, second
background_range = (datetime.datetime(2021, 10, 28, 13, 0, 0), datetime.datetime(2021, 10, 28, 15, 0, 0))

# Set the channel(s) to plot (use a list to combine multiple channels):
channels = [13,17]

# Set the time range for the plot:
plot_range = [datetime.datetime(2021, 10, 28, 11), datetime.datetime(2021, 10, 29, 0)]
```

FIGURE 5
The user interface of most Notebooks (here for the onset determination) consists of a simple drop-down menu accompanied by only a few lines of declaring variables. There are short instructions or a description between every cell that guide the user. This set of inputs yields an output shown in **Figure 6**.

Run the cell to produce a plot

```
In [8]:
flux, onset_stats, onset_found, peak_flux, peak_time, fig, bg_mean = \
    Event_onset.find_onset(viewing=w.view_drop.value, background_range=background_range, channels=channels,
                           resample_period=averaging, yscale='log', cusum_window=30, xlim=plot_range)
onset = onset_stats[-1]
peak_flux = peak_flux.values[0]
output = Event_onset.output
```

```
##### >Energy channels< #####
0.0788 - 0.1197 MeV
############################

##### >Flux peak< #####
                           flux
Time
2021-10-28 17:04:30  51912.726562
######################

##### >Onset time< #####
2021-10-28 15:46:30
######################

##### >Mean of background intensity< #####
19.934679
########################################

##### >Std of background intensity< #####
14.474801
########################################

##### >Particle species< #####
electrons
############################
```

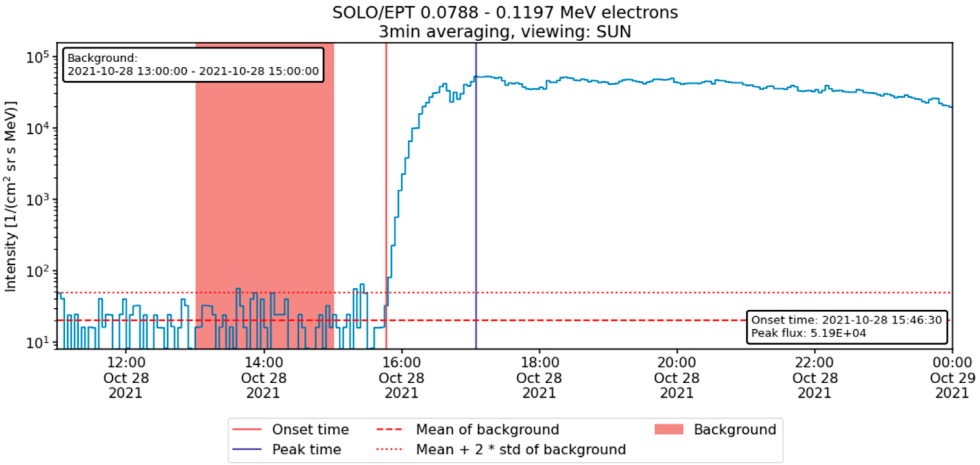


FIGURE 6
Example output for the SEP onset analysis Notebook for the SEP event on 28 October 2021 as observed by Solar Orbiter/EPT, based on the inputs shown in **Figure 5**. The red shaded area defines the pre-event background period, and the red and blue vertical lines indicate the found onset time and the peak intensity time, respectively. The intensity time series has been averaged to 1-min resolution.

2.3 Dynamic spectrum

Whereas usual time series data is presented as a two-dimensional representation of intensity as a function of time in a single energy channel, a dynamic spectrum presents all the energy channels of an instrument in a single plot. This allows one to determine, e.g., confirmation of velocity dispersion in an event immediately with only visual inspection, something that would otherwise become apparent after finding the onset time in a number of energy channels.

The Dynamic Spectrum tool plots the dynamic spectrum of a single instrument for a given time range. The plot is initialized with a grid that has time bins on the x-axis and energy bins on the y-axis. The bins in the grid are then colored according to $I \cdot E^2$, where E is the mean energy of the corresponding channel and I is the observed intensity at that time and energy bin. Physically, $I \cdot E^2$ represents the energy flux per logarithmic energy band carried by the SEPs. The motivation behind plotting this quantity instead of simply I is that this way the particle spectrum is flattened. We apply this procedure to account for the drastically varying intensity levels present in one energy spectrum, which can easily lead to poor visibility of the lowest intensities (observed at the highest energies) in a dynamic spectrum.

To plot the dynamic spectrum of an instrument, the user only needs to initialize the `Event` object presented in **Section 2.2**.

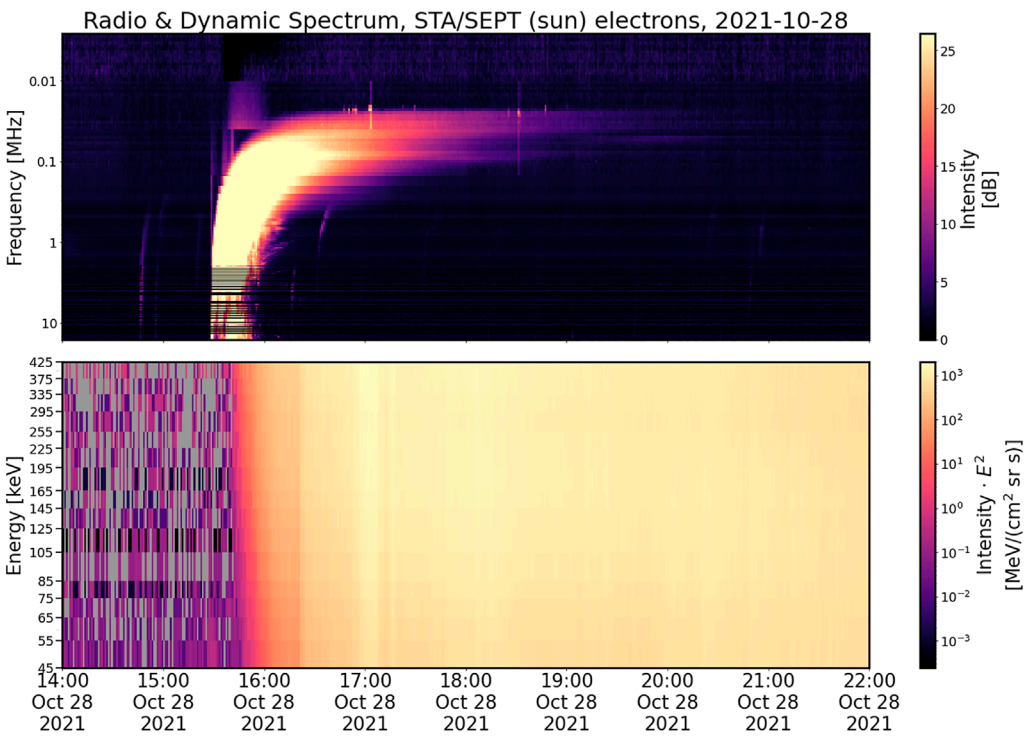


FIGURE 7
The output of the dynamic spectrum tool for the 28 October 2021 SEP event with radio plotting enabled. The upper panel shows the radio spectrum as observed by STEREO-A/SWAVES and the lower panel the dynamic electron spectrum of STEREO-A/SEPT.

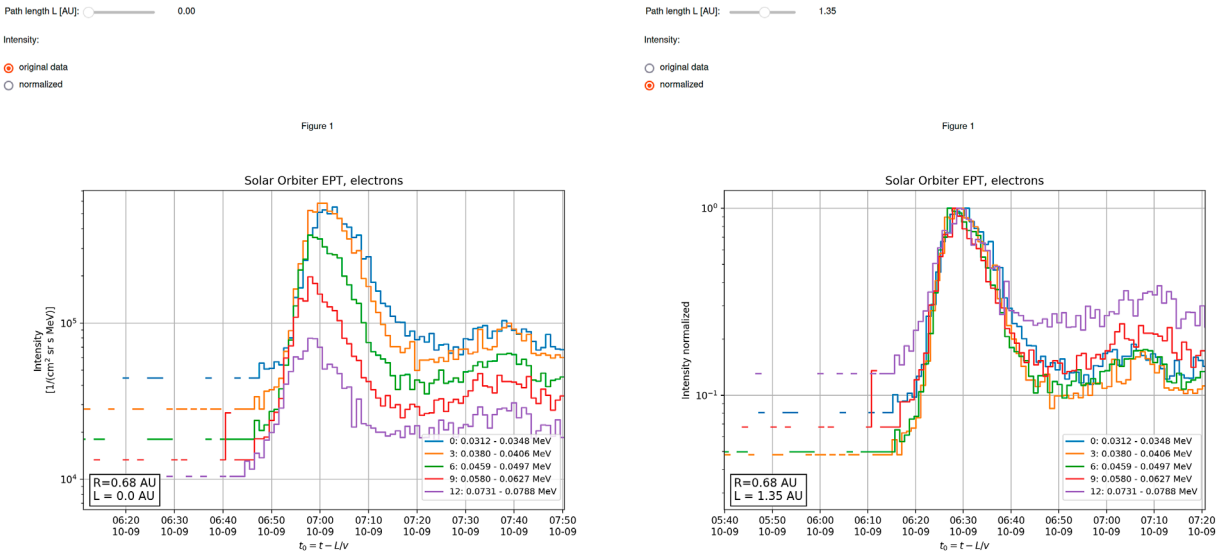


FIGURE 8
Unaltered (left) and time-shifted (right) electron intensity time series of the 9 October 2021 SEP event observed by Solar Orbiter/EPT. The bottom left panels inside both plots show the heliocentric distance (R) of the spacecraft at the time of the event and the path length (L) that corresponds to the time-shift. The time-shifted plot on the right has been normalized to the maximum intensity of the event and time-shifted by eye so that the rising flanks of the different channels align, yielding a travelled path length of 1.35 AU. At the time of the event Solar Orbiter was located at a distance of 0.68 AU from the Sun, which corresponds to a path length of 0.73 AU, assuming a nominal Parker spiral with solar wind speed of 400 km/s.

There are only two optional parameters that the user can change: `averaging` works exactly as it does in onset analysis notebook, with a Pandas-compatible time string, and cmap accepts the name of one of Matplotlib's (Hunter, 2007) colormaps as a string. By default, `averaging` is set to `None` and `cmap` is set to "`magma`".

Furthermore, the user also has the option to accompany the dynamic spectrum with a radio spectrum, as is shown in **Figure 7**. The radio spectrum in its current version only supports the SWAVES instruments onboard STEREO-A and B. It is planned to add support for the Wind/WAVES instrument in a future version. The maximum intensity in each spectrum is set to 70% of the maximum intensity level of the time window specified. This is done to avoid saturation and to enhance weaker signals. The radio spectrum can be used to show the onset of radio bursts at low frequencies (<10 MHz), local Langmuir waves and electrostatic disturbances at the spacecraft.

2.4 Visual time shift analysis

Velocity Dispersion Analysis (VDA) is a method to estimate the path length travelled by SEPs and the time when these SEPs were injected into the interplanetary medium. Often a VDA is not possible, for example, when the onset times in different energy channels do not show significant differences or when the onset times cannot be identified with high accuracy. For such cases, an alternative can be to use the rising phase at a fraction of the peak intensity instead of the classic onset of the event (e.g., Zhao et al., 2019). VDA also does not take into account the pitch-angle scattering of particles in the interplanetary medium, which is why care has to be exercised in its application (Laitinen et al., 2015). To still infer an SEP injection time, an alternative method, the Time Shift Analysis (TSA, e.g. Vainio et al., 2013) can be used, which usually employs only a single energy channel. Therefore, the information on the path length cannot be inferred. In our tool, we combine features of both methods by applying a TSA to a number of energy channels together. The basic assumption behind this tool is that the particles propagated along a common path length. The interactive time shift analysis tool presented here shows the particle intensities of the different energy channels of the same instrument as time series in a single plot (see **Figure 8** left). The tool then allows to interactively apply different path lengths to the data. For each path length the time profiles of all chosen energy channels are shifted backwards in time corresponding to the particle's travel time along the chosen path length:

$$t_0 = t - \frac{L}{v}, \tag{6}$$

where t is the initial timestamp of an observation without a shift, L is the assumed path length (i.e., along the interplanetary magnetic field line), and v is the speed of the particles. By varying L, the time series data is shifted differently far backwards. The plot is updated automatically during runtime. Since different energy channels detect particles of different speeds, the plotted time series of different energy channels will be shifted at different rates. We note that all energy channels cover a finite energy range, so there will always be a range of speeds within the particle population in any given energy channel. Nonetheless, we take the geometric mean energy of a channel as an approximation to represent the kinetic energy of all particles in that energy range. The relativistic speed of a particle is then calculated according to:

$$v = c\sqrt{1 - \left(\frac{mc^2}{E + mc^2}\right)^2}, \tag{7}$$

where E is the mean energy of a channel, m the mass of a particle, and c is the speed of light. A reasonable path length, which explains the observed velocity dispersion, is found when the rise phases of all chosen energy channels collapse, i.e., when they have been shifted so that they align at the same time. This time corresponds to the inferred injection time. There is also the option to normalize all displayed energy channels to the maximum intensity in the considered time frame, which helps to align the rise phases in different energy channels. **Figure 8** illustrates the usage of the tool by showing the unaltered intensity time series of multiple channels (left) next to normalized time series that have been time-shifted corresponding to a specific path length so that the onsets of the different channels align in a single onset (right).

3 Discussion

In this paper, we present the first tools for analyzing SEP events that are provided through the SERPENTINE project to the community. These Jupyter Notebooks are designed to alleviate the significant level of manual work that is an integral part of the study of any SEP event. They open up key pieces of energetic particle analysis for multi-spacecraft studies in an unprecedentedly easy way. The tools themselves require only a minimal amount of programming knowledge to utilize, while simultaneously giving the user access to powerful methods and visualization of data. There is, however, still room for some subjectivity in the interpretation of the results, e.g., for the onset and TSA tools. Also, the tools themselves require that the user is mindful of the methods and assumptions behind the curtain, in order to deliver scientifically valid analysis results. For example, it is still essential to read the corresponding data product descriptions and understand the specific caveats of every instrument before using the data. **Figure 3** gives an example on the importance of understanding such instrument caveats. Thus, the tools are aimed at experts who are familiar with such kind of analyses.

The development of these tools is an ongoing endeavor, and the source code is open to everyone on GitHub. In the future,

the user interface of the tools is planned to also support missions such as Wind and Parker Solar Probe, which so far are only partially supported. In addition, there are plans to include other data sets, such as X-ray and proton observations by GOES and radio data from other missions. Another goal is to provide access to higher level data products that are planned to be delivered by the SERPENTINE project, such as pitch-angle distributions or event catalogs. The existing tools will see further improvement, and new tools, such as VDA uncertainty estimation and SEP spectra analysis, are also under development. We encourage the scientific community to participate in the development by giving feedback about possible bugs and requested new functionalities *via* GitHub issues, but we are of course also open for direct communication *via*, e.g., email.

Data availability statement

The source code used in this study can be found online:

- SEPpy Python package code in GitHub repository SEPpy, archived at https://doi.org/10.5281/zenodo.7216077
- Solo-epd-loader Python package code in GitHub repository solo-epd-loader, archived at https://doi.org/10.5281/zenodo.7100451
- Python code and example Jupyter Notebooks in GitHub repository serpentine, archived at https://doi.org/10.5281/zenodo.7221070.

Author contributions

CP, JG, ND, EA, and RV discussed the tools. JG wrote the software for data loaders. CP, AY-L, EA, and JG wrote the software for onset determination. CP wrote the software for dynamic spectra and visual TSA. DM wrote the software for radio data. AY-L, DP, and SV tested the software and provided critical comments for the development. CP, JG, and ND prepared the first paper draft, and all authors were involved in the preparation of the final manuscript. All authors revised the manuscript before submission.

Funding

This study has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101004159 (SERPENTINE). CP, JG, RV, DP, and EA acknowledge the support of Academy of Finland (FORESAIL, grants 312357, 336809, and 336807). ND acknowledges the support of Academy of Finland (SHOCKSEE, grant 346902). DM acknowledges support from the Academy of Finland (RadioCME, grant 333859). DP and EA acknowledge the ERC under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 724391, SolMAG). EA acknowledges the support of the Academy of Finland (TRAMSEP, grant 322455).

Acknowledgments

The authors would like to thank everyone who helped to improve the tools described in this manuscript by providing feedback or contributing to the various open-source projects that it utilizes. The authors thank the two reviewers for their valuable feedback.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

The SunPy Community, Barnes, W. T., Bobra, M. G., Christe, S. D., Freij, N., Hayes, L. A., Ireland, J., et al. (2020). The SunPy project: Open source development and status of the version 1.0 core package. *Astrophys. J.* 890, 68. doi:10.3847/1538-4357/ab4f7a

Desai, M., and Giacalone, J. (2016). Large gradual solar energetic particle events. *Living Rev. Sol. Phys.* 13, 3. doi:10.1007/s41116-016-0002-5

Domingo, V., Fleck, B., and Poland, A. I. (1995). The SOHO mission: An overview. *Sol. Phys.* 162, 1–37. doi:10.1007/BF00733425

Dresing, N., Gómez-Herrero, R., Klassen, A., Heber, B., Kartavykh, Y., and Dröge, W. (2012). The large longitudinal spread of solar energetic particles during the 17 january 2010 solar event. *Sol. Phys.* 281, 281–300. doi:10.1007/s11207-012-0049-y

European Organization For Nuclear Research and OpenAIRE (2013). *European organization for nuclear research and OpenAIRE*. Geneva: CERN. doi:10.25495/7GXK-RD71

Fox, N. J., Velli, M. C., Bale, S. D., Decker, R., Driesman, A., Howard, R. A., et al. (2016). The solar Probe plus mission: Humanity's first visit to our star. *Space Sci. Rev.* 204, 7–48. doi:10.1007/s11214-015-0211-6

Gieseler, J., Dresing, N., Palmroos, C., Freiherr von Forstner, J. L., Price, D. J., Vainio, R., et al. (2022a). Solar-MACH: An open-source tool to analyze solar magnetic connection configurations. *Front. Astronomy Space Phys*. (submitted).

Gieseler, J., Palmroos, C., Dresing, N., Morosan, D., Asvestari, E., Yli-Laurila, A., et al. (2022b). Solar EneRgetic ParticlE aNalysis plaTform for the INner hEliosphere. SERPENTINE. doi:10.5281/zenodo.7221070

Gieseler, J., Palmroos, C., Dresing, N., Morosan, D., Asvestari, E., Yli-Laurila, A., et al. (2022c). Solar energetic particle time series analysis with Python. serpentine-h2020/SEPpy: 0.1.3. doi:10.5281/zenodo.7216077

Gieseler, J., Palmroos, C., Dresing, N., Morosan, D., Asvestari, E., Yli-Laurila, A., et al. (2022d). Solar energetic particle time series analysis with Python. jgieseler/solo-epd-loader: 0.1.10. doi:10.5281/zenodo.7100451

Harten, R., and Clark, K. (1995). The design features of the GGS wind and polar spacecraft. *Space Sci. Rev.* 71, 23–40. doi:10.1007/BF00751324

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Huttunen-Heikinmaa, K., Valtonen, E., and Laitinen, T. (2005). Proton and helium release times in SEP events observed with SOHO/ERNE. *Astron. Astrophys.* 442, 673–685. doi:10.1051/0004-6361:20042620

Kaiser, M. L., Kucera, T. A., Davila, J. M., Cyr, St.O. C., Guhathakurta, M., and Christian, E. (2008). The STEREO mission: An introduction. *Space Sci. Rev.* 136, 5–16. doi:10.1007/s11214-007-9277-0

Kaiser, M. L. (2005). The STEREO mission: An overview. *Adv. Space Res.* 36, 1483–1488. doi:10.1016/j.asr.2004.12.066

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., et al. (2016). Jupyter notebooks—A publishing format for reproducible computational workflows. *Position. Power Acad. Publ. Play. Agents Agendas* 87. IOS Press. doi:10.3233/978-1-61499-649-1-87

Laitinen, T., Huttunen-Heikinmaa, K., Valtonen, E., and Dalla, S. (2015). Correcting for interplanetary scattering in velocity dispersion analysis of solar energetic particles. *Astrophys. J.* 806, 114. doi:10.1088/0004-637X/806/1/114

Lintunen, J., and Vainio, R. (2004). Solar energetic particle event onset as analyzed from simulated data. *Astron. Astrophys.* 420, 343–350. doi:10.1051/0004-6361:20034247

Lucas, J. M. (1985). Counted data CUSUM's. *Technometrics* 27, 129–144. doi:10.1080/00401706.1985.10488030

McComas, D. J., Alexander, N., Angold, N., Bale, S., Beebe, C., Birdwell, B., et al. (2016). Integrated science investigation of the Sun (ISIS): Design of the energetic particle investigation. *Space Sci. Rev.* 204, 187–256. doi:10.1007/s11214-014-0059-1

McKinney, W. (2010). "Data structures for statistical computing in Python," in Proceedings of the 9th Python in Science Conference, Austin, Texas, June 28-July 3, 2010. doi:10.25080/Majora-92bf1922-00a

Müller, D., Cyr, St.O. C., Zouganelis, I., Gilbert, H. R., Marsden, R., Nieves-Chinchilla, T., et al. (2020). The Solar Orbiter mission. Science overview. *Astron. Astrophys.* 642, A1. doi:10.1051/0004-6361/202038467

Mumford, S. J., Freij, N., Stansby, D., Christe, S., Ireland, J., Mayer, F., et al. (2022). SunPy. Zenodo. doi:10.5281/zenodo.7074315

Paassilta, M., Papaioannou, A., Dresing, N., Vainio, R., Valtonen, E., and Heber, B. (2018). Catalogue of > 55 MeV wide-longitude solar proton events observed by SOHO, ACE, and the STEREOs at ≈ 1 $≈ 1$ AU during 2009 – 2016. *Sol. Phys.* 293, 70. doi:10.1007/s11207-018-1284-7

Page, E. S. (1954). Continuous inspection schemes. *Biometrika* 41, 100–115. doi:10.1093/biomet/41.1-2.100

Reames, D. V. (2021). Solar energetic particles. *Sol. Energetic Part. A Mod. Primer Underst. Sources, Accel. Propag.* 978. doi:10.1007/978-3-030-66402-2

Rodríguez-Pacheco, J., Wimmer-Schweingruber, R. F., Mason, G. M., Ho, G. C., Sánchez-Prieto, S., Prieto, M., et al. (2020). The energetic particle detector. Energetic particle instrument suite for the solar orbiter mission. *Astron. Astrophys.* 642, A7. doi:10.1051/0004-6361/201935287

Stansby, D., Harter, B., Sdc, M., Kemenade, H., Burrell, A., Hirch, M., et al. (2022). MAVENSDC/cdflib. Zenodo. doi:10.5281/zenodo.7011489

Stansby, D., Rai, Y., Argall, M., Jeffrey, B., Haythornthwaite, R., Teunissen, J., et al. (2021). heliopython/heliopy: Heliopy 0.15.4. Zenodo. doi:10.5281/zenodo.5090511

Vainio, R., Valtonen, E., Heber, B., Malandraki, O. E., Papaioannou, A., Klein, K.-L., et al. (2013). The first*SEPServer*event catalogue ∼68-MeV solar proton events observed at 1 AU in 1996–2010. *J. Space Weather Space Clim.* 3, A12. doi:10.1051/swsc/2013030

Zhao, L., Li, G., Zhang, M., Wang, L., Moradi, A., and Effenberger, F. (2019). Statistical analysis of interplanetary magnetic field path lengths from solar energetic electron events observed by wind. *Astrophys. J.* 878, 107. doi:10.3847/1538-4357/ab2041

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 20 December 2022
DOI 10.3389/fspas.2022.1076747

Check for updates

OPEN ACCESS

EDITED BY
Sophie A. Murray,
Dublin Institute for Advanced Studies (DIAS), Ireland

REVIEWED BY
Rui Liu,
University of Science and Technology of China, China
Anthony R. Yeates,
Durham University, United Kingdom

*CORRESPONDENCE
D. J. Price,
daniel.price@helsinki.fi

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 21 October 2022
ACCEPTED 25 November 2022
PUBLISHED 20 December 2022

CITATION
Price DJ, Pomoell J and Kilpua EKJ (2022), MAFIAT: Magnetic field analysis tools. *Front. Astron. Space Sci.* 9:1076747. doi: 10.3389/fspas.2022.1076747

COPYRIGHT
© 2022 Price, Pomoell and Kilpua. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

MAFIAT: Magnetic field analysis tools

D. J. Price*, J. Pomoell and E. K. J. Kilpua

Department of Physics, University of Helsinki, Helsinki, Finland

The magnetic twist is one of the key defining parameters of solar flux ropes (FRs). The routine computation of the winding of magnetic field lines, referred to as the twist, has the potential to lead to significant advancement in the field of solar physics and solar—terrestrial research, e.g., by enabling more accurate investigations of FR morphology, stability, and temporal evolution. However, this has been hampered by the axial-dependence of the solution and the availability of simpler, albeit approximate, methods. Here we introduce the Magnetic Field Analysis Tools (MAFIAT) python library and Jupyter notebooks for the computation and exploitation of this quantity. The required axis location is specified manually by the user, either with their own preferred method, or using the twist number calculated from the parallel current by Magnetic Field Analysis Tools. The notebooks allow users to create a variety of novel visualisations of FRs and their twist distributions for scientific study. Magnetic Field Analysis Tools is written in Python and is released under the BSD 3-Clause Licence. Code available at: https://github.com/pricedj/mafiat.

KEYWORDS

python (programming language), coronal mass ejection (CME), solar, software package, corona, magnetic field

1 Introduction

Coronal mass ejections (CMEs; Webb and Howard, 2012) are gigantic structures of magnetic field and plasma that regularly erupt from the Sun and serve as one of the primary drivers of space weather in the heliosphere. Their study is critical due to their vast and often adverse impact, ranging from human spaceflight to national power grids and other infrastructure (e.g., Eastwood et al., 2017; Kilpua et al., 2017). An integral component of CMEs are flux ropes, helical structures where magnetic field lines wind about a common axis (Chen, 2017). The understanding of CMEs and their impact requires a detailed knowledge of their magnetic fields and their associated flux ropes (FRs), from their formation in the corona to their eventual eruption and propagation (e.g., Kilpua et al., 2019). However, directly measuring coronal magnetic fields is very difficult and as a result modelling is usually employed (Wiegelmann et al., 2017).

Flux ropes possess a number of fundamental properties that are also useful to study in their own right. Here we focus on one of the most critical parameters, the magnetic twist, which describes the number of turns that a magnetic field line makes around the FR axis. Computations of the twist sheds light on how stable the FR is, and therefore how it evolves and the likelihood of it erupting (e.g., Amari et al., 2018;

Inoue et al., 2018; Zhong et al., 2021). For example, in the case of the kink instability, a FR is said to be unstable and likely to erupt if twisted beyond a critical value (Török and Kliem, 2003). There are several distinct quantities that are commonly called twist which, in general, do not measure the same thing. As a result, 'twist' values must always be considered alongside the methodology employed (Liu et al., 2016).

A commonly used metric for quantifying twist is the twist number T_w which uses the parallel current and magnetic field to compute how many turns two infinitesimally close field lines wind about each other (Berger and Prior, 2006, Eq. 16). A simpler quantity is T_α, computed using the force-free parameter α at the footpoints of the FR (Inoue et al., 2011). However, these methods, and others, notably do not factor in the geometry of the field lines. The twist number T_g introduced by Berger and Prior (2006) (see their Eq. 12) takes this into consideration. T_g describes how many times each field line winds about the flux rope axis. It is defined as

$$T_g = \frac{1}{2\pi}\int_x \frac{d\varphi}{ds}ds = \frac{1}{2\pi}\int_x \hat{\mathbf{T}}(s)\cdot\hat{\mathbf{V}}(s)\times\frac{d\hat{\mathbf{V}}(s)}{ds}ds \qquad (1)$$

Where $x(s)$ defines the flux rope axis, φ the rotation angle made by the field line under consideration about $x(s)$, $\hat{\mathbf{T}}$ the unit tangent vector to $x(s)$ (i.e., the local direction of the flux rope axis), and $\hat{\mathbf{V}}$ the unit vector normal to $x(s)$ which intersects the field line under consideration. This method is more computationally expensive and more difficult to employ, but it factors in the geometry of the FR. For a comparison of these, and other, twist methods, see Liu et al. (2016).

This paper serves to introduce the newly released collection of Magnetic Field Analysis Tools (MAFIAT). The primary purpose of MAFIAT is to compute the quantity T_g and to provide useful visualisations of its distribution throughout the flux rope.

2 Methods

2.1 Implementation

MAFIAT is designed to support the analysis of magnetic fields in coronal simulations. It is written in Python 3 and is freely available from GitHub. Additionally, it is compatible with PyPI, enabling the installation of MAFIAT, along with its dependencies, with the widely used `pip` command-line tool. The primary workflow is in the form of the included Jupyter Notebooks, however, the functions can also be used in scripts as desired.

The list of required dependencies is kept as short as possible to minimise the potential for compatibility issues. At present, MAFIAT requires the widely used `ipykernel`, `k3d`, `matplotlib`, `numba`, `numpy`, and `scipy` packages. Additionally, it requires the newly-released `pySMSH` package which provides tools for working with grid-based simulation data.

2.2 Overview

The primary workflow of MAFIAT enables the computation of T_g for flux ropes in coronal simulation domains. This consists of four main steps:

1. Data loading and initialisation
2. Computing T_w to locate the FR axis
3. Computing T_g
4. Visualising the results

The first three steps take part in the `Computing_Tg` notebook, and the fourth step takes place in the `Visualising_Tg` notebook, both provided in the notebooks folder of the repository. The following sections describe these steps in more detail.

2.2.1 Initialisation

MAFIAT requires an input file with magnetic field data and details of the corresponding Cartesian rectilinear grid. From this, magnetic field and current density interpolators are initialised to facilitate the computation of T_w and the tracing of magnetic field lines through the simulation domain.

2.2.2 Axis location

After initialisation, the user is required to select a 2D plane that is orthogonal with one of the coordinate axes. It is advised that a plane bisecting the apex of the FR is chosen as closely as possible. Following this, the `compute_Tw_map` function is used to calculate T_w for the chosen plane.

The plane is then processed in a series of steps to identify the flux rope and facilitate locating the axis. First, any open field lines are removed, i.e., any field lines where both ends are not rooted in the photosphere. Second, any regions of small twist number ($|T_w| < 0.99$) are removed. This choice provides a good estimate of FR extent in many cases, but can be changed by the user as needed. Third, if a horizontal slice (i.e., a cut in z) was made then any regions with a positive (or negative) polarity are removed so that the FR only threads the plane once. Fourth, any points with the wrong sign of T_w are removed based on user selection. The fifth, and final, step is manual trimming of any other undesired regions of T_w. The remaining region is considered to correspond to the FR and its coordinates are saved for future use. We note that none of the steps above are strictly mandatory, and can be omitted.

Next, T_w is computed again at a higher resolution, for the region defined following the previous procedure and plotted with contours to assist in identifying the axis, which is considered to be at an extremum of T_w. The user should adjust the values of the contour levels until the axis is well-defined. In practice, this can mean adjusting the values by fractional amounts in a series of trial and error. Then the centre of the axis contour is taken forward as the axis of the FR (**Figure 1**).

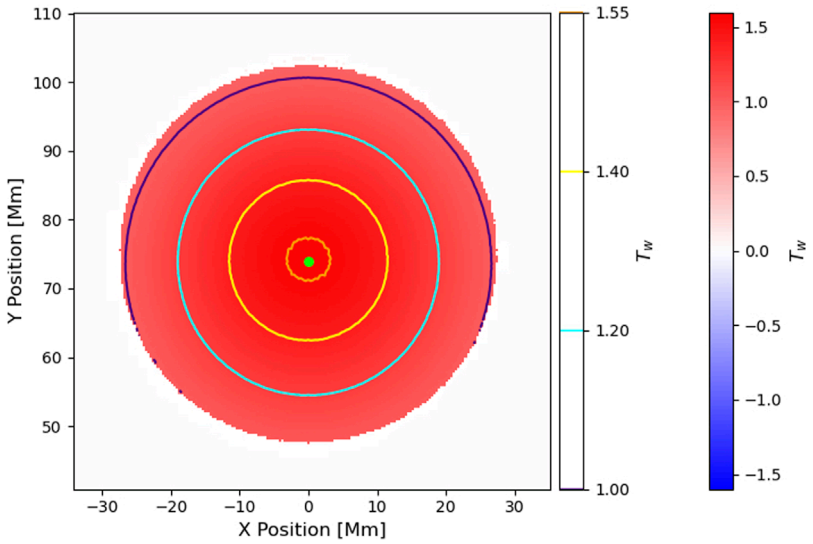


FIGURE 1
Output from the axis locating step, showing a high-resolution plot of T_w with contours of T_w overlaid. A lime dot indicates the centre of the highest contour, taken forward as the flux rope axis.

The axis location is then traced to determine the axis coordinates, and the saved FR coordinates from above are also traced to give the first visualisation of the FR and its axis so the user can look for any abnormalities (**Figure 2**).

2.2.3 Computing T_g

When the axis coordinates are confirmed, the corresponding unit tangent vectors ($\hat{T}$) are first computed. These are then used in a loop through each field line to find the unit normal vector from each point along the axis to a point on the chosen field line using the `find_V` function. For this step, the field lines are traced at a higher resolution than the axis to provide a greater search interval. For a given axis coordinate, unit vectors are computed to a range of field line coordinates. The pairing with the dot product closest to 0 is then taken as the sought-after normal vector and the process repeats for the next axis coordinate. To minimise mistaken matches in complex cases it is required that the rotation angle between consecutive unit normal vectors does not exceed a given value. Additionally, when all axis coordinates have been matched with a field line coordinate, the inner 80% (i.e., excluding 10% from each footpoint) of the paired unit tangent vectors and unit normal vectors are tested to ensure their dot products do not exceed 0.01. Where this occurs, the unit vector normals for the entire field line are returned as NaN because it indicates that the field line may not belong to the FR or potentially bad search parameters.

When the unit normals have been computed for each field line in the FR, the distance along the axis is calculated using the `calc_s` function. The `calc_dVds` function is then used to perform simple differencing to evaluate the latter part of Eq. **1**, namely, $\frac{d\hat{\mathbf{V}}(s)}{ds}$. The cross product of this with the unit vector normals is then computed. Finally the dot product of that result with the unit tangent normal is computed and all multiplied by $ds/2\pi$ for each point along the axis for each field line. The value for each step is saved as `Tg_comps`, i.e., the contribution to the T_g integral for each axis coordinate, and they are summed to return the final value of T_g for the field line.

2.2.4 Visualisation

The visualisation notebook reads the file generated by the computation notebook (stored in the Numpy npz format) and first allows the user to produce plots of T_g and T_w (**Figure 3**) for the FR coordinates and 2D plane selected for the axis determination (**Section 2.2.2**). Secondly, the user can visualise the FR and its axis. This is also done in the computation notebook, but is repeated here as a form of validation that the file was read in correctly, and for creating plots without having to redo the computation steps. Finally, it provides a visualisation of the FR whereby each segment of the field lines are coloured based on their local contribution to the total T_g integral for that field line, which illustrates how T_g is locally distributed along the field line (**Figure 4**). The two field line plots can be generated for all field lines intersecting the chosen plane or for the field lines from the plane that satisfy a T_g threshold.

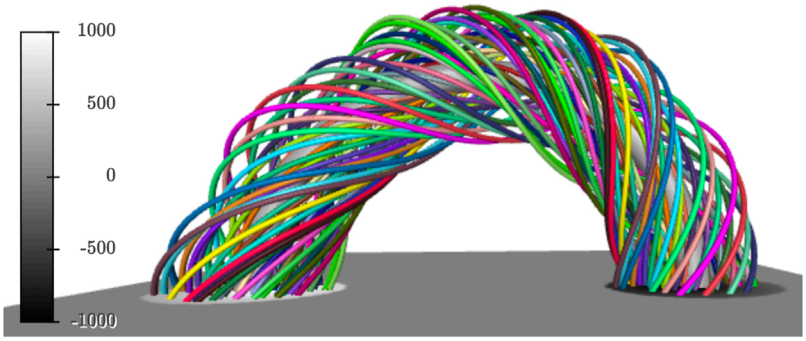


FIGURE 2
Output from the first visualisation of the flux rope. The axis is shown by a thicker grey line while the other field lines are shown by randomly coloured thinner lines. B_z from the lower boundary of the domain is also shown.

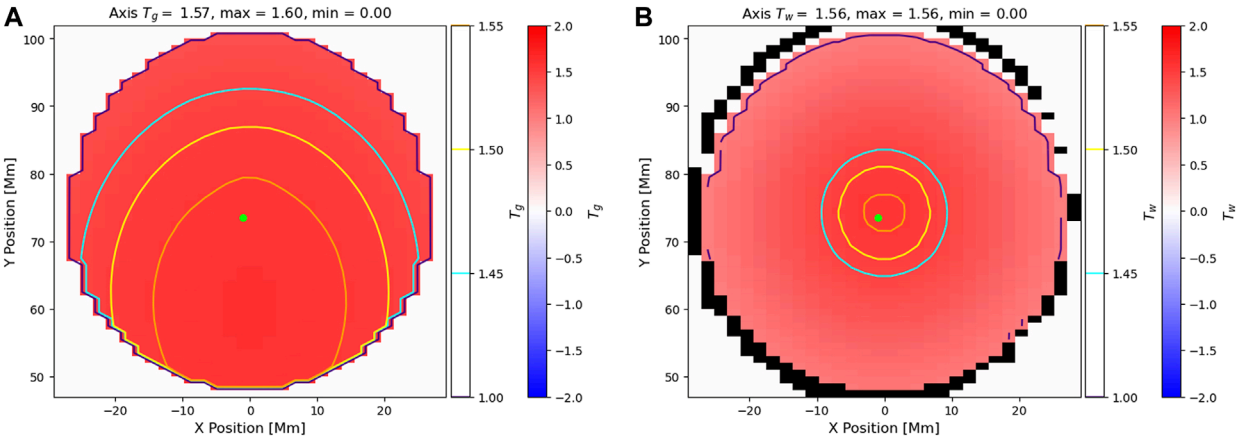


FIGURE 3
Output from the visualisation notebook showing the T_g **(A)** and T_w **(B)** planes with their corresponding contours overlaid. A lime dot indicates the centre chosen in **Figure 1**. The dot is slightly displaced due to the difference in resolution.

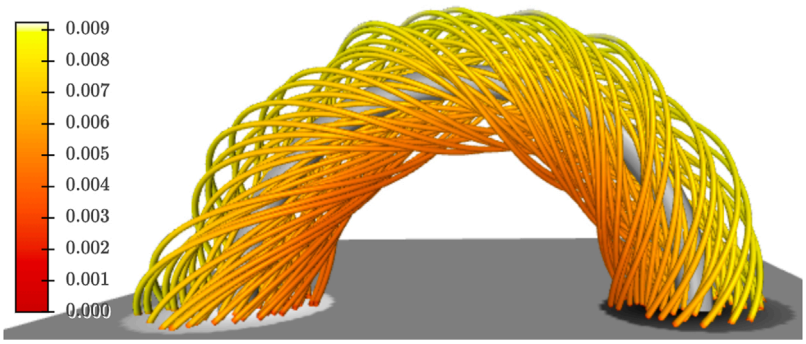


FIGURE 4
Output from the visualisation notebook. The axis is shown by a thicker grey line while the other field lines are shown by thinner lines coloured based on their local contribution to the total T_g integral. B_z at the lower boundary of the domain is also shown.

3 Discussion

We have presented a tool to investigate one of the most critical flux rope parameters, the magnetic twist, using the geometrically dependent T_g quantity and a suite of advanced visualisations such as the local contribution plot (**Figure 4**) which enables users to see how twist is distributed along the field lines of their flux ropes. The present workflow is liable to exclude part of the FR structure because T_w is initially used to define which field lines belong to the FR for the purposes of computing T_g. This means the Notebook provided is more suited to studying the distribution of the twist instead of FR identification. However, the user is free to select points in other ways instead of rigidly following the Notebook as written. In future versions we will implement alternative examples and provide them as Notebooks.

A current limitation for MAFIAT is that it is not suitable for low-lying FRs where field lines make multiple crossings through the photosphere because the field lines are traced to the photosphere at each end. Furthermore, for T_g in particular, FRs that are inclined at their footpoints may yield underestimations for T_g if it is not possible to find a unit normal from the axis to the target field line without going below the photosphere. Improved field line tracing and local interpolation are under development for future versions to handle these situations, which will ensure the found unit normals are more robustly computed and reduce the dependency of accuracy on field line resolution.

Presently all computations are serial. However, given that the computation of T_g is independent for each field line, they are able to be computed simultaneously. This will be addressed in a future version to improve performance. Furthermore, we plan to implement additional visualisation features such as a framework for the simple selection and investigation of individual field lines in the FR. These, and other features, highlight the fact that MAFIAT will remain under active development, with a number of improvements planned over the coming year.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article.

Author contributions

DP developed the MAFIAT software with contributions from JP. All authors contributed to manuscript preparation and revision, and read and approved the submitted version.

Funding

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 724391 (SolMAG) and 101004159 (SERPENTINE). The work leading to these results has been carried out in the Finnish Centre of Excellence in Research of Sustainable Space (Academy of Finland grant number 336807). JP acknowledges the Academy of Finland Project 343581 (SWATCH).

Acknowledgments

This research has made use of NASA's Astrophysics Data System.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Amari, T., Canou, A., Aly, J.-J., Delyon, F., and Alauzet, F. (2018). Magnetic cage and rope as the key for solar eruptions. *Nature* 554, 211–215. doi:10.1038/nature24671

Berger, M. A., and Prior, C. (2006). The writhe of open and closed curves. *J. Phys. A Math. Gen.* 39, 8321–8348. doi:10.1088/0305-4470/39/26/005

Chen, J. (2017). Physics of erupting solar flux ropes: Coronal mass ejections (CMEs)—Recent advances in theory and observation. *Phys. Plasmas* 24, 090501. doi:10.1063/1.4993929

Eastwood, J. P., Biffis, E., Hapgood, M. A., Green, L., Bisi, M. M., Bentley, R. D., et al. (2017). The economic impact of space weather: Where do we stand? *Risk Anal.* 37, 206–218. doi:10.1111/risa.12765

Inoue, S., Kusano, K., Büchner, J., and Skála, J. (2018). Formation and dynamics of a solar eruptive flux tube. *Nat. Commun.* 9, 174. doi:10.1038/s41467-017-02616-8

Inoue, S., Kusano, K., Magara, T., Shiota, D., and Yamamoto, T. T. (2011). Twist and connectivity of magnetic field lines in the solar active region. *Astrophys. J.* 738, 161. doi:10.1088/0004-637X/738/2/161

Kilpua, E. K. J., Lugaz, N., Mays, M. L., and Temmer, M. (2019). Forecasting the structure and orientation of earthbound coronal mass ejections. *Space weather.* 17, 498–526. doi:10.1029/2018SW001944

Kilpua, E., Koskinen, H. E. J., and Pulkkinen, T. I. (2017). Coronal mass ejections and their sheath regions in interplanetary space. *Living Rev. Sol. Phys.* 14, 5. doi:10.1007/s41116-017-0009-6

Liu, R., Kliem, B., Titov, V. S., Chen, J., Wang, Y., Wang, H., et al. (2016). Structure, stability, and evolution of magnetic flux ropes from the perspective of magnetic twist. *Astrophys. J.* 818, 148. doi:10.3847/0004-637X/818/2/148

Török, T., and Kliem, B. (2003). The evolution of twisting coronal magnetic flux tubes. *Astron. Astrophys.* 406, 1043–1059. doi:10.1051/0004-6361:20030692

Webb, D. F., and Howard, T. A. (2012). Coronal mass ejections: Observations. *Living Rev. Sol. Phys.* 9, 3. doi:10.12942/lrsp-2012-3

Wiegelmann, T., Petrie, G. J. D., and Riley, P. (2017). Coronal magnetic field models. *Space Sci. Rev.* 210, 249–274. doi:10.1007/s11214-015-0178-3

Zhong, Z., Guo, Y., and Ding, M. D. (2021). The role of non-axisymmetry of magnetic flux rope in constraining solar eruptions. *Nat. Commun.* 12, 2734. doi:10.1038/s41467-021-23037-8

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 12 January 2023
DOI 10.3389/fspas.2022.1048318

Check for updates

OPEN ACCESS

EDITED BY
Leslie Lamarche,
SRI International, United States

REVIEWED BY
Yue Deng,
University of Texas at Arlington, United States
Chao Xiong,
Electronic Information School, Wuhan University, China

*CORRESPONDENCE
Theodoros Sarris,
tsarris@ee.duth.gr

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 19 September 2022
ACCEPTED 30 November 2022
PUBLISHED 12 January 2023

CITATION
Sarris TE, Tourgaidis S, Pirnaris P, Baloukidis D, Papadakis K, Psychalas C, Buchert SC, Doornbos E, Clilverd MA, Verronen PT, Malaspina D, Ahmadi N, Dandouras I, Kotova A, Miloch WJ, Knudsen D, Olsen N, Marghitu O, Matsuo T, Lu G, Marchaudon A, Hoffmann A, Lajas D, Strømme A, Taylor M, Aikio A, Palmroth M, Heelis R, Ivchenko N, Stolle C, Kervalishvili G, Moretto-Jørgensen T, Pfaff R, Siemes C, Visser P, van den Ijssel J, Liu H-L, Sandberg I, Papadimitriou C, Vogt J, Blagau A and Stachlys N (2023), Daedalus MASE (mission assessment through simulation exercise): A toolset for analysis of in situ missions and for processing global circulation model outputs in the lower thermosphere-ionosphere.
Front. Astron. Space Sci. 9:1048318.
doi: 10.3389/fspas.2022.1048318

Daedalus MASE (mission assessment through simulation exercise): A toolset for analysis of *in situ* missions and for processing global circulation model outputs in the lower thermosphere-ionosphere

Theodore E. Sarris[1]*, Stelios Tourgaidis[1], Panagiotis Pirnaris[1], Dimitris Baloukidis[1], Konstantinos Papadakis[1,2], Christos Psychalas[1], Stephan Christoph Buchert[3], Eelco Doornbos[4], Mark A. Clilverd[5], Pekka T. Verronen[6], David Malaspina[7], Narghes Ahmadi[7], Iannis Dandouras[8], Anna Kotova[8], Wojciech J. Miloch[9], David Knudsen[10], Nils Olsen[11], Octav Marghitu[12], Tomoko Matsuo[13], Gang Lu[14], Aurélie Marchaudon[8], Alex Hoffmann[15], Dulce Lajas[15], Anja Strømme[15], Matthew Taylor[15], Anita Aikio[16], Minna Palmroth[2], Roderick Heelis[17], Nickolay Ivchenko[18], Claudia Stolle[19], Guram Kervalishvili[20], Therese Moretto-Jørgensen[21], Robert Pfaff[22], Christian Siemes[23], Pieter Visser[23], Jose van den Ijssel[23], Han-Li Liu[14], Ingmar Sandberg[24], Constantinos Papadimitriou[24], Joachim Vogt[25], Adrian Blagau[12] and Nele Stachlys[25]

[1]Department of Electrical Computer Engineering, Democritus University of Thrace, Xanthi, Greece, [2]Department of Physics, University of Helsinki, Helsinki, Finland, [3]Swedish Institute of Space Physics (IRF), Uppsala, Sweden, [4]Royal Netherlands Meteorological Institute KNMI, Utrecht, Netherlands, [5]British Antarctic Survey (UKRI-NERC), Cambridge, United Kingdom, [6]Space and Earth Observation Centre, Finnish Meteorological Institute, Helsinki, Finland, [7]Laboratory for Atmospheric and Space Physics, University of Colorado, Boulder, CO, United States, [8]IRAP, Université de Toulouse, CNRS CNES, Toulouse, France, [9]Department of Physics, University of Oslo, Oslo, Norway, [10]Department of Physics and Astronomy, University of Calgary, Calgary, AB, Canada, [11]DTU Space, Technical University of Denmark, Copenhagen, Denmark, [12]Institute for Space Sciences, Bucharest, Romania, [13]Ann and H.J. Smead Department of Aerospace Engineering Sciences, Boulder, CO, United States, [14]High Altitude Observatory, National Center for Atmospheric Research, Boulder, CO, United States, [15]European Space Research and Technology Centre, European Space Agency, Noordwijk, Netherlands, [16]Space Physics and Astronomy Research Unit, University of Oulu, Oulu, Finland, [17]Center for Space Sciences, University of Texas at Dallas, Dallas, TX, United States, [18]Div of Space and Plasma Physics, Royal Institute of Technology KTH, Stockholm, Sweden, [19]Leibniz Institute of Atmospheric Physics, Rostock, Germany, [20]GFZ German Research Centre for Geosciences, Potsdam, Germany, [21]NASA Ames Research Center, Moffett Field, CA, United States, [22]Heliophysics Science Division, NASA/Goddard Space Flight Center, Greenbelt, MD, United States, [23]Faculty of Aerospace Engineering, Delft University of Technology, Delft, Netherlands, [24]Space Applications Research Consultancy, Athens, Greece, [25]School of Science, Jacobs University Bremen, Bremen, Germany

Daedalus MASE (Mission Assessment through Simulation Exercise) is an open-source package of scientific analysis tools aimed at research in the Lower Thermosphere-Ionosphere (LTI). It was created with the purpose to assess the performance and demonstrate closure of the mission objectives of Daedalus, a mission concept targeting to perform *in-situ* measurements in the LTI. However, through its successful usage as a mission-simulator toolset, Daedalus MASE has evolved to encompass numerous capabilities related to LTI science and modeling. Inputs are geophysical observables in the LTI, which can be obtained either through *in-situ* measurements from spacecraft and rockets, or through Global Circulation Models (GCM). These include ion, neutral and electron densities, ion and neutral composition, ion, electron and neutral temperatures, ion drifts, neutral winds, electric field, and magnetic field. In the examples presented, these geophysical observables are obtained through NCAR's Thermosphere-Ionosphere-Electrodynamics General Circulation Model. Capabilities of Daedalus MASE include: 1) Calculations of products that are derived from the above geophysical observables, such as Joule heating, energy transfer rates between species, electrical currents, electrical conductivity, ion-neutral collision frequencies between all combinations of species, as well as height-integrations of derived products. 2) Calculation and cross-comparison of collision frequencies and estimates of the effect of using different models of collision frequencies into derived products. 3) Calculation of the uncertainties of derived products based on the uncertainties of the geophysical observables, due to instrument errors or to uncertainties in measurement techniques. 4) Routines for the along-orbit interpolation within gridded datasets of GCMs. 5) Routines for the calculation of the global coverage of an *in situ* mission in regions of interest and for various conditions of solar and geomagnetic activity. 6) Calculations of the statistical significance of obtaining the primary and derived products throughout an *in situ* mission's lifetime. 7) Routines for the visualization of 3D datasets of GCMs and of measurements along orbit. Daedalus MASE code is accompanied by a set of Jupyter Notebooks, incorporating all required theory, references, codes and plotting in a user-friendly environment. Daedalus MASE is developed and maintained at the Department for Electrical and Computer Engineering of the Democritus University of Thrace, with key contributions from several partner institutions.

KEYWORDS

lower thermosphere ionosphere, *in situ* measurements, global circulation model, daedalus mission, daedalus MASE, GCM, LTI

1 Introduction

Daedalus MASE comprises a suite of modeling tools targeting processes related to ion-neutral interactions in the Lower Thermosphere and Ionosphere (LTI), a key interface region between Earth's atmosphere and space. Within this region, the atmosphere transitions from being well-mixed and electrically neutral, to heterogeneous and partly ionized. Interactions between ions and neutrals maximize within the LTI, and in particular at altitudes from 100 to 200 km. These interactions lead to electrical currents, Pedersen and Hall conductivities and Joule or frictional heating, all of which maximize within this altitude range. Their approximation is a subject of extensive research, as these processes determine the energetics and dynamics within this region. Whereas the physics of these processes is well understood and is captured in Global Circulation Models (GCMs), their quantification is still largely unknown and shows large discrepancies between different models or even parameterizations within the same model. The reason is that their exact quantification requires the simultaneous and co-located measurement of an extensive list of many relevant parameters. Such comprehensive list of

measurements includes ion, neutral and electron densities (N_i, N_n, N_e), ion and neutral composition (n_{ix} and n_{nx}), ion, electron and neutral temperatures (T_i, T_e and T_n), ion drifts v_i), neutral winds (u_n), electric field ($\vec{E}$), and magnetic field ($\vec{B}$); in the following these are termed primary observables or geophysical observables. Whereas some of the above geophysical observables can be provided *via* remote sensing, the combination of all parameters at the same location can only be achieved *via in situ* measurements from a fully instrumented satellite. This has never been achieved to date, as altitudes from 100 to 200 km constitute the least visited region of the near-Earth environment (Sarris. (2019); Palmroth et al. (2021)). This leads to large discrepancies between estimations of key parameters, such as Pedersen and Hall conductivities, electrical currents and energy inputs due to Joule heating, which, in turn, greatly affects our ability to model and predict, for example, neutral density enhancements, that are of key importance to satellite drag calculations.

Daedalus MASE was initially developed to assess the performance of the mission concept Daedalus: proposed to the European Space Agency's Earth Explorer program (Sarris et al., 2020), Daedalus targets to perform *in situ* height-resolved measurements in the 100–200 km region from an eccentric low-perigee orbit. The main motivation for the Daedalus mission and its over-arching mission objective is to provide the first simultaneous and comprehensive set of *in situ* measurements of all physical quantities describing the lower thermosphere and ionosphere with the goal to explore and investigate the dominant processes that determine the energetics, dynamics, and chemistry of the region. Daedalus aims to make significant advancements and contributions to LTI physics/process understanding (e.g., Palmroth et al. (2021); Karlsson et al. (2020)), climatological specification/empirical models (e.g., Emmert et al. (2021)) and applications (e.g., Crisp et al. (2020)). One of the purposes of the development of Daedalus MASE was to demonstrate how the combination of all relevant primary observables enables the quantification of various products related to ion-neutral interaction processes. The purpose of Daedalus MASE was also to demonstrate that the mission requirements are sufficient to reach the Daedalus baseline mission objectives (ESA, 2020). This implies meeting performance metrics in terms of a) uncertainties of products, b) global statistics of products over the mission lifetime, and c) retrievals of profiles as a function of altitude, as is further discussed in the methodology section below.

Even though the initial development of Daedalus MASE targeted to demonstrate closure of the mission objectives of the Daedalus mission, the set of tools that have been developed during the Phase-0 Science and Requirements Consolidation Study of Daedalus have evolved to encompass capabilities that are of interest and can be applied to: a) science studies based on Global Circulation Models of the LTI, b) processing of *in situ* data by LTI sounding rockets and c) planned future missions targeting to perform measurements in the thermosphere–ionosphere, such as the Geospace Dynamics Constellation (GDC), a multi-spacecraft mission targeting to sample the upper atmosphere, currently under formulation by NASA.

The data underpinning the mission performance demonstration exercises performed by Daedalus MASE consist of global self-consistent simulations of the comprehensive LTI environment, that were performed using the National Center for Atmospheric Research (NCAR) Thermosphere-Ionosphere-Electrodynamics General Circulation Model (TIEGCM) (Qian et al., 2014). TIEGCM is a three-dimensional representation of the coupled thermosphere and ionosphere system based on first principles, that solves the momentum, energy and continuity equations for neutral and ion species at each time step. The variables available from the TIEGCM runs include the geophysical observables listed above, namely, N_i, N_n, N_e, n_{ix}, n_{nx}, T_i, T_eT_n, v_i, u_n, $\vec{E}$ and $\vec{B}$. From the gridded dataset of these observables, Daedalus MASE enables the following: a) The derivation of higher level products, such as heating terms, heat exchange rates, conductivities, currents, collision frequencies, etc., and related errors, as described below in **Sections 2.1–2.3**. b) The extraction of synthetic time series through the interpolation of the gridded TIEGCM datasets along satellite orbit tracks, and related coverage/sampling times within regions of interest, as described below in **Sections 2.4**, **2.5**. c) The calculation of statistical distributions of both primary observables and derived products within regions of interest and as a function of geomagnetic activity levels according to TIEGCM, and the comparison of these statistical distributions with the statistical representation of the LTI as reconstructed from along-track simulated measurements of an *in situ* mission over the mission lifetime, as described below in **Section 2.6**. d) The design of 3D graphics of primary and derived products, which enables a unique representation of the processes under investigation, as described below in **Section 2.7**.

A full solar cycle simulation was performed as part of the Daedalus mission performance demonstration, spanning from 2009 to 2019, with an output time resolution of 2 h. This output time resolution was selected so as to correspond to the orbital period of the Daedalus spacecraft along its elliptical orbit, which is also 2 h, while minimizing the large volumes of output data and the execution time. It is noted that the selected output time resolution introduces some limitations, as, for example, the along-track extraction of simulated measurements may only represent spatial variations, instead of both spatial and temporal variation; however, for the purpose of demonstrating the functionality of Daedalus MASE this was considered sufficient. Furthermore, the demonstrations of the statistical representation of the LTI as reconstructed from along-track simulated measurements and its comparison with the statistical representation of various products from the ensemble of the

gridded data are expected to yield similar results, in terms of the statistical similarity between the two representations. It is also noted that typical time steps of TIEGCM are on the order of 2 min, and that for the comparison of actual satellite along-track measurements with model runs under the same conditions such higher time resolutions should be used.

In the following, in **Section 2** we provide details on the methodology employed by each tool in Daedalus MASE, including a brief description of the corresponding code. Examples from the use of Daedalus MASE with gridded TIEGCM data are included in **Section 3**, including usage limitations. Finally, the scalability of Daedalus MASE and potential applications beyond the Daedalus mission frame are described in **Section 4**, including, but not limited to, its potential use as mission simulator for other *in situ* missions, such as the upcoming GDC mission, as well as sounding rocket experiments targeting processes in the LTI.

2 Methods

In this section we describe the methodology that is followed in each of the Daedalus MASE modules, providing a brief description of the functionality of each module, the inputs and outputs used, and the theoretical background of the estimations that are performed. Daedalus MASE is based on the assumption that all required geophysical observables that are related to ion-neutral interactions in the LTI are available. Depending on the application for which Daedalus MASE is used, these geophysical observables can be obtained in any of the following ways: a) *via in situ* measurements made along the orbit of a satellite or rocket platform; b) *via* physics-based Global Circulation Models (GCMs), such as TIEGCM which is used in the demonstrations presented herein, but also other GCMs, such as the Whole Atmosphere Community Climate Model-eXtended (WACCM-X) (Liu et al., 2010) or the Global Ionosphere Thermosphere Model (GITM) (Ridley et al., 2006); c) *via* empirical models, such as the combination of MSIS (Hedin. (1991); Picone et al. (2002); Emmert et al. (2021)) for neutral parameters, the International Reference Ionosphere (IRI) (Bilitza, 2018) for plasma parameters and the Horizontal Wind Model (HWM) (Drob et al. (2008); Drob et al. (2015)) for neutral winds, even though it is noted that in this case these geophysical observables may well be not self-consistent; d) *via* combinations of the above. The geophysical observables related to ion-neutral interactions in the LTI are summarized in **Table 1**, including commonly used instruments from spacecraft and/or rockets that measure each observable *in situ*. They are divided into plasma (ionosphere) parameters, neutral (thermosphere) parameters and fields. In the rest of this paper, the quantities listed in **Table 1** are termed primary products or simply geophysical observables.

For demonstrations and examples of Daedalus MASE functionality herein, data are obtained from global self-consistent simulations of the comprehensive LTI environment, obtained through NCAR's TIEGCM. Primary observable time series are extracted from TIEGCM grids through interpolation along realistic orbit tracks, with user-defined errors representing instrument and/or measurement errors and/or estimated through external user-specified parametric instrument simulators.

Daedalus MASE is composed as a set of modules written in Python, each of which can be used as a stand-alone package. These modules are accompanied by Jupyter Notebooks: a Jupyter Notebook is an open-source web application that integrates in a comprehensive way code, the underlying equations and theory, the output of computations performed by the code, visualizations of the code results (including 2D and 3D plots, projections on a sphere, along-orbit plots, etc), and multiple other multimedia resources, along with explanatory text. These are all provided in a single repository, allowing scientists to easily access all elements of the programming process.

An overview of the repository, named *DaedalusMASE*, is presented in **Figure 1**. The repository is publicly available as a GitHub package, and can be downloaded at: https://github.com/DaedalusMASE/DaedalusMASE. In its current form, *DaedalusMASE* is built in the form of six main modules. These are: a) *derived_products* module, used for the calculation of the derived products; b) *collision_frequencies* module, which implements different models of ion-neutral collision frequencies related to different models of interaction cross-sections; c) *interpolation* module, which is used for interpolation of TIEGCM parameters along a satellite orbit; d) *error_propagation* module, which is used for the estimation of instrumental errors on the derived products; e) *coverage_calculator* module, which estimates the coverage of regions of interest of the Daedalus mission; and f) *global_statistics* module, which investigates quantitatively whether the dataset of a variable obtained *via* an *in situ* sampling scheme is statistically significant. Each of the modules contains a folder with the corresponding source code, a documentation folder which includes the API documentation of the source code in. html format, and a folder which includes Jupyter notebooks that can be used either as stand-alone simulation tools or as tutorials on the use of each module. The *DaedalusMASE* repository includes also a folder with sample data, which contains input data needed by the modules, such as TIEGCM NetCDF files from a sample run, the corresponding geomagnetic indices, and sample orbital data.

Further to the above modules comprising Daedalus MASE, below we also present a set of Python routines that are used for the visualization of 3D datasets of GCMs. The visualization is performed using the Blender software package, and the code presented herein allows integrating the outputs of GCM model

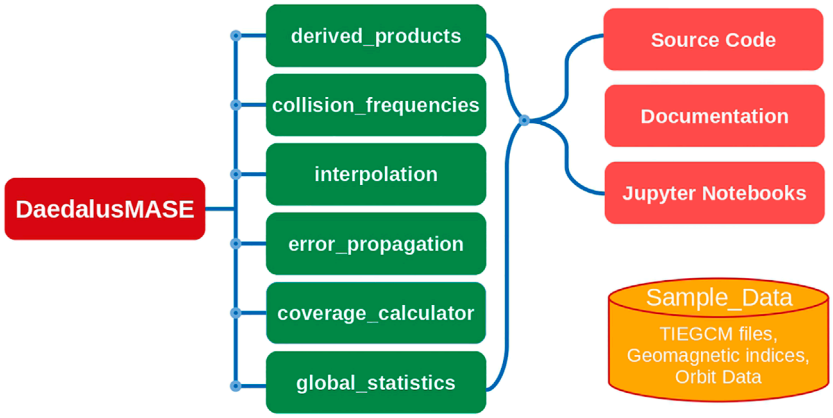


FIGURE 1
Overiview of the modules and structure of the Daedalus MASE repository.

TABLE 1 List of primary products or geophysical observables in Daedalus MASE and commonly used instruments for their derivation.

Geophysical Observables in Daedalus MASE			
	Abbrev.	Geophysical Observable	Commonly used instruments
Ionosphere	$\vec{v}_i$	Ion Drift velocity	Thermal Ion Imager, Ion Drift Meter, Retarding Potential Analyzer
	T_i	Ion Temperature	
	T_e	Electron Temperature	Langmuir Probe
	N_i	Ion Number Density	Mutual Impedance Probe
	N_e	Electron Number Density	
	n_{ix}	Ion Composition	Ion Mass Spectrometer
Thermosphere	$\vec{u}_n$	Neutral Wind Velocity	Ram and Cross-Track Wind Sensors
	ρ	Neutral Mass Density	Accelerometer
	T_n	Neutral Temperature	Neutral Mass Spectrometer
	n_{nx}	Neutral Composition	
Fields	$\vec{B}$	Magnetic Field	Magnetometer
	$\vec{E}$	Electric Field	Electric Field Instrument

results, which are in the form of NetCDF files, with Blender, for creating the meshes and textures in 3D plots and animations. The repository is publicly available as a GitLab package, and can be downloaded at: https://gitlab.com/eelcodoornbos/blender_gcm_slices.

In the following we outline the functionality of each of the tools and modules comprising Daedalus MASE, based on processing of the above primary products listed in **Table 1**.

2.1 Daedalus MASE derived products module

One of the main purposes of Daedalus MASE, and indeed an overarching theme of the Daedalus mission, is to explore ion-neutral interaction processes within the LTI, focusing in particular on processes within the 100–200 km altitude range. Ion-neutral interaction processes within this region require simultaneous and co-located measurements of all relevant

parameters in order to be resolved quantitatively, and these measurements can only be provided *in situ* by a spacecraft equipped with all relevant instruments. The goal of this module is to demonstrate the quantification of key physical processes related to ion-neutral interactions. These physical processes are directly linked to a list of quantities that can be derived analytically from the geophysical observables listed in **Table 1**. These are listed in **Table 2** and are classified in terms of: a) heating sources, b) conductivity/cross-sections/collision frequencies, and c) electrical currents/magnetospheric forcing in the LTI, and are discussed in further detail below.

Calculations of the derived products are performed with the *daedalusmase_derived_products* package of *DaedalusMASE*, which is composed of a series of modules and sub-modules, as detailed in **Supplementary Tables S1–S7**. In the following we present the corresponding routines that comprise these modules, along with analytic theoretical descriptions of the formulas and the assumptions used. In the folder *Jupyter_Notebooks* of the repository, there are example codes in the form of Jupyter notebooks which demonstrate the use of the routines.

2.1.1 Heating sources module

The first set of derived products in **Table 2**, marked in red, lists key heating sources and heat transfer processes in the LTI. In particular, a recurring unknown quantity of great significance in the energy budget of the LTI is Joule, or frictional, or Ohmic heating rate, and how it drives or evolves in concert with the plasma and neutral dynamics; at the same time, characterizing the variability of Joule heating within the latitude and altitude region where it maximizes is a critical missing piece in LTI processes. In further detail:

2.1.1.1 Joule, frictional, and ohmic heating rates

Joule heating (q_j) (Strangeway. (2012), eq. 38) and the relevant equivalent expressions for frictional heating (q_f) (Strangeway. (2012), Eq. **28**) and Ohmic heating (q_Ω) (Lu et al. (1995), Eq. **(3)**) provide three distinct estimation methods of the same local energy dissipation process viewed from different perspectives, and are expected to yield similar values. Estimating all three provides redundancy, by partially relying on different inputs, but with anticipated differences in performance as a function of altitude. They also enable a cross-comparison providing confirmation of the physics at play. Joule heating is given by equation:

$$q_j = \vec{J}_\perp \cdot \vec{E}^\star = eN_e\left(\vec{v}_{i,\perp} - \vec{u}_{e,\perp}\right)\cdot\left(\vec{E} + \vec{u}_n \times \vec{B}\right) \tag{1}$$

where e is the electron charge, N_e is the electron number density, $\vec{v}_{i,\perp}$ and $\vec{u}_{e,\perp}$ are the ion and electron velocities perpendicular to the geomagnetic field, $\vec{J}_\perp = eN_e(\vec{v}_{i,\perp} - \vec{u}_{e,\perp})$ is the electric current perpendicular to the geomagnetic field, $\vec{E}$ the electric field, $\vec{B}$ the magnetic field, $\vec{u}_n$ the bulk velocity of the neutrals, and $\vec{E}^\star = \vec{E} + \vec{u}_n \times \vec{B}$ is the electric field in the reference frame of the neutrals.

At altitudes where $\nu_{en} \ll \Omega_e$, the electrons are magnetized, thus they move at a velocity (in the neutral reference frame):

$$\vec{v}^*_{e\perp} \approx \frac{\vec{E}^\star \times \vec{B}}{B^2} \tag{2}$$

which in the satellite rest frame is:

$$\vec{v}_{e\perp} \approx \vec{u}_n + \frac{\left(\vec{E} + \vec{u}_n \times \vec{B}\right)\times \vec{B}}{B^2} \tag{3}$$

This is valid at all ionospheric heights above the D region (>90 km).

The parallel electron mobility is large enough to produce a very large parallel conductivity $\sigma_\parallel$ compared to the Pedersen and Hall conductivities, σ_P and σ_H, respectively, which means that the electrons generally move more easily along the magnetic field. This means that they tend to sort out any field-aligned (i.e. parallel to the magnetic field) electric fields, and thus, that the electric field tends to be perpendicular to the magnetic field, or else that:

$$\vec{E}_\parallel = 0 \tag{4}$$

Inserting Eq. **3** and Eqs **4**–**1**:

$$q_j = eN_e\left(\vec{v}^*_i - \vec{v}^*_e\right)\cdot\vec{E}^*_\perp = eN_e\left(\vec{v}_i - \vec{v}_e\right)\cdot\left(\vec{E}_\perp + \vec{u}_n \times \vec{B}\right) \tag{5}$$

$$q_j = eN_e\left[\vec{v}_i - \left(\frac{\vec{E}_\perp + \vec{u}_n \times \vec{B}}{B^2} + \vec{u}_n\right)\right]\cdot\left(\vec{E}_\perp + \vec{u}_n \times \vec{B}\right) \tag{6}$$

using the identity $(\vec{a}\times\vec{c})\cdot\vec{a} = 0$, this reduces to:

$$q_j = eN_e\left(\vec{v}_i - \vec{u}_n\right)\cdot\left(\vec{E}_\perp + \vec{u}_n \times \vec{B}\right) = eN_e\vec{v}^*_{i\perp}\vec{E}^\star_\perp \tag{7}$$

meaning that the Joule heating rate can be estimated by the ion current times the electric field. For an ion population that consists of i species with N_i number densities, where $i = O_2^+, NO^+, O^+, \ldots$, Eq. **7** becomes:

$$q_j = e\sum_{i=O_2^+,NO^+,O^+,\ldots} N_i\vec{v}_i\left(\vec{E}_\perp + \vec{u}_n \times \vec{B}\right) \tag{8}$$

where:

$$N_e = \sum_{i=O_2^+,NO^+,O^+,\ldots} N_i \tag{9}$$

The expression for Ohmic heating is given as:

$$q_\Omega = \sigma_P\left(\vec{E} + \vec{u}_n \times \vec{B}\right)^2 \tag{10}$$

The frictional heating rate is calculated as:

$$q_f = m_i\nu_{in}N_e|\vec{v}_{i,\perp} - \vec{u}_{n,\perp}|^2 \tag{11}$$

where m_i is the ion mass and ν_{in} is the ion-neutral collision frequency.

Joule, frictional and Ohmic heating rates in Daedalus MASE are calculated by the *joule*, *frictional* and *ohmic* routines of the *daedalusmase_derived_products.mod_heating_sources.sub_heating_rates* sub-module.

TABLE 2 List of derived products in Daedalus MASE and contributing primary observables.

Derived products in Daedalus MASE		
Derived product (References)	Theoretical equation for estimation	Geophysical observables
Joule heating (Strangeway. (2012), eq. 38)	$q_j = eN_e(\vec{v}_i - \vec{u}_e)\cdot(\vec{E} + \vec{u}_n \times \vec{B})$	$\vec{v}_i$, N_e, $\vec{E}$, $\vec{B}$
Ohmic heating (Lu et al. (1995), Eq. 3)	$q_\Omega = \sigma_P\lvert\vec{E} + \vec{u}_n \times \vec{B}\rvert^2$	$\vec{u}_n$, $\vec{E}$, $\vec{B}$, products of σ_P
Frictional heating (Strangeway. (2012), Eq. 28)	$q_f = m_i\nu_{in}N_e\lvert\vec{v}_{i,\perp} - \vec{u}_{n,\perp}\rvert^2$	$\vec{v}_i$, $\vec{u}_n$, N_e, products of ν_{in}
Heat transfer from ions to neutrals Killeen et al. (1984)	$q_{\Delta T_{in}} = N_e\nu_{in}\frac{m_i}{m_i+m_n}3k_B(T_i - T_n)$	T_i, T_n, N_e, products of ν_{in}
Heating and cooling terms between electrons, neutrals and ions	see text for theoretical description and formulas	—
Pedersen conductivity (Schunk and Nagy. (2009), Eq. 5)	$\sigma_P = \frac{e}{B}\left(N_e\frac{\Omega_e\nu_{en}}{\Omega_e^2+\nu_{en}^2} + \sum_i N_i\frac{\Omega_i\nu_{in}}{\Omega_i^2+\nu_{in}^2}\right)$	N_e, N_i, $\lvert\vec{B}\rvert$, products of ν_{in} and ν_{en}
Hall conductivity (Schunk and Nagy. (2009), Eq. 5)	$\sigma_H = \frac{e}{B}\left(N_e\frac{\Omega_e^2}{\Omega_e^2+\nu_{en}^2} - \sum_i N_i\frac{\Omega_i^2}{\Omega_i^2+\nu_{in}^2}\right)$	N_e, N_i, $\lvert\vec{B}\rvert$, products of ν_{in} and ν_{en}
Parallel conductivity (Schunk and Nagy. (2009), Eq. 5)	$\sigma_\parallel = N_e\frac{e^2}{\nu_{en}m_e}$	N_e, products of ν_{en}
Ion-neutral collision frequency (Sangalli et al. (2009), Eq. 6)	$\nu_{in} = \frac{e}{m_i}\frac{\lvert\vec{E}+\vec{v}_i\times\vec{B}\rvert}{\lvert\vec{v}_{i,\perp}-\vec{u}_{n,\perp}\rvert}$	$\vec{v}_i$, T_i, N_e, N_i, $\vec{E}$, $\vec{B}$, T_n, $\vec{u}_n$, n_{ix}
Perpendicular current (*via* vi, ve) (Richmond and Thayer. (2000), Eq. 10)	$j_\perp = eN_e(\vec{v}_i - \vec{v}_e)$	$\vec{v}_i$, N_e, $\vec{E}$, $\vec{B}$
Perpendicular, Pedersen, and Hall currents (Richmond and Thayer. (2000), Eq. 11)	$j_\perp = \vec{j}_P + \vec{j}_H = \sigma_P\vec{E}^* + \sigma_H\hat{b}\times\vec{E}^*$	$\vec{u}_n$, $\vec{E}$, $\vec{B}$, products of σ_P and σ_H
Magnetic forcing (Richmond and Thayer. (2000), Eq. 20)	$\vec{f}_{mag} = \vec{j}\times\vec{B}$	$\vec{u}_n$, $\vec{v}_i$, N_e, $\vec{E}$, $\vec{B}$
Mechanical Power (Lu et al. (1995), Eq. 2)	$q_{mech} = \vec{u}_n\cdot(\vec{j}\times\vec{B})$	$\vec{u}_n$, $\vec{v}_i$, N_e, $\vec{E}$, $\vec{B}$

2.1.1.2 Convection and wind heating

By expanding Eq. 10 for Ohmic heating, we obtain:

$$\begin{aligned} q_\Omega &= \sigma_P\left(\vec{E} + \vec{u}_n \times \vec{B}\right)^2 \\ &= \underbrace{\sigma_P E^2}_{q_c} + \underbrace{\sigma_P\lvert\vec{u}_n \times \vec{B}\rvert^2 - 2\sigma_P\vec{u}_n\cdot\left(\vec{E}\times\vec{B}\right)}_{q_w} \end{aligned} \tag{12}$$

In this formula, the first term marked as q_c is known as convection heating [Lu et al. (1995); Billett et al. (2018)], and corresponds to the Joule heating rate in the absence of neutral winds, whereas the second term marked as q_w is the neutral wind correction term (Billett et al., 2018), often termed "wind heating" (Lu et al., 1995), which gives a measure of the error in estimating Joule heating rate neglecting neutral winds. When neutral winds are driven frictionally by $\vec{E}\times\vec{B}$ convection, e.g., during substorm growth and expansion phases, the presence of neutral winds has in general the tendency to lower the convection heating, thus an estimation of Joule heating without taking into account the presence of neutral winds will lead to an over-estimation. On the other hand, when neutral winds have a dynamo effect, e.g., during substorm recovery (when $\vec{E}\times\vec{B}$ convection decreases, while the inertia of the massive neutral atmosphere supports the neutral winds a longer time), this is no longer true. Whereas numerous studies have shown that the neutral winds can affect Joule heating substantially [Lu et al. (1995); Thayer. (1998); Deng and Ridley. (2007)], the quantification of the contribution of the neutral winds requires detailed measurements of all relevant parameters. This module enables the separate computation and cross-comparison of the two terms, both in models and in cases that *in situ* measurements are available.

Convection and wind heating rates in Daedalus MASE are calculated by the *convection_heat* and *wind_heat* routines of the *daedalusmase_derived_products.mod_heating_sources.sub_heating_rates* sub-module.

2.1.1.3 Ohmic heating per unit mass

Also related to the heating sources is the estimate of Ohmic heating per unit mass. The rate of local temperature change in the thermosphere due to Joule heating is more directly related to the heating per unit mass, i.e., the volumetric heating rate divided by the mass density. This is because, although the volumetric heating is decreasing with altitude, the mass density decreases more rapidly, and thus less energy is required to significantly heat the more tenuous neutral gas (Richmond and Thayer, 2000). Thus, whereas the volumetric heating rate is largest in the E region, the heating per unit mass becomes larger higher up, in the F region. This is calculated according to:

$$q_{j,\rho} = \frac{\sigma_P\left(\vec{E} + \vec{u}_n \times \vec{B}\right)^2}{\rho} \tag{13}$$

where ρ is the neutral density.

Ohmic heating rate per unit mass in Daedalus MASE is calculated by the *ohmic_per_mass* routine of the *daedalusmase_derived_products.mod_heating_sources* module.

2.1.1.4 Ratio of joule heating over pressure

This module also enables the calculation of the ratio of Joule heating over pressure. This ratio has units of $1/s$ (since Joule heating has units of $W = kg \cdot m^2 s^{-3}$, and atmospheric pressure has units of $Pa = kgm^{-1}s^{-2}$), and provides an indication of the time until the accumulated Joule heating in a certain volume would equal the thermal energy that is present in that volume of the atmosphere in absence of convection and heat conduction.

The ratio of Joule heating over pressure in Daedalus MASE is calculated by the *ohmic_per_pressure* routine of the *daedalusmase_derived_products.mod_heating_sources* module.

2.1.1.5 Heat transfer rates between species

The routines of the *daedalusmase_derived_products.mod_heating_sources. sub_heat_transfer_rates* sub-module enable the calculation of the heat transfer rates between species, such as the heat transfer rates from ions to neutrals, electrons to neutrals and ions to/from electrons. It is noted that the friction between ions and neutrals as well as ionization result in different temperatures of ions, electrons and the neutral gas, and heat transfer due to elastic collision between these species. The rates can be derived from observed temperatures and collision frequencies. When different temperatures between ions and neutrals are observed, then the heat transfer between the two species can be estimated according to Killeen et al. (1984) as:

$$q_{\Delta T_{in}} = N_e \nu_{in} \frac{m_i}{m_i + m_n} 3k_B (T_i - T_n) \tag{14}$$

Similarly, for the ion-electron case:

$$q_{\Delta T_{ie}} = N_e \nu_{ie} \frac{m_i}{m_i + m_e} 3k_B (T_i - T_e) \tag{15}$$

and for the electron-neutral case:

$$q_{\Delta T_{en}} = N_e \nu_{en} \frac{m_e}{m_e + m_n} 3k_B (T_e - T_n) \tag{16}$$

where subscripts i, e and n denote ions, neutrals and electrons respectively, T, ν and m denote temperatures, collision frequencies and masses, respectively, for each species, and k_B is the Boltzmann constant.

In general, T_e and T_i are greater than T_n, thus a heat transfer from electrons and ions to the neutrals is observed. On the other hand, although in general $T_e > T_i$, at very low altitudes and/or during disturbed geomagnetic conditions, T_i can exceed T_e locally. In this case the heat flow between ions and electrons is reversed and the electrons are heated.

Heat transfer from ions to neutrals is calculated by the *heat_transfer_in* routine, and from electrons to neutrals by the *heat_transfer_en_elastic* routine. Finally, the energy exchange between ions and electrons is calculated by the *heat_transfer_ei* routine of the *daedalusmase_derived_products.mod_heating_sources. sub_heat_transfer_rates* sub-module.

2.1.1.6 Frictional heating rates

Frictional heating arises due to the differential velocity between species. Whereas with Eq. **11**, we calculated the total ion-neutral frictional heating (q_f), here we calculate the frictional heating between each individual ion and neutral species. Furthermore, we calculate frictional heating between ions and electrons and between different ion species. The frictional heating rate between ions and neutrals is given as:

$$qF_{in} = N_i m_i \sum_n \frac{\nu_{in} m_n (u_n - u_i)^2}{m_i + m_n} \tag{17}$$

The frictional heating rate between ions and electrons is:

$$qF_{ie} = N_e m_e \sum_i \frac{\nu_{ei} m_i (u_e - u_i)^2}{m_i + m_e} \tag{18}$$

The frictional heating rate between different ion species is:

$$qF_{ij} = N_j m_j \sum_i \frac{\nu_{ji} m_i (u_j - u_i)^2}{m_i + m_j} \tag{19}$$

where N refers to number density, m is the mass, ν is the collision frequency and u is the velocity. Subscripts i,j refer to different ion species, e denotes electrons, n denotes neutrals.

It is noted that the sub-module named *sub_frictional_heating_rates* includes an extensive set of routines for calculating the corresponding heating rates separately for O^+, O_2^+, NO^+ and N^+ ions. This allows the contribution of various species to the total heating to be evaluated.

2.1.1.7 Electron cooling rates

The *daedalusmase_derived_products.mod_heating_sources. sub_cooling_rates* sub-module includes routines for the calculation of inelastic collisions between electrons and neutrals. Specifically, includes routines for calculating electron losses due to N_2 rotational [*sub_colling_rates.N2_rot*] and vibrational [*sub_colling_rates.N2_vib*] excitation, O_2 rotational [*sub_colling_rates.O2_rot*] and vibrational [*sub_colling_rates.O2_vib*] excitation and O fine structure [*sub_colling_rates.O_fine*]. The relevant formulas can be found in Schunk and Nagy. (2009).

2.1.2 Conductivities module

Module *daedalsumase_derived_products.mod_conductivities* includes routines for the calculation of Pedersen (σ_P), Hall (σ_H) and parallel ($\sigma_\parallel$) conductivities as described in Richmond and Thayer. (2000), using the following equations (see also **Table 2**):

$$\begin{aligned} \sigma_P &= \frac{e}{B}\left(N_e \frac{\Omega_e \nu_{en}}{\Omega_e^2 + \nu_{en}^2} + \sum_i N_i \frac{\Omega_i \nu_{in}}{\Omega_i^2 + \nu_{in}^2} \right) \\ &= \frac{e}{B}\left(N_e \frac{\kappa_e}{1 + \kappa_e^2} + \sum_i N_i \frac{\kappa_i}{1 + \kappa_i^2} \right) \end{aligned} \tag{20}$$

$$\sigma_H = \frac{e}{B}\left(N_e\frac{\Omega_e^2}{\Omega_e^2+\nu_{en}^2} - \sum_i N_i\frac{\Omega_i^2}{\Omega_i^2+\nu_{in}^2}\right) = \frac{e}{B}\left(N_e\frac{\kappa_e^2}{1+\kappa_e^2} - \sum_i N_i\frac{\kappa_i^2}{1+\kappa_i^2}\right) \quad (21)$$

$$\sigma_\parallel = N_e\frac{e^2}{\nu_{en}m_e} \quad (22)$$

where ν_{en} is the electron neutral collision frequency, Ω_e and Ω_i are the electron and ion gyrofrequencies respectively, e is the electron charge, N_e is the electron number density, N_i is the ion number density and κ_i is the ratio of the gyrofrequency over the collision frequency of each ion.

The Pedersen conductivity is calculated by the routine *pedersen_cond*, the Hall conductivity is calculated by the routine *hall_cond* and the parallel conductivity is calculated by the routine *parallel_cond*.

2.1.3 Collision frequencies and cross sections module

Conductivities, collision cross-sections and collision frequencies, are key parameters that affect heating in the LTI, and are also critical in both LTI and magnetosphere modelling. In the Earth's LTI system, the interactions of ions and neutrals influence the velocities, temperatures and densities of both species, thus playing a vital role in momentum and energy exchange between the thermosphere and the ionosphere. These interactions are characterised through the ion neutral collision frequencies and subsequently (fundamentally) through the ion-neutral cross sections, which are fundamental parameters in the coupling between the neutral atmosphere and the ionosphere. However, collision cross-sections, as documented in, e.g., Schunk and Nagy. (2009); Richmond. (2017), have been determined primarily through laboratory experiments, which are not necessarily representative of the real LTI environment, and are believed to have systematic biases.

2.1.3.1 Collision frequencies

The mod_collision_freqs_cross_sections.sub_collision_frequencies sub-module includes routines for the calculations of the collision frequencies between electrons, ions and neutrals. The laboratory-derived relevant formulas can be found in Schunk and Nagy. (2009). For the case of ion-neutral collisions, another option is to use the ion momentum equation (Sangalli et al. (2009); Sarris et al. (2020)) in order to derive the corresponding collision frequency. The ion momentum equation is given as:

$$m_iN_i\left(\frac{\partial}{\partial t}+\vec{v}_i\nabla\right)\vec{v}_i = q_iN_i\left(\vec{E}+\vec{u}_i\times\vec{B}\right) - m_iN_i\nu_{in}\left(\vec{v}_i-\vec{u}_n\right) - m_iN_i\nu_{ie}\left(\vec{v}_i-\vec{v}_e\right) + m_iN_i\vec{g} - \nabla P_i \quad (23)$$

where q_i id the ion charge, $\vec{g}$ is the gravity acceleration and P_i is the ion thermal pressure. Assuming a homogenous plasma and neglecting gravity and pressure contributions:

$$m_iN_i\frac{\partial}{\partial t}\vec{v}_i = q_iN_i\left(\vec{E}+\vec{u}_i\times\vec{B}\right) - m_iN_i\nu_{in}\left(\vec{v}_i-\vec{u}_n\right) - m_iN_i\nu_{ie}\left(\vec{v}_i-\vec{v}_e\right) \quad (24)$$

Taking the expression for motion perpendicular to the geomagnetic field and assuming a steady state:

$$0 = q_iN_i\left(\vec{E}_\perp+\vec{u}_i\times\vec{B}\right) - m_iN_i\nu_{in}\left(\vec{v}_{i,\perp}-\vec{u}_{n,\perp}\right) - m_iN_i\nu_{ie}\left(\vec{v}_{i,\perp}-\vec{v}_{e,\perp}\right) \quad (25)$$

Collisions between electrons and ions become important (relatively to ion-neutral collisions) only in the upper ionosphere, where, though, ions and electrons have almost same velocities perpendicular to $\vec{B}$ ($E\times B$ drift), thus $\vec{v}_{i,\perp}-\vec{v}_{e,\perp}\approx 0$ and we can neglect the third term on the right-hand side of Eq. **(26)**:

$$q_iN_i\left(\vec{E}_\perp+\vec{u}_i\times\vec{B}\right) = m_iN_i\nu_{in}\left(\vec{v}_{i,\perp}-\vec{u}_{n,\perp}\right) \quad (26)$$

Finally, solving for ν_{in}, yields:

$$\nu_{in} = \frac{q_i}{m_i}\frac{|\vec{E}_\perp+\vec{v}_i\times\vec{B}|}{|\vec{v}_{i,\perp}-\vec{u}_{n,\perp}|} \quad (27)$$

2.1.3.2 Cross sections

Subsequently, mod_collision_freqs_cross_sections.sub_cross_sections sub-module calculates ion-neutral cross sections according to Banks and Kockarts. (1973) using the following formula:

$$\sigma_{in} = \frac{\nu_{in}}{N_n\sqrt{\frac{2k_BT_i}{m_i}}} \quad (28)$$

2.1.4 Currents and magnetic forcing module

An open question in the lower thermosphere and ionosphere is how atmospheric and magnetospheric forcing and drag between charged and neutral species affect the wind structure, convection patterns, and electric currents in the region. To this direction, the third set of derived products, marked in green in **Table 2**, includes calculations of currents flowing perpendicularly to the magnetic field. Perpendicular currents are calculated both through the differential velocity between ion drift and electron drift (given by $E\times B$ in the LTI) and through the Hall and Pedersen currents. The magnetic forcing, i.e. the Lorentz force, that plasma exerts on the neutral atmosphere through collisional coupling, is also listed. The perpendicular currents and magnetic forcing are calculated by module *mod_currents_magnetic_forcing*, as discussed in further detail in the corresponding sections below.

2.1.4.1 Current density

Within the lower ionosphere, Pedersen currents flow in the direction of the electric field, while the Hall currents flow in the direction opposite to the $\vec{E}\times\vec{B}$ direction, i.e. perpendicularly to the electric and the magnetic field. Pedersen currents flow a bit higher (~ 120 km) where the Pedersen conductivity maximizes, whereas the Hall currents flow lower, at the altitude where the Hall conductivity maximizes (i.e., ~ 115 km). The problem is that the current density $\vec{j}_\perp$ cannot be determined directly; $\vec{j}_\perp$ can be, in principle, inferred from magnetometer data, which is, however, not straightforward at altitudes where Pedersen, Hall and field-aligned currents co-exist and all contribute to the local magnetic field, i.e., roughly below 300 km. Alternatively, assuming quasi-neutrality, i.e., that the electron density N_e is equal to the sum of the ion species densities N_{O^+}, N_{O2^+}, and N_{NO^+}, and denoting the electron and ion drifts for each species with v_e, v_{O^+}, $v_{O_2^+}$, and v_{NO^+}, respectively, we can calculate the perpendicular currents by using:

$$\begin{aligned}\vec{j}_\perp &= e\left(N_{O^+}\vec{v}_{O^+}+N_{O2^+}\vec{v}_{O_2^+}+N_{NO^+}\vec{v}_{NO^+}\right.\\ &\quad\left.+N_{N^+}\vec{v}_{N^+}-N_e\vec{v}_e\right)\\ &\approx eN_e\left(\vec{v}_i-\vec{v}_e\right)\end{aligned}\tag{29}$$

Another way to calculate the perpendicular currents is by using Ohm's law. From that perspective one can calculate the Pedersen and Hall components of the perpendicular currents independently as follows:

$$\vec{j}_\perp=\sigma_P\vec{E}^\star+\sigma_H\hat{b}\times\vec{E}^\star=\vec{j}_P+\vec{j}_H\tag{30}$$

where σ_H is the Hall conductivity, $\hat{b}$ is the unit vector of the magnetic field, the perpendicular sign refers to quantities perpendicular to the magnetic field and the star sign denotes the electric field in the reference frame of the neutrals:

$$\vec{E}^\star=\vec{E}_\perp+\vec{u}_n\times\vec{B}\tag{31}$$

The Pedersen current is calculated by the routine *current_pedersen*, the Hall current is calculated by the routine *current_hall* and the perpendicular current is calculated by the routine *current_perp* of the *mod_currents_magnetic_forcing* module.

2.1.4.2 Magnetic forcing

The closure of currents in the magnetosphere and in the ionosphere produces a $\vec{j}_\perp\times\vec{B}$ force at each region. In the magnetosphere this force can slow down the magnetospheric convection and provide energy to the electromagnetic field, thus acting like a dynamo/generator that converts kinetic energy to electromagnetic energy; it is noted that in the magnetosphere one can also encounter motor/load regions, where plasma is accelerated (e.g., at reconnection sites). In the ionospheric counterpart, the $\vec{j}_\perp\times\vec{B}$ force balances the frictional drag on the ions by the neutral atmosphere preserving the ionospheric convection and consequently accelerates the neutrals in the direction of the plasma flow. The $\vec{j}_\perp\times\vec{B}$ force in the ionosphere consists of two components associated with the Pedersen and Hall currents. The Pedersen associated part, counteracts the drag force in the direction opposite to the $\vec{E}\times\vec{B}$ direction, while the Hall part balances the drag force in the direction of the electric field. Equal and opposite forces act also on the neutrals, which tend to drive neutral winds in the Thermosphere. Although the $\vec{j}_\perp\times\vec{B}$ part associated with the Pedersen current is of the same order as the Hall associated part, as the Pedersen currents flow higher than the Hall currents, the neutral density in the region is lower and thus are more efficient in driving neutral winds (Cowley, 2000). Daedalus MASE allows the investigation of the altitude-dependent contribution of the Pedersen and Hall currents in the total $\vec{j}_\perp\times\vec{B}$ forcing term.

In order to address the coupling between the thermosphere, the ionosphere and the magnetosphere, we apply Poynting's theorem to the entire system [e.g., Thayer and Vickrey. (1992); Lu et al. (1995)]:

$$\frac{\partial w}{\partial t}+\nabla\cdot\left(\frac{\vec{E}\times\vec{B}}{\mu_0}\right)+\vec{j}\cdot\vec{E}=0\tag{32}$$

where w is the electromagnetic energy density, μ_0 is the magnetic permeability of free space and $\vec{J}$ is the current density. Assuming a steady state (i.e., assuming that the change of the total electromagnetic energy in time can be ignored compared to the plasma motion), Eq. **32** becomes:

$$\nabla\cdot\left(\frac{\vec{E}\times\vec{B}}{\mu_0}\right)+\vec{j}\cdot\vec{E}=0\tag{33}$$

where the first term is the divergence of the electromagnetic energy flux, and the second term is the energy transfer rate. As explained above (see Eq. **4**) the electric field component perpendicular to the geomagnetic field ($\vec{E}_\perp$) is much larger than the parallel component ($\vec{E}_\parallel$), thus $\vec{J}\cdot\vec{E}$ roughly equals $\vec{J}_\perp\cdot\vec{E}_\perp$. Further analyzing the energy transfer rate leads to:

$$\vec{j}_\perp\cdot\vec{E}_\perp=\underbrace{\vec{j}_\perp\cdot\vec{E}^\star}_{q_j}+\underbrace{\vec{u}_n\cdot\left(\vec{j}_\perp\times\vec{B}\right)}_{\epsilon}\tag{34}$$

where ϵ is the mechanical power exerted on the neutrals by the $\vec{j}_\perp\times\vec{B}$ forcing (Lu et al., 1995) and $\vec{E}^\star$ is the electric field in the neutral wind reference frame (Eq. **31**).

The $\vec{j}_\perp\times\vec{B}$ forcing is calculated by the routine *JxB_forcing* and the mechanical power by the routine *mechanical_power* of the *mod_currents_magnetic_forcing* module.

2.1.5 TIEGCM utilities module

Apart from the modules for the calculation of the various derived products as discussed above, *daedalusmase_derived_products* includes several routines for processing TIEGCM outputs, combined into the *tiegcm_utils* module. Specifically, the *read_tiegcm* routine is used for reading TIEGCM output files, which by default are in Network Common

Data Form (NetCDF) format. The *convert_mmr* routine is used to convert TIEGCM densities from mass mixing ratio (*mmr*) to cm^{-3}. Furthermore, the *igrf_B* routine is used for the calculation of the IGRF magnetic field and the *electric_field* routine for the calculation of the electric field from $\vec{E}\times\vec{B}$ velocities.

It is noted that TIEGCM does not solve the ion momentum equations explicitly to calculate the ion velocities; instead, the ion velocities are calculated in post processing by using subroutine tgcmproc, which can be found in the list of TIEGCM post-processors available from NCAR, written in Fortran. In a similar manner, DaedalusMASE uses the ion and electron momentum equations to derive expressions for the ion and electron velocities. By neglecting the forces due to pressure gradient and gravity, the ion and electron velocities can be given as ((Richmond and Thayer, 2000):

$$\vec{v}_{i\perp}^{\,\star}=\frac{\nu_{in}\Omega_i\vec{E}_\perp^{\star}-\Omega_i^2\hat{b}\times\vec{E}_\perp^{\star}}{B\left(\nu_{in}^2+\Omega_i^2\right)} \tag{35}$$

and

$$\vec{v}_{e\perp}^{\,\star}=\frac{-\nu_{en}\Omega_e\vec{E}_\perp^{\,*}-\Omega_e^2\hat{b}\times\vec{E}_\perp^{\,*}}{B\left(\nu_{en}^2+\Omega_e^2\right)} \tag{36}$$

where the star sign denotes the corresponding velocities and electric field in the neutral wind reference frame. Module *tiegcm_utils* includes routines for the calculation of e, O^+, O_2^+, NO^+ and N^+ velocities.

2.1.6 Height integration module

Using inputs from the locally-computed derived products in TIEGCM, as described in the section above and listed in **Table 2**, this module enables also the calculation of height-integrated and hemispherically-integrated derived products. Such estimates are of particular importance for LTI processes, as, for example, in order to provide a quantitative assessment of high-latitude energy input, the hemispherically-integrated Joule heating in the Northern or Southern hemispheres needs to be estimated [e.g., Deng et al. (2009)]. Several previous studies have also studied Joule heating in two dimensions, corresponding to the altitude-integrated Joule heating [Thayer. (1998); Lu et al. (1995); Weimer. (2005); Deng et al. (2009)]. Furthermore, estimations of the height-integrated Pedersen conductivities in the E (100–150 km) and F (150–600 km) regions and their ratio has been used to obtain an estimate of the ratio of the Joule heating deposited in the E and F regions (Sheng et al., 2014).

Whereas height-integrated estimations of various products are commonly used in processing GCM outputs, there is a lack of an open-source tool that enables the computation of integrations with user-defined limits. This module enables the integration of GCM gridded data over a user-specified latitude and longitude, and also enables producing latitude-longitude maps of height-integrated Joule heating, Pedersen conductance (height-integrated Pedersen conductivity), Hall conductance (height-integrated Hall conductivity), and height-integrated Pedersen and Hall currents. Height integrations are performed with a trapezoidal integration scheme, according to:

$$\int_a^b f(x)\,dx=\sum_{k=1}^{N}\frac{f(x_{k-1})+f(x_k)}{2}\Delta x \tag{37}$$

where f is the altitude-resolved quantity being integrated, x is the altitude, a and b are the upper and lower limits of integration, and k corresponds to the discrete levels where property f is provided. It is noted that most commonly in GCMs properties are calculated in terms of pressure levels; thus Δx_k is not fixed, but rather increases with altitude. Height integration in Daedalus MASE is performed with the *integration_height* routine of the *daedalusmase_derived_products* module.

In cases where height integration needs to be performed for a discrete altitude interval as opposed to the entire altitude range in a GCM (e.g., for the E region, at 100–150 km), the module performs a re-gridding process by interpolating the property under height-integration at user-defined altitude ranges of fixed length. This enables the subsequent integration to be performed over the same altitude range, rather than integrating over the same number of pressure levels, in which case the altitude range would be variable throughout the globe. The regridding procedure is performed by the *regrid* routine, while the integration with user defined latitude, longitude and altitude limits is performed by the *integration* routine of the module. In the *Jupyter_Notebooks* folder of the Daedalus MASE repository there are two notebooks for the demonstration of the use of this module: the *daedalusmase_derived_products_regrid* performs the regridding procedure for a TIEGCM NetCDF file, and the *daedalusmase_derived_products_integration* calculates the heating (joule, convection or wind) in GW that is confined inside a user defined 3D region of the thermopshere.

Height (and 3D) integration in geographic coordinates is particularly useful for calculating global or hemispherical energy inputs; it is noted, however, that many LTI phenomena are organised with respect to the geomagnetic field lines, due to the strong impact of the geomagnetic field on the motion of charged particles [e.g., Richmond. (1995); Laundal and Richmond. (2017)]. Furthermore, in the ionosphere magnetic field lines are often assumed to be equipotential, and thus the electric potential equation can be simplified and solved as a two-dimensional problem over magnetic longitude and latitude. Field line integrated Hall and Pedersen conductivities (termed conductances) are also key quantities in studying magnetosphere-ionosphere coupling. Thus, for magnetically organized phenomena, such as ionospheric currents, a field-aligned integration scheme is more suitable and is a very useful feature of an LTI toolset. A relevant module performing integrations along magnetic field lines is not currently included in Daedalus MASE, but is planned to be included in subsequent versions.

2.1.7 Derived products plotting utilities module

The *mod_plot_utils* includes routines for the visualization of the *daedalusmase_derived_products* results. These include vertical profiles, map plots, polar plots for the scalar products and quiver plots for the vector products. Examples of module outputs are presented in **Figure 3** of the Results section.

2.1.8 Importing daedalus MASE derived products modules

In order to import the modules, the user must add the *daedalusmase_derived_products* source code path to the Python path. This is done as follows: the built-in Python modules *os* and *sys* should be imported first. The *os* module implements functions on pathnames, while the *sys* module contains parameters specific to the system. The *os. path.abspath()* function is first used to define the path of the *daedalusmase_derived_products* folder which contains the source code of the package; subsequently the *sys. path.append()* function is used to add the source code path to the Python path. Then the modules can be imported to the code. In the following, we present a code snippet which illustrates this procedure:

2.2 Daedalus MASE collision frequencies module

In the Earth's LTI system, the interactions of ions and neutrals influence the velocities, temperatures and densities of both species, thus playing a vital role in momentum transfer and energy exchange between the thermosphere and the ionosphere. These interactions are characterised through the ion neutral collision frequencies, ν_{in}, and subsequently (fundamentally) through the ion-neutral cross sections, σ_{in}. Ion-neutral collision frequencies depend on a number of terms, including the density and composition of the ion and neutral species, the ion and electron temperatures, and values for collision cross sections, σ_{in}; thus, through their effect on the determination of collision rates, collision cross-sections also affect Joule heating and conductivity estimations. Traditionally, collision cross-sections are obtained primarily through laboratory experiments of ion-neutral collisions and model extrapolations, and the resulting values for σ_{in} and ν_{in} are normally taken from published tables such as, for example, the tables in the text book by Schunk and Nagy. (2009). However, these estimations include large measurement uncertainties, and their accuracy has never been conclusively evaluated. The reason is that, whereas the microscopic processes underlying the estimation of σ_{in} are governed by fundamental physics, and whereas the underlying processes are the same in the laboratory environment and in the upper atmosphere, the observation conditions are largely different between the two environments, leading to different estimations, as the cross sections are not measured directly, but are rather inferred from macroscopic quantities. For example, laboratory experiments are not always performed under the similar conditions as in the ionosphere (e.g., they are often at lower temperatures, commonly around 300 K), leading to systematic uncertainties in both σ_{in} and ν_{in} in the upper atmosphere. The discrepancies are evident from comparisons of different estimates in the literature, such as between tables provided by Banks and Kockarts. (1973), Schunk and Nagy. (2009), and Richmond. (2017) and references therein. Due to the above discrepancies and uncertainties, collision cross-sections and collision rates are among the largest sources of errors in empirical models, GCMs and magnetosphere-ionosphere coupling simulations, to all of which they are key inputs. Together, the collision rates and collision cross sections represent the largest source of uncertainty in estimating the ionospheric conductivity, which is a key parameter in current coupled models of the LTI. Existing proxies of these parameters are based on semi-empirical relations and show large deviations due to lack of co-located measurements of all required parameters.

```
import os
import sys
#Define daedalusmase_derived_products package location
fpath = os.path.abspath('path_to_source_code/')
#add src path to python path
sys.path.append(fpath)

#import mod_collision_freqs_cross_sections module
import daedalusmase_derived_products.mod_collision_freqs_cross_sections.sub_cross_sections as cross_sec
#import mod_conductivities module
import daedalusmase_derived_products.mod_conductivities as cond
#import mod_currents_magnetic_forcing module
import daedalusmase_derived_products.mod_currents_magnetic_forcing as currents
#import mod_tiegcm_utils module
import daedalusmase_derived_products.mod_tiegcm_utils as tiegcm
#import mod_heating_sources module.sub_heating_rates
import daedalusmase_derived_products.mod_heating_sources.sub_heating_rates as heatings
#import mod_heating_sources module.sub_plot_utils
import daedalusmase_derived_products.mod_plot_utils as plots
```

The E region of the Ionosphere, between 90 and 150 km approximately, is where electric currents perpendicular to the magnetic field (i.e., Pedersen and Hall currents) are mostly concentrated. These currents are the result of the relative velocities between electrons and ions. As electrons and ions interact with the neutrals their velocities deviate from the $\vec{E}\times\vec{B}$ drift which leads to different ion and electron velocities. Electrons are strongly bounded to the magnetic field, $\vec{E}\times\vec{B}$ drifting through this ionospheric altitude range, thus their interactions with neutrals (i.e., electron-neutral collision frequencies) are less important. On the other hand, the ions start to deviate from $\vec{E}\times\vec{B}$ at approximately 200 km and at the bottom of the E region are fully entrained by the neutrals, which indicates the significance of ion-neutral collision frequencies in the region. The dominant ion species in the region are O_2^+ and NO^+ and the dominant neutral species are O, O_2 and N_2, thus the interactions between these species and the corresponding collision frequencies are the key parameters in the region, that affect the conductivity and the electric currents.

Further up, in the F region, between 150 and 500 km, O and O^+ are the dominant constituents, thus the interactions between the two are crucial in determining the momentum and energy exchange between the ionosphere and the thermosphere (Salah, 1993). Moreover, the $O-O^+$ momentum transfer collision cross sections are dominated by resonant charge exchange between O and O^+, whereas the contribution from polarisation can be neglected in the F region (Banks, 1966). The importance of $O-O^+$ collision frequency has been highlighted through many studies: for example, Moffett et al. (1990) researched the influence of $O-O^+$ collision frequency on the behaviour of the ionospheric F region, and found that a 70% increase in the collision frequency increased the peak electron density by as much as 25% at nighttime and by 10% at daytime, while they also found an increase of the height of the maximum density by 20 km at night. Moreover, Roble. (1988) studied the influence of the collision frequency on the structure of the neutral atmosphere and found that a 70% increase in the atomic oxygen collision frequency led to an increase of the exospheric neutral temperature by 80 K at daytime, while the global averaged Joule heating was increased by 80%. However, while estimating the $O-O^+$ cross section is crucial, it has never been measured directly in the LTI. Ion-neutral cross sections have been derived from laboratory measurements by extrapolation of the low-energy laboratory data to higher energies, although recent investigations indicate that these measurements might not be accurate or applicable (e.g., Archer et al. (2017)). Since the estimates of ion-neutral collision frequencies are currently not accurate and include large discrepancies, the assumptions that go into their estimations need to be well understood and the variations arising from different ion-neutral collision frequency models need to be considered.

Whereas in **Section 2.1.3.1** we calculated ion-neutral collision frequencies based on the formula of Sangalli et al. (2009) and the theoretical formulas of Schunk and Nagy. (2009), here we use a set of different formulas for the calculation of collision frequencies found in literature. The *daedalusmase_collision_frequencies* package aims to provide a computational compendium of all estimation methodologies and parameterizations, allowing their cross-comparison while revealing their discrepancies, e.g. as a function of altitude or temperature, and the quantitative effect of these discrepancies in the calculation of the ionospheric conductivities. In the following we present the corresponding routines that comprise *daedalusmase_collision_frequencies* modules, along with analytic theoretical descriptions of the formulas used. A list of the routines along with corresponding description is presented in **Supplementary Table S8**. The package is accompanied by a Jupyter notebook, in the *Jupyter_Notebooks* folder, that can be used as a stand-alone simulation tool as well as a tutorial for the use of the module.

2.2.1 Collision frequencies module

The *mod_collision_freqs* module includes all the routines needed for the calculation of ion-neutral collision frequencies based on different models. In its current version the module includes routines for the calculation of the collision frequencies between O^+, O_2^+ and NO^+ ions and O, O_2 and N_2 neutral species, based on Banks. (1966) (*vin_banks* routine), Schunk and Walker. (1973) (*vin_schunk_walker* routine), Schunk and Nagy. (2009) (*vin_schunk_nagy* routine), Richmond. (2017) (*vin_richmond* routine) and Ieda. (2020) (*vin_ieda* routine). Especially, for the O^+-O collision frequencies, *op_o_cols* routine includes the parameterizations and equations used in the studies by Dalgarno et al. (1964), Banks. (1966), Stubbe. (1968), Schunk and Walker. (1973), Salah. (1993), Pesnell et al. (1993), Hickman et al. (1997), Richmond. (2017), Schunk and Nagy. (2009) and Ieda. (2020). Furthermore and in order to access the impact of the differences of the ion-neutral collisions models on the conductivities, the module includes the routines *pedersen* and *hall* for the calculation of the Pedersen and Hall conductivities respectively.

2.2.2 Collision frequencies utilities module

The routines of the *mod_collision_freqs* module, need ion and electron temperatures, geomagnetic field, and ion, electron and neutral densities as inputs. These parameters, as was mentioned above, can be extracted either from GCMs or from empirical models. The *mod_utils* module, includes routines for running empirical models and extracting the needed parameters. More specifically, temperatures and densities of electrons and ions are derived from IRI 2016 (*run_iri* routine), the neutral temperature and densities from MSIS00 (*run_msis* routine) and the geomagnetic field from IGRF 2012 (*run_igrf* routine).

Furthermore, this module includes the definitions of the needed constants for the calculations.

2.2.3 Collision frequencies plotting module

The *mod_plot_utils* includes routines for the visualization of the *mod_collision_freqs* module results. These include vertical profiles of collision frequencies and conductivities, as well as temperature dependence of collision frequencies. **Figure 4** of the Results section, presents three example plots of the module.

2.2.4 Importing daedalus MASE collision frequencies modules

In order to import the modules, the user must follow the same procedure as described in **Section 2.1.8** In the following, we present a code snippet which illustrates this procedure, along with the calculation of Pedersen conductivity. The output plot of this code is presented in **Section 3.2**, **Figure 4A**.

```
import os
import sys
import datetime
import numpy as np
fpath = os.path.abspath("path_to_source_code")#define source code location
sys.path.append(fpath) #add src path to python path #add src path to python path
#import Daedalus MASE packages
from daedalusmase_collision_frequencies.mod_utils import allocations as alloc
from daedalusmase_collision_frequencies.mod_utils import const
import daedalusmase_collision_frequencies.mod_utils as utils
import daedalusmase_collision_frequencies.mod_collision_freqs as cols
import daedalusmase_collision_frequencies.e_n_collision_freqs as col_elec
import daedalusmase_collision_frequencies.mod_plot_utils as plots
e_n_collision_freqs
from daedalusmase_collision_frequencies.mod_utils import allocations as alloc
#Define simulation parameter#
timer_value=datetime.datetime(2015, 3, 17, 0, 0, 0)#simulation time
min_alt=100#minimum altitude in km
max_alt=500#maximum altitude in km
dalt_sim=1#altiltude resolution in km
lon_value=-117.5#longitude in deg
lat_value=36.5#latitude in deg

#main code
alt_range=np.arange(min_alt,max_alt,d_alt)
for i in range(0,len(alt_range)):
    B_enu,b_unit_enu=utils.run_igrf(timer_value, lat_value, lon_value, alt_range[i]) #run IGRF
    ne, Op, O2p, NOp, Np, Tn_iri, Ti, Te=utils.run_iri(timer_value, lat_value, lon_value, alt_range[i])
                                                        #run IRI
    O,O2,N,N2,He,H,Ar,Ntotal,Tn=utils.run_msis(timer_value, lat_value, lon_value, alt_range[i]) #run
                                                        MSIS
    bnorm = np.sqrt(B_enu[0]*B_enu[0] + B_enu[1]*B_enu[1] + B_enu[2]*B_enu[2]) #magn. field magnitude
    vinri,vinNOpr,vinO2pr,vinOpr,vinNOp_N2r,vinNOp_O2r,vinNOP_Or,vinO2p_N2r,vinO2p_O2r,vinO2p_Or,
                                                        vinOp_N2r,vinOp_O2r,vinOp_Or=cols.vin_richmond(
                                                        Ti,Tn,O2,O,N2) #collision frequencies according
                                                        to Richmond [2017]
    ven=col_elec(B_enu,N2,O2,O,Te)
    sigmaped_r,sigmaped_e_r,sigmaped_op_r,sigmaped_o2p_r,sigmaped_nop_r=cols.pedersen(bnorm,ne,Op,O2p,
                                                        NOp,ven,vinOpr,vinO2pr,vinNOpr) #Pedersen
                                                        conductivity

    #Outputs
    alloc.sigma_B.apend(sigmaped_r)
    alloc.sigma_e_B.apend(sigmaped_e_r)
    alloc.sigma_op_B.apend(sigmaped_op_r)
    alloc.sigma_o2p_B.apend(sigmaped_o2p_r)
    alloc.sigma_nop_B.apend(sigmaped_nop_r)

plots.plot_pedersen_contributions(time_sim,lat_sim,lon_sim,altmin_sim,altmax_sim,dalt_sim,savefig=True)
                                                    #ion contributions to Pedersen conduct.
```

2.3 Error propagation module

As described above in **Sections 2.1**, **2.2**, the calculation of the various derived products listed in **Table 2** involves a long list of geophysical observables, or primary products, as listed in **Table 1**. For an *in situ* spacecraft or rocket borne mission, these primary observables are obtained through instruments with errors that can vary as a function of altitude, density, latitude, longitude, solar illumination conditions, spacecraft charging, etc. Thus, the resulting error of derived products is dependent on the errors of primary geophysical observables in a complex way. The aim of the module named *daedalusmase_error_propagation* is to enable the calculation of the relative contributions of errors of primary observables into the resulting error of a derived observable. Thus, after derived products are computed as discussed in **Sections 2.1**, **2.2**, their uncertainties are estimated through standard propagation of variances through the respective analytical formulae. As errors are largely dependent on the instrumentation and the measurement methodology that is used, error estimates are introduced by the user per primary observable. As discussed in the Daedalus Report for Assessment (ESA, 2020), in the Phase-0 requirements definition study of Daedalus, assumed errors were introduced based on the corresponding targeted measurement uncertainty per observable. This enabled the cross-comparison of the effects that errors have in terms of their propagation into the resulting uncertainty in obtaining each of the derived observables.

As an example, the three heating terms mentioned in **Table 2** correspond to the same heating source, expressed in terms of different variables; similarly, two different methodologies were presented with respect to perpendicular current estimations. Thus, different corresponding instruments are used in their derivation, and the resulting errors are also expected to be different. The error propagation module enables the cross-comparison of the different estimation methodologies and their performance in terms of error in the various regions as a function of altitude and instrument characteristics (forward approach). Together with these heating terms, error propagation calculations were performed for all derived products listed in **Table 2**. Due to the multi-page extent of the formatted calculations of the resulting errors, the reader is referred to the Jupyter Notebooks that accompany the *daedalusmase_error_propagation* code. A list of the routines that comprise the module along with corresponding description is presented in **Supplementary Table S9**. We note that the errors introduced on the primary geophysical observables correspond to the total error, thus both random and systematic errors are injected. These are used to evaluate numerically the total errors propagating into the derived products. The values of the primary geophysical observables are obtained similarly to the modules described in **Sections 2.1**, **2.2** above, *via* GCMs. As an example call of the module, we present a code snippet for the calculation of the errors along altitude, over a specific latitude-longitude point:

```
import sys
sys.path.insert(0,"../source_code") #add src to Python path
import factors #import factors module
import productderivation as pD #import product derivation module
import supportfunctions #import support functions derivation module
import plots #import plots
import errorpropagation as EP #import error propagation module
filename="path_to_TIEGCM_file" #define input file
timer=5 #define timestep
lat_value=61 # TIEGCM index which corresponds to 63.75 deg #define latitude
lon_value=49 #  TIEGCM index which corresponds to -57.5 deg #define longitude
pD.models_input(filename, timer, lat_value, lon_value) #run models
pD.products(lat_value, lon_value) #calculate products

# factors.pedersen_con[0,49,:-1]
min_alt=100 #define minimum altitude
max_alt=400 #define maximum altitude
plots.plot_conductivities(lat_value, lon_value, min_alt, max_alt) #ploting

error_flag=True
EP.error(error_flag=error_flag, lat_value=lat_value, lon_value=lon_value) #run error propagation
plots.plot_conductivities_error(lat_value, lon_value, min_alt, max_alt) #plotting
```

2.4 Interpolation module

Physics based global circulation models such as TIEGCM, GITM and WACCM-X return results at a fixed number of locations, which most commonly are irregularly gridded. Hence a spatial interpolation scheme is needed to obtain an along-orbit estimate of any of the geophysical observables simulated in the GCM. In Daedalus MASE, an interpolation module named *daedalusmase_interpolation* is included as a standalone function and can be called by external programs as needed. The main program is written in python, however for large numbers of orbit points and large file sizes of the gridded datasets, interpolation can also be carried out by a subroutine written in C++, encapsulated in a python wrapper (F2PY3). The C++ subroutine adheres to the OpenMP interface, so the end user can opt for parallel interpolation. If opted for, the workload is distributed across as many processing threads as available, accelerating the execution. This is done as interpolating along orbit with a cadence of a few seconds through a gridded dataset consisting of thousands of points and for multi-year simulations is a computationally expensive process that requires parallel processing.

The method used in *daedalusmase_interpolation* is a trilinear interpolation scheme. A similar interpolation scheme was used by Elvidge et al. (2016) to interpolate data outputs from TIE-GCM and GITM to the position of the CHAMP spacecraft and then compare those with actual measurements. In *daedalusmase_interpolation_scheme*, the user can opt for first or second-order trilinear interpolation as well as other methods such as cubic splines. Given the orbit data of a specific mission, and depending on the method opted for, the interpolation scheme searches the 8, 16 or more neighboring grid points, deposits interpolating weights to each point and then interpolates those values to Daedalus' position. When used in a multi-threaded environment the program can interpolate multiple positions in parallel. A list of the routines that comprise the module along with corresponding description is presented in **Supplementary Tables S10**. In the following, we present a code snippet for the trilinear interpolation of neutral temperature along a satellite orbit, through a TIEGCM NetCDF file:

2.5 Coverage calculation module

A key aim of the Daedalus mission assessment exercise is to demonstrate the ability of the proposed mission concept to capture geophysical observables and derived products related to the mission objectives globally with sufficient statistics. This module, named *daedalusmase_coverage_calculator*, aims to estimate the coverage of regions of interest with along-track measurements. The present module of Daedalus MASE allows introducing orbital files exported in standard formats from common orbital propagators, such as FreeFlyer (https://ai-solutions.com/freeflyer-astrodynamic-software/), NASA's General Mission Analysis Tool (GMAT) (https://software.nasa.gov/software/GSC-17177-1)and Ansys Systems Tool Kit (STK) (https://www.ansys.com/products/missions/ansys-stk). Since key features in the ionosphere and thermosphere, such as electrical currents, auroral particle precipitation, plasma and neutral motion and magnetic disturbances are organized by the Earth's magnetic field, magnetic coordinates are a more natural system in which to represent geophysical observables in the LTI (Laundal and Richmond, 2017); thus, module *daedalusmase_coverage_calculator* includes function *AddMagneticCoordinates* that appends magnetic latitude and magnetic local time information for each position along orbit. Subsequently, the user can enter details of the regions of interest, such as latitude (or magnetic latitude) and longitude (or magnetic local time) ranges by calling the function *CreateNewBin*.

Based on these ranges, this module then performs estimates of the coverage in terms of total sampling time within these regions of interest over the mission lifetime. The module also enables estimating the total sampling time under user-defined conditions, such as solar illumination and the *Kp* index of geomagnetic activity. The calculations are executed by function *calculate_coverage* and the results are saved in a file. Prior to the calculation, the user can set the paths of the files by calling functions *set_orbit_files_path*, *set_coverage_results_files_path*, and *set_geomagnetic_indices_files_path*. The user must also call function *read_geomagnetic_indices* to specify the time period of interest.

```
import sys
sys.path.insert(0,"../source_code")#add src to Python path
import interpolationmase_mainfunc as inter #import interpolation module
ModelName="path_to_TIEGCM_file" #define input file input file
OrbitFile="path_to_orbit_file" #define input orbit
inter.runinterpolator(ModelName,OrbitFile,tgvar="TN",interpolation="Trilinear",Save=True,outfilename="
                                                    InterResults.nc") #run interpolator
```

The resulting data produced by the above calculations can be used directly to create charts. Past coverage calculations can be retrieved by calling function *load_coverage_results*. Charts can be created by functions *plot_coverage_bars*, *plot_coverage_bars_grouped_by_region*, *plot_coverage_polar_chart*, and *plot_orbit_kp_scatter*, *plot_orbit_heatmap*.

A list of the routines that comprise the module along with corresponding description is presented in **Supplementary Table S11**.

As an example, the following function presents the use of the *daedalusmase_coverage_calculator* module.

2.6 Global statistics module

A key question that often accompanies *in situ* sampling schemes is whether a specified orbital scenario will obtain, over the mission lifetime, statistically representative measurements within the regions of interest. For example, as part of its primary science objectives, Daedalus aims to quantify Joule heating (among other derived products) within the regions where it maximizes; thus, a key requirement of the Daedalus mission is for Joule heating to be estimated globally with sufficient statistics, so that the *in situ* measurements are representative of the variability of Joule heating and can enable reproducible parameterizations of Joule heating in global models. The related performance metrics of the Daedalus mission set the thresholds of when a statistically representative coverage can be obtained over the mission lifetime, which was defined in terms of the mean and variance of Joule heating within the identified regions of interest. The purpose of this module, named *daedalusmase_global_statistics*, is to investigate quantitatively whether the dataset of a variable obtained *via* an *in situ* sampling scheme is sufficiently large and diverse to determine the average value of both primary geophysical observables, such as those presented in **Table 1**, and derived products, such as those of **Table 2**, within each region of interest, to within a specified fraction of the median (as determined from models), and for a specified range of driving conditions. In the Daedalus mission performance simulations, using the Daedalus MASE code, this was demonstrated *via* the following scheme:

1. TIEGCM was run for the minimum mission lifetime of Daedalus, corresponding to a 3 year lifetime, with a temporal resolution corresponding to approximately the orbital period.
2. The gridded dataset of TIEGCM was sampled along orbit *via* an interpolation scheme (see module *daedalusmase_interpolation_scheme* described above).
3. Regions of interest (or bins) were defined by ranges of altitude, magnetic latitude, magnetic local time and the *Kp* index of geomagnetic activity.
4. Statistics were assembled within the regions of interest, and the mean and variance were calculated within the pre-set bins.
5. The statistical characteristics of the dataset of the along-orbit sampling of TIEGCM were compared to the characteristics of the dataset that included the full statistics of all TIEGCM gridded points within the regions of interest. Accordingly, a statistical distribution or Probability Density Function (PDF) of Joule heating was constructed for each of the defined regions of interest based on Joule heating measurements along the orbit, and the PDF was compared with the corresponding PDF from the full statistics of TIEGCM gridded points. For the comparison, the median value (Me) and Median Absolute Deviation (MADe) were calculated and inter-compared.

```
import sys
sys.path.append("../daedalusmase_coverage_calculator/")
# module imports
from data import *
from plot import *
from utils import *
from orbit_file_conversions import add_magnetic_coordinates
add_magnetic_coordinates( "orbit1.csv", "orbit2.csv" )
# --- initialization ---
create_new_bin( "SMP_A1", "Sample region", 22, 2,   70, 80,   100, 150,              4, 9,   20*60 )
read_geomagnetic_indices(2015, 2016)
# --- Calculation ---
BinMisses, BinHits, ResultsFilename, Duration = calculate_coverage( "Test", " ", "Orbit.csv", "2015" )
if(BinMisses==0 and BinHits==0 and Duration==0): # Load pre-executed calculation.
# --- Loading ---
  load_coverage_results( CoverageResults_Files_Path + "Orbit.csv.2015.CoverageResults.txt" )
else:
  print("Bin hits:", BinHits,"Bin misses:", BinMisses, "Output:", ResultsFilename)
# --- Ploting ---
plot_coverage_polar_chart( "Polar Chart", 1.4 )
plot_orbit_kp_scatter("Orbit.csv", 2015, "01-01-2015", "20-01-2015", 100, 220 )
plot_orbit_heatmap( "Orbit.csv" )
plot_coverage_bars( "Coverage Bars" )
plot_coverage_bars_grouped_by_region( "Coverage Bars by Region", [ ["SMP"], ["AEE", "AEM","AED"],  ["AFM
                                                ","CF","PCF"],  ["EPB", "SQ"],  ["EEJ"] ] )
```

6. The statistical significance of the along-orbit sampling when compared to truth was evaluated by performing the Wilcoxon Rank Sum Test for the two PDFs. This metric gives an estimate of what constitutes a statistically significant sample of measurement locations so as to enable the accurate parameterization of derived products.

The result of the above module was to provide a metric that allows estimating what constitutes a statistically significant sample of measurement locations so as to enable the accurate parameterization of the sampled primary and derived products. The sample of measurement locations, termed mission coverage performance, enables quantifying the sampling requirements of the mission and justifying the orbit selection. The mission sampling requirements thus evolve iteratively in concert with mission performance assessments in Daedlus MASE, chosen to ensure globally adequate statistical representation of the main derived products within predefined regions.

The calculation of the statistics per region of interest is performed by the function *calc_stats_for_orbit* for along-orbit sampling of TIEGCM and by the function *calc_stats_for_tiegcm* for the ensemble of the TIEGCM gridded points. The results are separated by user-specified regions of interest and are saved in files of NetCDF format. These data can be subsequently read by the function *load_results* and plotted by the functions *plot_variable*, *plot_variable_KpSeparated*, *plot_distributions* and *plot_ColorSpread_KpSeparated*. The user can also call *plot_comparison*, *plot_PDFperSubBin* and *execute_stat_test* in order to directly compare along-orbit statistics with global-TIEGCM statistics.

A list of the routines that comprise the module along with corresponding description is presented in **Supplementary Table S12**.

The following example shows the calculation of statistics based on data from a sample satellite orbit. Statistics are calculated based on gridded data produced by the TIEGCM model.

2.7 3D visualizations module

In order to demonstrate the nature of Daedalus *in situ* measurements, to clarify the mission's orbit geometry, and to provide global context to the measurements based on model output, a set of graphics and 3D visualizations were produced during the course of the Daedalus Phase-0 study. The scope of this section is to describe the process and present the code that results in a number of high-quality graphics and visualizations that can be produced based on GCM outputs.

The 3D visualizations involve the following steps:

1. Global model outputs of the thermosphere-ionosphere are produced using GCMs. The output data consist of NetCDF files.
2. Based on simulated orbit ephemeris data, observations from an *in situ* mission are simulated along orbit, utilizing the interpolation module, as described above.
3. Interfaces are created between the input data sources and graphics software, including Generic Mapping Tools https://doi.org/10.1029/2019GC008515 for 2D graphs and text annotations and Blender https://www.blender.org for 3D rendering. The interface to Generic Mapping Tools has

```
import sys
sys.path.append( "../daedalusmase_global_statistics/" )
from data import *
from plot import *
# Calculate statistics upon data from the satellite orbit.
calc_stats_for_orbit("../../Sample_Data/tiegcm_data/", "../../Sample_Data/orbital_data/", "./orbit.nc")
# Load the results
load_results( "./orbit.nc", "Ohmic" )
# Calculate statistics upon data produced by the TIEGCM model.
calc_stats_for_tiegcm("../../Sample_Data/tiegcm_data/", "./tiegcm/")
# Load the results
load_results( "./tiegcm/", "Ohmic" )
# Plot the results for a certain region (AAA) and a certain variable (Ohmic heating)
plot_variable( "AAA", "Ohmic" )
plot_variable_KpSeparated( "AAA", "Ohmic" )
plot_distributions( "AAA", "Ohmic")
plot_ColorSpread_KpSeparated( "AAA", "Ohmic" )
# Plot comparison charts between to result-files
plot_comparison("./tiegcm/", "./orbit.nc","Ohmic", " ean  values")
plot_PDFperSubBin("./tiegcm/", "./orbit.nc", "AAA", "Ohmic", "10-8W/m3")
# Print the result of statistical tests and comparisons
execute_stat_test("./tiegcm/", "./orbit.nc", "AAA", "Ohmic")
```

been created using the relatively new Python interface, PyGMT (https://doi.org/10.5281/ZENODO.4522136), which allows direct integration of GMT commands in a Python script or Jupyter notebook.

4. The visualisation is set up using the graphics software, and visual exploration of the data and models is initiated, resulting in a selection of (motion) graphics output.
5. The 3D graphics and motion graphics are edited in software such as Adobe Illustrator for static 2D graphics and Apple Motion for animations. In particular, the Motion software makes it easy to combine (composite) many 2D and 3D elements in a single animation, such as the 2D date/time annotations, AE-index graph and colour bars (created with Generic Mapping Tools) with the three separately 3D-rendered Earth and atmosphere model outputs (created with Blender) in 9.
6. Figures and animations are subsequently exported in high resolution formats. Examples of the animations can be found *via* the links in the presentation by Doornbos et al. (2020) as well as in the website: https://daedalus.earth.

It is noted that, for interfacing with Blender, two methods were explored. At first, Python scripts and Jupyter notebooks were used to create files outside of Blender, in formats that can be imported by a user *via* the standard Blender graphical user interface. These file formats include Wavefront Object files (.obj) for static geometry, Point Cache (.pc2) files for time-varying geometry, and PNG graphics for textures to be used on the 3D geometry. This approach required extensive overhead in terms of file management and setting up of scenes inside of Blender. A second approach that was explored was to integrate this code with Blender's Python API, effectively creating a plug-in for Blender in which collections of NetCDF model output files and satellite orbit parameters can be selected for rendering. In principle, this approach could lead to an easily reusable plugin package, however, open issues with the installation of the plug-in and Python package dependencies currently still prohibit easy adoption.

The resulting output of the 3D visualizations scheme consist of high-quality graphics in terms of content, style and resolution, as well as scientific figures and animations for a variety of audiences and for explaining and promoting the science and physics processes of the LTI. These visualizations enable to provide feedback, based on the visual exploration of the models and simulated data, on aspects related to the underlying physics of the processes explored, to the sampling characteristics, to the data processing approaches, *etc.* The aim of the present section and the dissemination of the code that produces these 3D graphics and visualizations is to support other scientists in producing high-quality and clear graphics of GCM outputs.

3 Results

In the following, we present some sample results from running the above modules of DaedalusMASE. These are not comprehensive and are only meant to be used as examples for using these modules, demonstrating some of their capabilities; the user is referred to the corresponding code and related online documentation for more information and for an outline of their full capabilities.

3.1 Sample results from daedalusMASE_derived_products

Using the *daedalusmase_derived_products* Jupyter notebook, the user can select to calculate various derived products as well as plot both primary and derived products in GCM simulation outputs. This module currently reads NetCDF files as formatted through standard TIEGCM simulation outputs, but can be easily adapted to use outputs from GITM, WACCM-X and other GCMs that provide a comprehensive list of LTI variables. Through a Graphical User Interface as shown in **Figure 2**, the user can select the TIEGCM input file, the pressure level and the timestep, and the desired parameters to be plotted. Also, one can select from a list of different maps to plot the results. Moreover, an orbit and nightshade can be added to the plots.

The comprehensive list of scalar products (both primary and derived) that can be calculated and/or plotted from this module includes: 1) Joule, Ohmic and frictional heating, 2) Convection heating and wind correction term, 3) Ohmic heating per unit mass and ratio of Joule heating over pressure, 4) Heat transfer to the neutral gas, 5) Pedersen, Hall and total perpendicular conductivities, 6) Ion neutral collision frequencies and Ion neutral cross sections.

The comprehensive list of vector products (both primary and derived) that can be calculated and/or plotted from this module includes: 1) Pedersen, Hall and total perpendicular currents, 2) Magnetic forcing vector, 3) Electric Field, 4) Magnetic Field, 5) Ion drifts, 6) Neutral winds.

It is noted that Daedalus MASE offers the possibility to plot altitude profiles for specific cases and over specific times and locations. These can be of use for multiple applications, including for ground-based investigations that utilize collision frequency models or rocket flights that do not incorporate neutral density measurements.

In **Figure 3** we present some examples of different formats for plotting the scalar and vector products: In **Figure 3A** we plot Joule heating rate in mW/m^3 in 2D latitude-longitude plot. In **Figure 3B** we present an altitude-latitude cut of Joule

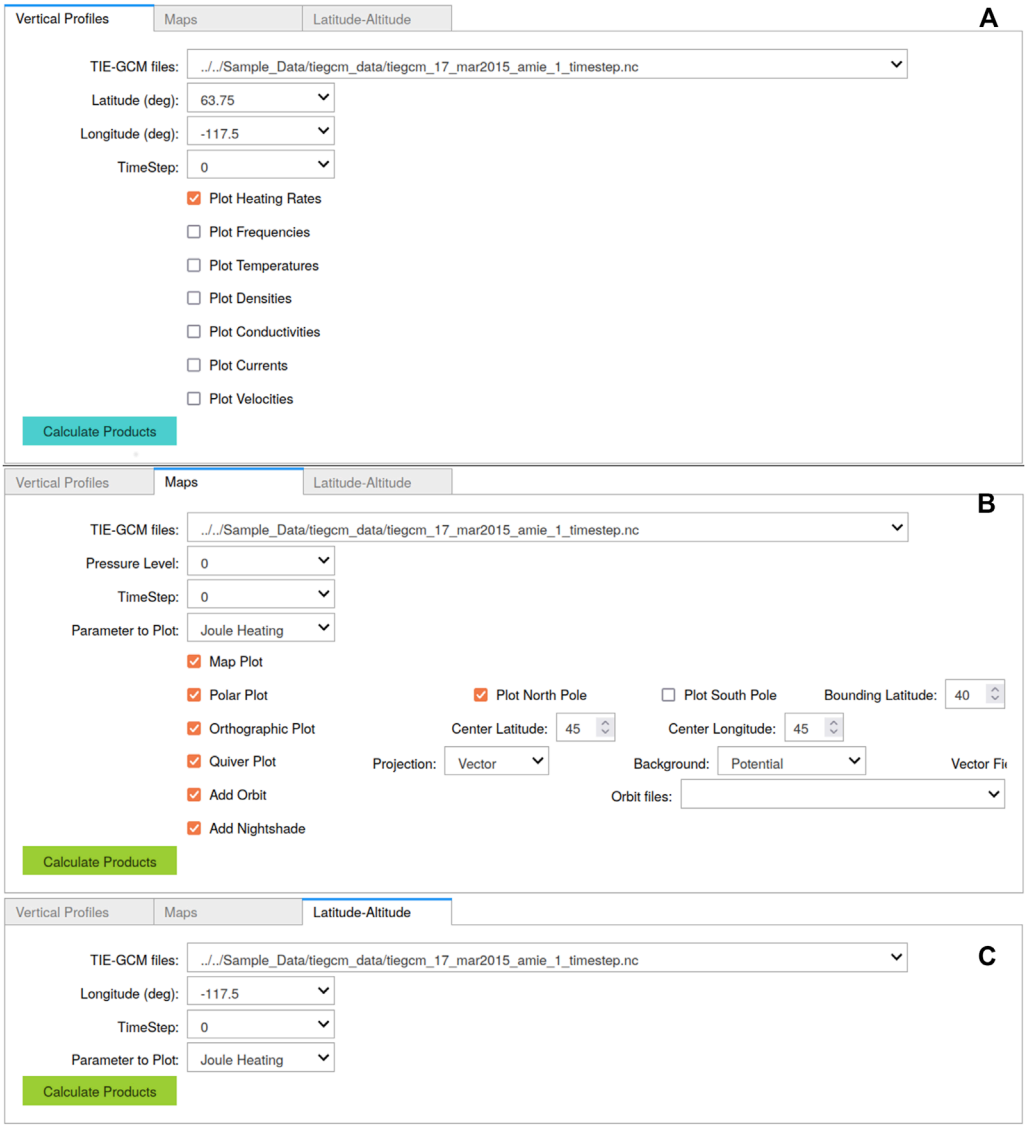


FIGURE 2
Graphical User Interface (GUI) for calculating and plotting: **(A)** vertical (altitude) profiles of various primary and derived products in TIEGCM; **(B)** derived products in TIEGCM as a function of latitude-longitude maps; and **(C)** latitude-altitude profiles of derived products in TIEGCM.

heating. A northern polar plot of the Pedersen conductivity is shown in Figure 3C. In Figure 3D a streamline plot of electric field plotted over a polar plot of the electric potential is shown. Lastly, in Figure 3E, an orthographic view of the global Joule heating rate distribution is presented.

3.2 Sample results from daedalusMASE_collision_frequencies

In calculations of the Pedersen conductivity, σ_P, the ratio (κ_i) of each species' gyrofrequency *versus* its collision frequency needs to be estimated. The collision frequency depends, in turn, on the species' density, thus neutral composition needs to be known. The variability of conductivities stems mainly from the variability of electron density N_e, of the neutral density N_n of various n species, from the temperature-dependent ion-neutral collision frequencies, ν_{in}, and, to a lesser extent, the magnetic field strength B, which remains rather constant with altitude and is well known in the limited range of LTI altitudes, at least compared to other geophysical observables. The above enter the equations for all three conductivities described in Section 2.1.2 (Eqs 20–22), namely, σ_P, σ_H, and $\sigma_\parallel$. To investigate the variability of collision frequencies between the different

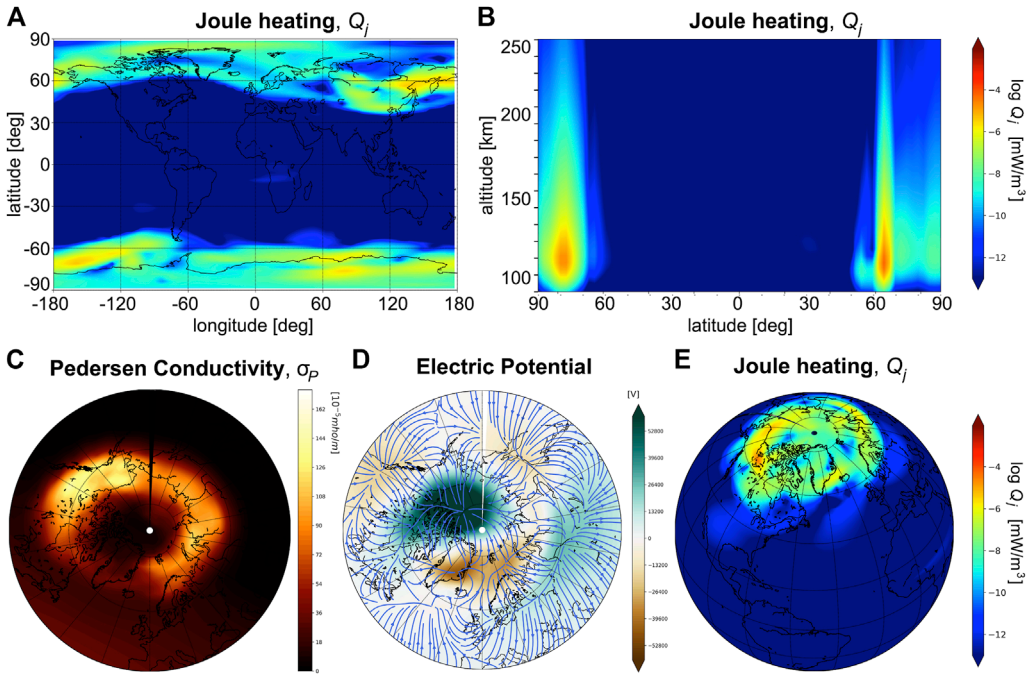


FIGURE 3
(A) 2D latitude-longitude map of Joule heating rate at 120 km. **(B)** Altitude-latitude plot of Joule heating rate along a meridional cut at fixed longitude. **(C)** Polar plot (north) of Pedersen Conductivity at 120 km, **(D)** Polar plot (north) of electric potential with overlaid streamlines of electric field, at 120 km. **(E)** Orthographic global plot of Joule heating at 120 km. All plots correspond to the peak of St. Patrick's day storm, 17 March 2015, 13:55 UT.

models, and further to demonstrate the effect that such a variability of collision frequencies has on conductivities, and, through that, on Joule heating and currents, in Figure 4 we plot: a) the contribution of electron and ion species to the Pedersen conductivity b) normalized collision frequencies of $O^+ - O$ (i.e., collision frequencies divided by the O density) as a function of altitude, based on the various approximations or models, as marked; c) altitude profiles of the ion-neutral collision frequencies, as obtained for the conditions of St. Patrick's day 2015 storm; and c) altitude profiles of Pedersen conductivities, calculated according to the different ν_{in} of panel d). The impact of the different ν_{in} models in the calculation of the Pedersen conductivity is evident in the F region (200–300 km) where significant discrepancies can be observed in σ_P.

3.3 Sample results from daedalusMASE_error_propagation

Using the Daedalus MASE error propagation calculations module, the user can select to calculate the error propagated onto various derived products, as listed in Table 2, as well as plot errors as a function of altitude or as maps in latitude and longitude. Through a Graphical User Interface as shown in the Figure 5A, the user can select the TIEGCM input file, the errors per geophysical observable, and the desired parameters for which to calculate the propagated error.

As an example of the propagation of errors in the calculation of derived products, in the Figure 5B we show the inter-comparison of the errors in obtaining Joule, Ohmic and frictional heating, the three different estimates of the same process by which the relative motion of plasma and neutrals results in heating in the LTI. The total uncertainty propagated onto the corresponding heating profiles, based on assumed uncertainties on the contributing (primary) variables, is shown in this figure as a function of altitude. The vertical profiles are obtained during the peak of a geomagnetic storm that occurred on 17 March 2015, also referred to as St Patrick's day storm 2015. This figure illustrates how the three estimates produce very similar profiles–at least in the self-consistent model world–and how the uncertainty of the methods is predicted to differ, as a function of altitude. Here, it is shown that the relative total uncertainty of at least one of the methods stays well below 15% throughout the E region.

The individual errors of the various geophysical observables that were introduced in the above exercise are consistent with the observational requirements in terms of total errors; however, this module allows realistic or actual measurement

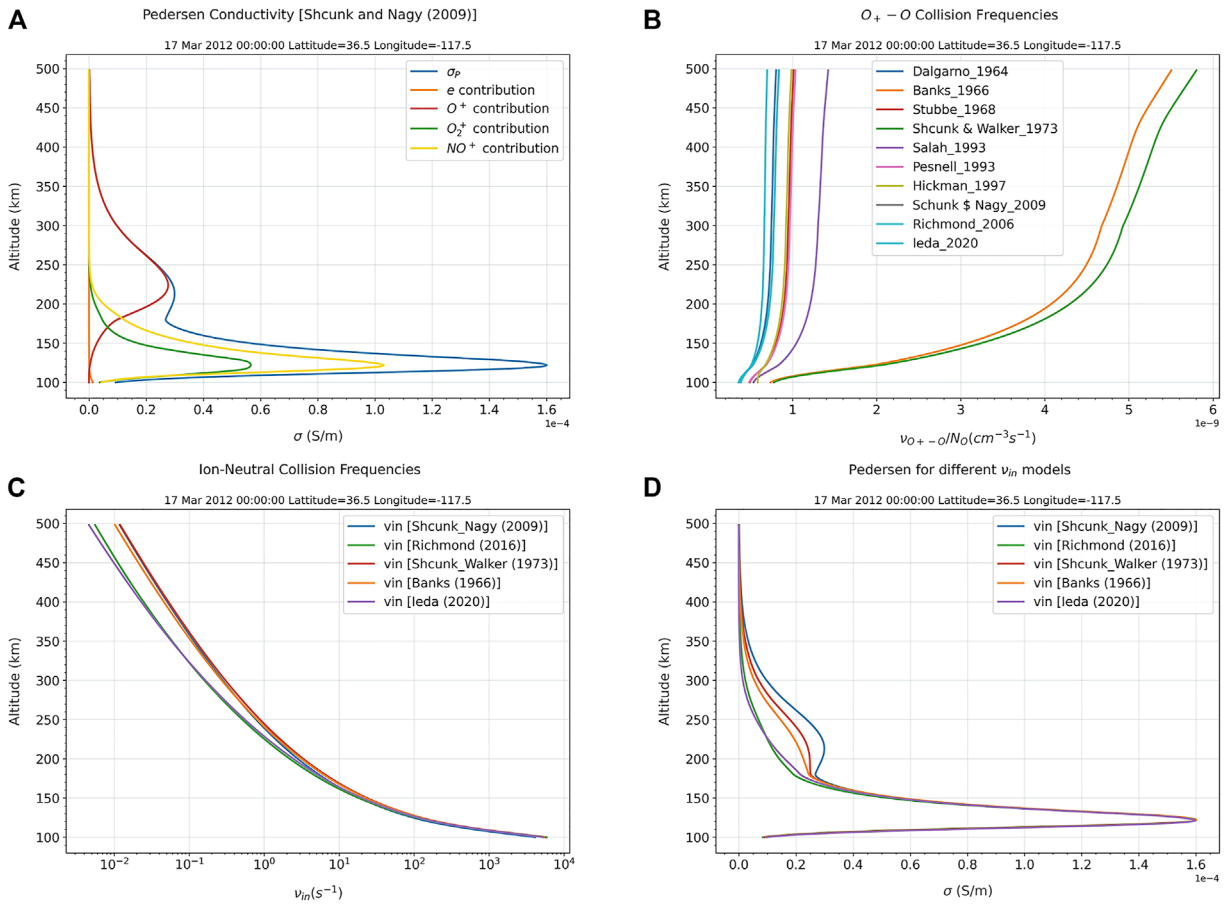


FIGURE 4
(A) Electron and ion contributions to the Pedersen conductivity **(B)** Normalized collision frequencies of $O^+ - O$ as a function of altitude, based on the various models, as marked; **(C)** altitude profiles of the ν_{in} collision frequencies during St. Patrick's day 2015 storm; and **(D)** altitude profiles of the Pedersen conductivity, calculated with ν_{in} from different models as marked.

errors to be introduced in a similar fashion. Furthermore, errors can be injected in this module after being estimated through parametric instrument simulators. Errors can thus be a function of the local environment, such as altitude, solar illumination, latitude, *etc.* In particular, the low altitudes of the LTI are known to introduce wake effects and ram perturbations to various geophysical observables; this module enables injecting estimated errors of measured quantities and propagating their effects onto the derived products.

3.4 Sample results from daedalusMASE_interpolation_scheme

In the following we present an example of the along-track interpolation, at 10-s intervals, which is used to produce synthetic time series data from GCM gridded datasets. In the case of the Daedalus Mission Assessment exercise, the interpolation of the GCM gridded dataset enabled the creation of synthetic geophysical observable time series, which were extracted along realistic orbit tracks. The data underpinning this exercise consisted of a mission lifetime global self-consistent simulation of the comprehensive LTI environment, which corresponded to 3 years; this simulation was then sampled along a 3-year set of simulated orbits, producing the synthetic time series.

The left-hand side of **Figure 6** shows an example of an orbit of the Daedalus spacecraft, with vectors of the neutral winds, ion drifts and electron drifts over-plotted along the orbit. The middle panels show a "dashboard" of instantaneous measurements, obtained through the along-orbit interpolation. The right-hand side panels show, in sequence from top to bottom, the geodetic latitude, longitude and altitude of the simulated spacecraft orbit, and the along-orbit interpolated time series of electron density. Further to the application for the Daedalus Mission Performance Demonstrations, the creation

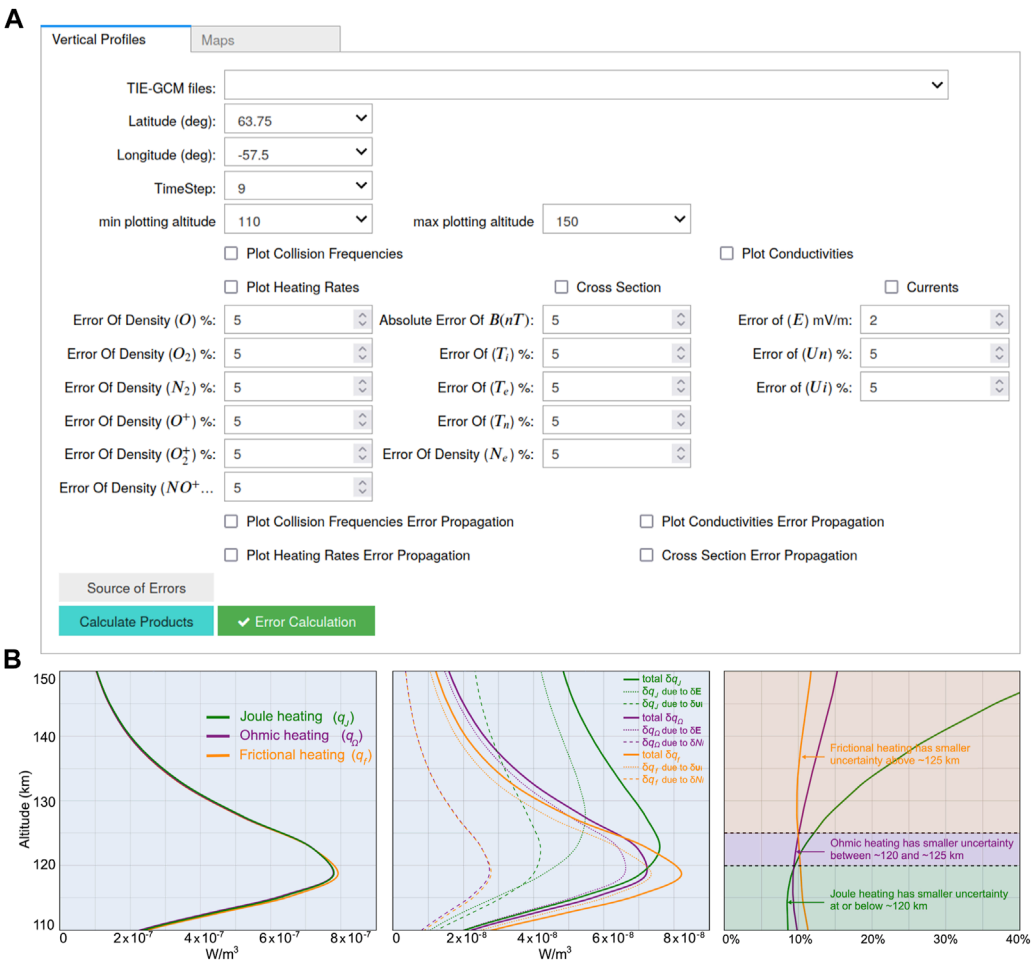


FIGURE 5
(A): a Graphical user Interface (GUI) for the selection of parameters of the error propagation, such as the location, derived product and errors per geophysical observables. **(B)**: Example of Joule heating total uncertainty propagation for a vertical profile at 63.75^{O} latitude, −57.5^{O} longitude, 2015-03-17 18:00, for the three estimation methods. Left, profiles as drawn from TIEGCM grid column. Middle, absolute total uncertainties, and principal contributors to total uncertainty (electric field δ E, ion drift δ vi, and ion number density δ Ni). Right, relative total uncertainties in E region.

of synthetic time series enables the cross-comparison of model data with measurements from satellites, as well as the creation of model altitude profiles for ground-based investigations that utilize collision frequency models or rocket flights.

3.5 Sample results from daedalusMASE_coverage_calculator

In support of the scene generation and performance demonstration modeling, Daedalus lifetime simulations were performed using TIEGCM, for the baseline duration of the Daedalus mission (3-year), with the following considerations: The time period for the 3-year TIEGCM scene generation runs was selected based on the phase of the solar cycle expected during the launch of Earth Explorer 10, originally planned for 2027 or 2028. Since geomagnetic activity is largely solar cycle-driven, the corresponding phase of the past solar cycle was simulated. The consensus prediction for the timing of the next solar cycle (solar cycle 25) is given in **Figure 7**, based on solar sunspot number prediction. It was thus expected that Daedalus' launch would occur during the descending phase of the solar cycle. The most recent descending phase was that of solar cycle 24, also shaded in the **Figure 7A**. The corresponding time period is from 1 January 2015, until 31 December 2017.

Subsequently, an orbital simulation was run, based on atmospheric drag predictions for the same time period. The orbital simulation includes apogee maintenance and perigee descent maneuvers to lower-most altitudes from nominal altitudes, aiming to address the coverage requirements, as stated in ESA. (2020). A summary of orbital information based on the orbital design of the Daedalus Mission Concept is

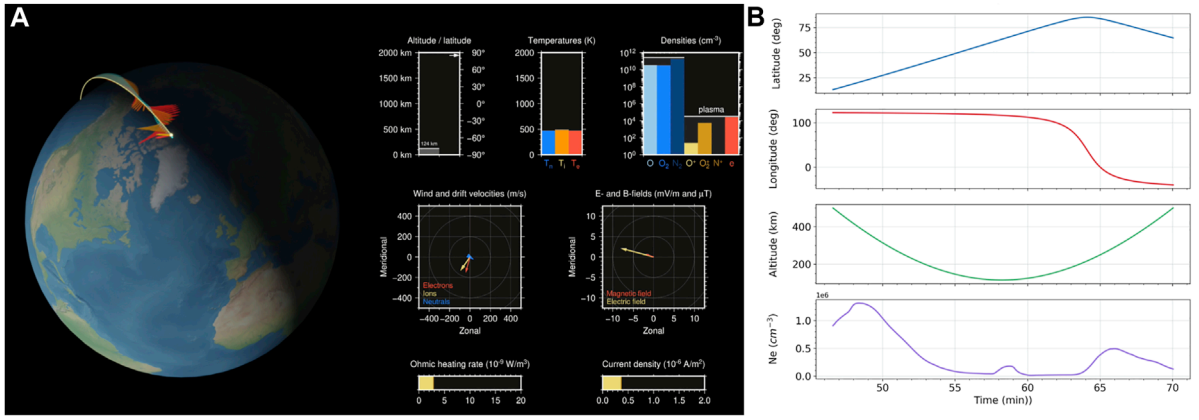


FIGURE 6
Example of the along-orbit interpolation of various primary geophysical observables: The **(A)** (globe) shows sample orbit of Daedalus, with vectors of neutral winds and ion drifts plotted along the orbit; the middle panels show a "dashboard" of instantaneous measurements, obtained through the along-orbit interpolation; the **(B)** show spacecraft latitude, longitude, altitude, and along-orbit time series of electron density.

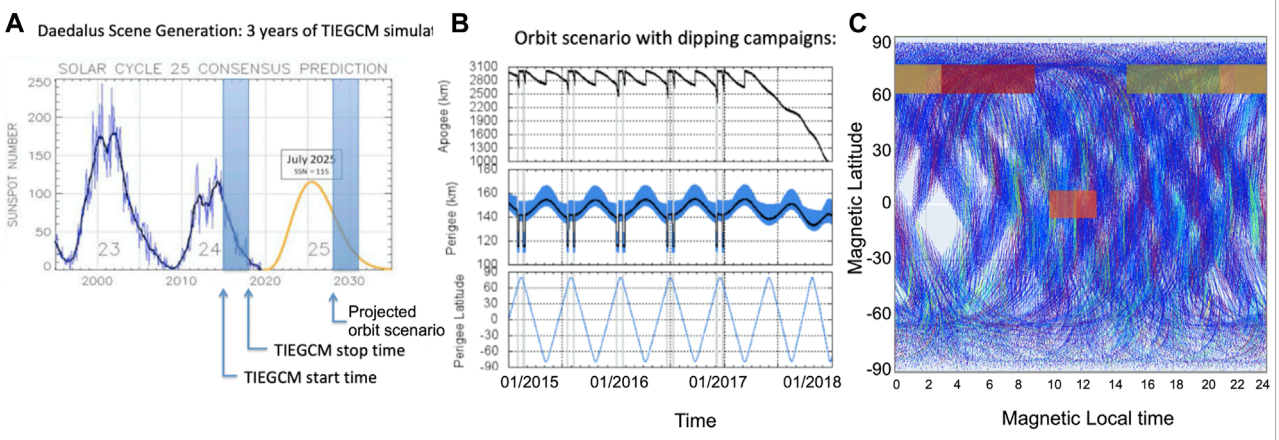


FIGURE 7
(A): Historical record of solar cycles 23 and 24, and consensus prediction of solar cycle 25. **(B)**: Orbital parameters based on simulations of a sample Daedalus orbital scheme. **(C)**: Latitude vs local time coverage in the 100–200 km altitude range; color-coded is the *Kp* index, ranging from 0 (in blue) to 9 (in red).

given in the Figure 7B, according to Sarris et al. (2020). An example of orbital tracks is shown in the Figure 7C, where the *Kp* index is plotted in color along Daedalus' orbit. Regions of interest are marked in colored rectangular stripes, and within these regions statistics of the obtained measurements are calculated using module *daedalusmase_global_statistics*.

3.6 Sample results from daedalusMASE_global_statistics

Based on the calculation of the derived products, as described Sections 3.1, 3.2, and the coverage calculations, as described in Section 3.5, the purpose of this module is to estimate the statistical significance of obtaining various primary and derived products with the along-sampling scheme of *in situ* measurements by the Daedalus mission concept.

Figure 8 demonstrates the comparison between the statistical distribution of Joule heating as calculated by the TIEGCM model, shown in the top panels, and Joule heating as sampled *in situ* along the orbit with a cadence of 1 s, as shown in the lower panels. An altitude profile of Joule heating is shown in the left panels, were the region of interest that is investigated is the auroral midnight sector, defined by Magnetic Latitudes of 60–75°, Magnetic Local Time between 21:00 and 03:00 and Altitude of 100–160 km. Ohmic heating

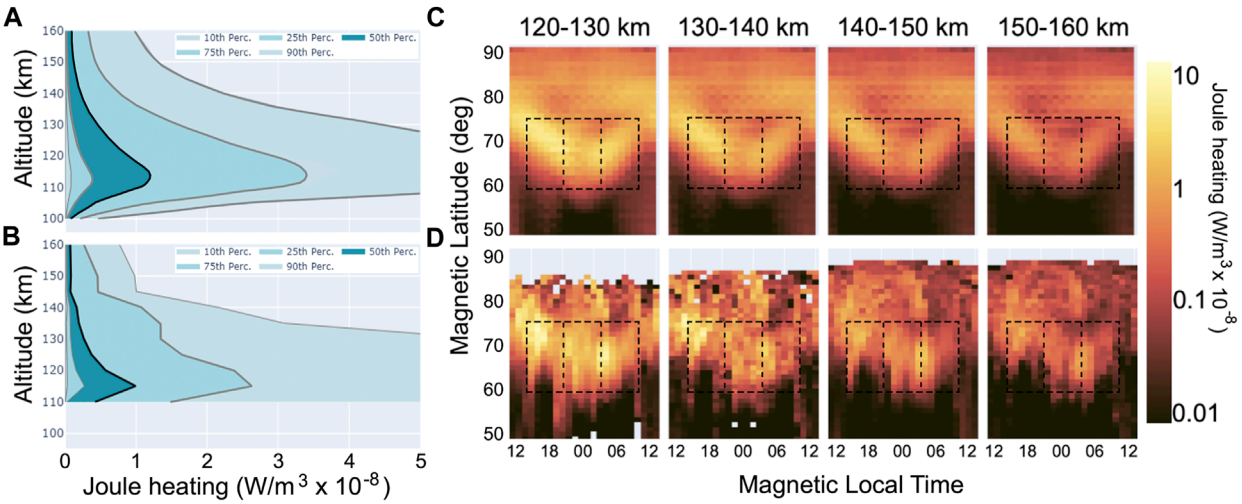


FIGURE 8
(A,B): Statistical distribution of Joule heating in altitude, as calculated by all gridded data of TIEGCM within magnetic latitudes from 60 to 75° and local time between 21:00 and 03:00 **(A)** and as sampled along orbit within the same region **(B)**. **(C,D)**: Statistical distribution of Joule heating in magnetic latitude and local time, for four altitude bins, as marked, based on all TIEGCM gridded data **(C)** and along-track sampling **(D)**.

as a function of magnetic latitude and local time is shown in the right panels. The lower panels visualize the accuracy to which each region can be described by the orbit samples.

Further to the visual inter-comparison shown in Figure 8, as described in Section 2.6, in order to evaluate the statistical significance of the along-orbit sampling when compared to truth (i.e., the gridded dataset), this module calculates the median values (Me) and Median Absolute Deviation (MADe) for the two datasets, and also performs the Wilcoxon Rank Sum Test for the two PDFs.

3.7 Sample results from daedalusMASE_3D_visualizations

In Figure 9, as an example of 3D visualizations, a snapshot of the evolution of key geophysical observables in the LTI are given: temperature, electron density and neutral winds are plotted as a function of pressure level in the Thermosphere during a geomagnetic storm event, as parameterized here *via* the auroral electrojet (AE) index of geomagnetic activity, following a fast-moving Coronal Mass Ejection (CME) that hit the Earth's magnetic field. The AE index, shown in the lower center part of Figure 9, provides a global, quantitative measure of auroral zone magnetic activity produced by enhanced ionospheric currents flowing below (i.e., at sub-auroral latitudes) and within the auroral oval; the red line indicates the snapshot time. As it can be seen in the movie of this event in the Supplementary Material, initially, solar EUV radiation dominates, causing diurnal variations. Later on in the simulation, intense variations are observed, primarily at high latitudes, causing travelling disturbances that move to lower latitudes, as well as longitude-dependent variability. These effects are caused by energetic particle precipitation (EPP) from the magnetosphere into the atmosphere, together with waves, tides and gravity waves from below and also by Joule heating (JH) of the lower thermosphere due to mega-Ampere currents that are closing within the same 100–200 km range where temperature and composition change drastically.

4 Summary and conclusion

Daedalus MASE routines and modules were originally designed as a mission simulator for the Daedalus mission concept, which targets to sample the LTI with *in situ* measurements down to extreme low altitudes, with a main focus in the under-sampled altitude range of 100–200 km. The underlying assumption for Daedalus MASE is that the primary observables used for the calculation of derived quantities are obtained by instruments capable of performing measurements *in situ*, and the goals of the mission simulator included the assessment of the closure of the mission objectives by the *in situ* sampling scheme of Daedalus, including, but not limited to, the demonstration of achieving the calculation of a set of derived products, an assessment of the errors that propagate onto these derived products, estimates of the coverage times from an *in situ* sampling scheme, and an assessment of the statistical significance of the *in situ* sampling scheme. However, beyond their initial usage as a mission

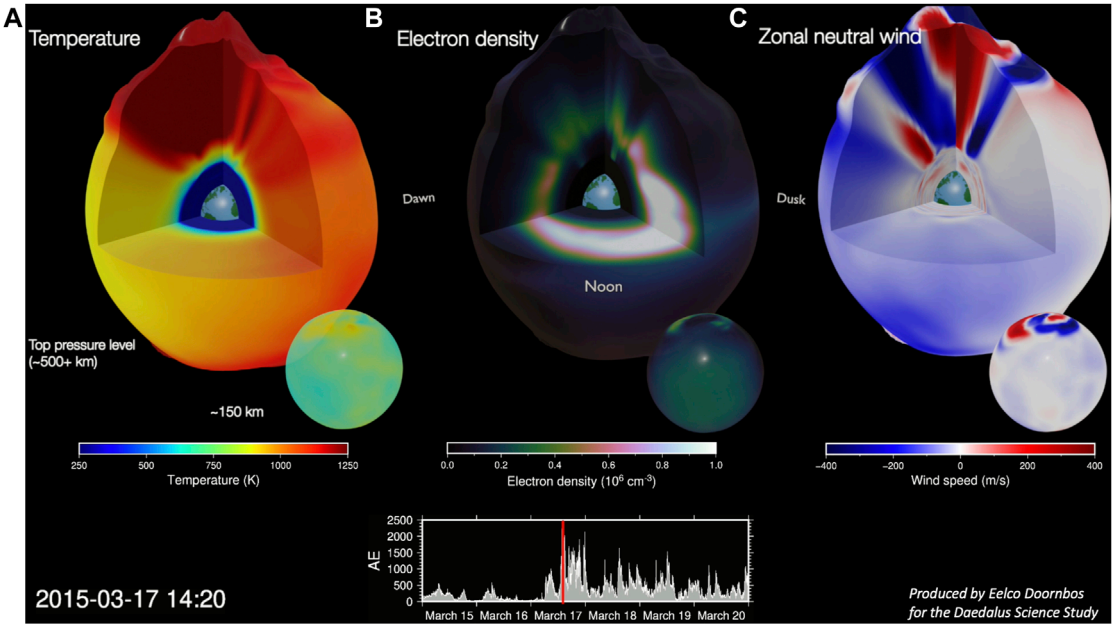


FIGURE 9
3D visualization of GCM model outputs: neutral temperature **(A)**, electron density **(B)** and zonal neutral wind **(C)**. The model output is from the 2015 St. Patrick's Day storm, showing the simulated state of the atmosphere on 17 March 2015, 14:20 UTC, during a period of significant forcing from above. Pressure level geopotential heights from the model have been exaggerated by 50 times to show vertical detail. Small spheres at the lower right side of each large sphere show the corresponding values at altitudes ~150 km.

simulator for the Daedalus mission, the modules comprising Daedalus MASE have evolved to include a number of tools that can be used for performing various scientific analyses of *in situ* spacecraft measurements of various observables, for processing and plotting gridded model datasets of Global Circulation Models of the Earth's thermosphere and ionosphere, and for performing comparisons between satellite data and models.

As an example, the Electric Field Instrument (EFI) onboard the Swarm satellites (Knudsen et al., 2017), which fly on near-polar, circular orbits with altitudes that range from 400 to 530 km, perform measurements of plasma densities and electron temperatures, both of which are also outputs of TIEGCM. By interpolating the TIEGCM gridded dataset along the Swarm orbits, the two time series can be inter-compared for similarities and for differences: the former allow using the larger picture, as captured by TIEGCM, to interpet the along-track observations, whereas the latter enable identifying missing physics and small-scale, sub-grid variability in the models. Similarly, neutral density data from the Swarm accelerometers (Siemes et al., 2016) as well as from Swarm GPS observations (van den Ijssel et al., 2020) can also be cross-compared with model data using Daedalus MASE. Furthermore, Daedalus MASE tools allow performing comparisons of statistical values as obtained from TIEGCM with the statistical ensemble of along-orbit data, such as those made by Swarm, gathered over longer time periods that enable classifications in areas of interest and as a function of geomagnetic conditions.

Similarly, the Ionospheric Connection Explorer (ICON) mission, which flies in a circular, low-inclination orbit at an altitude of 575 km, retrieves, among other parameters, *in situ* vertical ion drifts, the peak of ionospheric density, and altitude profiles of neutral temperature, wind vectors and densities of O and N_2. All the above quantities are output geophysical observables in TIEGCM, and Daedalus MASE allows their extraction and plotting, either along-track or as altitude profiles from TIEGCM, enabling their direct comparison. We note in particular the recent development of TIEGCM-ICON, driven by tidal perturbations derived from ICON observations and at high latitude by ion convection and auroral particle precipitation patterns from AMIE, the Assimilative Mapping of Ionospheric Electrodynamics model (Maute, 2017); Daedalus MASE routines are directly applicable to TIEGCM-ICON as well, assisting in comparative studies between ICON observations and TIEGCM-ICON model outputs. As further enhancements of Daedalus MASE, special pre-set routine settings can be devised that enable the extraction of, e.g., altitude profiles in the altitude ranges and vertical resolutions as described in the Level 2 data descriptions of ICON. In addition, the use of the modules described in **Sections 2.5**, **2.6** enables the calculation of statistical properties of long-term time series

and the estimation of their statistical significance in user-defined bins in terms of local time, latitude and geomagnetic activity.

In terms of processing of the outputs of GCMs (gridded datasets), potential scientific analyses enabled by Daedalus MASE include the calculation of various derived products based on geophysical quantities (or primary products) of GCMs on all grid-points of the GCM simulation, the production of maps, altitude profiles and latitude–local time cuts in GCMs, the conversion of geographic gridded points to magnetic coordinates, height integrations, and calculations of globally or hemispherically integrated quantities.

It is noted that the derived products calculated *via* Daedalus MASE are of prime interest in the E region and lower F region, as they relate to key unknown properties in these regions (see, e.g., Sarris. (2019); Palmroth et al. (2021)). It is also noted, however, that many of these quantities are not directly calculated in GCMs. Thus Daedalus MASE fills a gap by providing a comprehensive toolset for the direct calculation of each of these quantities. These can be readily integrated with existing codes, can be expanded to perform additional calculations and cross-comparisons, or can be used as a stand-alone toolset for the post-processing of GCM datasets, including plotting of GCM snapshots and movies in various forms, through 2D and 3D global maps and various altitude/latitude/longitude cuts.

The calculation of these derived products is accompanied by error propagation estimation routines, for the inclusion of measurement errors; errors can also be obtained through simplified parametric instrument simulators, or through analytic (e.g., altitude- or density-dependent) error models. Subsequently, derived products are computed, and their uncertainties are estimated through standard propagation of variance for the respective analytical formulae. Data from the solar cycle run of TIEGCM are available upon request (see data availability statement below).

These functionalities are potentially useful to the broader space physics community that use GCM outputs for LTI science. Applications are presented focusing on gridded datasets produced by NCAR's TIEGCM, however, the analysis tools of Daedalus MASE can be expanded to be applicable to other gridded geophysical observable data, that can be obtained from, e.g., the Whole Atmosphere Community Climate Model with thermosphere and ionosphere extension (WACCM-X), or the Global Ionosphere Thermosphere Model (GITM).

Further to the modules described herein, which enable various computations based on the gridded datasets, Daedalus MASE also includes additional code and modules that are specific to the Daedalus mission, and which are not described herein. These include, for example: an instrument-specific module for the calculation of the electric field that enables calculating the effect of spacecraft velocity on the resulting electric field and its dependence on the boom length; code enabling the reconstruction of pitch angle distributions of energetic particles based on a fixed number of particle detectors; simulations of the effect of energetic particles on the lower atmosphere. Furthermore, Vogt et al. (2022) expanded upon earlier work on the altitude reconstruction of primary and derived products based on multi-point measurements, addressing the question of how *in situ* measurements in the LTI can be extrapolated using parametric models of observables and derived variables: Using ensembles of model parameters, created by means of Monte Carlo simulations and synthetic measurements based on model predictions and relative uncertainties, they presented a methodology that allows for assessing how cost-critical elements of the Daedalus mission such as perigee and apogee distances affect the accuracy of the reconstruction of various ionospheric profiles.

The modules for the calculation of various derived products in Daedalus MASE are accompanied by Jupyter Notebooks that include the equations and code calculating these derived products, together with literature references. Most importantly, these Jupyter Notebooks include easy-to-use plotting functions, enabling 2D and 3D plotting on maps, altitude profiles and various user-defined cuts, enabling the quick processing and useability of GCM outputs. This is combined with having all relevant code in python, thus enabling the user to build upon the existing framework. Daedalus MASE also includes interpolation routines that can estimate primary and derived products at any location within a gridded dataset. This includes points along spacecraft and rocket orbits, altitude profiles, ground instrument lines of sight, *etc.*

Combining the above functionalities, together with derived product calculations and uncertainty estimations, Daedalus MASE enables performing coverage simulations for various assumed mission lifetimes and orbit options, in order to determine and demonstrate the statistical significance of the data collected and of representing global processes in the LTI *via in situ* measurements. This is of particular importance for a low-altitude mission such as Daedalus, where sampling the lowermost altitudes has significant implications for the mission duration, required propulsion and, overall, mission complexity, due to enhanced atmospheric drag. Thus, besides the proposed mission concept Daedalus, direct application of Daedalus MASE can be made for other *in situ* sampling schemes, such as the upcoming Geospace Dynamics Constellation (GDC), a mission concept that targets to study the coupling between the magnetosphere and the ionosphere/thermosphere (IT) system, and how that coupled system responds to external energy input. Daedalus MASE tools can also be applied

to rocket measurements: even though rockets often lack the comprehensive ensemble of all relevant instruments to measure ion-neutral interactions, Daedalus MASE allows substituting a measurement by modeled along-flight parameters, while also enabling sensitivity analyses of the effects of model and measurement errors on the calculated parameters.

Data availability statement

The datasets and source code presented in this study can be found in the following online repositories: The source code of the Daedalus MASE modules described in this study as well as sample datasets of TIEGCM outputs, geomagnetic indices and sample orbit files, can be downloaded from the Daedalus MASE repository, at link: https://github.com/DaedalusMASE/DaedalusMASE. The source code for the 3D visualizations of GCM outputs can be downloaded at: https://gitlab.com/eelcodoornbos/blender_gcm_slices. The names of the modules and subroutines in the above repositories can be found in the article/**Supplementary Material**.

Author contributions

TS coordinated the study and wrote the first draft. ST wrote **Sections 2.1**, **2.2**, **3.1**, **3.2** and corresponding code. PP wrote **Sections 2.3**, **3.3** and corresponding code. CHP contributed to the code discussed in **Sections 2.1** and **3.1**. PP and KP wrote **Sections 2.4**, **3.4** and corresponding code. DB wrote **Sections 2.5**, **2.6**, **3.5**, **3.6** and corresponding code. ED wrote the code described in **Sections 2.7**, **3.7**. TS, SB, ED, MC, ID, WM, DK, NO, OM, TOM, AM, MP, CS, NI, RP, TM-J, CS contributed to the conception and design of DaedalusMASE functionality as members of the Daedalus Mission Advisory Group. AH, DL, AS, MT contributed to the conception and design of Daedalus MASE functionality as ESA technical officers overlooking the Daedalus Science and Requirements Consolidation study. MC, PV, DM, NA, AK, WM, JV, AB, NS contributed to other parts of DaedalusMASE code, not described herein, and expanded upon functionality of DaedalusMASE. H-LL, GL, ST contributed to the TIEGCM model runs and model data collection. IS and COP contributed to the independent review and testing of the code. All authors contributed to the scientific analysis. All authors contributed to manuscript revision and approved the submitted version.

Funding

Funding for the development of the modules in DaedalusMASE was provided by the European Space Agency under contract number 4000127346 to the Democritus University of Thrace (study title: Daedalus Phase-0 Science and Requirements Consolidation Study). The research work by TS was supported by DUTH project KE82875. The research work by ST was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the HFRI PhD Fellowship grant (Fellowship Number: 400), and DUTH project KE82372.

Acknowledgments

Daedalus MASE was developed during the Phase 0 study of Daedalus, one of three candidate mission concepts for ESA's 10th Earth Explorer.

Conflict of interest

Authors IS and CP were employed by the company Space Applications Research Consultancy.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fspas.2022.1048318/full#supplementary-material

References

Archer, W. E., Knudsen, D., Burchill, J., Jackel, B., Donovan, E., Connors, M., et al. (2017). Birkeland current boundary flows. *J. Geophys. Res. Space Phys.* 122, 4617–4627. doi:10.1002/2016ja023789

Banks, P. (1966). Collision frequencies and energy transfer electrons. *Planet. Space Sci.* 14, 1085–1103. doi:10.1016/0032-0633(66)90024-9

Banks, P. M., and Kockarts, G. (1973). *Aeronomy*. New York, NY: Academic Press.

Bilitza, D. (2018). Iri the international standard for the ionosphere. *Adv. Radio Sci.* 16, 1–11. doi:10.5194/ars-16-1-2018

Billett, D., Grocott, A., Wild, J., Walach, M.-T., and Kosch, M. (2018). Diurnal variations in global joule heating morphology and magnitude due to neutral winds. *J. Geophys. Res. Space Phys.* 123, 2398–2411. doi:10.1002/2017ja025141

Cowley, S. (2000). Magnetosphere-ionosphere interactions: A tutorial review. *Magnetos. Curr. Syst. Geophys. Monogr. Ser.* 118, 91–106. doi:10.1029/GM118p0091

Crisp, N. H., Roberts, P. C., Livadiotti, S., Oiko, V. T. A., Edmondson, S., Haigh, S., et al. (2020). The benefits of very low Earth orbit for Earth observation missions. *Prog. Aerosp. Sci.* 117, 100619. doi:10.1016/j.paerosci.2020.100619

Dalgarno, A., Henry, R., and Stewart, A. (1964). The photoionization of atomic oxygen. *Planet. Space Sci.* 12, 235–246. doi:10.1016/0032-0633(64)90225-9

Deng, Y., Maute, A., Richmond, A. D., and Roble, R. G. (2009). Impact of electric field variability on joule heating and thermospheric temperature and density. *Geophys. Res. Lett.* 36, L08105. doi:10.1029/2008gl036916

Deng, Y., and Ridley, A. J. (2007). Possible reasons for underestimating joule heating in global models: E field variability, spatial resolution, and vertical velocity. *J. Geophys. Res.* 112, 12006. doi:10.1029/2006ja012006

Doornbos, E., Sarris, T., Tourgaidis, S., Pirnaris, P., Buchert, S., Liu, H., et al. (2020). "Visualizing models and observations of the thermosphere-ionosphere in support of the ESA EE10 candidate mission Daedalus," in *EGU general assembly 2020* (Vienna: European Geophysical Union), 7386. doi:10.5194/egusphere-egu2020-7386

Drob, D., Emmert, J., Crowley, G., Picone, J., Shepherd, G., Skinner, W., et al. (2008). An empirical model of the Earth's horizontal wind fields: HWM07. *J. Geophys. Res.* 113, 13668. doi:10.1029/2008ja013668

Drob, D. P., Emmert, J. T., Meriwether, J. W., Makela, J. J., Doornbos, E., Conde, M., et al. (2015). An update to the horizontal wind model (HWM): The quiet time thermosphere. *Earth Space Sci.* 2, 301–319. doi:10.1002/2014ea000089

Elvidge, S., Godinez, H. C., and Angling, M. J. (2016). Improved forecasting of thermospheric densities using multi-model ensembles. *Geosci. Model. Dev.* 9, 2279–2292. doi:10.5194/gmd-9-2279-2016

Emmert, J. T., Drob, D. P., Picone, J. M., Siskind, D. E., Jones, M.Jr, Mlynczak, M., et al. (2021). Nrlmsis 2.0: A whole-atmosphere empirical model of temperature and neutral species densities. *Earth Space Sci.* 8, e2020EA001321. doi:10.1029/2020ea001321

ESA (2020). "Report for assessment: Earth explorer 10 candidate mission Daedalus," in *European space agency. Tech. Rep.* (Noordwijk, Netherlands: European Space Agency).

Hedin, A. E. (1991). Extension of the msis thermosphere model into the middle and lower atmosphere. *J. Geophys. Res.* 96, 1159–1172. doi:10.1029/90ja02125

Hickman, A., Medikeri-Naphade, M., Chapin, C., and Huestis, D. (1997). Fine structure effects in the o+-o collision frequency. *Geophys. Res. Lett.* 24, 119–122. doi:10.1029/96gl03797

Ieda, A. (2020). Ion-neutral collision frequencies for calculating ionospheric conductivity. *J. Geophys. Res. Space Phys.* 125, e2019JA027128. doi:10.1029/2019ja027128

Karlsson, T., Andersson, L., Gillies, D., Lynch, K., Marghitu, O., Partamies, N., et al. (2020). Quiet, discrete auroral arcs–observations. *Space Sci. Rev.* 216, 16–50. doi:10.1007/s11214-020-0641-7

Killeen, T., Hays, P., Carignan, G., Heelis, R., Hanson, W., Spencer, N., et al. (1984). Ion-neutral coupling in the high-latitude F region: Evaluation of ion heating terms from dynamics explorer 2. *J. Geophys. Res.* 89, 7495–7508. doi:10.1029/ja089ia09p07495

Knudsen, D., Burchill, J., Buchert, S., Eriksson, A., Gill, R., Wahlund, J.-E., et al. (2017). Thermal ion imagers and Langmuir probes in the swarm electric field instruments. *J. Geophys. Res. Space Phys.* 122, 2655–2673. doi:10.1002/2016ja022571

Laundal, K. M., and Richmond, A. D. (2017). Magnetic coordinate systems. *Space Sci. Rev.* 206, 27–59. doi:10.1007/s11214-016-0275-y

Liu, H.-L., Foster, B., Hagan, M., McInerney, J., Maute, A., Qian, L., et al. (2010). Thermosphere extension of the whole atmosphere community climate model. *J. Geophys. Res.* 115, 15586. doi:10.1029/2010ja015586

Lu, G., Richmond, A., Emery, B., and Roble, R. (1995). Magnetosphere-ionosphere-thermosphere coupling: Effect of neutral winds on energy transfer and field-aligned current. *J. Geophys. Res.* 100, 19643. doi:10.1029/95ja00766

Maute, A. (2017). Thermosphere-ionosphere-electrodynamics general circulation model for the ionospheric connection explorer: Tiegcm-icon. *Space Sci. Rev.* 212, 523–551. doi:10.1007/s11214-017-0330-3

Moffett, R., Sellek, R., and Bailky, G. (1990). The influence of O+- O collision frequency on ionospheric F-region behaviour. *J. Atmos. Terr. Phys.* 52, 125–132. doi:10.1016/0021-9169(90)90075-x

Palmroth, M., Grandin, M., Sarris, T., Doornbos, E., Tourgaidis, S., Aikio, A., et al. (2021). Lower-thermosphere–ionosphere (lti) quantities: Current status of measuring techniques and models. *Ann. Geophys.* 39, 189–237. doi:10.5194/angeo-39-189-2021

Pesnell, W. D., Omidvar, K., and Hoegy, W. R. (1993). Momentum transfer collision frequency of O+-O. *Geophys. Res. Lett.* 20, 1343–1346. doi:10.1029/93gl01597

Picone, J., Hedin, A., Drob, D. P., and Aikin, A. (2002). Nrlmsise-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *J. Geophys. Res.* 107, 15–16. doi:10.1029/2002ja009430

Qian, L., Burns, A. G., Emery, B. A., Foster, B., Lu, G., Maute, A., et al. (2014). The NCAR TIE-GCM: A community model of the coupled thermosphere/ionosphere system. *Model. ionosphere–thermosphere Syst.* 2014, 73–83. doi:10.1002/9781118704417.ch7

Richmond, A. D. (2017). "Ionospheric electrodynamics," in *Handbook of atmospheric electrodynamics* (Boca Raton, Florida: CRC Press), 249–290.

Richmond, A. (1995). Ionospheric electrodynamics using magnetic apex coordinates. *J. Geomagn. Geoelec.* 47, 191–212. doi:10.5636/jgg.47.191

Richmond, A., and Thayer, J. (2000). Ionospheric electrodynamics: A tutorial. *Magnetos. Curr. Syst.* 118, 131–146. doi:10.1029/gm118p0131

Ridley, A., Deng, Y., and Toth, G. (2006). The global ionosphere–thermosphere model. *J. Atmos. Solar-Terrestrial Phys.* 68, 839–864. doi:10.1016/j.jastp.2006.01.008

Roble, R. G. (1988). Impact of uncertainties in the o+-o collisional cross-section on the global structure and dynamics of the thermosphere and ionosphere. *EOS Trans. AGU* 69, 417.

Salah, J. E. (1993). Interim standard for the ion-neutral atomic oxygen collision frequency. *Geophys. Res. Lett.* 20, 1543–1546. doi:10.1029/93gl01699

Sangalli, L., Knudsen, D., Larsen, M., Zhan, T., Pfaff, R., and Rowland, D. (2009). Rocket-based measurements of ion velocity, neutral wind, and electric field in the collisional transition region of the auroral ionosphere. *J. Geophys. Res.* 114, 13757. doi:10.1029/2008ja013757

Sarris, T. E., Talaat, E. R., Palmroth, M., Dandouras, I., Armandillo, E., Kervalishvili, G., et al. (2020). Daedalus: A low-flying spacecraft for *in situ* exploration of the lower thermosphere–ionosphere. *Geosci. Instrum. Method. Data Syst.* 9, 153–191. doi:10.5194/gi-9-153-2020

Sarris, T. E. (2019). Understanding the ionosphere thermosphere response to solar and magnetospheric drivers: Status, challenges and open issues. *Phil. Trans. R. Soc. A* 377, 20180101. doi:10.1098/rsta.2018.0101

Schunk, R., and Nagy, A. (2009). *Ionospheres: Physics, plasma physics, and chemistry*. Cambridge, England: Cambridge University Press.

Schunk, R., and Walker, J. (1973). Theoretical ion densities in the lower ionosphere. *Planet. Space Sci.* 21, 1875–1896. doi:10.1016/0032-0633(73)90118-9

Sheng, C., Deng, Y., Yue, X., and Huang, Y. (2014). Height-integrated pedersen conductivity in both E and F regions from COSMIC observations. *J. Atmos. Solar-Terrestrial Phys.* 115, 79–86. doi:10.1016/j.jastp.2013.12.013

Siemes, C., de Teixeira da Encarnacao, J., Doornbos, E., van den Ijssel, J., Kraus, J., Peresty, R., et al. (2016). Swarm accelerometer data processing from raw accelerations to thermospheric neutral densities. *Earth Planets Space* 68, 92. doi:10.1186/s40623-016-0474-5

Strangeway, R. J. (2012). The equivalence of joule dissipation and frictional heating in the collisional ionosphere. *J. Geophys. Res.* 117, 17302. doi:10.1029/2011ja017302

Stubbe, P. (1968). Frictional forces and collision frequencies between moving ion and neutral gases. *J. Atmos. Terr. Phys.* 30, 1965–1985. doi:10.1016/0021-9169(68)90004-4

Thayer, J. (1998). Height-resolved joule heating rates in the high-latitude E region and the influence of neutral winds. *J. Geophys. Res.* 103, 471–487. doi:10.1029/97ja02536

Thayer, J., and Vickrey, J. (1992). On the contribution of the thermospheric neutral wind to high-latitude energetics. *Geophys. Res. Lett.* 19, 265–268. doi:10.1029/91gl02868

van den Ijssel, J., Doornbos, E., Iorfida, E., March, G., Siemes, C., and Montenbruck, O. (2020). Thermosphere densities derived from swarm gps observations. *Adv. Space Res.* 65, 1758–1771. doi:10.1016/j.asr.2020.01.004

Vogt, J., Marghitu, O., Blagau, A., Pick, L., Stachlys, N., Buchert, S., et al. (2022). Daedalus ionospheric profile continuation (DIPCont). *Geoscientific Instrum. Methods Data Syst. Discuss.* 1–28, 12. doi:10.5194/gi-2022-12

Weimer, D. (2005). Improved ionospheric electrodynamic models and application to calculating joule heating rates. *J. Geophys. Res.* 110, A05306. doi:10.1029/2004ja010884

Copyright © 2023 Sarris, Tourgaidis, Pirnaris, Baloukidis, Papadakis, Psychalas, Buchert, Doornbos, Clilverd, Verronen, Malaspina, Ahmadi, Dandouras, Kotova, Miloch, Knudsen, Olsen, Marghitu, Matsuo, Lu, Marchaudon, Hoffmann, Lajas, Strømme, Taylor, Aikio, Palmroth, Heelis, Ivchenko, Stolle, Kervalishvili, M. Jørgensen, Pfaff, Siemes, Visser, van den Ijssel, Liu, Sandberg, Papadimitriou, Vogt, Blagau and Stachlys. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 24 January 2023
DOI 10.3389/fspas.2023.1082150

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Brian Harding,
University of California, Berkeley, United States
Alla V. Suvorova,
National Central University, Taiwan

*CORRESPONDENCE
Gonzalo Cucho-Padin,
gonzaloaugusto.cuchopadin@nasa.gov

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 27 October 2022
ACCEPTED 11 January 2023
PUBLISHED 24 January 2023

CITATION
Cucho-Padin G, Bhattacharyya D, Sibeck DG, Connor H, Youngblood A and Ardila D (2023), EXOSpy: A python package to investigate the terrestrial exosphere and its FUV emission.
Front. Astron. Space Sci. 10:1082150.
doi: 10.3389/fspas.2023.1082150

COPYRIGHT
© 2023 Cucho-Padin, Bhattacharyya, Sibeck, Connor, Youngblood and Ardila. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

EXOSpy: A python package to investigate the terrestrial exosphere and its FUV emission

Gonzalo Cucho-Padin[1,2]*, Dolon Bhattacharyya[3], David G. Sibeck[1], Hyunju Connor[4], Allison Youngblood[5] and David Ardila[6]

[1]Space Weather Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, United states, [2]Department of Physics, The Catholic University of America, Washington DC, MD, United states, [3]Laboratory of Atmospheric and Space Physics, University of Colorado, Boulder, CO, United states, [4]Geospace Physics Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, United states, [5]Exoplanets and Astrophysics Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, United states, [6]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, United states

The exosphere is the uppermost layer of the terrestrial atmosphere, mainly composed of atomic hydrogen (H) that resonantly scatters solar far-ultraviolet (FUV) photons at 121.56 nm, also referred to as Lyman-Alpha (Ly-α) emission. Analysis of this emission has been used to determine the global, three-dimensional, and time-dependent exospheric H density structure, which is essential to assess the permanent escape of H to space as well as to determine their role in governing the transient response of terrestrial plasma environment to space weather. Thus, Ly-α emission and its by-product, the H density, are highly desirable to the magnetospheric community. On the other hand, this emission can also be regarded as a significant source of contamination during studies of FUV targets such as O/B-type stars, planetary and exoplanetary atmospheres, and the circumgalactic medium, especially when observations are acquired from Earth-orbiting instruments. In this case, accurate specification of exospheric Ly-α photon flux and its subsequent removal is required by the planetary and astrophysics community studying solar/extra-solar system objects. This work introduces EXOSpy, an open-source python-based package that provides several models of terrestrial exospheric H density and calculates exospheric Ly-α emission with a high potential to contribute to investigations in both communities. We present several examples to demonstrate how EXOSpy can be used to (i) validate current and new exospheric models based on actual Ly-α radiance data, (ii) estimate exospheric contamination for a given instrument's line-of-sight and spatial location, and (iii) provide support for new space-based FUV instrument design.

KEYWORDS

terrestrial exosphere, FUV, lyman-alpha, atomic hydrogen, python

1 Introduction

In 1972, the Apollo 16 mission retrieved the first wide-field far-ultraviolet (FUV) imagery of the earth and revealed the vast extent of the terrestrial exosphere, also known as the "geocorona" (Carruthers et al., 1976). This outermost atmospheric layer extends from several hundreds of kilometers above the earth's surface (~500 km) out to several tens of earth radii ($\sim$ 60 R_E) (Baliukin et al., 2019). The most dominant component in this region is atomic hydrogen (H) that resonantly scatters FUV emission centered at the 121.56 nm wavelength (Ly-α). Furthermore, the exosphere is embedded within the many distinct plasma environments

spanning the topside ionosphere, plasmasphere, ring-current system, and solar wind. Charge exchange interactions between exospheric H atoms and ambient ions are well known to play a crucial role in governing the earth's transient response to space weather, both by dissipating magnetospheric ring current energy through the generation of energetic neutral atoms (ENAs) and by driving the ion transport responsible for plasmaspheric refilling after geomagnetic storms (Ilie et al., 2012; Krall et al., 2018; Cucho-Padin et al., 2020). Due to its relevance in magnetospheric physics, efforts have been made to estimate three-dimensional (3-D), time-dependent, and global H density distributions using sophisticated remote sensing techniques based on measurements of the terrestrial Ly-α emission (Bailey and Gruntman, 2011; Zoennchen et al., 2015; Cucho-Padin and Waldrop, 2018; 2019).

The planetary and astrophysics community also requires observations of Ly-α emission from extra-solar targets to understand stellar chromospheres and transition regions, model photo-chemistry in planetary/exoplanetary atmospheres, and measure the abundance of neutral hydrogen in the Inter-stellar medium (ISM) (Youngblood et al., 2016). However, such observations carried out from low-Earth orbits (LEO) and usually with optical spectrometers are significantly contaminated by the bright geocoronal emission, e.g., $\sim >$ 10 kR for a dayside limb observation at 600 km altitude (kR = kilo Rayleigh, 1 Rayleigh = $10^6/4\pi$ photons/sec/str/cm^2). Also, the terrestrial exosphere is dynamic, and its spatial density distribution responds to seasonal variations, solar cycles, and active geomagnetic conditions, ultimately resulting in variations of Ly-α radiance of $\sim >$ 500 R (Waldrop and Paxton, 2013; Cucho-Padin and Waldrop, 2019) that might be of similar magnitude to the emission from extra-terrestrial objects. These conditions motivate planetary scientists and astrophysicists to pursue sophisticated techniques to avoid and/or subtract the terrestrial Ly-α emission from their measurements. For example, Youngblood et al. (2016) conducted a stellar characterization study using observations from the Cosmic Origins Spectrograph (COS) onboard the Hubble Space Telescope (HST). To avoid geocoronal contamination, measurements were restricted to happen during the time of the year when the earth's heliocentric velocity would allow the geocoronal Ly-α emission to coincide with the attenuated stellar Ly-α line core. In other words, due to a Doppler effect, the Ly-α line of the target is not located exactly at 121.56 nm but red- or blue-shifted depending on velocity direction. Then, the geocoronal emission line was directly subtracted using sky background measurements. Another technique includes fitting geocoronal measurements to a Gaussian function (Wood et al., 2005) or other airglow templates (Bourrier, V. et al., 2018) to estimate the line-integrated intensity and subtract it off from the measurements.

A similar approach used by Bhattacharyya et al. (2017) to characterize the spatial distribution of H in the Martian exosphere using the Advance Camera for Survey (ACS) instrument onboard HST, is to observe Mars when the geocoronal Ly-α is sufficiently doppler-shifted with respect to the Mars line. This ensures that the geocoronal contamination can be subtracted off from the total observed intensity without dealing with complicated radiative transfer effects. However, an additional HST orbit is required to measure the background emission intensity since ACS is a broad-band UV imager.

It is evident to the UV astronomy community in general that an accurate specification of the terrestrial exospheric Ly-α emission is of utmost importance not only to increase the frequency and precision of target observations under less-restrictive conditions but also to increase the efficiency of space-based asset utilization to allow for better characterization of the target properties with more accurate estimations of the background contamination. In this work, we have developed a python-based package EXOSpy that brings together state-of-the-art and publicly available H exospheric models to be accessible to the scientific community through a set of simple functions. Furthermore, EXOSpy enables the user to calculate exospheric emissions that serve (i) to validate exospheric models with actual Ly-α radiance data, (ii) to estimate exospheric contamination that may affect extra-terrestrial observations, and (iii) to support UV instrument design. In this manuscript, we outline EXOSpy's architecture and highlight its key capabilities. In **Section 2**, we describe the main features of this package, such as the exospheric models included in it and the core operations to estimate Ly-α radiance. In **Section 3**, we show practical applications of EXOSpy. Finally, in **Section 4** we summarize the conclusions of this project and present future directions to improve this package.

2 Description of the EXOSpy python-based package

EXOSpy is an open-source python package that allows the user to explore several H exospheric models and generate column-integrated Ly-α intensities for single-pixel photometers or wide field-of-view (FOV) imagers that could potentially facilitate mission design efforts as well as aid the removal of exospheric background emission.

2.1 Terrestrial exospheric models

EXOSpy includes physics- and data-based exospheric models described in this section. We refer the reader to the cited references below as well as the EXOSpy's website https://exospy.readthedocs.io/ for further details.

Chamberlain (1963) developed a physics-based exospheric model to determine density distribution of species at altitudes where collisions between them can be neglected. Here, H atoms are assumed to be thermalized through neutral-neutral collisions at the boundary of the collisional and collisionless atmosphere, termed as the exobase (~500 km altitude), and then adopt ballistic or escaping trajectories to space. The H density distribution model is reported as an analytical altitude-dependent function that only requires values of H density (n_{exo}) and neutral temperature (T_{exo}) at the exobase, both of which can be specified by external models, such as the well-known model NRLMSIS2.0 (Emmert et al., 2021).

Bailey and Gruntman (2011) developed an exospheric model based on parametric fitting to a spherical harmonic function using single-day measurements of scattered Ly-α emission acquired by the Lyman-Alpha Detector (LAD) onboard the Two-Wide angle Imaging Neutral-atom Spectrometers (TWINS) mission. The approach used in this study leverages the linearity between the H density and its emission that occurs when the medium is optically thin, i.e., the density is low enough such that each H atom scatters a solar Ly-α photon only once. At earth, the exospheric optically thin region is located beyond 3 R_E geocentric distance. In addition, the TWINS/LAD apogee of $\sim$ 7.2 R_E and its nadir viewing geometry

TABLE 1 Data-based exospheric models included in the EXOSpy package. Solar radio flux at 10.7 cm (F10.7) and the geomagnetic disturbance index (DST) provide additional information regarding solar and geomagnetic conditions during the Ly-α data acquisition, which was used to generate the model. Values of F10.7 and DST are expressed as a day or multi-day averages. The reader is referred to the references for further details. DST values were obtained from the World Data Center for Geomagnetism, Kyoto. F10.7 values were obtained from the LASP Interactive Solar Irradiance Data Center, Colorado, United States.

References of the exospheric model	Instrument(s)	Dimension, range of validity [R_E]	EXOSpy alias	Avg. F10.7 [s.f.u.]	Avg. DST [nT]
Bailey and Gruntman, (2011)	TWINS/LAD	3-D, [3–8]	B11	67.8	+4.00
Zoennchen et al. (2015)	TWINS/LAD	3-D, [3–8]	Z15MAX, Z15MIN	69.57, 108.5	−4.99, −7.21
Connor and Carter, (2019)	XMM-Newton	1-D, [3–12]	C19O01, C19M03	205.8, 144.4	−5.08, −11.17
Zoennchen et al. (2022)	UVIS/HDAC	1-D, [3–15]	Z21	133.9	−27.87
Jung et al. (2022)	XMM-Newton	1-D, [3–12]	J22	69.5	−7.87
Cucho-Padin et al. (2022b)	LAICA	3-D, [6–20]	C22	146.2	−35.12

allows only partial observation of the exosphere, i.e., a single LAD line-of-sight (LOS) acquires emission starting from a radial position ~7.2R_E towards infinity on the opposite side, but it cannot observe emission behind the instrument where there is still significant exospheric density with approximate values of 10~100 cm^{-3} (Cucho-Padin et al., 2022b). Thus, exospheric density reconstructions using TWINS/LAD Ly-α data typically set 8R_E as an upper boundary since the estimation of the region behind the sensor may feature high uncertainty. Hence, retrieved 3-D, global H density distributions for this model are only valid from the 3 to 8 R_E region. Zoennchen et al. (2015) used a similar approach using multi-day data acquired by LAD/TWINS to generate a model for solar minimum conditions (with data acquired in 2008 and 2010) and solar maximum conditions (with data acquired in 2012). The region of validity for these two models is also restricted to a region from 3 to 8 R_E. Furthermore, Zoennchen et al. (2022) estimated an H density radial profile using Ly-α measurements from the Ultraviolet Imaging Spectrograph (UVIS) onboard the CASSINI spacecraft during an earth flyby on 18 August 1999, on its way to Saturn (Esposito et al., 2004; Werner et al., 2004). These radiance data were fitted to a spherical function to yield an exospheric profile valid from 3 to 15 R_E. Recently, Cucho-Padin et al. (2022b) derived a 3-D model of the outer exosphere using a single global image acquired by the Lyman-Alpha Imager CAmera (LAICA) onboard the Proximate Object Close Flyby with Optical Navigation (PROCYON) spacecraft when it was located at ~2000 R_E from earth. The H density model is reported as spherical harmonics coefficients with a valid region that ranges from 6 to 15 R_E.

Estimation of neutral densities is also possible through remote sensing of soft X-ray emissions, which are generated by the charge exchange interaction between heavy charged solar wind particles (e.g., O^{+7} and O^{+8}) and exospheric neutral atoms. Connor and Carter (2019) estimate H density in the vicinity of the terrestrial magnetosheath, i.e., the subsolar point at ~10 R_E, using simultaneous measurements of soft X-ray radiance acquired by the XMM-Newton mission and *in situ* ion density, acquired by Cluster and Geotail missions. The H density for two events during solar maximum conditions (October 2001 and May 2003) are reported as a spherical function. Following a similar methodology, Jung et al. (2022) used XMM-Newton radiance data and THEMIS ion density data to determine an H density profile during solar minimum conditions (November 2008) and it is also reported as a spherical function. Since both studies utilized data from magnetosheath (located within the optically thin exospheric region), we consider the valid region of their models to range from 3 R_E to ~12 R_E, where the bow shock (the outer boundary of magnetosheath) is typically located (Fairfield, 1971). **Table 1** provides a summary of data-based models included in this package.

EXOSpy provides the user with functions to generate and visualize H density distributions. For example, the function draw1DHmodel receives a 1-D model alias (see **Table 1**) and an altitude range (in units of R_E) as inputs to produce the corresponding H density profile, as a numpy array, as well as a density vs altitude graph. Similarly, the function draw3DHmodel generates a 3-D H density profile and plots for the meridional and equatorial planes and spherical shell for a given geocentric radius. In **Figure 1**, panels (A) (B) and (C) show plots for the 3-D H density distribution model derived by Zoennchen et al. (2015) for solar maximum conditions (EXOSpy's alias Z15MAX). Specifically, panel (A) shows the equatorial plane, panel (B) shows a meridional (pole-to-pole) plane containing the subsolar point, and panel (C) shows a spherical shell at a geocentric radius of 5 R_E. In addition, panel (D) shows a comparison between 1-D H density profiles along the Sun-Earth line generated with the models included in the package. To determine the spatial distribution of densities, EXOSpy uses the Geocentric Solar Ecliptic (GSE) coordinate system, in which the x-axis points sunwards from earth's center, the z-axis points to the ecliptic north pole, and the y-axis completes the orthogonal triad. Also, we implemented a python notebook (Example1. ipynb located in the project's github) to show the use of EXOSpy's functions to create all graphs in **Figure 1**. The reader can access the file through the link https://github.com/gcucho/EXOSpy/blob/main/Examples/Example1.ipynb.

2.2 Calculation of column-integrated lyman-alpha emission

Estimation of the column-integrated line of sight Ly-α intensity ($I(\hat{n})$) observed by a photometric instrument (e.g., a single-pixel photometer) requires information of its spatial position, the direction of its line-of-sight (LOS) vector and the exospheric volumetric emission rate (ε) which is generated by the interaction of solar Ly-α photons and atmospheric particles such as atomic H and molecular Oxygen (O_2).

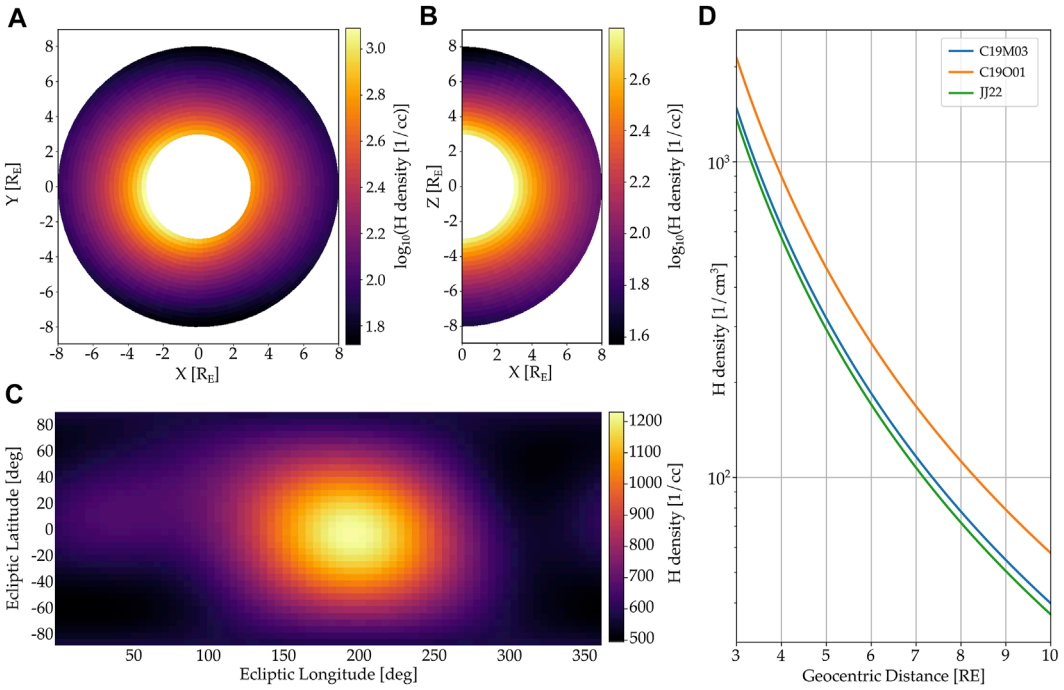


FIGURE 1
Visualization of H exospheric models included in the EXOSpy package. Panel **(A)** shows the equatorial plane of the 3-D H density profile derived by Zoennchen et al. (2015) for solar maximum conditions (Z15MAX). Panel **(B)** depicts the meridional (pole-to-pole) plane at the ecliptic longitude $\phi = 0°$ (noon). Panel **(C)** shows a spherical shell of H density at $r = 5R_E$. In addition, panel **(D)** shows a comparison between three 1-D H density models included in this package along the Sun-Earth line. All the plots use the Geocentric Solar Ecliptic (GSE) coordinate system, and radial distances are expressed in units of R_E.

To calculate $I(\hat{n})$ within the terrestrial exosphere, EXOSpy solves the following equation (Bhattacharyya et al., 2017):

$$I(\hat{n}) = \frac{\Psi(\hat{n})}{10^6}\int_0^{L_{max}} \Big(\varepsilon_0(l)\,T(\tau_{\mathrm{H}}(l))\,e^{-\tau_{\mathrm{O_2}}(l)} + \varepsilon_m(l)\,T(\tau_{\mathrm{H}}(l))\,e^{-\tau_{\mathrm{O_2}}(l)}\Big)\,dl + I_{IP}(\hat{n}), \quad (1)$$

where $\hat{n}$ represents the 3-D direction of the instrument LOS, the terms $\varepsilon_0(l)$ and $\varepsilon_m(l)$ are the volume emission rates due to single and multiple scattering of Ly-α photons by H atoms given in units of photons/cm^3/sec, the terms $\tau_{\mathrm{H}}(l) = \sigma_H \int_0^L n_H(l)dl$ and $\tau_{\mathrm{O_2}}(l) = \sigma_{\mathrm{O_2}} \int_0^L n_{\mathrm{O_2}}(l)dl$ are the optical depth of the medium due to H atoms and O_2 molecules, σ_H represents the Ly-α photon scattering cross-section and $\sigma_{\mathrm{O_2}}$ represents the Ly-α photon absorption cross-section by O_2. The line integral is evaluated from the beginning of the LOS ($l = 0$) to an appropriate upper boundary with a significant density of Ly-α emitters ($l = L_{\max}$). Also, $T(\tau)$ is the Holstein function (Holstein, 1947) that estimates the photon transmission probability in a medium with optical depth $\tau(l)$ and is given by the relation

$$T(\tau) = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty} e^{-x^2} e^{\tau e^{-x^2}}\,dx, \quad (2)$$

In addition, the term $\Psi(\hat{n})$ is the scattering function that estimates the probability of photon scattering in the direction $\hat{n}$ and is formulated as

$$\Psi(\hat{n}) = \frac{11}{12} + \frac{1}{4}\cos^2(\theta(\hat{n})), \quad (3)$$

where $\theta(\hat{n})$ is the angle between $\hat{n}$ and the $+X_{GSE}$ axis (Brandt and Chamberlain, 1959). The term $I_{IP}(\hat{n})$ is the interplanetary Ly-α background, in units of Rayleigh, emitted by atomic H of interstellar origin.

The use of **Eq. 1** requires volume emission rates for single and multiple scattering that can be estimated by a radiative transfer (RT) model. An RT model computes the single and multiple resonance scattering of photons by atmospheric particles and, for this purpose, requires solar photon flux and the cross-section of the particle-photon interaction as inputs to the model. For the analysis of the exosphere, the RT model uses three inputs: an H density profile (1-D for spherically symmetric distribution) to estimate the Ly-α photon scattering rate, an O_2 density profile to estimate the Ly-α photon absorption rate, and the incoming solar Ly-α flux usually expressed as irradiance at 121.56 nm in units of W/m^2. As a result, the RT model provides a spatial distribution of volume emission rates.

At high altitudes, where molecular oxygen is not found anymore ($\tau_{\mathrm{O_2}} = 0$, $e^{-\tau_{\mathrm{O_2}}} = 1$), and the medium is optically thin ($\tau_{\mathrm{H}} \ll 1$, $T(\tau_{\mathrm{H}}) = 1$), i.e., only single scattering of solar Ly-α occurs ($\varepsilon_m = 0$ and $\varepsilon_0 = g^* \times n_{\mathrm{H}}$), **Eq. 1** can be simplified as

$$I(\hat{n}) = \frac{g^*\Psi(\hat{n})}{10^6}\int_0^{L_{max}} n_H(l)\,dl + I_{IP}(\hat{n}) \quad (4)$$

where $n_{\mathrm{H}}(l)$ is the atomic H density in units of 1/cm^3 and g^* (also referred to as g-factor) is the scattering rate in units of photons/sec given by the relation

$$g^* = f_0 \sigma_H \delta v_D \sqrt{\pi} \quad (5)$$

where f_0 represents the incoming line center solar Ly-α flux in photons/cm^2/sec/Å, σ_H is the cross-section of the H Ly-α scattering

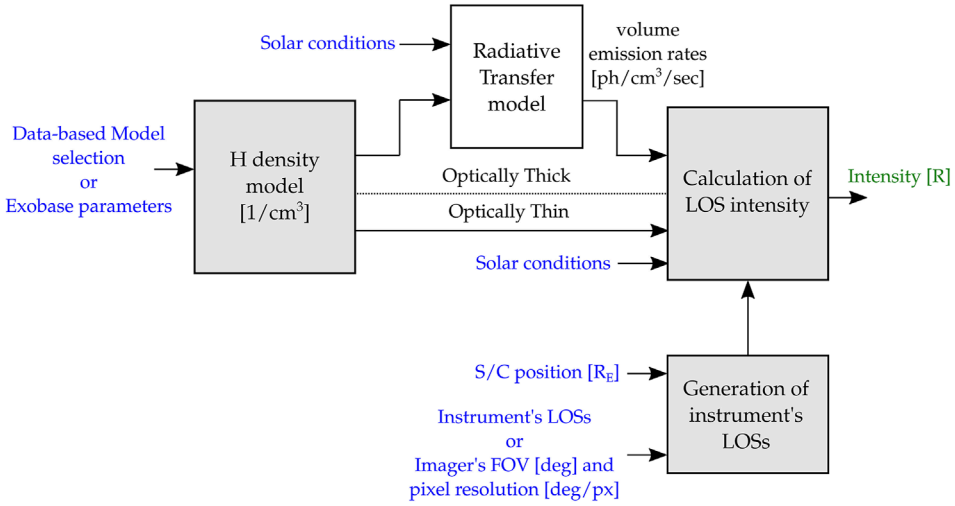


FIGURE 2
EXOSpy architecture to calculate column-integrated Ly-α intensities. Blue labels indicate user inputs, gray blocks are coded modules in EXOSpy and the white block indicate the module to be included in a future release of the package. Estimation of intensities for the optically thick region requires the use of a radiative transfer model which is not included in this version of EXOSpy; however, we provide a set of volume emission rate profiles for several solar conditions needed to estimate Ly-α intensities (see text for further details). EXOSpy provides a function to generate a synthetic imager when the FOV and pixel resolution are provided. Intensities values are generated in units of Rayleighs.

cross section at line center, and δv_D is the Doppler width of the Ly-α emission line. **Eq. 4** is only valid for the optically thin region of the exosphere, i.e., altitudes beyond 3 R_E geocentric distance.

Figure 2 depicts a block diagram that explains the steps to calculate line of sight column-integrated Ly-α intensities using EXOSpy. If the user selects a physics-based exospheric model that extends from 500 km altitude to 10 R_E geocentric distance, **Eq. 1** is used to calculate intensities. As mentioned before, it requires volume emission rates for single and multiple scattering obtained from an RT model. This current version of EXOSpy does not include an RT model (white box in **Figure 2**), but it is considered to be part of a future release. To overcome this problem, we have included seven files with distributions of volume emission rates (ε_0 and ε_m) generated with Chamberlain-based H density profiles (see **Section 2.1**) and exobase parameters obtained from the NRLMSIS2.0 model corresponding to the equinox (March 20) from 2014 to 2020. This period represents solar maximum to solar minimum variations. In addition, O_2 densities, which typically extend up to $\sim$ 800 km altitude due to their smaller scale height in comparison to H, are also obtained from the NRLMSIS2.0 model. Solar irradiance data is included in each file and are retrieved from the LASP Interactive Solar Irradiance Datacenter (LISIRD) (https://lasp.colorado.edu/lisird/data/composite_lyman_alpha/). EXOSpy's function Chamberlain implements the physics-based, 1-D Chamberlain model, which was used to generate the volume emission rates included in the package, and the function CalculateLOSfromVER implements Eq. **1** without considering Interplanetary Ly-α background.

On the other hand, if the user selects a data-based exospheric model, **Eq. 4** is used to calculate Ly-α intensity since these models are only valid for the optically thin region of the exosphere (see **Table 1**). In this case, the user should provide solar conditions as irradiance data which can be retrieved from LASP/LISIRD. EXOSpy's function GenerateIntensityOpticallyThin implements Eq. **4** without considering Interplanetary Ly-α background.

Then, the user should provide a set of spacecraft positions and LOS directions in GSE coordinates to calculate the line of sight column-integrated Ly-α intensities. Also, EXOSpy allows the user to create a synthetic image comprised of individual LOSs when a FOV (in units of degrees) and a pixel resolution (in units of degrees/pixel) are provided.

Hence, EXOSpy calculates only the exospheric Ly-α emission which is the first term in the right hand side part of **Eqs. 1**, **4**. Interplanetary Ly-α background (I_{IP}) can be retrieved from physics-based models (e.g. (Pryor et al., 1992)) or data provided by the Solar Wind ANisotropy (SWAN) instrument onboard the SOlar Heliospheric Observatory (SOHO) mission. The SOHO/SWAN instrument is located at Earth-Sun L1 point (Lagragian point), observing the interplanetary medium in Ly-α using a hydrogen cell. Its main product is a 1-day average all-sky map of the interplanetary Ly-α background.

2.3 Technical details of the EXOSpy package

The source code of EXOSpy is located in the following Github repository: https://github.com/gcucho/EXOSpy/. Also, a release of the package has been uploaded to a Zenodo repository and the download link can be found in (Cucho-Padin et al., 2022).

Furthermore, the package is part of the Python Package Index (PyPi) repository and its latest version (v2.3) can be installed to a personal computer using the command: pip install EXOSpy==2.3. For this, the user is required to have python 3.x and the pip application. Additionally, the following packages are needed: NumPy, SciPy, scikit-learn, and they will be installed automatically if they are not present in the user's computer.

In this work, we have developed four examples to show how to use EXOSpy. These examples are available within the source code and their locations are indicated in the manuscript. The user can also go to our

project's website https://exospy.readthedocs.io/ for more information regarding these examples.

3 Applications

3.1 Validation of H exospheric models

Current and future models of exospheric density distributions can be assessed using EXOSpy and Ly-α radiance data from several space-based instruments such as the Geocorona Imager (GCI) onboard the Imager for Magnetopause-to-Aurora Global Exploration (IMAGE) mission, the Global UltraViolet Imager (GUVI) onboard the Thermosphere, Ionosphere, Mesosphere Energetics and Dynamics (TIMED) mission, and the LADs onboard the TWINS mission, among others.

We propose a validation methodology that is comprised of four steps: (i) select a Ly-α instrument and identify its orbit and instrument LOS(s), (ii) identify the exospheric region (optically thin or optically thick) observed by the instrument, (iii) set the testing exospheric model and calculate column-integrated Ly-α intensities using EXOSpy's functions, and (iv) calculate the model performance. To demonstrate our approach, we show an example using TWINS/LAD measurements (step i). NASA's TWINS mission is comprised of two satellites (TWINS1 and TWINS2) with a highly elliptical orbit, an apogee of 7.2 R_E, orbital precession of ≈1 year, and a 90° longitudinal separation between satellites to allow for stereoscopic observations. Each TWINS spacecraft has two LADs (single-pixel photometers) that rotate 180° around a constant nadir pointing vector (McComas et al., 2009). TWINS radiance and ephemeris data are publicly available in NASA's Space Physics Data Facility (SPDF) repository. Panel (A) in **Figure 3** shows TWINS1/LAD1 observations for three rotations acquired on 11 June 2008, 08:20 UT. These observations have been pre-processed to remove stellar contamination and interplanetary Ly-α background as per the procedure detailed in (Cucho-Padin and Waldrop, 2018). The black line indicates the TWINS1 orbit, the green lines are LAD1's LOSs, and the blue arrow indicates the initial clockwise rotation of the LAD instrument around the nadir vector. Due to the LOSs viewing geometry, these observations serve to evaluate exospheric models beyond 3 R_E geocentric distance, i.e., optically thin region (step ii). EXOSpy enables the user to input a new exospheric model as spherical harmonic coefficients or as discrete 3-D spatial locations in GSE spherical coordinates using the class exosgrid. In this example, we use the model developed by Zoennchen et al. (2015) for solar minimum conditions, which is included in the package (Z15MIN). Then, we estimated the line of sight column-integrated Ly-α intensities using EXOSpy's function *GenerateIntensityOpticallyThin*. Panel (B) in **Figure 3** shows a comparison between the calculated intensities based on Z15MIN model in blue line (step iii) and the actual data measured by the TWINS1/LAD1 instrument on 19 June 2008, in the dashed red line. Finally, these values can be evaluated using several performance indicators such as minimum square error or the Pearson correlation coefficient, among others (step iv). Panel (B) shows an evident disagreement between TWINS/LAD observations and calculated intensities in the minima regions that are possibly associated with the multi-day approach used to generate this exospheric model. Specifically, Z15MIN was implemented using Ly-α observations during the periods June 19–21, 2008, and June 19–21, 2010. The intensities shown in the blue line were obtained using Eq. **4** and only considering solar flux for 19 June 2008 (used in Eq. **5**). Hence, the exospheric model can reflect the average behavior of density distributions for several days in two different years and may exclude the formation of daily structural patterns in the density distributions. Nevertheless, further analysis should be done to draw firm conclusions regarding the validity of this model.

The python notebook Example2. ipynb uploads the pre-processed ephemeris and attitude data from TWINS1/LAD1 and shows the use of EXOSpy's functions to generate column-integrated Ly-α intensities. The reader can access the file in https://github.com/gcucho/EXOSpy/blob/main/Examples/Example2.ipynb.

3.2 Estimation of exospheric contamination in current space-based UV instruments

Observations of extra-terrestrial bodies in UV might be contaminated by the bright exospheric emission, and its removal or avoidance is crucial for the correct analysis of the measurements.

To demonstrate how EXOSpy may help quantify this background emission, we estimate the Ly-α intensities as seen from a Low earth Orbit (LEO) satellite. Specifically, we simulate the orbit and viewing geometry of the TIMED/GUVI instrument, which are similar to those of the HST mission. Panel (A) in **Figure 4** shows in blue line the orbit of the satellite within the XY ecliptic plane at an altitude of 600 km (optically thick region). Red dots indicate positions when the spacecraft acquires data, and green lines are examples of measurements' LOSs only for two positions for better visualization. TIMED/GUVI is a spectrograph with a rotating mirror that allows LOS observations of the earth's limb from look angles (LA) that range from 0 to 20°. A LOS has LA = 0° when its direction is perpendicular to the zenith and LA >0 degrees when its direction is closer to the earth. The satellite location is represented by the angle between $+X_{GSE}$ axis and the zenith, also referred to as the solar zenith angle (SZA). For each satellite position (red dot), we simulated LOSs covering 20° LA with a 1-degree resolution. Then, we calculate the column-integrated Ly-α intensity using EXOSpy's function CalculateLOSfromVER as well as the volume emission rates for the optically thick region during solar minimum conditions (year 2020).

Panel (B) in **Figure 4** shows the resulting Ly-α intensities. Two typical features of exospheric emission are shown in this plot. First, the Ly-α intensity is reduced as the satellite transits from dayside (SZA = 0°) to dusk (SZA = 90°) due to the thicker atmosphere the solar Ly-α flux encounters on its path, ultimately resulting in fewer photons reaching the dusk/nightside region. Second, the presence of denser molecular oxygen at low altitudes absorbs Ly-α photons such that LOSs with smaller tangential distances may produce lower intensities. For instance, for a satellite position SZA = 0°, a LOS with LA = 0° yields ∼ 10 kR while a LOS with LA = 20° yields ∼ 8 kR since its path crosses lower down in the atmosphere, which has a larger amount of O_2 molecules. The python notebook Example3. ipynb shows the use of EXOSpy's functions to generate TIMED/GUVI's LOSs and their column-integrated Ly-α intensities. The reader can access the file in https://github.com/gcucho/EXOSpy/blob/main/Examples/Example3.ipynb.

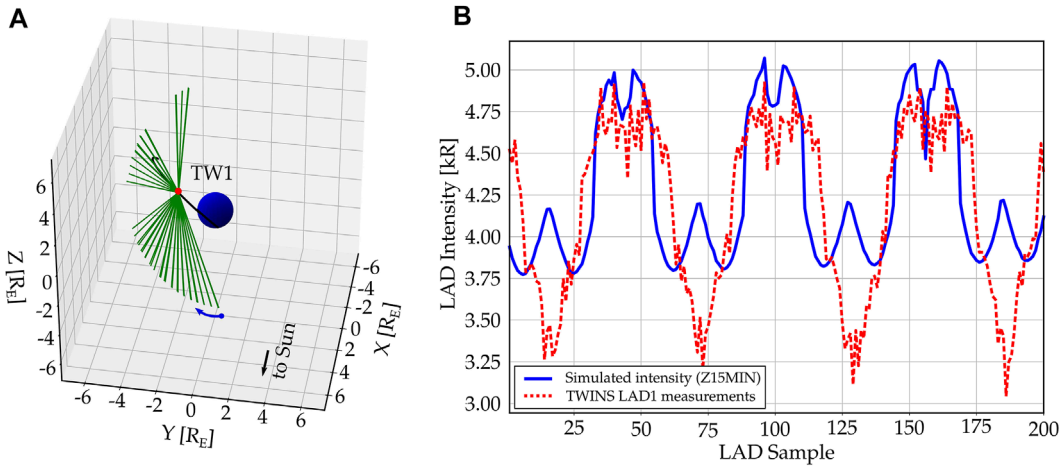


FIGURE 3
Simulation of TWINS1/LAD1 observations of the optically thin exosphere. In Panel **(A)**, the black line indicates TWINS1 (TW1) orbit, green lines are LAD's LOSs rotating 180° around a nadir pointing vector, and the blue arrow indicates the initial direction of rotation. In Panel **(B)**, the blue line shows the calculated intensity for each LOS using the Z15MIN model and Eq. 4, and the dashed red line shows the actual data from TWINS1/LAD1 instrument.

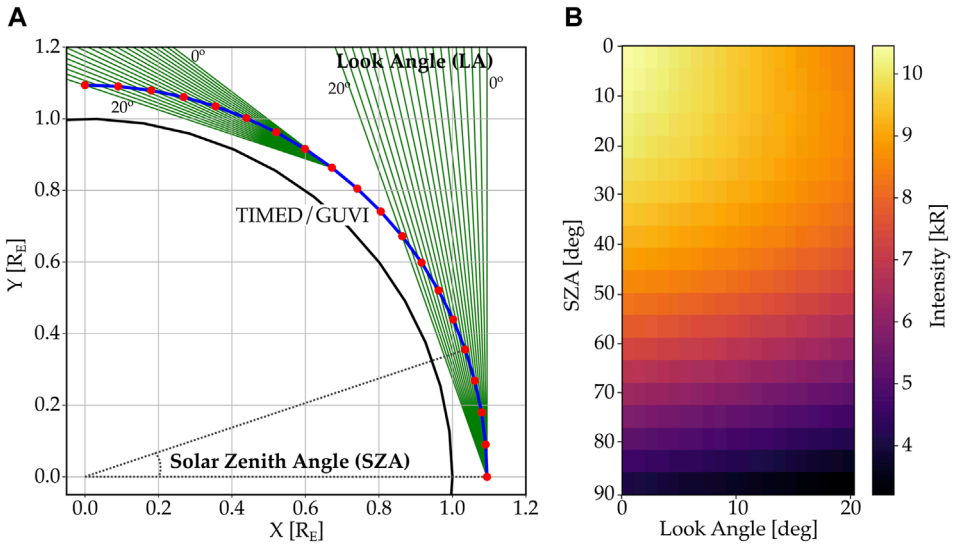


FIGURE 4
Simulation of TIMED/GUVI observations of the optically thick exosphere. Panel **(A)** shows the trajectory of the GUVI instrument projected to the XY ecliptic plane as a blue line. Red dots indicate the position of the satellite where measurements are carried out. Green lines are LOSs for several look angles, LA ϵ [0,20] degrees. Panel **(B)** shows the calculated intensities by EXOSpy using the volume emission rate for the year 2020, i.e., during solar minimum conditions.

3.3 UV instrument design

The EXOSpy package can be used as part of the designing process of space-based UV instruments for several mission objectives such as estimation of exospheric H density distributions or observation of extra-terrestrial bodies in FUV wavelengths other than Ly-α.

Here we describe how the EXOSpy package can be used to design a wide FOV imager to observe the exosphere in Ly-α. This task typically aims to determine several optical parameters such as FOV (in degrees), pixel resolution (in degrees/pixel), sensor responsivity at Ly-α (in counts/sec/R), integration time (in seconds), the optimal ephemeris, and the optimal pointing viewing geometry. In this case, the designing methodology is an iterative process wherein an initial set of the above-listed parameters are used to generate an image that is then evaluated against certain requirements such as signal-to-noise ratio (SNR) or its capability to yield additional scientific products, e.g., H densities with lower uncertainty. If these requirements are not met, the parameters are modified, and a new iteration is performed. For this application, we simulate an image to observe the optically thin region of the exosphere with the following parameters: a square FOV

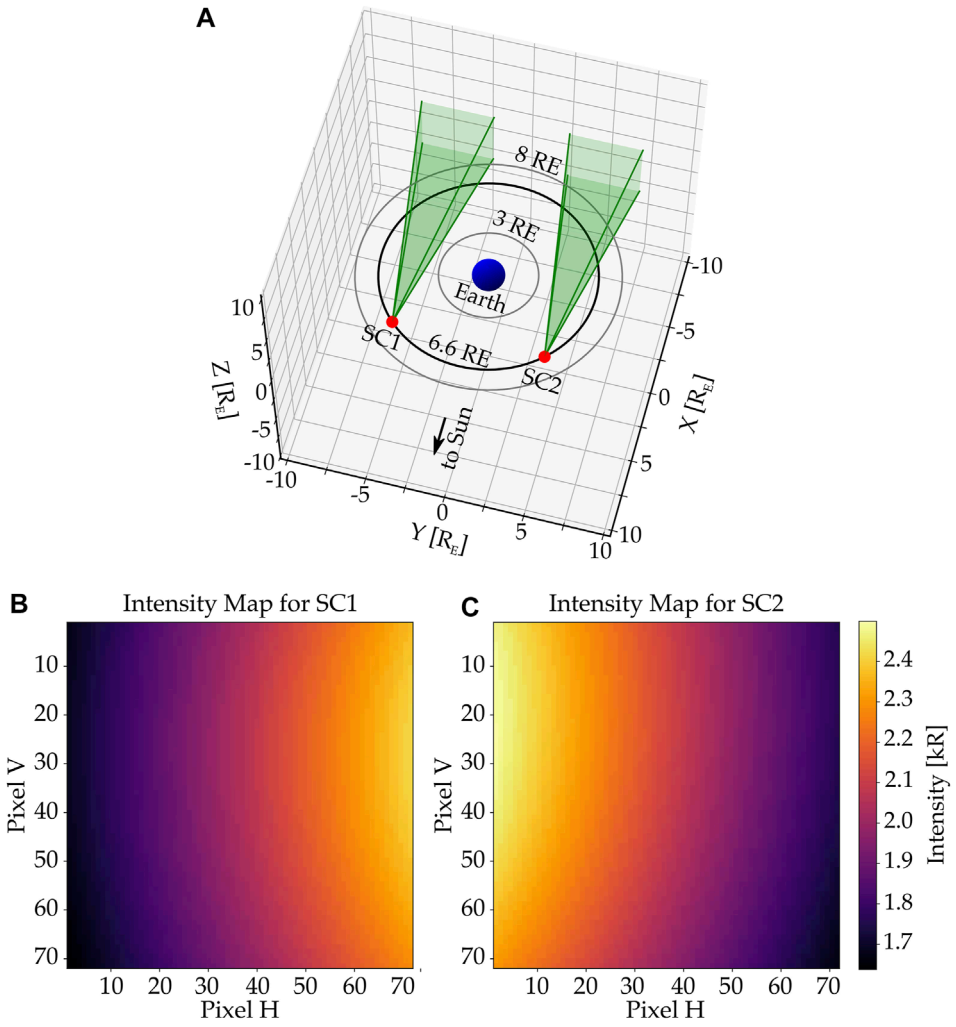


FIGURE 5
Simulation of a wide FOV imager to observe the optically thin exosphere. Panel **(A)** shows the viewing geometry for an spacecraft located in a geosynchronous orbit with radius 6.6 R_E. Red dots indicate the positions where the imager acquire an image. See the text for further details of optical parameters. Panels **(B,C)** show the calculated intensities by EXOSpy using the 3-D H model Z15MAX (valid only from 3 to 8 R_E).

of 18°, a pixel resolution of 18/72°/pixel, a geosynchronous orbit with a radius of 6.6 R_E over the ecliptic plane, and an anti-sunward pointing viewing geometry. Panel (A) in **Figure 5** shows a simulation of this instrument where the red dots indicate two acquisition periods, green sectors show the square FOV, and the black line represents the orbit. To generate synthetic intensities, we select H density distributions during solar maximum conditions, i.e., Z15MAX model (see **Table 1**). The solar irradiance used in this example was 0.0086 W/m^2 obtained from LISIRD/LASP website for 20 June 2012. Panels (B) and (C) show the resulting 72 × 72 pixel image for each satellite position. The python notebook Example4. ipynb shows the use of EXOSpy's function to generate a wide field image for the different pixel LOSs by calculating their corresponding column-integrated Ly-α intensities. The reader can access the file in https://github.com/gcucho/EXOSpy/blob/main/Examples/Example4.ipynb. It is worth noting that these images show a dawn/dusk asymmetry in the intensities, which, at this altitude, is directly associated with an asymmetric distribution of H densities. In addition, we adopt a sensor responsivity at Ly-α equal to 5 ×10^{-3} counts/sec/R (typical for a CCD imaging sensor) and an integration time of 10 min. As a result, the number of digital counts in the images range from ∼5100 to ∼7200 counts. Assessment of this parameter setting can be done using a threshold SNR. In optical systems, the main source of noise is the Poisson-distributed shot noise which, in turn, simplifies the derivation of the measurement's SNR to $\sqrt{N}$, where N is the number of digital counts. Hence, the SNR for these images range from ∼71 to ∼84.

Unlike spectrograph-based instruments that can resolve narrow wavelengths bands, imagers rely on optical filters or a combination of them to generate a narrow responsivity function. Nevertheless, imagers are still the best option to obtain a global view of the target. For example, future astrophysics missions might be interested in observing very distant galaxies in Ly-α using imagers. Depending on the galaxy's distance and its velocity with respect to the observer, they may be red-shifted (z) from 121.56 nm. For instance, observation of a target with a red-shift parameter of $z = 0.2$ indicates that an emitted 121.56 nm photon would be observed as 145.87 nm at earth. The proximity of these wavelengths requires an effective selection of optical filters to generate a responsivity function able to suppress the exospheric

Ly-α emission. For this purpose, an accurate specification of this bright emission under different solar conditions is required. This task can be achieved using the similar approach presented in **Section 3.2** to estimate column-integrated Ly-α emissions for a given satellite position and instrument's LOSs.

4 Summary and future work

In this work, we have developed EXOSpy, an open-source python-based package to explore current terrestrial exospheric models and calculate their Ly-α emission. Through examples provided in this project, we demonstrate the capabilities of EXOSpy (i) to validate exospheric models with actual Ly-α radiance data, (ii) to estimate exospheric contamination that may affect extra-terrestrial observations, and (iii) to support UV instrument design.

Future development of the EXOSpy package includes (i) the implementation of an open-source 3-D Radiative Transfer model that can automatically provide volume emission rates for a given 3-D H density profile, (ii) the capability for directly downloading actual Ly-α data from TWINS/LAD, IMAGE/GEO and TIMED/GUVI instruments and subsequent comparison with synthetically-generated column-integrated Ly-α measurements, and (iii) the capability for downloading 1-day averaged interplanetary Ly-α background from the SOHO/SWAN instrument.

Data availability statement

The EXOSpy's source code and Jupyter notebook used to generate figures in this paper can be found https://zenodo.org/record/7245359 in Zenodo. TWINS Lyman-alpha data are publicly available through the NASA Space Physics Data Facility.

Author contributions

GC-P conceptualized this project, developed the python package and wrote the manuscript. DB co-developed the python package and reviewed the manuscript, DS supervised the work, secured funding, and reviewed the manuscript. HC provided feedback regarding the validation of exospheric models and reviewed the manuscript. AY provided feedback regarding contamination in astrophysics applications and reviewed the manuscript. DA provided feedback regarding UV instrument design and reviewed the manuscript. All authors contributed to the article and approved the submitted version.

Funding

NASA Cooperative Agreement 80NSSC21M0180G: Partnership for Heliophysics and Space Environment Research (NS) NASA Heliophysics United States Participating Investigator Program under Grant WBS516741.01.24.01.03 WBS516741.01.24.01.03 (DS).

Acknowledgments

We acknowledge the support from the International Space Science Institute on the ISSI team 492, titled "The earth's Exosphere and its Response to Space Weather". Also, HC gracefully acknowledges support from the NASA grants, 80NSSC20K1670 and 80MSFC20C0019.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Bailey, J., and Gruntman, M. (2011). Experimental study of exospheric hydrogen atom distributions by Lyman-alpha detectors on the TWINS mission. *J. Geophys. Res. Space Phys.* 116, 1–9. doi:10.1029/2011JA016531

Baliukin, I. I., Bertaux, J.-L., Quémerais, E., Izmodenov, V. V., and Schmidt, W. (2019). SWAN/SOHO Lyman-alpha mapping: The hydrogen geocorona extends well beyond the Moon. *J. Geophys. Res. Space Phys.* 124, 861–885. doi:10.1029/2018JA026136

Bhattacharyya, D., Clarke, J., Bertaux, J.-L., Chaufray, J.-Y., and Mayyasi, M. (2017). Analysis and modeling of remote observations of the Martian hydrogen exosphere. *Icarus* 281, 264–280. doi:10.1016/j.icarus.2016.08.034

Bourrier, V., Ehrenreich, D., Lecavelier des Etangs, A., Louden, T., Wheatley, P. J., Wyttenbach, A., et al. (2018). High-energy environment of super-Earth 55 cancri e - i. far-uv chromospheric variability as a possible tracer of planet-induced coronal rain. *A&A* 615, A117. doi:10.1051/0004-6361/201832700

Brandt, J. C., and Chamberlain, J. W. (1959). Hydrogen radiation in the night sky. *Astrophys. J.* 130, 670–682.

Carruthers, G. R., Page, T., and Meier, R. R. (1976). Apollo 16 Lyman alpha imagery of the hydrogen geocorona. *J. Geophys. Res.* 81, 1664–1672. doi:10.1029/ja081i010p01664

Chamberlain, J. W. (1963). Planetary coronae and atmospheric evaporation. *Planet. Space Sci.* 11, 901–960. doi:10.1016/0032-0633(63)90122-3

Connor, H. K., and Carter, J. A. (2019). Exospheric neutral hydrogen density at the nominal 10 R_E subsolar point deduced from XMM-Newton X-Ray observations. *J. Geophys. Res. Space Phys.* 124, 1612–1624. doi:10.1029/2018JA026187

Cucho-Padin, G., Ferradas, C., Waldrop, L., and Fok, M. C. H. (2020). Understanding the role of exospheric density in the ring current recovery rate. *AGU Fall Meet. Abstr.* 2020, SM037.

Cucho-Padin, G., Kameda, S., and Sibeck, D. G. (2022). The Earth's outer exospheric density distributions derived from procyon/laica uv observations. *J. Geophys. Res. Space Phys.* 127, e2021JA030211. doi:10.1029/2021JA030211

Cucho-Padin, G., and Waldrop, L. (2019). Time-dependent response of the terrestrial exosphere to a geomagnetic storm. *Geophys. Res. Lett.* 46, 11661–11670. doi:10.1029/2019gl084327

Cucho-Padin, G., and Waldrop, L. (2018). Tomographic estimation of exospheric hydrogen density distributions. *J. Geophys. Res. Space Phys.* 123, 5119–5139. doi:10.1029/2018ja025323

Emmert, J. T., Drob, D. P., Picone, J. M., Siskind, D. E., Jones, M.Jr., Mlynczak, M. G., et al. (2021). NRLMSIS 2.0: A whole-atmosphere empirical model of temperature and neutral species densities. *Earth Space Sci.* 8, e2020EA001321. doi:10.1029/2020ea001321

Esposito, L. W., Barth, C. A., Colwell, J. E., Lawrence, G. M., McClintock, W. E., Stewart, A. I. F., et al. (2004). The cassini ultraviolet imaging spectrograph investigation. *Space Sci. Rev.* 115, 299–361. doi:10.1007/s11214-004-1455-8

Fairfield, D. H. (1971). Average and unusual locations of the Earth's magnetopause and bow shock. *J. Geophys. Res.* 76, 6700–6716. doi:10.1029/JA076i028p06700

Holstein, T. (1947). Imprisonment of resonance radiation in gases. *Phys. Rev.* 72, 1212–1233. doi:10.1103/physrev.72.1212

Ilie, R., Liemohn, M. W., Toth, G., and Skoug, R. M. (2012). Kinetic model of the inner magnetosphere with arbitrary magnetic field. *J. Geophys. Res. Space Phys.* 117, 7189. doi:10.1029/2011ja017189

Jung, J., Connor, H. K., Carter, J. A., Koutroumpa, D., Pagani, C., and Kuntz, K. D. (2022). Solar minimum exospheric neutral density near the subsolar magnetopause estimated from the xmm soft x-ray observations on 12 november 2008. *J. Geophys. Res. Space Phys.* 127, e2021JA029676. doi:10.1029/2021ja029676

Krall, J., Glocer, A., Fok, M.-C., Nossal, S. M., and Huba, J. D. (2018). The unknown hydrogen exosphere: Space weather implications. *Space weather.* 16, 205–215. doi:10.1002/2017SW001780

McComas, D. J., Allegrini, F., Baldonado, J., Blake, B., Brandt, P. C., Burch, J., et al. (2009). The two wide-angle imaging neutral-atom spectrometers (TWINS) NASA mission-of-opportunity. *Space Sci. Rev.* 142, 157–231. doi:10.1007/s11214-008-9467-4

Pryor, W. R., Ajello, J. M., Barth, C. A., Hord, C. W., Stewart, A. I. F., Simmons, K. E., et al. (1992). The Galileo and Pioneer Venus ultraviolet spectrometer experiments: Solar Lyman-alpha latitude variation at solar maximum from interplanetary Lyman-alpha observations. *Astrophysical J.* 394, 363. doi:10.1086/171589

Waldrop, L., and Paxton, L. J. (2013). Lyman-alpha airglow emission: Implications for atomic hydrogen geocorona variability with solar cycle. *J. Geophys. Res. Space Phys.* 118, 5874–5890. doi:10.1002/jgra.50496

Werner, S., Keller, H., Korth, A., and Lauche, H. (2004). Uvis/hdac lyman-alpha observations of the geocorona during cassini's Earth swingby compared to model predictions. *Adv. Space Res. Ionos. Atmos. Incl. CIRA* 34, 1647–1649. doi:10.1016/j.asr.2003.03.074

Wood, B. E., Redfield, S., Linsky, J. L., Müller, H.-R., and Zank, G. P. (2005). Stellar lyα emission lines in the *Hubble space telescope* archive: Intrinsic line fluxes and absorption from the heliosphere and astrospheres. *Astrophysical J. Suppl. Ser.* 159, 118–140. doi:10.1086/430523

Youngblood, A., France, K., Loyd, R. O. P., Linsky, J. L., Redfield, S., Schneider, P. C., et al. (2016). The muscles treasury survey. II. Intrinsic LYαAND extreme ultraviolet spectra of K and M dwarfs with exoplanets. *Astrophysical J.* 824, 101. doi:10.3847/0004-637x/824/2/101

Zoennchen, J. H., Connor, H. K., Jung, J., Nass, U., and Fahr, H. J. (2022). Terrestrial exospheric dayside h-density profile at 3–15 r_E from uvis/hdac and twins lyman-α data combined. *Ann. Geophys.* 40, 271–279. doi:10.5194/angeo-40-271-2022

Zoennchen, J. H., Nass, U., and Fahr, H. J. (2015). Terrestrial exospheric hydrogen density distributions under solar minimum and solar maximum conditions observed by the TWINS stereo mission. *Ann. Geophys.* 33, 413–426. doi:10.5194/angeo-33-413-2015

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 09 February 2023
DOI 10.3389/fspas.2022.1049677

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Pablo Gutiérrez-Marqués,
Max Planck Institute for Solar System Research, Germany
Hamood Ur Rehman,
University of Okara, Pakistan

*CORRESPONDENCE
Baptiste Cecconi,
baptiste.cecconi@observatoiredeparis.psl.eu

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 20 September 2022
ACCEPTED 07 November 2022
PUBLISHED 09 February 2023

CITATION
Cecconi B, Louis CK, Bonnin X, Loh A and Taylor MB (2023), Time-frequency catalogue: JSON implementation and python library.
Front. Astron. Space Sci. 9:1049677.
doi: 10.3389/fspas.2022.1049677

COPYRIGHT
© 2023 Cecconi, Louis, Bonnin, Loh and Taylor. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Time-frequency catalogue: JSON implementation and python library

Baptiste Cecconi[1]*, Corentin K. Louis[2], Xavier Bonnin[1], Alan Loh[1] and Mark B. Taylor[3]

[1]LESIA, Observatoire de Paris, Université PSL, Sorbonne Université, Université Paris Cité, CNRS, Meudon, France, [2]School of Cosmic Physics, DIAS Dunsink Observatory, Dublin Institute for Advanced Studies, Dublin, Ireland, [3]H H Wills Physics Laboratory, University of Bristol, Bristol, United Kingdom

TFCat (Time-Frequency Catalogue) is a data interchange format based on JSON (JavaScript Object Notation), which has been initially designed for exchanging low frequency radio events and features. It defines several types of JSON objects and how they are combined to represent data related to temporal-spectral features of a time spectrogram (a.k.a., dynamic spectrum), their properties, and their temporal and spectral extents. This implementation is inheriting from the GeoJSON file format. The TFCat python library is implementing this specification and provides a software interface permitting to create, update and validate TFCat objects efficiently.

KEYWORDS

radioastronomy, data format, python library, catalog, JSON

1 Introduction

At the lower end of the radio frequency domain, i.e., below ≈ 100 MHz, the electromagnetic signatures are tracers of energetic and unstable particle populations rather than atomic and molecular transitions. Such features are usually identified with their spectral-temporal (also referred to as "time-frequency") shape in time varying spectrograms (also referred to as dynamic spectra). Such time-frequency features are related to Earth and planetary magnetospheres (magnetospheric and auroral radio emissions), Earth and planetary lightnings (electrostatic discharges), Solar activity (solar radio bursts), as well as astrophysical radio sources such as Pulsars or Fast Radio Bursts. For a review of the low frequency planetary radio emissions, see Zarka (2004). **Figure 1** shows an example dynamic spectrum recorded by the Cassini/RPWS (Radio and Plasma Waves Science) instrument (Gurnett et al., 2004) displaying Solar and Terrestrial low frequency radio features.

Since the early 1960's, a corpus of catalogues identifying and documenting such Jovian radio emissions (Barrow and Desch, 1980; Barrow, 1981; Leblanc et al., 1981, Leblanc et al., 1983; Aubier and Genova, 1985; Leblanc et al., 1989, Leblanc et al., 1990; Riihimaa, 1991; Leblanc et al., 1993; Marques et al., 2017), and solar radio bursts (Maxwell et al., 1963; Bernold, 1980; Kayser et al., 1988; Zhang et al., 2018) have been

published, or are available online (such as the Bleien Solar Radio Data legacy web page[1]), providing access to: lists of event bounding boxes in the temporal-spectral plane; more refined contour shapes; or linear features. However, there was no standard exchange format, leading to difficulties in sharing and reusing such catalogues.

The TFCat (Time-Frequency Catalogue) specification has been developed with the aim of facilitating the recording, rese and processing of low frequency radio temporal-spectral events. In the past two years, several catalogues have been published using the TFCat format, and tools are now implementing interfaces on top of this specification, for labelling observations, displaying catalogues, or even using the catalogue shapes to select and process observational data (Cecconi et al., 2021a; Fogg et al., 2021; Louis et al., 2021; Taubenschuss et al., 2021; Wu et al., 2021, Wu et al., 2022b, Wu et al., 2022a).

In this paper, we will present the needs and use cases identified in the preliminary stages of the TFCat specification project. We then detail the proposed catalogue model and its implementation. The associated python library is also presented, with a series of examples and applications.

2 Needs and use cases

The MASER service (Measuring, Analysing and Simulating Emissions in the Radio range) is a science ready toolkit for low frequency radio astronomy (Cecconi et al., 2020). During its inception, the storing, processing and sharing of labelled features in radio frequency dynamic spectra has been identified as one of the major community needs. The main complexity is the shape representation of the features, as a series of coordinates in the temporal-spectral plane. The MASER team has thus explored various applicable solutions based on existing tools and standards in the astronomy, heliophysics and Earth sciences.

The astronomy community has developed an interoperability framework called the "Virtual Observatory" and maintained by the IVOA (International Virtual Observatory Alliance, https://ivoa.net). They define many standards allowing to build complex interoperable data infrastructures. For the purpose of catalogues and events, three standards are applicable: the VOTable (Virtual Observatory Table) format (Ochsenbein et al., 2019), the VOEvent (Virtual Observatory Event) model (Seaman et al., 2011) and the STC (Space-Time Coordinate) model (Rots, 2007) or its upcoming successor[2]. Those formats and models are very well designed for astronomical catalogues and events, including time intervals, spectral ranges, spatial regions on the sky (most often using Right Ascension (RA) and Declination (Dec) coordinates). However, there is currently no way to build and share more detailed geometries mixing temporal and spectral coordinates using those tools. The heliophysics community has also defined a set of standards including a heliophysics event list (HPEventList[3]), which is using CSV (comma separated values) and VOTable as serialization formats. The same restriction applies as for the astronomy applications.

The description of two-dimensional (2D) geometries or coverage information can be done using various tools and methods. The solar physics community has been using the "chaincodes" method (see, e.g., the *SunPy* implementation[4]), which describes a 2D geometry specifying a starting point on an image and then listing the pixel per pixel path along the feature contour. Such implementation is computationally very efficient, but is providing geometry in pixel coordinates, requiring the access to the original data (and metadata) to reuse the described feature. In the astronomy framework, the MOC (HEALPix –Hierarchical Equal Area isoLatitude Pixelisation– Multi-Order Coverage map) standard (Fernique et al., 2019) is currently allowing to describe spatial and temporal complex coverage maps, however the spectral dimension is not considered yet in the standard. Finally in the Earth science community, feature geometries are using the GeoJSON format (Butler et al., 2016). This format is using geographic coordinates to describe features and proposes several classes of features (such as points, segments or polygons).

In order to facilitate the adoption by catalogue providers (e.g., authors of publications), the selected solution must be easy to implement and should avoid dependencies on too specific software. On the user side, the same statement applies with the other requirements that it should be easy for tool and science platform developers to implement the solution.

Besides the description of the temporal-spectral feature geometries, it shall be possible to set additional properties for each feature (e.g., a label, an identifier linking to the original observation, or any property or statistical quantity related to the observed feature). Such properties shall be defined properly, so that users can reuse and process them. Furthermore, the catalogue shall contain metadata covering the scope of the catalogue itself (such as authors, references, a title). Metadata on the source datasets, from which the feature catalogue is derived, should also be referenced. Finally the temporal and spectral axes must be properly defined within the catalogue.

1 http://soleil.i4ds.ch/solarradio/.

2 Astronomical Coordinates and Coordinate Systems (2021) Proposed IVOA Recommendation. Version 1.0. https://www.ivoa.net/documents/Coords/20210924/index.html.

3 Heliophysics Event List: https://spase-group.org/docs/conventions/HDMC-Event-List-Specification-v1.0.4.pdf.

4 SunPy.Chaincode: https://docs.sunpy.org/en/stable/generated/api/sunpy.net.helio.Chaincode.html.

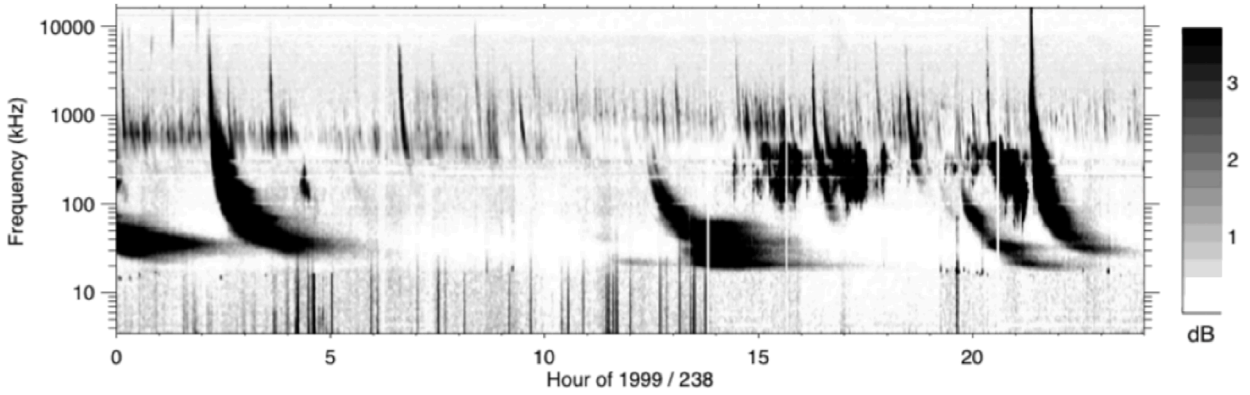


FIGURE 1
Low frequency radio dynamic spectrum recorded by Cassini/RPWS on 1999-08-26. The spectrogram is showing Solar radio bursts as well as terrestrial auroral kilometric radiation (AKR).

3 Proposed catalogue model

The TFCat model provides a way to describe a series of features referenced by their coordinates (or set of coordinates) in the temporal-spectral plane, as observed from a radio instrument (a ground based telescope or onboard a spacecraft), and identified by an agent (a scientist or a software). Each feature of the catalogue is defined by a set of temporal and spectral coordinates, and a geometry type. Features can have additional properties, depending on the science case, which content shall be defined in the catalogue. The temporal and spectral coordinate frame in use in the catalogue shall be defined uniquely in a catalogue file, or identified by unique coordinate system name. The catalogue can have general properties (e.g., header type metadata). The TFCat base model elements are:

- *TFCat Geometry*—There are 7 types of TFCat Geometry: *Point*, *MultiPoint*, *LineString*, *MultiLineString*, *Polygon*, *MultiPolygon*, with an additional *GeometryCollection*, which is a set of any of the six other geometry types. Each geometry is composed of a set of coordinate pairs. A *LineString* is defined by a list of connected *Point* features, forming a continuous path. A *Polygon* is defined by a closed list of connected *Point* features, forming a closed contour. The feature type definitions are inspired from the GeoJSON specification (Butler et al., 2016).
- *TFCat Feature*—A TFCat Feature contains one or several TFCat Geometries, and a set of properties, provided as a set of (key, value) pairs.
- *TFCat Field*—A TFCat Field is defining the TFCat feature properties. Its minimal content includes a name, a description, a data type and a UCD (Unified Content Descriptors) (Cecconi et al., 2021b). It may also include a unit [following the VOUnit –Virtual Observatory Unit–specification (Derriere et al., 2014)], or a set of allowed values, when applicable.
- *TFCat Property*—A TFCat Property is a (key, value) pair. There are two types of TFCat Properties:
  - *TFCat Feature Property*—TFCat feature properties are used in a TFCat Feature. They must be defined with a corresponding TFCat Field.
  - *TFCat Collection Property*—TFCat collection properties are used in a TFCat Feature Collection. They should provide generic information on the catalogue, such as: title, authors, reference, instrument, etc.
- *TFCat Coordinate Reference System*—A TFCat Coordination Reference System (CRS) contains a description of the time axis, a description of the spectral axis, and a reference position. The TFCat CRS could be of three types: *name*, *link* or *local*.

3.1 Javascript object notation implementation

The TFCat model has been implemented using the JSON format (Bray, 2014), reusing the structure of the GeoJSON specification. The TFCat JSON specification is available (Cecconi et al., 2022b). The specification is maintained on a git repository[5]. When applicable the astronomy standards have been implemented (UCD (Preite Martinez et al., 2018), VOUnit (Derriere et al., 2014), and list of known keywords, like, e.g., reference positions).

5 TFCat specification: https://gitlab.obspm.fr/maser/catalogues/catalogue-format.

The TFCat CRS specification aligns on GeoJSON CRS definition. Currently, the TFCat CRS must be defined in the TFCat object instance. However, a set of predefined time axis are available within the specification (e.g., *unix, jd, mjd*, etc).

3.2 Catalogue validation

A JSON Schema document[6] has been prepared to validate the catalogues. However, the JSON schema only covers part of the specification, since some constraints could not be implemented therein (like, e.g., the need to have a TFCat Field defined for each TFCat Feature Property). An example of TFCat validation using the JSON schema is provided in **Section 4.1**.

A full validator (based on the TFCat specification) has been implemented in *STILTS*[7] (Starlink Tables Infrastructure Library Tool Set, Taylor, 2006), version 3.4-7 or later.

This java-based tool can be called on the command line:

```
java -classpath stilts.jar uk.ac.starlink.tfcat.
    Validate <filename.json>
```

Listing 1: Validating a TFCat file

4 Time-frequency catalogue python library

Since the TFCat JSON implementation is closely inheriting from GeoJSON, many tools from the Earth science community can be reused. First, the TFCat python library[8] has been developed starting from the code of a python GeoJSON library[9]. The TFCat library also makes use of the *shapely* library[10], which can be used for managing and processing TFCat features in general.

4.1 Creating a time-frequency catalogue feature collection

The two main elements required to start preparing a TFCat feature collection is the choice of the CRS and the list of feature geometries (with its set of time and spectral coordinates). In the following code snippet, a CRS object is created with a time axis using the *unix* timestamp representation, a spectral axis (frequency in kHz) and a reference position set to the Juno spacecraft.

```
from tfcat import CRS
crs = CRS({
    "type": "local",
    "properties": {
        "name": "Time-Frequency",
        "time_coords_id": "unix",
        "spectral_coords": {
            "type": "frequency",
            "unit": "kHz",
        },
    "ref_position_id": "juno",
    }
})
```

Listing 2: Example CRS definition

In the following example, we create a TFCat feature with a TFCat *Polygon* geometry and two custom feature *properties* called *feature_type* and *integrated_flux_density*. The names and values of feature properties are not constrained by the TFCat specification. However, the specification requires to include TFCat Fields defining any TFCat feature property that may be included in a Feature. The set of coordinates are available in two variables called *tt* and *ff*, containing respectively the time coordinates (as a *unix* timestamp), and the frequency coordinates (in kHz). Note, that a *Polygon* geometry requires that the last coordinate point of the input sequence should be the same as the first one (closed polygon condition). It also requires that the segments forming the polygon are not crossing over (no cross-over condition). The polygon must also be curled counter-clockwise. We use the *shapely* package to check the latter condition.

```
from tfcat import Polygon, Feature
from shapely.geometry import LinearRing
# packing coordinates into a list
coords = list(zip(tt, ff))
# checking if list of coordinates is curled in
    the direct trigonometric sense (counter-
    clockwise)
if not LinearRing(coords).is_ccw:
    coords = coords[::-1]
# creating the TFCat Feature
feature = Feature(
    geometry=Polygon([coords]),
    properties={
        "feature_type": "Io-A",
        "integrated_flux_density": 1.22e-19,
```

6 TFCat JSON Schema v1.0: https://voparis-ns.obspm.fr/maser/tfcat/v1.0/schema#.

7 STILTS website: http://www.starlink.ac.uk/stilts/.

8 TFCat Python library: https://gitlab.obspm.fr/maser/catalogues/tfcat.

9 GeoJSON: https://github.com/jazzband/geojson.

10 shapely: https://shapely.readthedocs.io/en/stable/.

```
    },
)
```

Listing 3: Creating a TFCat Polygon

Each *properties* must be defined at the main level of the feature collection object, using a set of TFCat Field objects, as shown below.

```
fields = {
    "feature_type": {
        "info": "Feature Type",
        "datatype": "str",
        "ucd": "meta.id",
    },
    "integrated_flux_density": {
        "info": "Integrated Flux Density",
        "datatype": "float",
        "ucd": "phot.flux.density;stat.mean",
        "unit": "W m-2 Hz-1",
    }
}
```

Listing 4: Defining Fields

The feature collection can also include collection-level *properties*, listing global metadata for the collection, such as the author, the instrument, etc. The next code snippet shows a set of collection level properties. We recommend to use keywords compliant with the EPNcore (Europlanet core) metadata dictionary (Erard et al., 2022) for Solar System related collections. A *data_source_reference* property may be included to refer to the data collection used to derive the catalogue. The value of the data source reference should be a DOI if it is available for the progenitor resource. However, any type of well-known community reference could be used, such as, e.g., a SPASE (Roberts et al., 2018) resource identifier or a NASA/PDS (Planetary Data System) identifier[11]. For authors, we recommend to use a list of dictionaries containing *GivenName*, *FamilyName*, *ORCID* and *Affiliation* (using ROR[12] when possible). The current version of the TFCat specification does not define mandatory properties, nor does it define the object type of collection level properties. This may be implemented in a future version, in order to improve the interoperability and reusability.

```
properties = {
    "instrument_host_name": "Juno"
    "instrument_name": "Waves",
    "title": "Catalogue of Juno Radio emissions",
    "authors": [
        {
            "GivenName": "Corentin K.",
            "FamilyName": "Louis",
            "ORCID": "https://orcid.org/0000-0002-9552-8822",
            "Affiliation": "https://ror.org/051sx6d27"
        },
    ],
    "target_name": "Jupiter",
    "target_class": "planet",
    "target_region": "magnetosphere",
    "feature_name": "radio emissions",
    "bib_reference": "10.1029/2021JA029435",
    "data_source_ref": "http://doi.org/10.25935/6jg4-mk86",
    "publisher": "PADC/MASER",
    "version": "1.0",
}
```

Listing 5: Defining collection level properties

Finally, the collection can be built, assuming a list of features is available in an object called *features*. The JSON schema URI can also be referenced. The default value is set to the latest released TFCat JSON schema URI (i.e., https://voparis-ns.obspm.fr/maser/tfcat/v1.0/schema# at the time of writing).

```
from tfcat import FeatureCollection
SCHEMA_URI = "https://voparis-ns.obspm.fr/maser/tfcat/v1.0/schema#"
# creating the collection
collection = FeatureCollection(
    schema=SCHEMA_URI,
    features=features,
    properties=properties,
    fields=fields,
    crs=crs,
)
```

Listing 6: Finalizing a TFCat collection

The TFCat collection can then be checked against the JSON Schema.

```
from tfcat.validate import validate
validate(collection)
```

Listing 7: Validating a TFCat collection

11 Citing PDS4 Data: https://pds.nasa.gov/datastandards/citing/.

12 Research Organisation Registry: https://ror.org.

Writing the collection into a file is straightforward, since the TFCat module implements the same interfaces as the *json* python module for file system interactions (i.e., *dump(), dumps(), load(), loads()* methods):

```
# write out the collection on disk
from tfcat import dump
filename = "tfcat.json"
with open(filename, "w") as f:
    dump(collection, f)
```

Listing 8: Dumping into File

4.2 Loading a time-frequency catalogue feature collection

The TFCat module also provides capabilities to load TFCat JSON files from the local file system, or from a remote location:

```
from tfcat import TFCat
local_cat = TFCat.from_file(<filepath>)
remote_cat = TFCat.from_url(<URL>)
```

Listing 9: Loading local or remote TFCat file

In the following example, we load TFCat file available in Taubenschuss et al. (2021). The script first displays the coordinate reference system (using the *crs* property), and the geometry type of the first feature. It then plots the two first features of the catalogue (see **Figure 2**), and then shows the use of the temporal and spectral converters, which can be used to load the time-frequency coordinates as an *astropy.time.Time* object for the time component, and an *astropy.units.Quantity* for the spectral component.

```
from tfcat import TFCat
tf_file = 'cassini_faraday_patches_2006.json'
cat = TFCat.from_file(tf_file)
cat.crs
# {"properties": {"name": "Time-Frequency", "
    ref_position_id": "cassini-orbiter",
# "spectral_coords": {"type": "frequency", "unit
    ": "Hz"}, "time_coords_id": "unix"},
# "type": "local"}
cat.feature(0).geometry.type
# 'MultiPoint'
cat.plot_feature(0)
cat.plot_feature(1)
cat.crs.time_converter(cat.feature(0).tmin).isot
# '2006-09-12T09:02:26.750'
cat.crs.spectral_converter(cat.feature(0).fmax)
# <Quantity 43469. Hz>
```

Listing 10: Example script

4.3 Module installation and dependencies

The TFCat module is distributed under MIT licence. The module dependencies are limited to a small set of external modules: *astropy* (Astropy Collaboration et al., 2022), *matplotlib* (Hunter, 2007), *shapely* (Gillies et al., 2007) and *jsonschema* (Berman et al., 2022). It is available on PyPI[13] and can thus be installed with pip:

```
pip install tfcat
```

Listing 11: Install TFCat with pip

5 Applications

TFCat has already been used in several published data collections (Cecconi et al., 2021b; Fogg et al., 2021; Louis et al., 2021; Taubenschuss et al., 2021; Smith et al., 2022; Wu et al., 2021, Wu et al., 2022b, Wu et al., 2022a). In order to showcase the capabilities of the TFCat format and its python implementation, 3 Jupyter notebook tutorials have been prepared, and are available in Cecconi et al. (2022a) [14]:

- *TFCat MultiPoint feature and feature processing* (MultiPoint_Example.ipynb) is showing how to use TFCat together with *shapely* to process a *MultiPoint* geometry into a *Polygon.*
- *TFCat MultiPolygon feature and data filtering example* (MultiPolygon_Example.ipynb) is showing how to use TFCat together with *shapely* and *das2* (Piker et al., 2019) to use a *MultiPolygon* geometry to select data and perform statistical analysis.
- *TFCat Multiple Catalogue Example* (Multiple_Catalogue_Example.ipynb) is showing how to compare two catalogues containing features derived from the same physical events.

The TFCat model and python library are used in the *SPectrogram Analysis and Cataloguing Environment* (SPACE) Labelling Tool (Louis et al., 2022). TFCat features can also be loaded and displayed in *Autoplot* (Faden et al., 2010). Full

13 https://pypi.org/project/TFCat/.

14 https://doi.org/10.25935/qwpa-7t05.

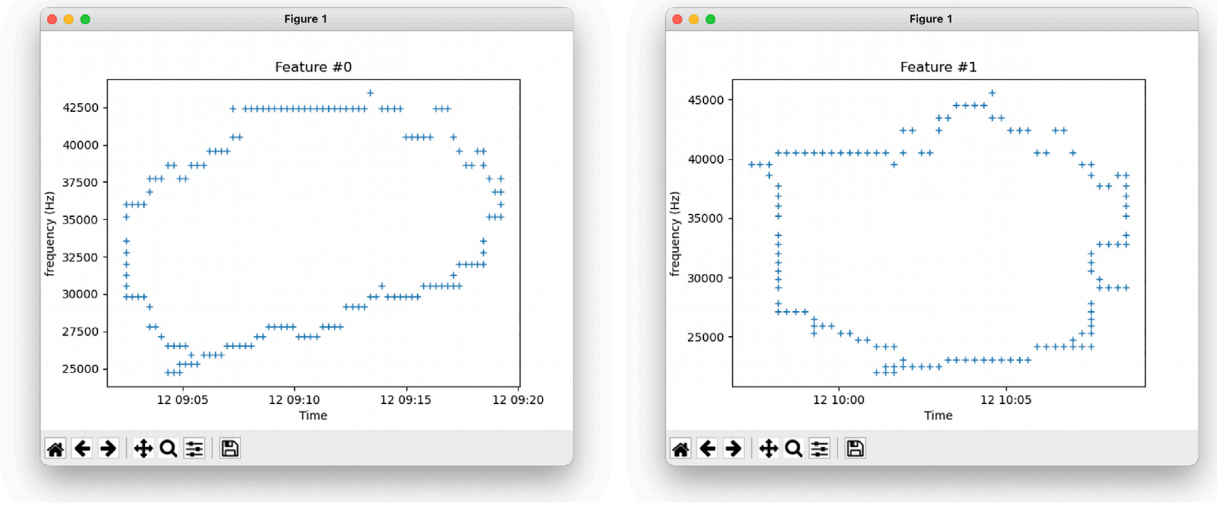


FIGURE 2
Example plots from Listing 10.

support is not currently available, but an example Jython script is provided[15] by the Autoplot team.

6 Conclusion and future prospects

The TFCat format and its associated Python library are providing the low frequency radio astronomy community with a novel tool enabling interoperability and reusability of temporal-spectral features. The published catalogues are demonstrating its usefulness through various use cases. The proposed notebooks are showing its usability. Furthermore, the accessibility is guaranteed by the open access policies of the current TFCat catalogue publishers. The findability of the TFCat catalogues can be implemented using existing community data discovery interfaces (such as EPN-TAP (Erard et al., 2022) for Solar System sciences). In order to enable the findability of the features, an implementation using the TAP Table Access Protocol, (Dowler et al., 2019), is under study.

The TFCat format is now used in publications and tools to store, share and process temporal-spectra features. The GeoJSON inherited specification allows to reuse python based libraries (such as *shapely*) for processing the geometries. At Observatoire de Paris, the MASER team is integrating TFCat in their machine learning pipelines related to planetary and solar radio emissions, as well as in their radio data flagging pipelines (for excluding radio frequency interferences in the data processing).

The TFCat collection properties are currently loosely defined: (a) the list of keywords is not defined in the TFCat specification; and (b) there is no metadata associated to them. This will be improved in a future major version, since this would require to refactor the already published collection. The specification foreseen for the TFCat collection properties would have the same definition as the TFCat Field items, with an additional *value* attribute, containing the value of the property. Such specification would improve TFCat reusability and interoperability, and it would make the format more compatible with other formats, such as VOTable.

Discussions are also ongoing with other communities dealing with temporal-spectral features, such as transient astrophysics (e.g., pulsars or fast radio bursts studies), meteor radio echoes, or even space based *in-situ* particle observations, for which the spectral axis would then represent an energy, mass or mass per charge axis.

Data availability statement

The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Author contributions

BC and CL developed the TFCat specification. BC developed the TFCat JSON Schema. BC, XB, and AL developed the TFCat python library. MT reviewed the TFCat specification, and developed the TFCat validator. BC wrote the first version

15 Autoplot Jython script: https://github.com/autoplot/scripts/tree/master/formats/tfcat.

of the papier. XB, AL, and MT commented and proofread the manuscript.

Funding

The Europlanet 2024 Research Infrastructure (EPN2024RI) project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871149. CKL's work at DIAS is supported by the Science Foundation Ireland Grant 18/FRL/6199. BC and MT's contributions for this study are supported by the EPN2024RI project. BC, XB, and AL acknowledge additional funding provided in France by the Centre National d'Études Spatiales (CNES), and Action Spécifique Observatoire Virtuel (ASOV). Support from Paris Astronomical Data Centre (PADC) is acknowledged.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., et al. (2022). The astropy project: Sustaining and growing a community-oriented open-source project and the latest major release (v5.0) of the core package. *Astrophys. J.* 935, 167. doi:10.3847/1538-4357/ac7c74

Aubier, M. G., and Genova, F. (1985). A catalogue of the high frequency limit of the Jovian decameter emission observed by Voyager. *Astron. Astrophys. Suppl. Ser.* 61, 341–351.

Barrow, C. H. (1981). A catalogue of Jupiter's decametric emission observed by Voyager-1 and by Voyager-2 in the range 15-40 MHz. *Astron. Astrophys. Suppl. Ser.* 46, 111–114.

Barrow, C. H., and Desch, M. D. (1980). Non-Io decametric radiation from Jupiter at frequencies above 30 MHz. *Astron. Astrophys. Suppl. Ser.* 86, 339–341.

Berman, J., Sterling, C., Taprest, R., Nezbeda, H., chen, W., Rosen, S., et al. (2022). python-jsonschema/jsonschema: v4.16.0. Zenodo. doi:10.5281/zenodo.7064213

Bernold, T. (1980). A catalogue of fine structures in type IV solar radio bursts. *Astron. Astrophys. Suppl. Ser.* 42, 43–58.

Bray, T. (2014). The JavaScript object notation (JSON) data interchange format, RFC 7159. doi:10.17487/RFC7159

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., and Schaub, T. (2016). The GeoJSON format. *IETF RFC* 7946. doi:10.17487/RFC7946

Cecconi, B., Bonnin, X., Loh, A., and Taylor, M. (2022a). TFCat (Time-Frequency catalogue): JSON implementation and Python library. Supplementary material. *PADC*. doi:10.25935/QWPA-7T05

Cecconi, B., Debisschop, L., and Lamy, L. (2021a). A catalogue of Jovian decametric radio observations from January 1978 to December 1990 (Version 1.0) [Data set]. PADC. doi:10.25935/1FQM-FS07

Cecconi, B., Loh, A., Sidaner, P. L., Savalle, R., Bonnin, X., Nguyen, Q. N., et al. (2020). Maser: A science ready toolbox for low frequency radio astronomy. *Data Sci. J.* 19, 1062. doi:10.5334/dsj-2020-012

Cecconi, B., Louys, M., Preite Martinez, A., Derrière, S., Ochsenbein, F., Erard, S., et al. (2021b). UCD1+ controlled vocabulary - updated list of terms version 1.4. *IVOA Endorsed Note*. doi:10.5479/ADS/bib/2021ivoa.spec.0616C

Cecconi, B., Taylor, M. B., Bonnin, X., and Loh, A. (2022b). JSON implementation of time-frequency radio catalogues: TFCat (version 1.0). *PADC*. doi:10.25935/6068-8528

Derriere, S., Gray, N., Demleitner, M., Louys, M., and Ochsenbein, F. (2014). Units in the VO version 1.0. *IVOA Recomm.* doi:10.5479/ADS/bib/2014ivoa.spec.0523D

Dowler, P., Rixon, G., Tody, D., and Demleitner, M. (2019). Table Access Protocol version 1.1. *IVOA Recomm.* doi:10.5479/ADS/bib/2019ivoa.spec.0927D

Erard, S., Cecconi, B., Le Sidaner, P., Demleitner, M., and Taylor, M. B. (2022). Epn-tap: Publishing solar system data to the virtual observatory. *IVOA Recomm.* version 2.0. doi:10.5479/ADS/bib/2022ivoa.spec.0822E

Faden, J. B., Weigel, R. S., Merka, J., and Friedel, R. H. W. (2010). Autoplot: A browser for scientific data on the web. *Earth Sci. Inf.* 3, 41–49. doi:10.1007/s12145-010-0049-0

Fernique, P., Boch, T., Donaldson, T., Durand, D., O'Mullane, W., Reinecke, M., et al. (2019). Moc - HEALPix multi-order coverage map version 1.1. *IVOA Recomm.* doi:10.5479/ADS/bib/2019ivoa.spec.1007F

Fogg, A. R., Jackman, C., Waters, J., Bonnin, X., Lamy, L., Cecconi, B., et al. (2021). Bursts of auroral kilometric radiation individually selected from wind/waves data. PADC. doi:10.25935/HFJX-XX26

Gillies, S., van der Wel, C., Van den Bossche, J., Taves, M. W., Arnott, J., Ward, B. C., et al. (2007). Shapely (Version 2.0) [Computer software]. Available at https://github.com/shapely/shapely.

Gurnett, D. A., Kurth, W. S., Kirchner, D. L., Hospodarsky, G. B., Averkamp, T. F., Zarka, P., et al. (2004). The cassini radio and plasma wave investigation. *Space Sci. Rev.* 114, 395–463. doi:10.1007/s11214-004-1434-0

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Kayser, S. E., Bougeret, J. L., Fainberg, J., and Stone, R. G. (1988). A catalog of interplanetary type III storms. *Astron. Astrophys. Suppl. Ser.* 73, 243–253.

Leblanc, Y., de La Noe, J., Genova, F., Gerbault, A., and Lecacheux, A. (1981). A catalogue of Jovian decametric radio observations from January 1978 to December 1979. *Astron. Astrophys. Suppl. Ser.* 46, 135–149.

Leblanc, Y., Gerbault, A., Denis, L., and Lecacheux, A. (1993). A catalogue of Jovian decametric radio observations from January 1988 to December 1990. *Astron. Astrophys. Suppl. Ser.* 98, 529–546.

Leblanc, Y., Gerbault, A., and Lecacheux, A. (1989). A catalogue of Jovian decametric radio observations from January 1982 to December 1984. *Astron. Astrophys. Suppl. Ser.* 77, 425–438.

Leblanc, Y., Gerbault, A., Lecacheux, A., and Boudjada, M. Y. (1990). A catalogue of Jovian decametric radio observations from January 1985 to December 1987. *Astron. Astrophys. Suppl. Ser.* 86, 191–207.

Leblanc, Y., Gerbault, A., Rubio, M., and Genova, F. (1983). A catalogue of Jovian radio observations from January 1980 to December 1981. *Astron. Astrophys. Suppl. Ser.* 54, 135–148.

Louis, C. K., Jackman, C. M., Mangham, S. W., Smith, K. D., O'Dwyer, E. P., Empey, A., et al. (2022). The "spectrogram analysis and cataloguing environment" (space) labelling tool. arXiv astro-phIM. doi:10.3389/fspas.2022.1001166

Louis, C. K., Zarka, P., and Cecconi, B. (2021). Catalogue of Jupiter radio emissions identified in the Juno/Waves observations (Version 1.0) [Data set], PADC. doi:10.25935/NHB2-WY29

Marques, M. S., Zarka, P., Echer, E., Ryabov, V. B., Alves, M. V., Denis, L., et al. (2017). Statistical analysis of 26 yr of observations of decametric radio emissions from Jupiter. *Astron. Astrophys.* 604, A17. doi:10.1051/0004-6361/201630025

Maxwell, A., Hughes, M. P., and Thompson, A. R. (1963). Catalog of type II(Slow-Drift) and type IV(continuum) solar radio bursts. *J. Geophys. Res.* 68, 1347–1354. doi:10.1029/JZ068i005p01347

Ochsenbein, F., Taylor, M., Donaldson, T., Williams, R., Davenhall, C., Demleitner, M., et al. (2019). VOTable format definition version 1.4. *IVOA Recomm.* doi:10.5479/ADS/bib/2019ivoa.spec.1021O

Piker, C., Granroth, L., Mukherjee, J., Pisa, D., Cecconi, B., Kopf, A., et al. (2019). Lightweight federated data networks with das2 tools. *Earth Space Sci. Open Archive*. doi:10.1002/essoar.10500359.1

Preite Martinez, A., Louys, M., Cecconi, B., Derriere, S., Ochsenbein, F., The Semantic IVOA working group, (2018). The UCD1+ controlled vocabulary Version 1.3. *IVOA Recomm.* doi:10.5479/ADS/bib/2018ivoa.spec.0527M

Riihimaa, J. J. (1991). Evolution of the spectral fine structure of Jupiter's decametric S-storms. *Earth Moon Planets* 53, 157–182. doi:10.1007/BF00057430

Roberts, D. A., Thieman, J., Génot, V., King, T., Gangloff, M., Perry, C., et al. (2018). The SPASE data model: A metadata standard for registering, finding, accessing, and using heliophysics data obtained from observations and modeling. *Space weather.* 16, 1899–1911. doi:10.1029/2018sw002038

Rots, A. H. (2007). Space-time coordinate metadata for the virtual observatory version 1.33. *IVOA Recomm.* doi:10.5479/ADS/bib/2007ivoa.spec.1030R

Seaman, R., Williams, R., Allan, A., Barthelmy, S., Bloom, J., Brewer, J., et al. (2011). Sky event reporting metadata version 2.0. *IVOA Recomm.* doi:10.5479/ADS/BIB/2011ivoa.spec.0111S

Smith, K. D., Louis, C. K., Jackman, C. M., Fogg, A. R., O'Dwyer, E., Waters, J. E., et al. (2022). Catalogue of auroral kilometric radiations (AKR) observed by the polar spacecraft (1.2). Zenodo. doi:10.5281/zenodo.7260994

Taubenschuss, U., Cecconi, B., and Lamy, L. (2021). Catalogue of faraday rotation patches identified in saturn kilometric radiation (SKR) observations by cassini/RPWS/HFR (Version 1.0) [Data set]. PADC. doi:10.25935/R11G-6J63

Taylor, M. B. (2006). "Stilts - a package for command-line processing of tabular data,". Editors C. Gabriel, C. Arviset, D. Ponz, and S. Enrique (San Francisco: Astronomical Society of the Pacific Conference Series), 351, 666.*Astronomical Data Analysis Softw. Syst. XV*

Wu, S., Ye, S. Y., Fischer, G., Wang, J., Long, M. Y., Menietti, J. D., et al. (2021). Statistical study of spatial distribution and polarization for saturn narrowband emissions. Supplementary material: Event catalogue (Version 1.0) [Data set]. PADC. doi:10.25935/ZMMZ-CA38

[Dataset] Wu, S., Ye, S. Y., Fischer, G., Taubenschuss, U., Jackman, C. M., O'Dwyer, E., et al. (2022a). Saturn anomalous myriametric radiation, a new type of saturn radio emission revealed by cassini. supplementary material: Catalogue of events. (Version 1.0) [Data set]. PADC. doi:10.25935/8MAY-4320

Wu, S., Zarka, P., Lamy, L., Taubenschuss, U., Cecconi, B., Ye, S. Y., et al. (2022b). Catalogue of observations of the first harmonic of saturn kilometric radiation during cassini's grand finale (Version 1.0) [Data set]. PADC. doi:10.25935/T033-QS72

Zarka, P., Treumann, R. A., Ryabov, B. P., and Ryabov, V. B. (2004). Radio and plasma waves at the outer planets. *Astrophys. Space Sci.* 33, 293–300. doi:10.1023/a:1012221527425

Zhang, P. J., Wang, C. B., and Ye, L. (2018). A type III radio burst automatic analysis system and statistic results for a half solar cycle with Nançay Decameter Array data. *Astron. Astrophys.* 618, A165. doi:10.1051/0004-6361/201833260

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 16 February 2023
DOI 10.3389/fspas.2023.1091967

Check for updates

OPEN ACCESS

EDITED BY
Angeline G. Burrell,
United States Naval Research Laboratory, United States

REVIEWED BY
Fabio Lepreti,
University of Calabria, Italy
Nickolay Ivchenko,
Royal Institute of Technology, Sweden

*CORRESPONDENCE
B. Cecconi,
baptiste.cecconi@observatoiredeparis.psl.eu

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 07 November 2022
ACCEPTED 09 January 2023
PUBLISHED 16 February 2023

CITATION
Cecconi B, Aicardi S and Lamy L (2023), Jupiter radio emission probability tool. *Front. Astron. Space Sci.* 10:1091967. doi: 10.3389/fspas.2023.1091967

COPYRIGHT
© 2023 Cecconi, Aicardi and Lamy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Jupiter radio emission probability tool

B. Cecconi[1,2]*, S. Aicardi[3] and L. Lamy[1,2,4]

[1]LESIA, Observatoire de Paris-PSL, CNRS, Sorbonne Université, Université Paris Cité, Paris, France, [2]ORN, Observatoire de Paris-PSL, Université d'Orléans, CNRS, Nançay, France, [3]DIO, Observatoire de Paris-PSL, CNRS, Paris, France, [4]Aix Marseille Université, CNRS, CNES, LAM, Marseille, France

Jupiter is a source of intense radio emissions in the decametric wavelength range observable from ground (above ~10 MHz) and from space (down to a few kHz). The strong anisotropy of the Jovian radio sources results in characteristic shapes in the temporal-spectral domain, which can be used to identified the various types of Jovian radio components. The *Jupiter Probability Tool* provides users with Jovian radio emission observability predictions, depending on the observers location, and the radio emission class. The application can be used for observation planning or data analysis for ground or space observations.

KEYWORDS

jupiter, radio emissions, space physics, online application, observation planning, python

1 Introduction

Jupiter low frequency radio emissions have been studied since their discovery (Burke and Franklin, 1955), with ground (from 10 to 40 MHz) and space observatories (down to a few kHz), the ground based observations being limited by the ionospheric cutoff at ~10 MHz. The Jovian radio spectrum displays several components, which can be identified in dynamic spectra (time series of spectra), thanks to their temporal-spectral arc-shaped characteristic features (see, e.g.: Zarka, 2000; 2007). Moreover, a subset of Jovian radio components are driven by the interaction between the Jovian magnetic field and the Galilean moons, specifically with the moon Io (Bigg, 1963), but also with Europa and Ganymede (Louis et al., 2017a; Zarka et al., 2018a). The arc-shaped temporal-spectral geometry is a consequence of the radio emission mechanism, which produces a strongly anisotropic beaming pattern. Hence the observer's location is a key parameter for prediction the observability of Jovian radio emissions. Jupiter observation probability maps have been produced with the first published catalogues (see e.g.: Bigg, 1963; Leblanc et al., 1981), relating the observer's longitude in the Jovian System III frame (Higgins et al., 1997) and the phase of Io.

Planning or analysing Jupiter radio observations thus requires to know the observation geometry: observation date, observer's location and phase of the Jovian moons. In this article, we present an online application providing the probability of observing Jovian radio components depending on the observation and Jovian system geometries.

2 Use cases and requirements

The first use case is the analysis of a Jupiter low frequency radio observation. As presented in the introduction section, the interpretation of Jupiter radio emission observations requires an detailed knowledge of the geometry of the Jovian system, together with the observer's location. The observation configuration shall be displayed in a two-dimensional diagram with axes being (a) the longitude of the observer (usually in Jupiter system III longitude) also known as "Central

Meridian Longitude" (or CML), and (b) the selected Moon phase with respect to the observer. Such a diagram is referred to as a "Phase-CML" map. Many Phase-CML maps have been published (see, e.g., Leblanc et al., 1981; Marques et al., 2017; Zarka et al., 2018a). Comparing the selected observation configuration with Phase-CML maps greatly facilitates the Jupiter decametric radio emission observation interpretation. A second critical aspect of the Jovian radio emissions is their shape in the temporal-spectral domain. Comparing the observed shape with predicted ones would also strengthen the scientific interpretation (see, e.g., Louis et al., 2017b).

The second use case is the preparation of a Jupiter low frequency radio observation. The same geometry configuration display against Phase-CML maps allows to select observation times with higher probability of detecting the studied radio component. Furthermore, in order to prepare a ground based observation, the knowledge of Jupiter's elevation as seen from the observer's location is also required.

A series of design requirements has been derived from the two use cases.

- *Time range*: The observation time (or time range) shall be configurable.
- *Predefined observer*: Space missions with a low frequency instrument (e.g., Cassini, Juno, Wind, STEREO-A, STEREO-B, Galileo...), as well as major ground based low frequency radio observatories (e.g., Long Wavelengths Array, Nançay Decameter Array...) shall be easily configurable.
- *Custom ground-based observer*: For ground based observation, it shall be possible to set the location of an observatory (e.g., for radio amateur observatories).
- *Jovian Moon Control*: In the recent studies of Jovian radio emissions, Europa and Ganymede controlled radio emissions have been reported (Louis et al., 2017a; Zarka et al., 2018a) in addition to the long-studied Io-controlled emissions. Control by Callisto and Amalthea are also mentioned in some publications (Marques et al., 2017; Zarka et al., 2018a). The interface shall allow users to select the moon of interest (i.e., Io, Europa, Ganymede, Callisto and Amalthea).
- *Phase-CML maps*: The various published probability or occurrence Phase-CML maps shall be available for comparison (Lamy et al., 2022), which includes Phase-CML maps from Leblanc et al. (1993); Louis et al. (2017a); Marques et al. (2017); Zarka et al. (2018a,b); Jácome et al. (2022) at the time of writing.
- *Observational geometry*: The observational geometry shall be displayed on top of a selected Phase-CML map.
- *Jupiter elevation*: The elevation of Jupiter shall be displayed on the interface in a way the user can tell if the planet is observable at a given time.
- *Dynamic Spectra modeling*: An estimated model of the observed Jovian dynamic spectrum shall be available.

3 Existing tools

We have identified a set of existing tools serving the identified needs (observation planning and observation interpretation).

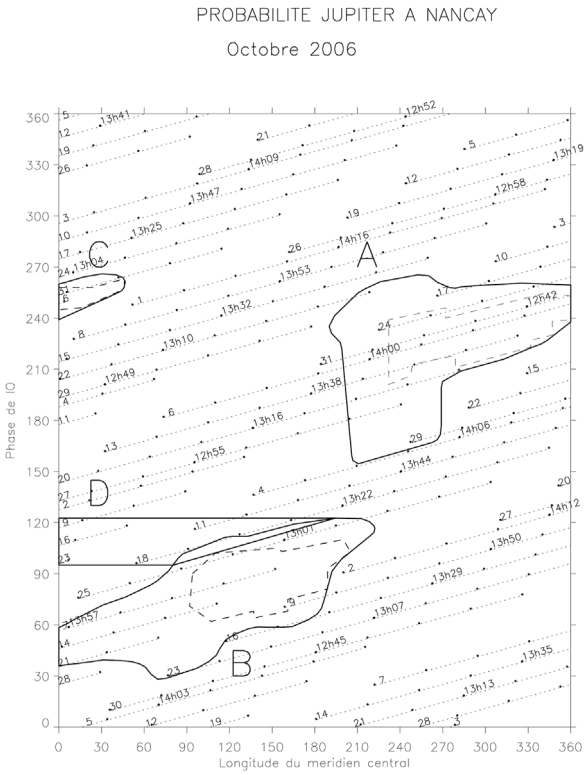


FIGURE 1
NDA provided probability maps for Io-controlled Jovian radio emission, on October 2006. The figure presents the observer's CML on the horizontal axis, and the Io phase on the vertical axis. The dotted lines are the trajectory of the observer in this plane. At the start of each dotted line, the starting day of month of the line is indicated. Each dotted line covers 8 h. The transit time of Jupiter is also provided on the middle tick along this trace. The labels and titles are in French.

3.1 Nançay decameter array probability maps

The Nançay Decameter Array (NDA, Lamy et al., 2017) is a phased array located in Nançay Radioastronomy Observatory (ORN), routinely observing Jupiter and the Sun. In addition to the data products[1], the NDA team is providing its users with a series of monthly Io Phase-CML probability maps, as well as the time of the Jupiter transit at the observer's location, as shown in **Figure 1**.

3.2 Radio jupiter pro

The Radio Sky Publishing[2] team developed tools to help the preparation of Jupiter radio observation, in the frame of the RadioJOVE citizen science project (Thieman et al., 2006; Fung et al., 2020). The *Radio Jupiter Pro* application[3] specifically provides Io Phase-CML probability map, plots indicating the elevation of Jupiter

1 NDA propability maps: https://realtime.obs-nancay.fr/dam/data_dam_affiche/data_dam_affiche.php?lang=en&planete=jupiter&mode=proba

2 Radio Sky Publishing: https://radiosky.com/softwarehome.html

3 Radio Jupiter Pro: https://radiosky.com/rjpro3ishere.html

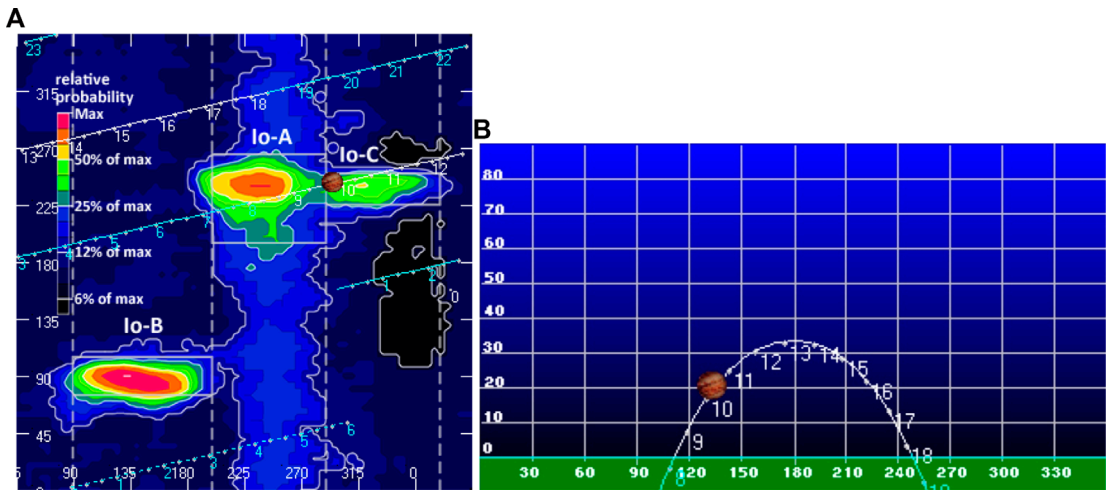


FIGURE 2
Radio Jupiter Pro features: **(A)** Io Phase-CML probability map with the observer's location marked with the Jupiter planet symbol; **(B)** elevation of Jupiter at the location of the observer. Tick marks are labelled every hour on the tracks of Jupiter on each panel.

and the Sun at the observer's location, as well as their location within the antenna beam, when this information is configured. See **Figure 2**.

3.3 Jupiter radio map

Jupiter Radio Map[4] is a java based application providing the Jovian radio observation probability, in an Io Phase-CML map. This application has been developed by a Japanese team (Kochi National College of Technology, Kochi). The tool displays an Io Phase-CML map, overlaid with the trace of the observational geometry, as selected on the user interface. It also used to be published as an iOS application.

3.4 ExPRES

The ExPRES (Exoplanetary and Planetary Radio Emission Simulator, Louis et al., 2019) is a radio observation modeling code dedicated to planetary radio emissions. It allows to construct dynamic spectra predictions, for an observation geometry and a set of radio emission conditions.

4 Online application

The "Jupiter Probability Tool" application has been designed following the requirements presented in **Section 2**. The tools cited in **Section 3** are already implementing part of project requirements. Hence, our application's graphical user interface implements some interface features found in existing tools (e.g., the location of Jupiter on the Phase-CML map, similarly to the Radio Jupiter Pro tool, as shown in **Figure 2**). However, since the previous tools are not open source, none of their code have been reused.

The application is using the SPICE kernel system (Acton et al., 2018) for computing observational and planetary ephemerides. It also displays contextual data, when available, such as observational data (e.g., from the NDA database) and pre-computed modeled data from the ExPRES modeling tool.

The application has been developed as a joint project between the MASER (Measuring, Analysing and Modeling of Emissions in the Radio range) service (Cecconi et al., 2020), and the NDA team (Lamy et al., 2017), with support of PADC (Paris Astronomical Data Centre).

4.1 Graphical user interface

Figure 3 shows a screenshot of the tool user interface. The tagged elements of this interface are described in **Table 1**. The online Jupiter Probability Tool application (Aicardi et al., 2022) is currently available at: https://jupiter-probability-tool.obspm.fr.

4.2 Development details

The application is developed in python, using the *flask*[5] web development library (Grinberg, 2018). The Solar System bodies and spacecraft ephemerides are retrieved using the *python-webgeocalc*[6] library, accessing a dedicated WebGeoCalc server (Acton et al., 2018) installed at the Observatoire de Paris. That server is configured to serve locally SPICE kernels for NASA, ESA and JAXA space mission. The NDA Jupiter data are retrieved using the *das2* (Piker, 2017) protocol, connecting to a *das2* server implemented and maintained by the Nançay Data Centre (CDN), at the Nançay Radio Observatory (ORN). The ExPRES simulation runs are retrieved from a local

4 Jupiter Radio Map: http://jupiter.kochi-ct.jp/jrm/

5 flask: https://flask.palletsprojects.com/en/2.2.x/

6 Python-Webgeocalc: https://webgeocalc.readthedocs.io/en/latest/

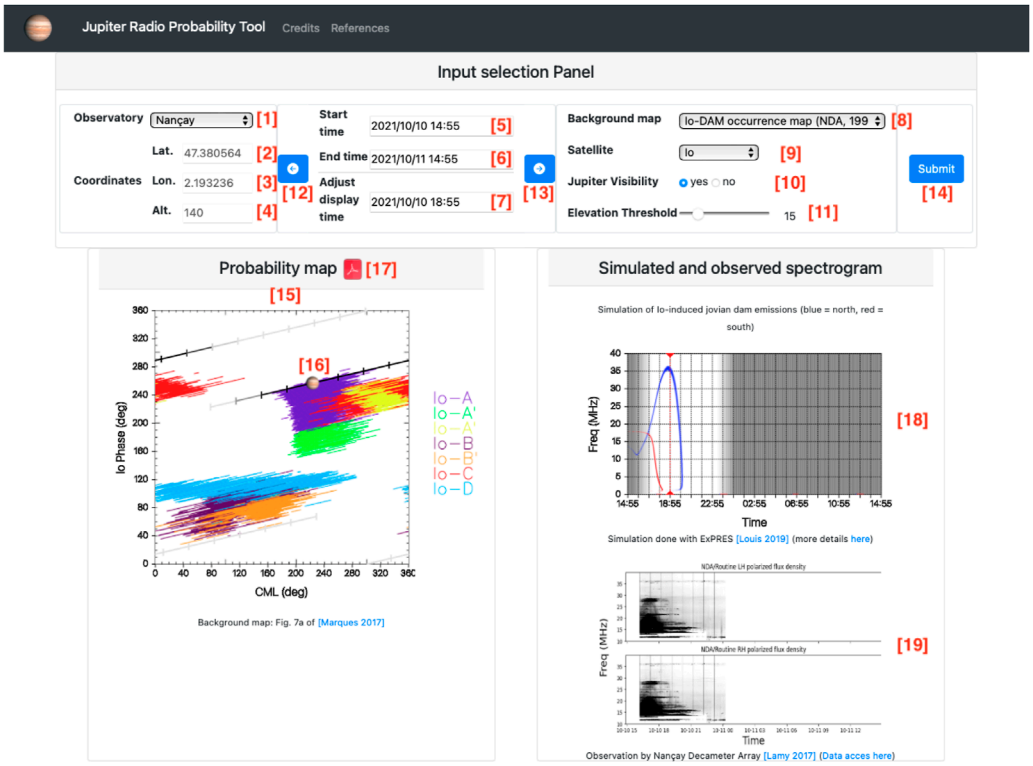


FIGURE 3
Screenshot of the online application. The numbers are referring to the numbered items in **Table 1**.

server as CDF[7] files, and are accessed using the *spacepy*. *PyCDF* (Niehof et al., 2022) module. The application also uses the *Pillow*[8], *numpy* (Harris et al., 2020), *astropy* (The Astropy Collaboration et al., 2018), *matplotlib* (Hunter, 2007) and *sqlalchemy* (Bayer, 2012).

4.3 Ephemerides computation

The observational geometry is computed in a two-step process. First the location of the observer is retrieved with a *STATE_VECTOR* query to the WebGeoCalc server, using the *LATITUDINAL* representation (providing latitude, longitude and distance in the selected frame), the *IAU_JUPITER* reference frame, and the aberration correction set to *CN + S* (see WebGeoCalc documentation for details). In the following listings, we assume the `timestamp` variable is a iterable containing the list of times to be used for computation (list of `datetime.datetime` objects). The `kernels` variable contains the list of meta-kernels to be usedfor the current computation. For Earth-based observatories, the *Solar System Kernels* are selected. For space mission, the specific meta-kernel has to added. Finally, the `observer` variable contains the observer's name.

```
from webgeocal import Calculation
import numpy as np
API = "https://voparis-webgeocalc2.obspm.fr/geocalc/api/"
calc_jupiter = Calculation(
    kernels=kernels,
    api=API,
    times=[timestamp.isoformat() for timestamp in timestamps],
    calculation_type="STATE_VECTOR",
    target="JUPITER",
    observer=observer,
    aberration_correction="CN+S",
    reference_frame="IAU_JUPITER",
    state_representation='LATITUDINAL',
    verbose=False
)
res_jupiter = calc_jupiter.run()
cml = (180-np.array(res_jupiter.get("LONGITUDE")))%360
date_jupiter = res_jupiter.get("TIME_AT_TARGET")
```

Listing 1: First State Vector query to Webgeocalc: computing *CML* and *time at target*.

The CML of the observer is then:

$$\mathrm{CML} = (180 - \mathrm{Long}_{\mathrm{obs}}) \bmod 360 \tag{1}$$

the result of this call also provides a *TIME_AT_TARGET*. The second query retrieves the location of the selected moon in the *IAU_JUPITER*

7 Common Data Format: https://cdf.gsfc.nasa.gov/
8 Pillow: https://pillow.readthedocs.io/en/stable/

TABLE 1 Features of the application graphical user interface, with their type. ***List*** items are drop-down menus with predefined values. ***Number*** items are text boxes with decimal numbers input. ***Epoch*** items are text boxes with date and time input. ***Switch*** items are buttons to activate a feature. ***Button*** items are used to trigger an action. ***Display*** items are showing the result of the query. ***Cursor*** items are actionable objects allowing to select a location within a range.

Tag	Type	Description
1	List	*Observatory selection*: A list of predefined observatories. At the time of writing of the paper, the list of observatories is: NDA (Nançay Decameter Array, Nancay, France), UTR-2 (Kharkiv, Ukraine), Iitate (Japan), LWA (Long Wavelength Array, New Mexico, United States), I-LOFAR (Irish LOFAR station), and the Juno spacecraft. In order to manually enter the location of an observatory, use the "Custom (Earth-based)" entry
2,3,4	Number	*Observer's coordinates*: For ground based observatories, the latitude 1) in *degrees*, longitude 2) in *degrees*, and altitude in *m* above see level of the observatory (automatically filled in upon selection of the predefined observatory)
5,6	Epoch	*Time range selection*: Start time 5) and End time 6) for the display
7	Epoch	*Sliding Time selection*: Selected observation configuration within the time range
8	List	*Phase-CML map selection*: A list of Phase-CML maps
9	List	*Satellite selection*: A list of Jovian moon: Io, Europa, Ganymede, Callisto, Amalthea
10	Switch	*Jupiter visibility*: Activate the Jupiter visibility from the observer feature
11	Switch	*Jupiter elevation threshold*: Set the elevation threshold if the "Jupiter visibility" feature is active
12,13	Button	*Time selection*: Move the time range backward 12) or forward 13)
14	Button	*Submit*: Submit the form and update the display panels
15	Display	*Probability map*: The selected Phase-CML map 8) is displayed, with the track of the observation geometry [using the time range 5) to 6)] and the "Jupiter" symbol placed at the time 7). The panel can be used to select the display time by clicking on the observation track. If the "Jupiter visibility" feature is active, the track shows shading depending on the elevation of Jupiter. The reference of the map is given below the figure
16	Cursor	*Observational geometry*: The marker of the selected observational geometry if a "Jupiter" planet symbol
17	Button	*Download map*: The figure can be downloaded as a PDF file
18	Display	*Modeled dynamic spectrum*: A modeled dynamic spectrum, using ExPRES, with a background shading if the "Jupiter visibility" feature is active. A vertical dashed line with markers at the top and bottom is showing the display time
19	Display	*Observation*: If observations are available for the selected observatory, and the selected time range, they are displayed in this panel. Currently, only NDA data can be displayed here

reference frame, with a similar *STATE_VECTOR* query, using the *TIME_AT_TARGET*, and no aberration correction. In the following listing, the `satellite` variable contains the name of the selected moon.

```
calc_moon = Calculation(
    kernels=kernels,
    api=API,
    times=date_jupiter,
    calculation_type="STATE_VECTOR",
    target=satellite,
    observer="JUPITER",
    aberration_correction="NONE",
    reference_frame="IAU_JUPITER",
    state_representation="LATITUDINAL",
    verbose=False
)
res_moon = calc_moon.run()
phase = (180+cml+np.array(res_moon.get("LONGITUDE"))
    )%360
```

Listing 2: Second State Vector query to Webgeocalc: computing the moon's *Phase*.

The Phase of the moon is then:

$$\text{Phase} = \left(180 + \text{CML} + \text{Long}_{\text{moon}}\right) \bmod 360 \tag{2}$$

The elevation of Jupiter for ground observatories is retrieved with a *STATE_VECTOR* call on the *WebGeoCalc* server, using the *RA_DEC* representation. The obtained sky coordinates are transformed into alt-azimuthal coordinates using *astropy* and the observatory location.

5 Summary and perspectives

The Jupiter Probability Tool application is a science ready tool for preparing and analysing Jupiter radio observations. It has already been used in several studies. Louis et al. (2021) prepared a series of joint Jupiter decametric observations on three international LOFAR stations (Ireland, France and Germany) using the Jupiter Probability Tool. Lamy et al. (2022a) is a comment on a published paper. The authors made use of the application to support their argumentation. Lamy et al. (2022b) also made use of the application to confirm that the observed emissions were Io-controlled emissions.

A series of improvement and new features are planned for the next versions of the application. Firstly, new observatories shall be implemented, especially space missions with low frequency radio instrumentation, such as, e.g., Cassini; STEREO-A; STEREO-B; WIND; Galileo; JUICE; Voyager 1; Voyager 2; and Mars Express.

Improvement of the application to adhere to the FAIR (Findable, Accessible, Interoperable, and Reusable) principles (Wilkinson et al., 2016) are also in preparation. One of the planned feature in this context is the addition of provenance (Servillat et al., 2022) information to the output figures (with the provision of the list of citations to be used if the figures are included in a scientific publication), thus improving the reusability of the application products.

Data availability statement

The code of the application is currently not open (except the lines presented in the paper). The data used in this application are all openly available: ExPRES collection (Louis et al., 2020), NDA Jupiter Routine collection (Lamy et al., 2021), Phase-CML maps (Lamy et al., 2022), and the Juno SPICE kernel dataset (Semenov et al., 2017).

Author contributions

BC and LL prepared the application requirements. SA developed the application. BC and LL reviewed and commented the application during the development process. BC wrote the first version of the papier. SA and LL commented and proofread the manuscript.

Funding

The work is supported by the Europlanet 2024 Research Infrastructure project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871149.

Acknowledgments

The authors acknowledge support from Observatoire de Paris, CNES (Centre National d'Etudes Spatiale) and CNRS (Centre National de la Recherche Scientifique) through two of its funding programmes: PNP (Programme National de Planétologie) and PNST (Programme National des relations Soleil-Terre). Support from Paris Astronomical Data Centre (PADC) is acknowledged.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Acton, C., Bachman, N., Semenov, B., and Wright, E. (2018). A look towards the future in the handling of space science mission geometry. *Planet. Space Sci.* 150, 9–12. doi:10.1016/j.pss.2017.02.013

Aicardi, S., Cecconi, B., and Lamy, L. (2022). *Jupiter radio probability tool*. PADC/MASER. doi:10.25935/TV9M-HX48

Bayer, M. (2012). "Sqlalchemy," in *The architecture of open source applications volume II: Structure, scale, and a few more fearless hacks* Editors A. Brown, and G. Wilson Available at: http://aosabook.org/en/sqlalchemy.html.

Bigg, K. E. (1963). Influence of the satellite Io on jupiter's decametric emission. *Nature* 203, 1008–1010. doi:10.1038/2031008a0

Burke, B. F., and Franklin, K. L. (1955). Observations of a variable radio source associated with the planet jupiter. *J. Geophys. Res.* 60, 213–217. doi:10.1029/jz060i002p00213

Cecconi, B., Loh, A., Sidaner, P. L., Savalle, R., Bonnin, X., Nguyen, Q. N., et al. (2020). Maser: A science ready toolbox for low frequency radio Astronomy. *Data Sci. J.* 19, 1062. doi:10.5334/dsj-2020-012

Fung, S. F., Typinski, D., Flagg, R., Ashcraft, T., Greenman, W., Higgins, C., et al. (2020). Propagation teepee: A possible high-frequency (15–30 mhz) remote lightning signature identified by citizen scientists. *Geophys. Res. Lett.* 47, e2020GL087307. doi:10.1029/2020GL087307

Grinberg, M. (2018). *Flask web development: Developing web applications with python*. O'Reilly Media, Inc.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Higgins, C., Carr, T. D., Reyes, F., Greenman, W., and Lebo, G. (1997). A redefinition of Jupiter's rotation period. *J. Geophys. Res.* 102, 22033–22041. doi:10.1029/97ja02090

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Jácome, H. R. P., Marques, M. S., Zarka, P., Echer, E., Lamy, L., and Louis, C. K. (2022). Search for jovian decametric emission induced by Europa on the extensive Nançay decameter array catalog. *Astronomy Astrophysics* 665, A67. doi:10.1051/0004-6361/202244246

Lamy, L., Cecconi, B., and Aicardi, S. (2022). *Collection of jovian radio emission probability and occurrence maps*. PADC/MASER. doi:10.25935/1ZBG-EK27

Lamy, L., Cecconi, B., Aicardi, S., Louis, C. K., K. Louis, C., et al.Station de Radioastronomie de Nancay, Observatoire de Paris, Universite PSL, CNRS, Univ. Orleans, 18330 Nancay, France (2022a). Comment on "Locating the source field lines of Jovian decametric radio emissions" by YuMing Wang et al. *Earth Planet. Phys.* 6, 10–12. doi:10.26464/epp2022018

Lamy, L., Colomban, L., Zarka, P., Prangé, R., Marques, M. S., Louis, C. K., et al. (2022b). Determining the beaming of Io decametric emissions: A remote diagnostic to probe the io-jupiter interaction. *J. Geophys. Res. Space Phys.* 127, 30160. doi:10.1029/2021ja030160

Lamy, L., Le Gall, A., Cecconi, B., Loh, A., Renaud, P., Denis, L., et al. (2021). *Nançay decameter array (NDA) routine jupiter observation data collection*. PADC/CDN. doi:10.25935/DV2F-X016

Lamy, L., Zarka, P., Cecconi, B., Klein, K.-L., Masson, S., Denis, L., et al. (2017). 1977–2017: 40 Years of decametric observations of jupiter and the Sun with the Nançay decameter array. *Planet. Sol. Heliospheric Radio Emiss. (PRE VIII)* 8, 455–466. doi:10.1553/pre8s455

Leblanc, Y., Gerbault, A., Denis, L., and Lecacheux, A. (1993). A catalogue of Jovian decametric radio observations from January 1988 to December 1990. *Astron. Astrophys. Suppl. Ser.* 98, 529.

Leblanc, Y., Noë, J. d. l., Genova, F., Gerbault, A., and Lecacheux, A. (1981). A catalogue of jovian decametric radio observzations from January 1978 to December 1979. *Astron. Astrophys. Suppl. Ser.* 46, 135.

Louis, C., Cecconi, B., and Loh, A. (2020). *ExPRES jovian radio emission simulations data collection (version 01)*. PADC/MASER. doi:10.25935/KPGE-ZB59

Louis, C. K., Hess, S. L. G., Cecconi, B., Zarka, P., Lamy, L., Aicardi, S., et al. (2019). ExPRES: An exoplanetary and planetary radio emissions simulator. *Astronomy Astrophysics* 627, A30. doi:10.1051/0004-6361/201935161

Louis, C. K., Jackman, C. M., Griessmeier, J.-M., Wucknitz, O., McKenna, D. J., Murphy, P., et al. (2021). *Observing Jupiter's radio emissions using multiple LOFAR stations: A first case study of the io-decametric emission using the Irish IE613, French FR606 and German DE604 stations*. MNRAS in press. doi:10.48550/arxiv.2111.09599

Louis, C. K., Lamy, L., Zarka, P., Cecconi, B., and Hess, S. L. G. (2017a). Detection of jupiter decametric emissions controlled by Europa and Ganymede with voyager/PRA and Cassini/RPWS. *J. Geophys. Res. Space Phys.* 122, 9228–9247. doi:10.1002/2016ja023779

Louis, C. K., Lamy, L., Zarka, P., Cecconi, B., Imai, M., Kurth, W. S., et al. (2017b). Io-Jupiter decametric arcs observed by Juno/Waves compared to ExPRES simulations. *Geophys. Res. Lett.* 1, 9225–9232. doi:10.1002/2017gl073036

Marques, M. S., Zarka, P., Echer, E., Ryabov, A. V., Alves, M. V., Denis, L., et al. (2017). Statistical analysis of 26 yr of observations of decametric radio emissions from Jupiter. *Astronomy Astrophysics* 604, A17. doi:10.1051/0004-6361/201630025

Niehof, J. T., Morley, S. K., Welling, D. T., and Larsen, B. A. (2022). The SpacePy space science package at 12 years. *Front. Astronomy Space Sci.* 9, 612. doi:10.3389/fspas.2022.1023612

Piker, C. W. (2017). *Das2: Interface Control document*. Zenodo. doi:10.5281/zenodo.3588535

The Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., et al. (2018). The astropy project: Building an open-science project and status of the v2.0 core package. *Astrophys. J.* 156, 123. doi:10.3847/1538-3881/aabc4f

Semenov, B. V., Acton, C. H., and Costa Sitja, M. (2017). *JUNO SPICE KERNELS V1.0, JNO-J/E/SS-SPICE-6-V1.0*. NASA Planetary Data System. doi:10.17189/1520117

Servillat, M., Bonnarel, F., Louys, M., and Sanguillon, M. (2022). "Practical provenance in Astronomy," in *Astronomical society of the pacific conference series*. Editors J. E. Ruiz, F. Pierfedereci, and P. Teuben (Astronomical Society of the Pacific Conference Series), 532, 619.

Thieman, J., Flagg, R., Sky, J., Higgins, C., Garcia, L., Reyes, F., et al. (2006). The radio JOVE project: Amateurs working with professionals. *Planet. Radio Emiss. VI*, 225–230.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* 3, 160018. doi:10.1038/sdata.2016.18

Zarka, P., Marques, M. S., Louis, C., Ryabov, V. B., Lamy, L., Echer, E., et al. (2018a). Jupiter radio emission induced by Ganymede and consequences for the radio detection of exoplanets. *Astronomy Astrophysics* 618, A84. doi:10.1051/0004-6361/201833586

Zarka, P., Marques, M. S., Louis, C., Ryabov, V. B., Lamy, L., Echer, E., et al. (2018b). Radio emission from satellite-Jupiter interactions (especially Ganymede). *Planet. RADIO Emiss. VIII* 8, 45–58. doi:10.1553/pre8s45

Zarka, P. (2000). "Radio emissions from the planets and their moons," in *Radio astronomy at long wavelengths, geophysical monograph series*. Editors R. G. Stone, K. W. Weiler, M. L. Goldstein, and J.-L. Bougeret (Washington, DC: American Geophysical Union), 167–178. doi:10.1029/GM119p0167

Zarka, P. (2007). Plasma interactions of exoplanets with their parent star and associated radio emissions. *Planet. Space Sci.* 55, 598–617. doi:10.1016/j.pss.2006.05.045

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 23 February 2023
DOI 10.3389/fspas.2023.1076726

Check for updates

OPEN ACCESS

EDITED BY
Leslie Lamarche,
SRI International, United States

REVIEWED BY
Stefano Markidis,
KTH Royal Institute of Technology, Sweden
Feng Wang,
Guangzhou University, China

*CORRESPONDENCE
Will T. Barnes,
will.t.barnes@nasa.gov

†These authors have contributed equally to this work and share first authorship

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 21 October 2022
ACCEPTED 31 January 2023
PUBLISHED 23 February 2023

CITATION
The SunPy Community, Barnes WT, Christe S, Freij N, Hayes LA, Stansby D, Ireland J, Mumford SJ, Ryan DF and Shih AY (2023), The SunPy Project: An interoperable ecosystem for solar data analysis.
Front. Astron. Space Sci. 10:1076726.
doi: 10.3389/fspas.2023.1076726

COPYRIGHT
© 2023 The SunPy Community, Barnes, Christe, Freij, Hayes, Stansby, Ireland, Mumford, Ryan and Shih. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

The SunPy Project: An interoperable ecosystem for solar data analysis

The SunPy Community, Will T. Barnes[1,2*†], Steven Christe[1†], Nabil Freij[3,4†], Laura A. Hayes[5†], David Stansby[6†], Jack Ireland[1], Stuart J. Mumford[7], Daniel F. Ryan[8] and Albert Y. Shih[1]

[1]NASA Goddard Space Flight Center, Greenbelt, MD, United States, [2]Department of Physics, American University, Washington, DC, United States, [3]Lockheed Martin Solar and Astrophysics Laboratory, Palo Alto, CA, United States, [4]Bay Area Environmental Research Institute, Moffett Field, CA, United States, [5]ESTEC, European Space Agency, Noordwijk, Netherlands, [6]Advanced Research Computing Centre, University College London, London, United Kingdom, [7]Aperio Software Ltd., Leeds, England, [8]University of Applied Sciences and Arts Northwest Switzerland, Windisch, Switzerland

The SunPy Project is a community of scientists and software developers creating an ecosystem of Python packages for solar physics. The project includes the sunpy core package as well as a set of affiliated packages. The sunpy core package provides general purpose tools to access data from different providers, read image and time series data, and transform between commonly used coordinate systems. Affiliated packages perform more specialized tasks that do not fall within the more general scope of the sunpy core package. In this article, we give a high-level overview of the SunPy Project, how it is broader than the sunpy core package, and how the project curates and fosters the affiliated package system. We demonstrate how components of the SunPy ecosystem, including sunpy and several affiliated packages, work together to enable multi-instrument data analysis workflows. We also describe members of the SunPy Project and how the project interacts with the wider solar physics and scientific Python communities. Finally, we discuss the future direction and priorities of the SunPy Project.

KEYWORDS

solar physics, sunpy, heliophysics, Python (programming language), data analysis—methods

1 Introduction

The SunPy Project is an organization whose mission is to develop and facilitate a high-quality, easy-to-use, community-led, free and open-source solar data analysis ecosystem based on the scientific Python environment. The vision of the project is to build a diverse and inclusive solar physics and heliophysics community that supports scientific discovery and enables reproducibility through the development of accessible, open-source software (Bobra et al., 2020). To achieve this mission and to make this vision a reality, the SunPy Project maintains and guides the development of a number of Python packages including the sunpy core package, and organizes educational activities around the use of Python for solar-physics research.

As the scientific Python environment matured in the early 2010s (Hunter, 2007; Harris et al., 2020; Virtanen et al., 2020), the development of a Python package devoted to solar physics became viable. This led to the founding of the SunPy Project in 2011 by

scientists at NASA Goddard Space Flight Center. The goal of the SunPy Project at that time was to develop a package that provided the core functionality needed for solar data analysis in Python. To distinguish the software package from the wider project, this original package is now known as the sunpy core package (The SunPy Community et al., 2020). As the SunPy Project and sunpy grew, an ecosystem of affiliated packages (**Section 2.2**) was developed to keep the sunpy core package from becoming too large and difficult to manage.

The SunPy Project is committed to the principles of open development. All code is hosted and openly-developed on GitHub[1] in order to enable anyone to contribute code or provide feedback. All packages within the SunPy Project must be under an Open Source Initiative (OSI)[2] approved license. Discussion is hosted on several open communication channels which include weekly community calls, mailing lists, a Discourse forum, and instant messaging *via* Matrix[3]. Additionally, the SunPy Project has a code of conduct[4] to ensure that communication within the project is open, considerate, and respectful.

The aim of this paper is to give a high level description of the SunPy Project, including its various components, and to describe the direction of the project in the coming years. **Section 2** describes the various Python packages that form the project, including both the sunpy core package (**Section 2.1**) and the various affiliated packages (**Section 2.2**). **Section 3** gives an overview of the roles within the project and describes how to become involved with the SunPy Project. **Section 4** describes the various activities of the project within the broader solar physics community. Finally, **Section 5** lays out a vision for the future of the SunPy Project.

2 Code

2.1 The sunpy core package

The sunpy package is the central pillar of the SunPy Project (The SunPy Community et al., 2020) and provides the fundamental tools for accessing, loading, and interacting with solar physics data in Python. As we will discuss in **Section 2.2**, sunpy functions as one part of a larger ecosystem of packages for doing solar physics research in Python. While other packages in the ecosystem may focus on particular analysis techniques or analyzing data from specific instruments, the sunpy "core" package is focused on providing general tools for working with solar physics data. As an example, coordinate transformations between common solar coordinate systems are provided by the sunpy core package because they are needed for the analysis of nearly all solar imaging data and are critical for performing multi-instrument studies. However, correcting an AIA image to account for instrument degradation would not belong in sunpy because it is specific to data from one instrument. This allows the sunpy core package to be relatively small in size, thereby assuring its maintainability over time.

The primary components of the sunpy package are described briefly in the following paragraphs. For a more in-depth description of each of these components, see The SunPy Community et al. (2020, **Section 4**). The full documentation of the sunpy Application Programming Interface (API) is provided in the hosted online documentation[5].

2.1.1 Components of the core package

To search for and download data, sunpy provides the `Fido` interface for searching across a variety of data providers [e.g., the Virtual Solar Observatory (VSO)[6], or the Joint Science Operations Center (JSOC)[7]] maintained within the solar community. Internally, `Fido` is both an interface that defines the search API for creating data queries as well as a collection of client classes that provide a translation between this user-facing API and the search parameters accepted by individual data providers. A complete list of all supported data sources is provided in the documentation for using `Fido`[8]. **Section 4.1.1** of (The SunPy Community et al., 2020) also provides a comprehensive discussion of the data sources that `Fido` searches by default. Additionally, `Fido` can also be extended to search additional data sources that may not be included in sunpy (e.g., the Solar Orbiter Archive, see **Section 2.2.2**). Attributes such as time, wavelength, and instrument name, among others, can be used to filter these search results. By providing a single interface to many disparate data sources, sunpy, *via* `Fido`, easily enables multi-instrument research workflows.

Once a user has downloaded data, the `TimeSeries` and `Map` objects can be used to load and visualize time series and two-dimensional image data, respectively. These objects hold the data alongside the associated metadata in order to perform metadata-aware operations such as concatenation for time series or cropping for image data. In the case of `Map`, a World Coordinate System (WCS, e.g., Greisen and Calabretta, 2002) is also constructed from the associated metadata to enable easy mapping between pixel and world coordinates *via* `astropy`. In nearly all cases, solar image data is stored in the FITS format (Wells et al., 1981) which has an accompanying well-defined metadata standard (Pence et al., 2010). The accompanying metadata for each `Map` object adheres to this standard. Solar time series data, however, do not have a standard metadata or file format and are stored in a variety of file formats, including FITS, netCDF, JSON, or plain text. As such, the metadata associated with each `TimeSeries` object is much more sparse compared to `Map`, but at minimum will include the time of each observation as well as some information about the associated instrument that made the observation.

Additionally, by extending the astropy coordinates framework (See **Section 3.3** of The Astropy Collaboration et al., 2018 for more details), sunpy provides definitions of, and transformations between, common solar coordinate systems. Coordinates expressed

1 https://github.com/sunpy/sunpy
2 https://opensource.org
3 https://matrix.org/
4 https://sunpy.org/coc
5 The sunpy API is fully documented here: https://docs.sunpy.org
6 https://sdac.virtualsolar.org/cgi/search
7 http://jsoc.stanford.edu
8 https://docs.sunpy.org/en/stable/guide/acquiring_data/fido.html

using these frames can be used to represent the positional information of solar features and events. sunpy implements both observer-dependent (e.g., helioprojective Cartesian) and observer-independent (e.g., Stonyhurst heliographic) coordinate frames (Thompson, 2006). Each Map object instance also carries with it the corresponding coordinate frame of that image and the coordinate of the observer as defined by the position of the observatory given in the associated metadata.

2.1.2 Testing infrastructure

Sunpy includes thousands of unit, regression and integration tests that are run using the pytest testing framework. This test suite is run on every pull request opened on the sunpy GitHub repository using GitHub Actions to ensure that contributions to the codebase do not lead to unexpected regressions. A full description of our testing practices can be found in our developer documentation[9].

2.1.3 Release schedule

There is a new release of the core package with feature enhancements approximately every 6 months. Every other release is designated a long term support (LTS) release and receives bug fixes for a year rather than for 6 months. Additionally, there are bug fix releases every month. For each release a digital object identifier (DOI) is automatically generated and a record is created on Zenodo.[10] By providing regularly scheduled, versioned releases of sunpy, the SunPy Project enables reproducibility. For example, if a researcher is attempting to reproduce a result from a paper that used sunpy v2.0.2, she can create a new virtual environment and install that exact version of sunpy, even if the current version is many versions ahead of v2.0.2.

This release process is completely automated through GitHub Actions.[11] When a release is tagged, an action is triggered that tests the package on all supported versions of Python and all supported operating systems. If the packages build successfully, they are automatically uploaded to Python Package Index (PyPi) and subsequently the release is updated on conda-forge.

2.2 Affiliated packages

As the sunpy package grew and the amount of domain- and instrument-specific code being developed in Python increased, it became increasingly challenging to store and maintain the functionality needed for all solar physics research in one package. As such, the affiliated package system was introduced (Mumford and Christe, 2014) so that the sunpy core package could be generic enough for other packages to build on. The goal of this system is to support and promote software packages outside the scope of the sunpy core package, and to provide guidance to developers in implementing and maintaining the specific functionality provided by an affiliated package. This fosters code-ownership while ensuring the set of affiliated packages are interoperable and follow a set of common standards (**Section 2.2.1**). The SunPy Project provides development support through our community development efforts and by providing a package template as a foundation. In addition, affiliated packages are advertised at conferences and workshops where a SunPy poster, talk, or tutorial is given.

As a result of the creation of the affiliated package ecosystem, components of the sunpy core package that were tied directly to specific instruments or data analysis methods have recently been moved out into other affiliated packages. One example of this is aiapy (Barnes et al., 2020), a package for processing data from the Atmospheric Imaging Assembly (AIA, Lemen et al., 2012) on the *Solar Dynamics Observatory* (SDO, Pesnell et al., 2012). Prior to version 2.1, sunpy included functionality for calibrating level 1 AIA data. In 2019, in collaboration with the SunPy Project, the AIA instrument team began developing aiapy to provide a number of AIA-specific analysis routines in Python, including the aforementioned calibration software. aiapy became an affiliated package in 2020 and the AIA-specific functionality that previously lived in sunpy was deprecated and subsequently removed. This relocation of the code allows the AIA instrument team to have full autonomy over their calibration routines and release updates to their software on a more frequent timescale than that of the sunpy core package. At the same time, aiapy users and developers are able to take full advantage of the SunPy Project ecosystem.

Outside of the current list of affiliated packages, current and future NASA and ESA missions[12], as well as ground-based telescopes, such as the Daniel K. Inouye Solar Telescope (DKIST), have begun developing user tools for data analysis and/or pipelines for data calibration built on top of the SunPy ecosystem. While these packages are not yet affiliated, the SunPy Project has assisted in coordinating development efforts between these teams in order to foster a more interoperable ecosystem.

2.2.1 Application process

The affiliated package application process is completed in the open on GitHub and is open to all, both individuals and larger collaborations (e.g., instrument teams). To begin the process, an applicant opens an issue on the SunPy Project website GitHub repository[13] and provides details about the package, including the package name, the maintainers, a link to the code repository, and a link to the documentation. The Affiliated Package Liaison (**Section 3.2**) then selects a SunPy Project member to review the candidate affiliated package against the following criteria:

- *functionality*—is the package relevant to the solar physics community?
- *integration*—does the package make use of the existing ecosystem?

9 A complete guide to running the tests and the associated infrastructure can be found here: https://docs.sunpy.org/en/latest/dev_guide/contents/tests.html

10 The most current release on Zenodo can be found here: https://doi.org/10.5281/zenodo.7314636

11 The GitHub Actions templates used are available here: https://github.com/OpenAstronomy/github-actions-workflows

12 This includes, but is not limited to, the Interface Region Imaging Spectrometer (IRIS), several instruments on *Solar Orbiter*, as well as the X-Ray Telescope (XRT) and the Extreme ultraviolet Imaging Spectrometer (EIS) onboard *Hinode*

13 https://github.com/sunpy/sunpy.org

- *documentation*—is there hosted documentation, including examples and an API reference?
- *testing*—are there automatically run tests and is the coverage extensive?
- *duplication*—does the package duplicate existing functionality in the ecosystem?
- *community*—is there a code of conduct and do the developers engage the wider community?
- *development status*—is the project actively maintained, including versioned releases?

The assigned project member then scores the package in each category using a "stoplight" system (i.e., a package is scored green, orange, or red in each category). A detailed description of each criterion and the scoring for each is available on the affiliated package page of the SunPy Project website[14]. The submitting author of the affiliated package may also request an alternate reviewer, in which case the Affiliated Package Liaison will assign a new SunPy Project member to review the package. At the end of the review, the candidate package is either accepted, marked as provisional, or not accepted. If the package is accepted, the affiliated package is added to the list of affiliated packages on the SunPy Project website. If the package is marked as provisional or is not accepted, the reviewer and the Affiliated Package Liaison will work with the package authors to help them achieve provisional or accepted status. Accepted affiliated packages are reviewed once a year to ensure the interoperability of the ecosystem does not regress and that affiliated packages are actively maintained.

In all cases, the goal of the affiliated package review process is to broaden the ecosystem of tools for solar data analysis in Python. These criteria are not meant to be exclusionary, but rather to ensure interoperability and consistency across the ecosystem for the benefit of both users and developers. Interoperability in this context, means that affiliated packages should make use of the existing `sunpy` core data structures, (e.g., `Map` and `Timeseries`), *in lieu* of their own custom data structures. In the context of searching for and downloading data, affiliated packages should use the `Fido` interface and extend `Fido` for additional data sources as needed.

2.2.2 Current ecosystem

At the time of writing, the SunPy Project has a rich and growing ecosystem of affiliated packages. In addition to the `sunpy` core package, the affiliated package ecosystem includes:

- `aiapy` for functionality specific to the AIA instrument (Barnes et al., 2020)
- `ndcube` for generic handling of *N*-dimensional data sets with a world coordinate system (WCS) (Ryan et al., 2021).
- `pfsspy` for magnetic-field extrapolation (Stansby et al., 2020)
- `sunkit-instruments` for instrument-specific code that does not have a dedicated package (Ryan et al., 2022).
- `sunkit-image` for solar-specific image analysis or reduction techniques (Freij et al., 2022).
- `sunpy-soar`[15] for querying the Solar Orbiter Archive (SOAR)[16].

To demonstrate how several of the affiliated packages can be used together with `sunpy` in a scientific workflow, we show an example in **Figure 1** of how coronal loop structures can be analyzed using potential magnetic field extrapolations and multi-point extreme ultraviolet (EUV) observations. We have included a Jupyter notebook that illustrates each step of this workflow in the GitHub repository that accompanies this paper[17].

First, we use the `Fido` interface provided by `sunpy` to search for and download a synoptic magnetogram from the Helioseismic Magnetic Imager (HMI, Scherrer et al., 2012) on SDO for Carrington rotation 2255 which began on 2022-03-08. This is shown in the left panel in **Figure 1A**. Next, we identify active region NOAA 12976 which appeared near disk center, as seen from SDO, at 2022-03-29 21:04. The red box overlaid on the synoptic magnetogram is centered on the active region when it appeared at disk center at a Carrington longitude of 65°.

Since we are interested in the EUV observations of active region 12976, we also use `Fido` to query the VSO for data from AIA on SDO and the Extreme Ultraviolet Imager (EUVI) on the *Solar Terrestrial Relations Observatory* (STEREO, Howard et al., 2008). Additionally, we use the `sunpy-soar` package to allow `Fido` to search for and download data from the SOAR. Here, we query the SOAR for data from the Extreme Ultraviolet Imager (EUI, Rochus et al., 2020) on *Solar Orbiter* (Müller et al., 2020).

Figure 1B shows full disk EUV images from AIA (left), the full-sun imager (FSI) on EUI (middle), and EUVI on the STEREO-A spacecraft (right). We use the `aiapy` package to correct the AIA image (middle panel) for instrument degradation and update the pointing information. The red box in each panel is centered on active region 12976, as seen from the respective spacecraft, and has a width and height of 700 arcseconds. The top right panel of **Figure 1** shows the Stonyhurst heliographic longitude and radius (in AU) of SDO, STEREO-A and *Solar Orbiter* as derived from the observer location metadata of each image.

Viewing the active region from the vantage points of these three spacecraft (separated by >90°), we gain a better understanding of its three-dimensional structure. Additionally, we use the `pfsspy` package to compute a potential field extrapolation from the corresponding synoptic magnetogram as shown in **Figure 1A**. We trace field lines from areas of negative magnetic flux inside the red box corresponding to active region 12976. The resulting field lines are overlaid in green on top of the cutouts from each EUV image in **Figure 1C**. Each field line traced using `pfsspy` is an `astropy` coordinate object expressed in terms of a Carrington heliographic coordinate frame defined in `sunpy`. As such, it is straightforward to transform each field line to the observer-dependent coordinate frame of each image as defined by the corresponding observatory using the plotting functionality

14 https://sunpy.org/project/affiliated

15 https://github.com/sunpy/sunpy-soar

16 https://soar.esac.esa.int/soar/

17 The GitHub repository for this paper, including the complete text and all code to generate **Figure 1**, can be found at https://github.com/sunpy/sunpy-frontiers-paper

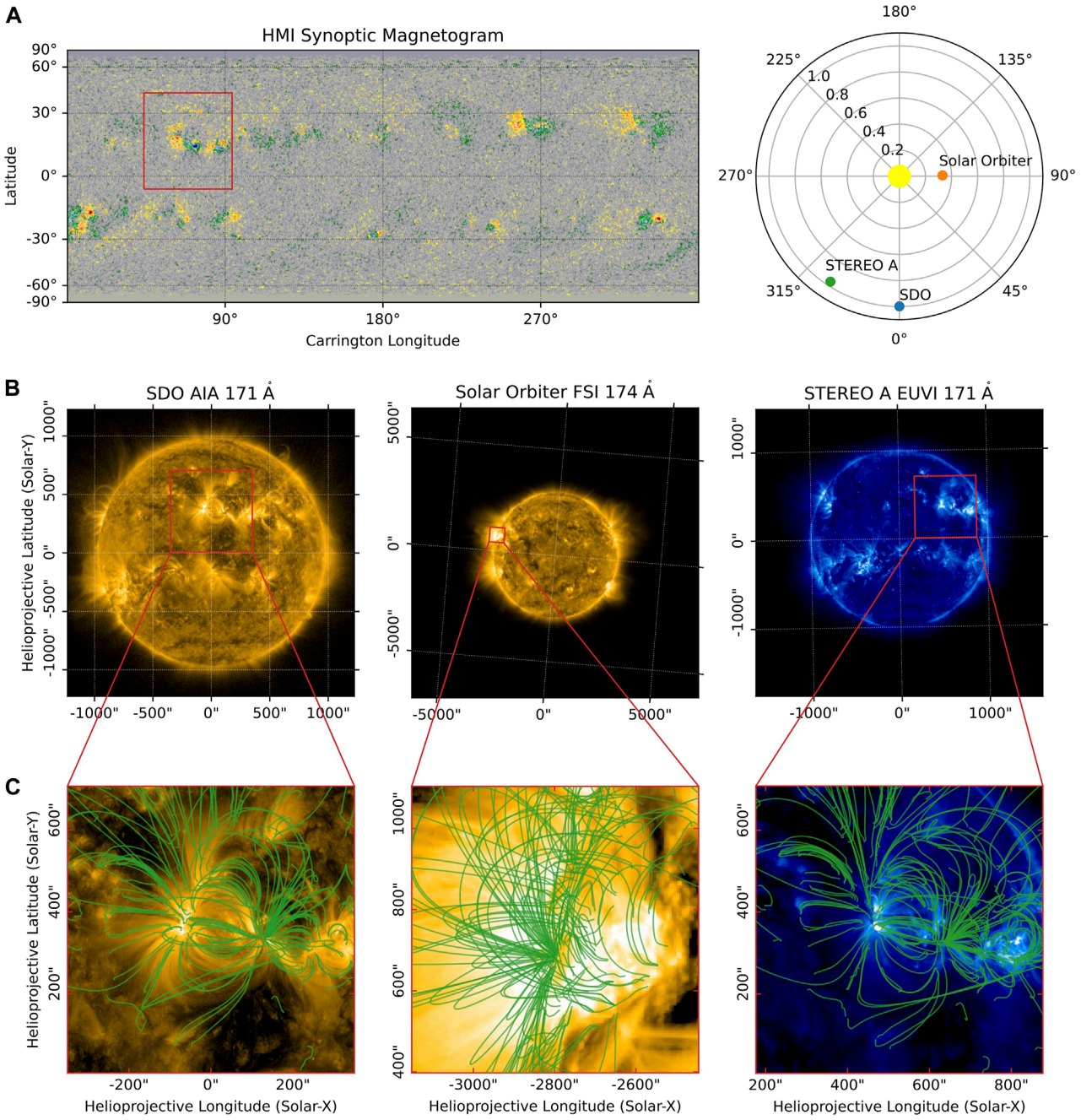


FIGURE 1
Illustration of multiple affiliated packages, including `sunpy`, `sunpy_soar`, `aiapy`, and `pfsspy`, working together. **(A)** The left panel shows the HMI synoptic magnetogram for Carrington rotation 2255. The red box is centered on the active region. The right panel shows the Stonyhurst heliographic longitude and radius (in AU) for SDO, STEREO A, and Solar Orbiter on 2022-03-29. **(B)** Full disk images from SDO AIA at 171 Å (left), SolO FSI at 174 Å (middle), and STEREO-A EUVI at 171 Å (right). All three images were downloaded using `sunpy` along with `sunpy_soar` to query the SOAR for the *Solar Orbiter* image. The AIA image was calibrated using `aiapy`. The red box in each panel is centered on the AR shown in the top panel. **(C)** Cutouts of the regions denoted in each image in **(B)**. `pfsspy` is used to compute a potential magnetic field solution from the magnetogram in **(A)** and trace field lines through the resulting volume. These field lines, shown in green, are transformed to the appropriate coordinate system of each instrument using `sunpy`.

provided in `astropy`. The interoperability between `astropy`, `sunpy`, `sunpy-soar`, `aiapy`, and `pfsspy` allows us to easily examine the three-dimensional magnetic structure of the active region and see to what extent the derived potential field corresponds to the EUV emission as observed by these three spacecraft.

3 People

3.1 Board and lead developers

The current structure of the SunPy Project is governed by the SunPy Project board (Christe, 2018). The board is a self-electing oversight board which delegates the majority of the day-to-day operations of the project to a lead developer, who in turn delegates it to members of the community. The lead developer has overall responsibility for the large scale organization of the `sunpy` core package, and ensures that pull requests comply with stated standards and align with the goals of the SunPy Project. The deputy lead developer supports the lead developer and fills in when the lead is absent. The board's role is to steer the overall direction of the SunPy Project and consists of scientists and researchers who are not necessarily involved directly with the day-to-day development of the `sunpy` core package.

3.2 Community roles

There are several specific community (or executive) roles within the SunPy Project that perform important duties related to the overall development and maintenance of the project. These roles encompass a range of responsibilities from the development of the core package and affiliated packages to project communication and liaison. The community roles are held by members of the wider solar community who are actively involved in the SunPy Project. Anyone interested in a community role is encouraged to apply.

At present, there are nine community roles within the SunPy Project. From the development side, these include the Lead Developer and the Deputy Lead Developer who are responsible for the development of the `sunpy` core package, support the development of affiliated packages, and lead the development community. To assist the Lead/Deputy Developers, there are several development community roles which include:

- Continuous Integration Maintainer
- Release Manager
- Webmaster
- Communication and Education Lead who is responsible for the overall engagement with the wider community
- Lead Newcomer and Summer of Code mentor who assists new contributors and oversees the Google Summer of Code project
- Affiliated Package Liaison who is responsible for overseeing the affiliated package review process (**Section 2.2.1**) and working with developers of current and candidate affiliated packages

3.3 Maintainers and contributors

The development of the `sunpy` core package depends principally on an established team of volunteers that support the Lead and Deputy Lead Developers. These volunteer *maintainers* are given commit access to the `sunpy` repository and are predominantly, though not exclusively, scientists from the solar community who use `sunpy` in their work. In addition to this group of core maintainers, there is a steady influx of new *contributors*, averaging around 20–25 people per year. These contributors enable a wider range of features and code changes than would otherwise be normally possible due to the time constraints of the established team of volunteers. Within this subset of maintainers are members who maintain the specific sub-packages within `sunpy` like `sunpy.map` or `sunpy.coordinates`. These individuals are selected due to either their specific knowledge of the topic or their expertise with these sub-packages.

Contributing to the SunPy Project includes a wide range of activities, not all of them programming related. These include reporting bugs by raising issues on GitHub, requesting features, writing code and tests, providing feedback on pull requests, correcting or adding documentation, helping people who have problems or questions in communication channels and more. The SunPy Project is always looking for any new volunteers or people willing to contribute their time.

4 Community

4.1 Engagement with the solar physics community

In order for the SunPy Project to maintain and grow the `sunpy` core package and affiliated packages within the ecosystem, engagement with the wider solar physics community is critical. The mission of the SunPy Project is to be community-led, and the development is driven by the needs of the solar physics community. To facilitate this, the SunPy Project is building a community for which there is inclusive and open communication between those developing `sunpy` and those using `sunpy` in their scientific research. Active contributions from users in terms of bug reports, issues encountered with code or documentation, and feature requests are all vital to the sustainability and future of the SunPy Project. We emphasize that being part of the SunPy Project does not necessarily mean writing software. Contributions in the form of feedback and suggestions are equally important.

To foster communication, the SunPy Project supports several communication platforms (**Section 4.1.1**) through which users and developers can regularly interact. The SunPy Project posts on solar physics noticeboards about recent releases and regularly advertises `sunpy` and affiliated packages at scientific conferences, providing tutorials and support. We also ask that if `sunpy` is used for scientific work that it is cited in the literature[18], thereby increasing its visibility to the scientific community and ultimately contributing to the continued growth and development of the package. More recently, the SunPy Project has improved communications and established relationships with data providers such as VSO and the SOAR, and teams supporting both operating and developing instruments and missions. The SunPy Project is always looking for ways to improve the accessibility of the project and to grow the community.

18 See this page for a guide on how to cite `sunpy` in published works: https://sunpy.org/about#acknowledging-or-citing-sunpy

4.1.1 Communication channels

Over the years, the usage of `sunpy` and affiliated packages within the solar physics community has increased, and with that methods to communicate within the SunPy community have also increased. At the time of writing, several distinct communication channels are available. These include:

- Multiple GitHub repositories for bug reports and feature requests. These are listed under the SunPy Project GitHub organization[19].
- Real time messaging[20].
- Mailing lists[21].
- An online community forum[22].
- Weekly public calls that anyone can participate in[23].

Each has their own distinct purpose, and was created as a need arose for their existence. For example, the GitHub repository is used for the development of `sunpy` and issues and bugs can be raised there. However some scientists may not be familiar with GitHub and would like to ask a general question on how to do something within `sunpy`. For this, the mailing list, community forum, or real time Matrix chat may be the most appropriate. We actively encourage users and those interested in contributing to use any or all of these communication channels.

In addition to the main communication platforms specific to the SunPy Project, we maintain a presence within other communication channels used by the wider heliophysics community, including Helionauts[24] and communication channels used by the Python in Heliophysics Community.

4.2 Python in Heliophysics Community (PyHC)

The Python in Heliophysics Community (PyHC)[25] (Barnum et al., 2022) is a project with similar goals as the SunPy Project, but focuses on the wider Heliophysics community (Burrell et al., 2018). These include providing coding standards (Annex et al., 2018), curating a list of participating projects[26], hosting bi-monthly community meetings, and organizing an inaugural summer school for early career researchers. The SunPy Project is actively involved in PyHC, with `sunpy` being one of the core PyHC packages. SunPy Project members regularly attend community meetings and present updates. The SunPy Project also took part in the PyHC 2022 summer school. Moving forward, PyHC and the SunPy Project will continue to collaborate and build upon efforts to use `sunpy` and affiliated packages within the larger heliophysics Python ecosystem.

4.3 Collaboration with the wider python ecosystem

The `sunpy` package forms part of the wider Python scientific ecosystem, requiring active collaboration with other scientific Python packages. Whenever possible, we aim to contribute to relevant open-source projects rather than duplicating functionality. As an example, large parts of `sunpy` depend on core functionality developed in the `astropy` package, including support for handling units, times, and coordinates.

The SunPy Project is sponsored by NumFOCUS, "a non-profit supporting open code for better science"[27]. NumFOCUS provides financial and organizational support for several important packages (e.g., `numpy`, `pandas` and `xarray`) and facilitates collaboration between packages throughout the scientific Python ecosystem. One example of this is the annual NumFOCUS summit that brings together the leaders of these packages to discuss interoperability, funding sources and other high-level topics that improve the Python ecosystem as a whole.

In addition, the SunPy Project is a member of the OpenAstronomy organization[28]. OpenAstronomy was created to collaborate on outreach, organize conferences such as Python in Astronomy, develop common tooling for infrastructure, and apply to internship programs such as Google Summer of Code (GSoC)[29] and Outreachy[30]. GSoC has been an invaluable source of programming effort for the SunPy Project over the past decade. The contributions from participants in this program have been crucial to `sunpy`. Examples of successful projects include the conversion to using `astropy.time` and creating a new Python API wrapper for Helioviewer.org[31]. As the focus of the OpenAstronomy organization is the broader astrophysics and astronomy community, the SunPy Project's participation has enabled closer ties within the rapidly growing Python-in-astronomy landscape.

5 The future of the SunPy Project

Development within the SunPy Project is driven by, and for, the solar physics community, responding to the needs of researchers for data analysis tools and techniques, and software for working with data from new missions. This means both the `sunpy` core package and other affiliated packages are continually changing and expanding. In September 2022 several members of the SunPy Project met at a coordination meeting to discuss the future of the project. Two key areas that emerged were updating the governance structure,

19 https://github.com/sunpy

20 Links to join the Matrix chat can be found at https://sunpy.org/help.html

21 https://groups.google.com/g/sunpy

22 https://community.openastronomy.org/

23 https://sunpy.org/jitsi

24 https://helionauts.org/

25 https://heliopython.org/

26 https://heliopython.org/projects/

27 https://numfocus.org/

28 https://openastronomy.org/

29 https://summerofcode.withgoogle.com/

30 https://www.outreachy.org/

31 https://hvpy.readthedocs.io/

and creating a roadmap for future development. The roadmap provides:

1. a set of priorities for developers to work on in the medium term.
2. a well scoped list of work items that funding can be sought for.
3. a mechanism to solicit input from the wider solar physics community on the medium term priorities for the SunPy Project.

At the time of writing, items in the draft roadmap include:

- Improving support and functionality for data with spectral coordinates (e.g., rastering spectrometers)
- Enabling multi-dimensional data sets (i.e., beyond 2D images).
- Improving support for running `sunpy` on cloud infrastructure.
- Creating a consistently structured set of documentation across all the SunPy Project packages.
- Adding functionality to rapidly visualize large data sets.
- Restructuring the project governance to, among other things, transform the lead developer positions into a multi-person steering committee and create an ombudsperson role.

The next step is consultation with the wider solar physics community. We invite feedback on this proposed roadmap *via* any of the aforementioned communication channels (**Section 4.1.1**) or by opening an issue on the repository used for tracking high-level, project-wide tasks and suggestions[32].

6 Conclusion

In this paper, we have summarized the SunPy Project and its various components, including the code developed and maintained by the project (**Section 2**), the people that comprise the project (**Section 3**) and the community that the project serves (**Section 4**). In particular, we have discussed how the `sunpy` package and the wider set of affiliated packages form a software ecosystem for solar physics research in Python and illustrated the types of analyses that such an ecosystem enables (**Figure 1**). Finally, we have summarized a tentative roadmap to steer the direction of the SunPy Project in the coming years. Importantly, we hope that such a high level description will provide a more clear understanding of the SunPy Project and the wider ecosystem and will encourage contributions of all forms from the global solar physics community.[33]

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: AIA and EUVI data are available through the VSO: https://sdac.virtualsolar.org/cgi/search. EUI data are available through the SOAR: https://soar.esac.esa.int/soar/. HMI synoptic magnetogram data are available through the JSOC: http://jsoc.stanford.edu/.

Author contributions

WB contributed to writing the text and created **Figure 1**. SC, NF, LH, and DS contributed to writing the text. All authors contributed to manuscript planning and revision and have read and approved the submitted version.

Funding

WB, AS, and SM are supported by an award from the NASA Research Opportunities in Space and Earth Sciences (ROSES) Open-Source Tools, Frameworks, and Libraries (OSTFL) program. NF is supported by NASA under contract NNG09FA40C (*IRIS*) and NNG04EA00C (SDO/AIA). LH is supported by an ESA Research Fellowship.

Acknowledgments

We thank everyone who has supported and contributed to the SunPy Project in any manner. `sunpy` makes use of astropy, a community-developed core Python package for Astronomy (Astropy Collaboration et al., 2022). We acknowledge financial contributions from Google as part of the Google Summer of Code program and from the European Space Agency as part of the Summer of Code in Space program. We acknowledge financial contributions from NumFOCUS for "Improving the Usability of sunpy's Data Downloader". Additionally, we acknowledge funding from the Solar Physics Division of the American Astronomical Society for SunPy workshops and tutorials at annual meetings.

Conflict of interest

Author SM was employed by Aperio Software Ltd.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

32 https://github.com/sunpy/sunpy-project

33 https://github.com/sunpy/sunpy-frontiers-paper

References

Annex, A., Alterman, B. L., Azari, A., Barnes, W., Bobra, M., Cecconi, B., et al. (2018). Python in heliophysics community (PyHC) standards. doi:10.5281/zenodo.2529131

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., et al. (2022). The astropy project: Sustaining and growing a community-oriented open-source project and the latest major release (v5.0) of the core package. *Astrophysical J.* 935, 167. doi:10.3847/1538-4357/ac7c74

Barnes, W., Cheung, M., Bobra, M., Boerner, P., Chintzoglou, G., Leonard, D., et al. (2020). Aiapy: A Python package for analyzing solar EUV image data from AIA. *J. Open Source Softw.* 5, 2801. doi:10.21105/joss.02801

Barnum, J., Masson, A., Friedel, R. H., Roberts, A., and Thomas, B. A. (2022). Python in heliophysics community (pyhc): Current status and future outlook. *Adv. Space Res.* doi:10.1016/j.asr.2022.10.006

Bobra, M., Bin, C., Christe, S., Hewett, R., Freij, N., Mumford, S., et al. (2020). SunPy proposal for enhancement 10: Vision and mission statement (SEP 0010). doi:10.5281/zenodo.7020094

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—An overview of python in heliophysics. *J. Geophys. Res. Space Phys.* 123 (12), 402. doi:10.1029/2018JA025877

Christe, S. (2018). SunPy proposal for enhancement 2: SunPy organization definition (SEP 0002). doi:10.5281/zenodo.3261663

Freij, N., Ireland, J., Mumford, S., Barnes, W., Chaubey, V., Paul, J. A., et al. (2022). sunpy/sunkit-image: v0.4.2. doi:10.5281/zenodo.6578722

Greisen, E. W., and Calabretta, M. R. (2002). Representations of world coordinates in FITS. *Astronomy Astrophysics* 395, 1061–1075. doi:10.1051/0004-6361:20021326

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Howard, R. A., Moses, J. D., Vourlidas, A., Newmark, J. S., Socker, D. G., Plunkett, S. P., et al. (2008). Sun Earth connection coronal and heliospheric investigation (SECCHI). *Space Sci. Rev.* 136, 67–115. doi:10.1007/s11214-008-9341-4

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Lemen, J. R., Title, A. M., Akin, D. J., Boerner, P. F., Chou, C., Drake, J. F., et al. (2012). The atmospheric imaging assembly (AIA) on the solar Dynamics observatory (SDO). *Sol. Phys.* 275, 17–40. doi:10.1007/s11207-011-9776-8

Müller, D., Cyr, O. C. S., Zouganelis, I., Gilbert, H. R., Marsden, R., Nieves-Chinchilla, T., et al. (2020). The solar orbiter mission - science overview. *Astronomy Astrophysics* 642, A1. doi:10.1051/0004-6361/202038467

Mumford, S., and Christe, S. (2014). SunPy proposal for enhancement 4: Packages affiliated with the SunPy project (SEP 0004). doi:10.5281/zenodo.3261752

Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., and Stobie, E. (2010). Definition of the flexible image transport system (fits), version 3.0. *Astronomy Astrophysics* 524, A42. doi:10.1051/0004-6361/201015362

Pesnell, W. D., Thompson, B. J., and Chamberlin, P. C. (2012). The solar Dynamics observatory (SDO). *Sol. Phys.* 275, 3–15. doi:10.1007/s11207-011-9841-3

Rochus, P., Auchère, F., Berghmans, D., Harra, L., Schmutz, W., Schühle, U., et al. (2020). The solar orbiter EUI instrument: The extreme ultraviolet imager. *Astronomy Astrophysics* 642, A8. doi:10.1051/0004-6361/201936663

Ryan, D., Mumford, S., Freij, N., Inglis, A., Leonard, D., Christe, S., et al. (2022). sunpy/sunkit-instruments: v0.3.1. doi:10.5281/zenodo.7190661

Ryan, D., Mumford, S., Sharma, Y., ankitBhope, A., Freij, N., et al. (2021). sunpy/ndcube: v2.0.1. doi:10.5281/zenodo.5715161

Scherrer, P. H., Schou, J., Bush, R. I., Kosovichev, A. G., Bogart, R. S., Hoeksema, J. T., et al. (2012). The helioseismic and magnetic imager (HMI) investigation for the solar Dynamics observatory (SDO). *Sol. Phys.* 275, 207–227. doi:10.1007/s11207-011-9834-2

Stansby, D., Yeates, A., and Badman, S. (2020). pfsspy: A Python package for potential field source surface modelling. *J. Open Source Softw.* 5, 2732. doi:10.21105/joss.02732

The Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., et al. (2018). The astropy project: Building an open-science project and status of the v2.0 core package. *Astronomical J.* 156, 123. doi:10.3847/1538-3881/aabc4f

The SunPy Community, Barnes, W. T., Bobra, M. G., Christe, S. D., Freij, N., Hayes, L. A., et al. (2020). The sunpy project: Open source development and status of the version 1.0 core package. *Astrophysical J.* 890, 68. doi:10.3847/1538-4357/ab4f7a

Thompson, W. T. (2006). Coordinate systems for solar image data. *Astronomy Astrophysics* 449, 791–803. doi:10.1051/0004-6361:20054262

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2

Wells, D. C., Greisen, E. W., and Harten, R. H. (1981). Fits - a flexible image transport system. *Astronomy Astrophysics Suppl. Ser.* 44, 363.

Appendix

Here we provide a glossary of terms used throughout this paper:

- SunPy Project: The board and lead/deputy developers, the community roles, maintainers, and every package under its supervision.
- `sunpy`: The core package for using Python for scientific research in solar physics.
- SunPy ecosystem: The collection of packages that use or interface with `sunpy` and support scientific research in solar physics, including `sunpy`.
- Affiliated package(s): Solar physics related functionality outside the scope of the `sunpy` core package and that satisfies the standards enumerated in **Section 2.2.1**.

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 28 February 2023
DOI 10.3389/fspas.2022.1058810

Check for updates

OPEN ACCESS

EDITED BY
Sophie A. Murray,
Dublin Institute for Advanced Studies (DIAS), Ireland

REVIEWED BY
Will Barnes,
American University, United States
Qiang Hu,
University of Alabama in Huntsville, United States

*CORRESPONDENCE
Jan Gieseler,
jan.gieseler@utu.fi

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 30 September 2022
ACCEPTED 14 November 2022
PUBLISHED 28 February 2023

CITATION
Gieseler J, Dresing N, Palmroos C, Freiherr von Forstner JL, Price DJ, Vainio R, Kouloumvakos A, Rodríguez-García L, Trotta D, Génot V, Masson A, Roth M and Veronig A (2023), Solar-MACH: An open-source tool to analyze solar magnetic connection configurations. *Front. Astron. Space Sci.* 9:1058810. doi: 10.3389/fspas.2022.1058810

COPYRIGHT
© 2023 Gieseler, Dresing, Palmroos, Freiherr von Forstner, Price, Vainio, Kouloumvakos, Rodríguez-García, Trotta, Génot, Masson, Roth and Veronig. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Solar-MACH: An open-source tool to analyze solar magnetic connection configurations

Jan Gieseler[1]*, Nina Dresing[1], Christian Palmroos[1], Johan L. Freiherr von Forstner[2,3], Daniel J. Price[4], Rami Vainio[1], Athanasios Kouloumvakos[5], Laura Rodríguez-García[6], Domenico Trotta[7], Vincent Génot[8], Arnaud Masson[9], Markus Roth[10] and Astrid Veronig[11]

[1]Space Research Laboratory, Department of Physics and Astronomy, University of Turku, Turku, Finland, [2]Institute of Experimental and Applied Physics, Kiel University, Kiel, Germany, [3]Now at Paradox Cat GmbH, München, Germany, [4]Department of Physics, University of Helsinki, Helsinki, Finland, [5]Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD, United States, [6]Space Research Group, Universidad de Alcalá, Alcalá de Henares, Spain, [7]The Blackett Laboratory, Department of Physics, Imperial College London, London, United Kingdom, [8]Institut de Recherche en Astrophysique et Planétologie, CNRS, CNES, Université Paul Sabatier, Toulouse, France, [9]ESAC Science Data Centre, European Space Agency, Madrid, Spain, [10]Thüringer Landessternwarte, Tautenburg, Germany, [11]Kanzelhöhe Observatory for Solar and Atmospheric Research, Institute of Physics, University of Graz, Graz, Austria

The Solar MAgnetic Connection HAUS[1] tool (Solar-MACH) is an open-source tool completely written in Python that derives and visualizes the spatial configuration and solar magnetic connection of different observers (i.e., spacecraft or planets) in the heliosphere at different times. For doing this, the magnetic connection in the interplanetary space is obtained by the classic Parker Heliospheric Magnetic Field (HMF). In close vicinity of the Sun, a Potential Field Source Surface (PFSS) model can be applied to connect the HMF to the solar photosphere. Solar-MACH is especially aimed at providing publication-ready figures for the analyses of Solar Energetic Particle events (SEPs) or solar transients such as Coronal Mass Ejections (CMEs). It is provided as an installable Python package (listed on PyPI and conda-forge), but also as a web tool at solar-mach.github.io that completely runs in any web browser and requires neither Python knowledge nor installation. The development of Solar-MACH is open to everyone and takes place on GitHub, where the source code is publicly available under the BSD 3-Clause License. Established Python libraries like `sunpy` and `pfsspy` are utilized to obtain functionalities when possible. In this article, the Python code of Solar-MACH is explained, and its functionality is demonstrated using real science examples. In addition, we introduce the overarching SERPENTINE project, the umbrella under which the recent development took place.

1 Heliophysics Archives USer group at ESA.

KEYWORDS

Python (programming language), software package, solar energetic particle (SEP), corona, coronal mass ejection (CME), spacecraft, heliosphere, PFSS

1 Introduction

The *Solar energetic particle analysis platform for the inner heliosphere* (SERPENTINE, 2021–2024) is a 42-month long project funded through the H2020-SPACE-2020 call of the European Union's Horizon 2020 framework programme. The project addresses several outstanding questions on the origin of solar energetic particle (SEP) events and provides an advanced data analysis and visualization platform that will benefit the whole heliophysics community. It utilizes the most recent European and US missions, i.e., Solar Orbiter (Müller et al., 2020), Parker Solar Probe (Fox et al., 2016) and BepiColombo (Benkhoff et al., 2021). These observations are complemented with supporting data from several current missions near Earth's orbit as well as ground-based radio imaging and spectroscopic observations by the European Low Frequency Array (LOFAR; van Haarlem et al., 2013).

SEP events are large and sporadic outbursts of charged particle radiation from the solar corona that are related to solar eruptions such as flares and coronal mass ejections (CMEs; e.g., Reames, 1999). They can be classified as impulsive and gradual events, based on their duration and the duration of the related solar X-ray flare. Impulsive SEP events are associated with impulsive flares and narrow CMEs, and they are enriched in electrons, ^{3}He isotope and heavy ions. Gradual SEP events are associated with gradual solar X-ray flares and fast and wide CMEs, and their abundances resemble nominal coronal abundances (Reames, 2013; Desai and Giacalone, 2016). Gradual events are usually broader in their helio-longitudinal extent than impulsive events and their intensities are also typically larger, making them the main concern of spacecraft operations and crews (Vainio et al., 2009).

The primary reason for the broad spatial extent of some gradual events is still unresolved. It could be due to a broad source, like a global coronal shock driven by a CME (e.g., Lario et al., 2016), due to efficient particle transport processes across the heliospheric magnetic field, or as a result of both mechanisms (e.g., Dresing et al., 2012; Rodríguez-García et al., 2021). The main objective of SERPENTINE is to pinpoint the primary causes of large gradual and widespread SEP events. To address this objective, SERPENTINE will answer the following open science questions:

Q1: What are the primary causes for widespread SEP events observed in the heliosphere?

Q2: What are the shock acceleration mechanisms responsible for accelerating ions from thermal/suprathermal energies to near-relativistic energies in the corona and in the interplanetary medium?

Q3: What is the role of shocks in electron acceleration in large gradual and widespread events? How does it relate to ion acceleration and what is its importance relative to flare acceleration?

To reach these goals and also to broaden the impact of the project, SERPENTINE will develop and release to the community a platform of tools for analyzing SEP events. Furthermore, the tools may also be useful for the broader heliospheric community looking at different aspects of solar activity or solar wind phenomena. Part of this platform will be a JupyterHub server that provides free access to the tools developed by the SERPENTINE project as Jupyter Notebooks, without requiring any installations beyond a web browser. This manuscript and the accompanying papers (Kouloumvakos et al., 2022; Palmroos et al., 2022; Trotta et al., 2022) will present the first batch of these tools to the heliophysics community.

Because SEPs are measured *in situ* as enhancements of the energetic particle fluxes, the presence of multiple, well-separated observers is indispensable to study widespread SEP events (e.g., Dresing et al., 2014; Richardson et al., 2014). The new space missions in combination with established spacecraft form a fleet that is ideal for this purpose, as it covers varying heliocentric distances and large longitudinal ranges around the Sun. The different orbits of the multiple spacecraft, in combination with varying source regions at the Sun, constantly form new constellations, building the baseline for in-depth SEP event analyses. The first released tool of SERPENTINE, Solar-MACH, provides the user with a quick overview of these specific constellations for a selected time, as shown in the example of **Figure 1**. "Solar-MACH" is an abbreviation for *Solar MAgnetic Connection HAUS*, with HAUS standing for the Heliophysics Archives USer group at ESA. "MACH" is intended to be pronounced like the *Mach* number.

2 Method

The main functionality of Solar-MACH is to provide the user with a polar plot showing a Sun-centered top view of the heliographic equatorial plane with the constellation of different observers for a given time, as shown in **Figure 1**. Additional information can be added, such as indicating the magnetic connection of each observer to the Sun as given by an idealized

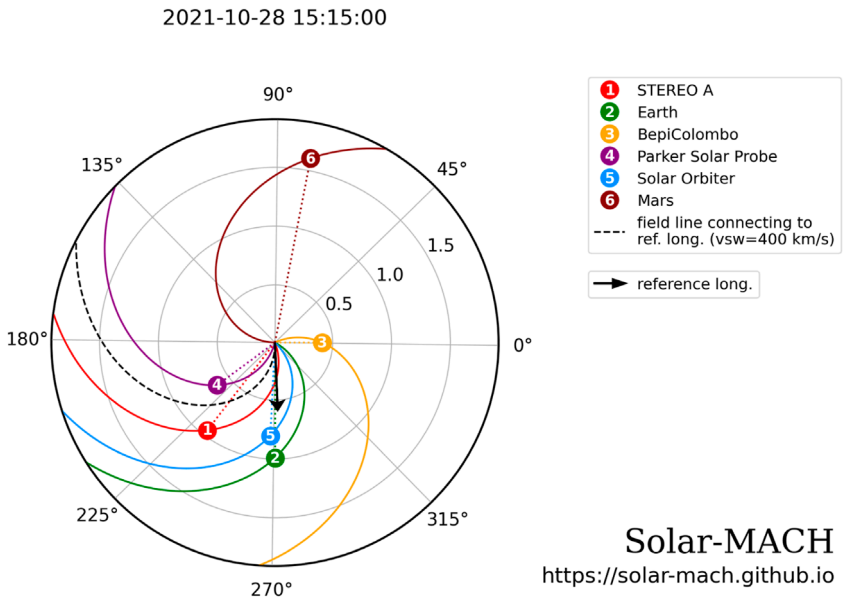


FIGURE 1
Solar-MACH plot for the time of the ground-level enhancement (GLE) event on 28 October 2021. Numbered symbols indicate the observers' locations and the spiral lines corresponding HMF lines connecting them to the Sun. Radial distance is provided in astronomical units (AU), and the angular information is given in Carrington longitude. The arrow points out the freely choosable location of a "reference" (e.g., a solar flare) at the Sun, and the dashed spiral line indicates a corresponding HMF line originating at that position.

classic Parker heliospheric magnetic field line, whose curvature is dependent on the radial solar wind speed. This and other options are described in **Section 2.1**.

All software operations are performed within Python 3 (e.g., Van Rossum and Drake, 2009); the corresponding package `solarmach` is described in **Section 2.2**. In addition, a web tool is provided that requires neither Python knowledge nor installation. This web tool and its technical background are presented in **Section 2.4**.

2.1 General functionality

2.1.1 Ephemeris

After the user provides a specific date and time, the spatial coordinates of all requested observers are obtained dynamically from JPL Horizons, an online Solar System data and ephemeris computation service that is maintained by the Solar System Dynamics Group of the Jet Propulsion Laboratory (JPL). In addition to the web interface, JPL Horizons provides an API for programmatic control. This API is utilized through version 4.0.5 (Mumford et al., 2022) of the `sunpy` open-source software package (The SunPy Community et al., 2020), which itself uses Astropy's (Astropy Collaboration et al., 2013; Astropy Collaboration et al., 2018; Astropy Collaboration et al., 2022) affiliated package astroquery (Ginsburg et al., 2021) for this functionality. The ephemeris information is returned per observer within Python as a SkyCoord object, which inherently provides coordinate system transformations. This allows easily supporting different heliographic coordinate systems. At the moment, the user can decide whether all operations should be carried out in Stonyhurst or Carrington coordinates (e.g., Thompson, 2006), with the latter being the default setting.

2.1.2 Magnetic connectivity

One important property for analyzing SEP events is the magnetic connection of each observer to the Sun. A simple approximation for this connection can be obtained by assuming an ideal Archimedean spiral configuration for the heliospheric magnetic field (Parker, 1958), which is defined in spherical coordinates by:

$$\varphi(r) = \varphi_0 + \frac{\omega(\vartheta)}{v_{sw}} \cdot (R - r) \cdot \cos\vartheta, \tag{1}$$

with heliographic longitude φ as a function of radial distance r, heliographic latitude ϑ, differential solar rotation frequency $\omega(\vartheta)$, radial solar wind speed v_{sw}, as well as radial distance R and longitude φ_0 of the observer. The differential solar rotation frequency $\omega(\vartheta)$ takes into account that the Sun's rotation speed varies with latitude. Here, we apply an empirical model by Poljančić Beljan et al. (2017) that describes $\omega(\vartheta)$ by $14.50 - 2.87\sin^2\vartheta$ (deg/day), which corresponds to

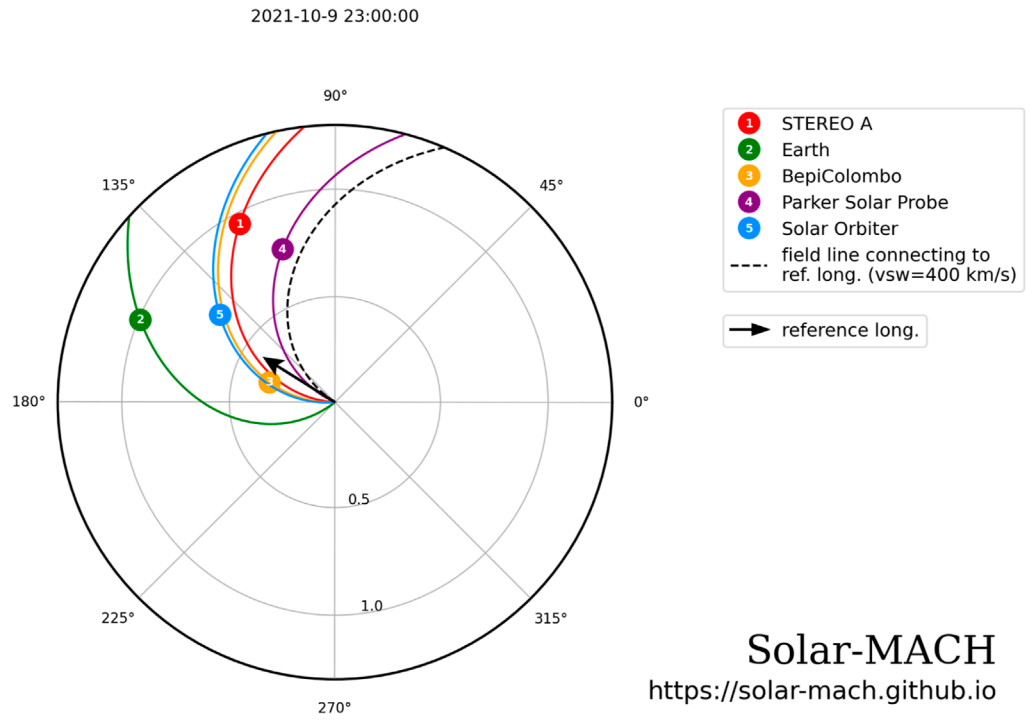
In [1]:
```
from solarmach import SolarMACH
```

In [2]:
```
# necessary options
body_list = ['STEREO-A', 'Earth', 'BepiColombo', 'PSP', 'Solar Orbiter']
vsw_list = [400, 400, 400, 400, 400, 400]   # position-sensitive solar wind speed per body in body_list
date = '2021-10-9 23:00:00'

# optional parameters
coord_sys = 'Carrington'                        # 'Carrington' (default) or 'Stonyhurst'
reference_long = 148                            # longitude of reference (None to omit)
reference_lat = 0                               # latitude of reference (None to omit)
plot_spirals = True                             # plot Parker spirals for each body
plot_sun_body_line = False                      # plot straight line between Sun and body
long_offset = 157                               # longitudinal offset for polar plot; defines where Earth's
                                                #     longitude is (by default 270, i.e., at "6 o'clock")
reference_vsw = 400                             # define solar wind speed at reference
return_plot_object = True                       # figure and axis object of matplotib are returned, allowing
                                                #     further adjustments to the figure
transparent = False                             # make output figure background transparent
numbered_markers = True                         # plot each body with a numbered marker
filename = 'Solar-MACH_'+date.replace(' ', '_')  # define filename of output figure
```

In [3]:
```
# initialize
sm = SolarMACH(date, body_list, vsw_list, reference_long, reference_lat, coord_sys)
```

In [4]:
```
# make plot
fig, ax = sm.plot(plot_spirals=plot_spirals, plot_sun_body_line=plot_sun_body_line,
                  reference_vsw=reference_vsw, transparent=transparent, numbered_markers=numbered_markers,
                  long_offset=long_offset, return_plot_object=return_plot_object)
```



In [5]:
```
# obtain data as Pandas DataFrame
sm.coord_table
```

Out[5]:

	Spacecraft/Body	Carrington longitude (°)	Carrington latitude (°)	Heliocentric distance (AU)	Longitudinal separation to Earth's longitude	Latitudinal separation to Earth's latitude	Vsw	Magnetic footpoint longitude (Carrington)	Longitudinal separation between body and reference_long	Longitudinal separation between body's mangetic footpoint and reference_long	Latitudinal separation between body and reference_lat
0	STEREO-A	118.204872	7.288958	0.957061	-38.867683	1.065832	400	177.311234	-29.795128	29.311234	7.288958
1	Earth	157.072554	6.223126	0.998714	0.000000	0.000000	400	218.953335	9.072554	70.953335	6.223126
2	BepiColombo	163.086313	2.285930	0.327839	6.013759	-3.937196	400	183.348797	15.086313	35.348797	2.285930
3	PSP	108.963800	3.702572	0.762690	-48.108755	-2.520554	400	156.403906	-39.036200	8.403906	3.702572
4	Solar Orbiter	142.918039	2.346947	0.682696	-14.154516	-3.876179	400	185.425985	-5.081961	37.425985	2.346947

FIGURE 2
Example of using the Python package `solarmach` in a Jupyter Notebook for the SEP event on 9 October 2021.

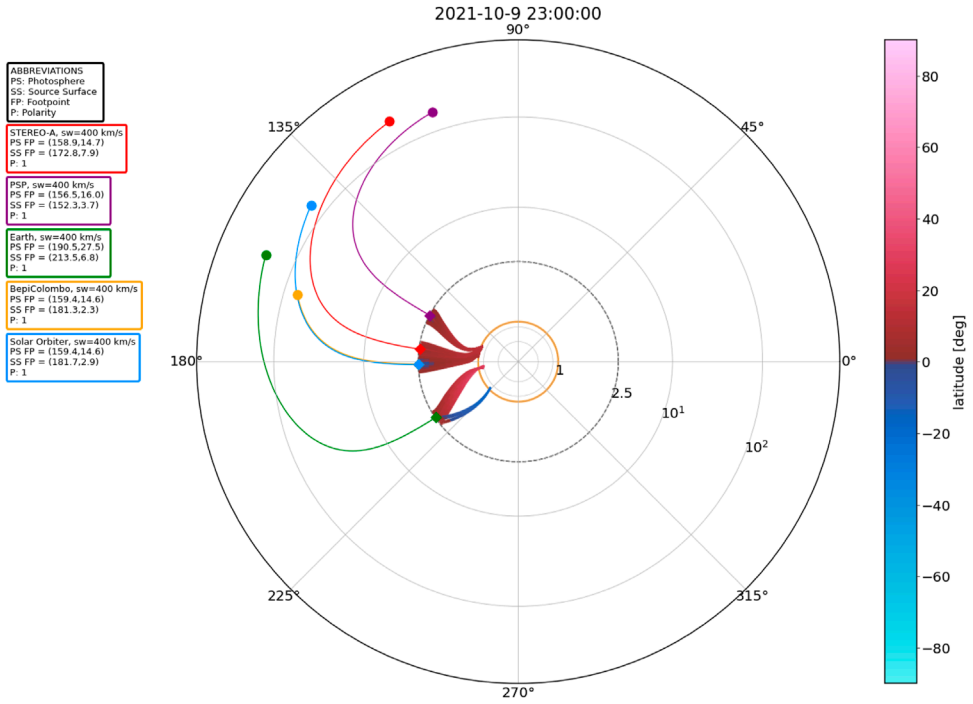


FIGURE 3
PFSS backmapping output of Solar-MACH for the SEP event on 9 October 2021. Note the different scaling of the radial axis compared to Figure 2 (see text for details). The yellow circle represents the photosphere at one solar radius.

14.50 (deg/day) $= 1.678241 \cdot 10^{-4}$ (deg/s) at the heliographic equator. For the two-dimensional visualization, all spherical coordinates are projected to the heliographic equatorial plane. The resulting field lines are shown in Figure 1 as solid spiral lines, with color-coding of the respective spacecraft. As the spatial coordinates are dynamically obtained for each spacecraft, the solar wind speed v_{sw} is the only free parameter that needs to be provided for each observer in units of km/s. By default, this is assumed to be 400 km/s. Automatically obtaining real measurements of the solar wind speed is planned as an option for the future. Applying Eq. 1 for $r = R_{\odot}$ provides the magnetic footpoint coordinate for each spacecraft, that is, the heliographic longitude where the assumed magnetic field line originates at the Sun. This information is especially important in order to connect *in situ* measurements of SEPs with remote-sensing observations of the associated source regions at the Sun. We must point out, though, that this analysis is just an estimation that may deviate significantly from reality. Employing this tool for magnetic backmapping purposes comes with the known caveats of the simple assumptions of ballistic backmapping, which are a constant and totally radial solar wind speed as well as the absence of interplanetary structures (e.g., interplanetary CMEs) that could deform the ideal Parker field.

2.1.3 Reference location

To further assist this analysis, the location of a so-called "reference" at the Sun (and a corresponding v_{sw}) can be provided to the tool, for example, the position of a flare that is assumed to be related with the SEP event. This information is then used on the one hand to visually indicate the position of the reference (i.e., flare) in the plot with a black arrow (cf. Figure 1) and its corresponding heliospheric magnetic field line, which helps to quickly estimate which spacecraft are possible observers of particles originating at this flare. On the other hand, the separation angles between the position of the reference and the magnetic footpoints of the different spacecraft are calculated. This information can then be obtained for further use in a table that also contains all other coordinate parameters provided by JPL Horizons, as demonstrated in the bottom of Figure 2.

2.1.4 Plotting options

In addition to the functional options described so far, Solar-MACH provides some plotting settings that enable users to have several options of visualization customization:

- Deactivate plotting of the heliospheric magnetic field lines.
- Add a straight line from the Sun to each observer, indicating the line of sight.

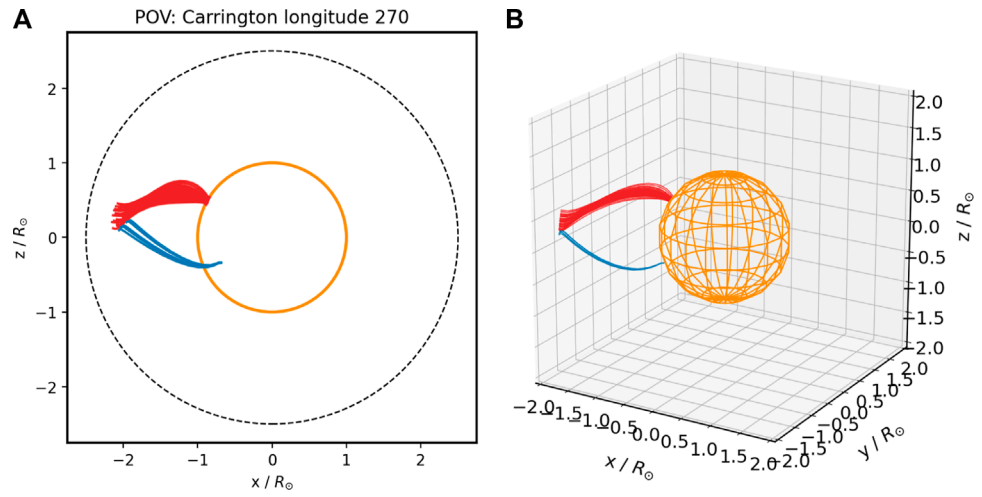


FIGURE 4
Detailed view of Earth's magnetic connection from the PFSS backmapping example shown in **Figure 3**. **(A)** Two-dimensional side view with the point of view (POV) at Carrington longitude 270° (i.e., the observer is located at *six o'clock* in **Figure 3**), **(B)** freely rotatable three-dimensional view. The two colors of the field lines indicate the corresponding magnetic polarity.

- Use numbered markers for the spacecraft to help with color vision deficiency issues or if the plot needs to be converted to grayscale.
- Provide a longitudinal offset for the polar plot to define where the Earth is located (by default 270°, i.e., at *six o'clock*).
- Make the background of the plot transparent.

2.2 Python package solarmach

The Solar-MACH Python code is made available as an installable Python package called `solarmach` (Gieseler et al., 2022b) that is listed on PyPI and conda-forge, i.e., it can be installed using either the `pip` or the `conda` command line tool, which are by far the most widely used means of installing Python packages. Installing the package in this way ensures that all other required packages are also installed.

An example of the standard workflow for the package is shown in a Jupyter Notebook in **Figure 2**. This specific Notebook is also available from the subfolder `examples` in the corresponding GitHub repository. The workflow consists of the following steps (each step is represented by one code cell in the example Notebook in **Figure 2**):

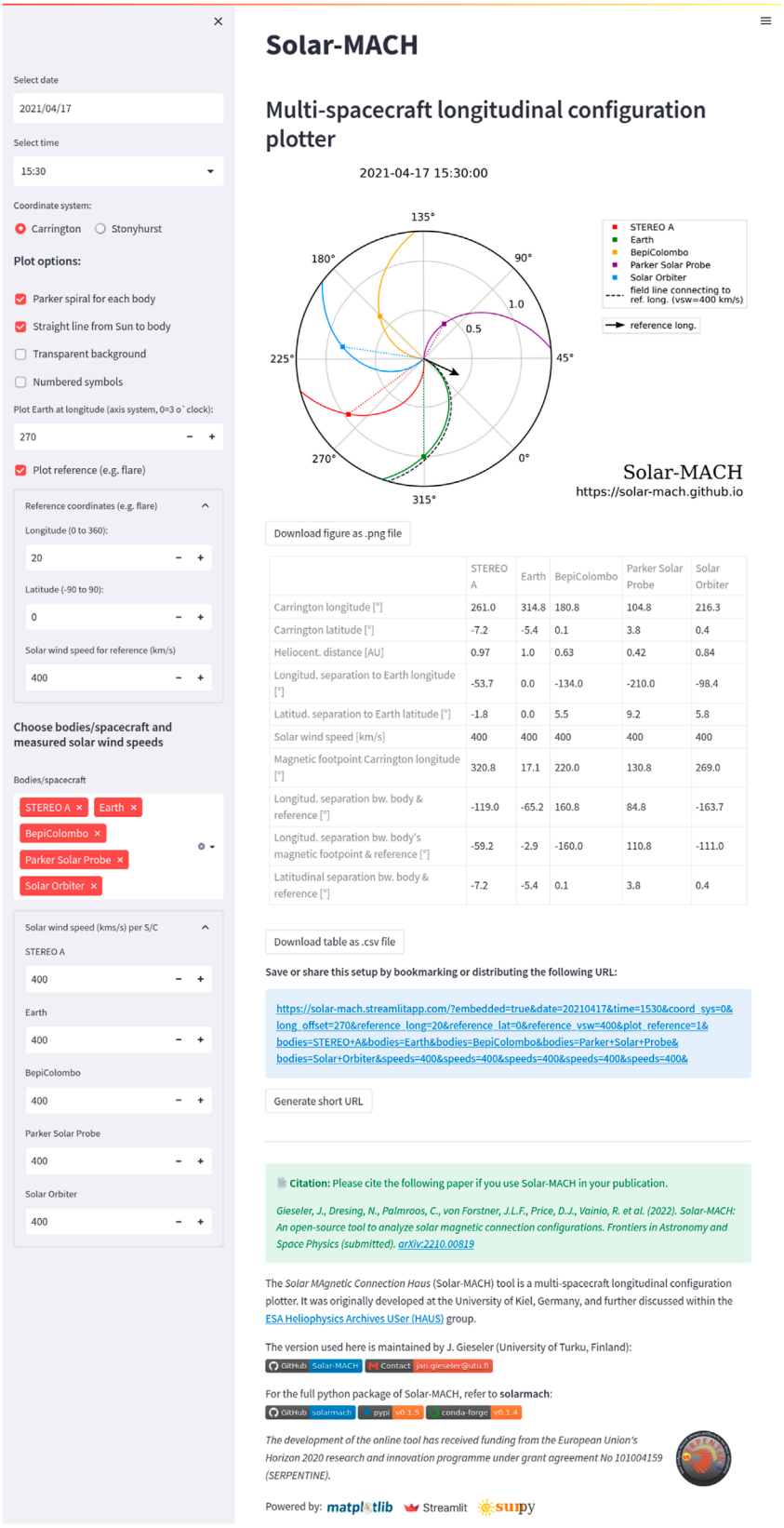
1. Import the `SolarMACH` class
2. Provide necessary options and optional parameters
3. Initialize the `SolarMACH` object with these settings
4. Generate a plot from the `SolarMACH` object (optionally)
5. Obtain the data as a Pandas DataFrame from the `SolarMACH` object (optionally)

The Pandas DataFrame obtained in the last step does not only display the dataset in a tabular form, but also provides the user with a ready-to-use dataset for further analyses, ultimately enabling fast diagnostic pipelines (McKinney, 2010). Furthermore, through the optional settings in step 2, it is possible to either provide a file name and ending under which the plot will then be saved (where the file type is automatically determined, e.g., "file.png" or "file.pdf"), or to return the matplotlib (Hunter, 2007) plot object in order to further manipulate it.

A Jupyter Notebook with more detailed descriptions and various use cases is available in the SERPENTINE software repository (Gieseler et al., 2022c). This Notebook contains examples on how to connect the Solar-MACH outputs with other Python functionalities. One example is the creation of an animated GIF file that shows a time-lapse of the spacecraft constellation over a given time period (see here for an example). In addition, this extended Notebook provides the functionality to continue the interplanetary magnetic connections to the solar corona with a PFSS model, which is described in more detail in **Section 2.3**.

2.3 Further backmapping with PFSS

The magnetic field configuration close to the Sun in the corona differs drastically from the heliospheric magnetic field (HMF) and cannot be described using the simple Parker spiral approach given by **Eq. 1**. In order to further extend the magnetic connectivity from an observer towards the solar surface, a



	STEREO A	Earth	BepiColombo	Parker Solar Probe	Solar Orbiter
Carrington longitude [°]	261.0	314.8	180.8	104.8	216.3
Carrington latitude [°]	-7.2	-5.4	0.1	3.8	0.4
Heliocent. distance [AU]	0.97	1.0	0.63	0.42	0.84
Longitud. separation to Earth longitude [°]	-53.7	0.0	-134.0	-210.0	-98.4
Latitud. separation to Earth latitude [°]	-1.8	0.0	5.5	9.2	5.8
Solar wind speed [km/s]	400	400	400	400	400
Magnetic footpoint Carrington longitude [°]	320.8	17.1	220.0	130.8	269.0
Longitud. separation bw. body & reference [°]	-119.0	-65.2	160.8	84.8	-163.7
Longitud. separation bw. body's magnetic footpoint & reference [°]	-59.2	-2.9	-160.0	110.8	-111.0
Latitudinal separation bw. body & reference [°]	-7.2	-5.4	0.1	3.8	0.4

FIGURE 5
The Streamlit web version at solar-mach.github.io.

Potential Field Source Surface (PFSS) model (e.g., Mackay and Yeates, 2012, and references therein) is applied by using version 1.1.2 (Stansby et al., 2022) of the `pfsspy` open-source software package (Stansby et al., 2020). The main purpose is to calculate a set of open magnetic field lines in a defined circular area around the point where an idealized HMF line connects an interplanetary observer to the source surface, at which the field lines are forced by the PFSS model to be radial. This is illustrated in **Figure 3** for the situation on 9 October 2021. The outer colored field lines correspond to those of **Figure 2**, only that the radial distance is now plotted in a semilogarithmic scale, so that the situation close to the Sun can be depicted in more detail. The radial axis is given in units of solar radii, with a linear scale up to where the source surface is assumed (this is a free parameter, by default set to 2.5 solar radii). Further out, a logarithmic scale is used, which explains the slightly different shape of the HMF lines compared to **Figure 2**. Below the source surface, for each HMF line a set of open field lines is calculated and traced back to the photosphere. Color-coding gives additional information about the corresponding latitudes, as depicted by the color bar on the right. **Figure 4** presents alternative ways to illustrate the results, where the set of coronal field lines obtained for the HMF line connecting to the Earth is shown in a two-dimensional side view (**Figure 4A**) or in a freely rotatable three-dimensional view (**Figure 4B**). These figures illustrate a situation in which the magnetic connection below the source surface is ambiguous as the bundle of field lines splits, each leading to a different region at the photosphere. Although the tool cannot provide an unambiguous backmapped magnetic footpoint location, such a result contains important information for the user, which are valuable for further analyses.

At the moment, the PFSS functionality of Solar-MACH is only provided through the Jupyter Notebook in the SERPENTINE software repository (Gieseler et al., 2022c) that has already been mentioned in **Section 2.2**. This part has not yet been added to the `solarmach` package, but this is planned for the near future, subsequently making it available through the Streamlit web tool (cf. **Section 2.4**), too. In order to be able to obtain Helioseismic and Magnetic Imager (HMI) synoptic maps through the Joint Science Operations Center (JSOC) at Stanford University, which are needed as input for the PFSS model in the current version, the user needs to register once at http://jsoc.stanford.edu/ajax/register_email.html. For the future, especially for the integration into the Streamlit web tool, we aim at removing this requirement.

2.4 Streamlit web tool

While all the functionality of Solar-MACH depends fully on Python, a web tool is provided online at solar-mach.github.io that completely runs in the web browser, features a fully graphical user interface (GUI), and requires neither Python knowledge nor installation. It is implemented using the open-source Python package streamlit. This package provides an easy way to set up a GUI to Python functions that is made available through a web server. It is possible to run it locally on a computer (if Python and all other requirements are installed), offering a full GUI to the user instead of running the code in a Jupyter Notebook or terminal. But the main purpose is to deploy the Streamlit application to a web hosting service, because only then it provides all the underlying Python functionalities without the need to install any kind of (Python) software. A screenshot of the web tool is shown in **Figure 5**. We emphasize that the possibility to use the web tool is invaluable for getting started quickly in a non-code environment, which is especially important for new users.

In addition to quickly obtaining an overview of the observer constellation when a SEP event just took place, another important functionality of the web tool is to bookmark or share a given combination of options that has been set through the GUI. After all settings have been done, a specific URL for this configuration is shown at the bottom of the web page (blue box in **Figure 5**). An option is provided next to it to generate a short link of this URL because it contains all settings and is thus rather long. An advantage of this configuration is that almost all settings can be provided through the URL, which thus can be used as a simple API. This API is not documented yet, but should be self-explanatory using a given example.

For further use of the Solar-MACH results, the user can save the plot by clicking with the right mouse button on it and using the web browser functionality, exactly like in the Jupyter Notebook version mentioned in **Section 2.2** and **Section 2.3**. In addition, the web tool provides a download button for the plot that might be helpful, especially when using mobile devices like smartphones or tablets. The web tool also automatically displays a table with all ephemeris, magnetic footpoint, and separation information, which can be downloaded as a comma-separated values (CSV) file.

The Streamlit application itself consists of a rather small code base that is open source (Gieseler et al., 2022a). It collects all necessary parameters from the user through the GUI, hands them over to the `solarmach` package described in **Section 2.2**, and presents the returned results. The online version provided at solar-mach.github.io is automatically redeployed whenever the corresponding GitHub repository is updated. In this process, the latest `solarmach` version available at PyPI is installed. Because the GUI needs to be adjusted manually to any new functionality, the Streamlit version may not always offer the latest additions to the `solarmach` package.

3 Discussion

This paper introduces the Solar-MACH tool to the scientific community. Solar-MACH automatically obtains and visualizes the constellation of the heliospheric spacecraft fleet for a given time and provides a first estimation of the magnetic connectivity of all observers. Here, we illustrated the general workflow of the tool, presented the corresponding Python package `solarmach` and its PFSS extension, and described the important web version of it provided through the `streamlit` library. While the tool is intended to have a very simple interface, it is important to point out that for the magnetic backmapping several assumptions are made in the background. Those results should therefore only be interpreted by users with sufficient knowledge of the topic.

The development process of Solar-MACH started at the University of Kiel, Germany, and further discussion took place within the ESA Heliophysics Archives USer (HAUS) group. Now, the main development continues within the SERPENTINE EU Horizon 2020 project, which has been presented in the beginning of this paper. It is important to note that the project is, and remains, fully open source (under the BSD 3-Clause License) and very much encourages everyone to contribute, either by actual code or by any other kind of feedback.

The software is completely developed in Python and uses established open-source Python libraries such as `pfsspy`, `streamlit`, and `sunpy`. The development takes place continuously in public GitHub repositories (separated for the Python package `solarmach` and the web application), which are also the main entry point for any kind of participation. We encourage users to give feedback on bugs or possible new features by submitting GitHub issues. This should also be the starting point for code contributions. But of course we are also open to feedback through direct communication. All code versions released on GitHub are automatically archived at Zenodo with a DOI, so that each version (or the latest) can be cited independently.

Solar-MACH is already used widely in the scientific community (e.g., Badman et al., 2022; Hu et al., 2022; Mierla et al., 2022; Papaioannou et al., 2022; Rodríguez-García et al., 2022), but the development is still ongoing with the introduction of additional functionalities. Some important future plans have already been mentioned in this paper, such as fully integrating the PFSS extension into Solar-MACH. Further ideas are listed in the GitHub repository's issues section and contain, among others, using dynamically obtained solar wind speeds for the HMF lines, adding trajectories of spacecraft, or introducing features related to CME analyses, such as providing a reference cone instead of an arrow. Furthermore, it is planned to incorporate functions that have already been demonstrated in the Jupyter Notebooks into the core `solarmach` package, for example, the creation of animated GIF files with time-lapses of the spacecraft constellation. For this, like for all other Solar-MACH functions, one important aspect is to provide them in a way such that they remain accessible to all levels of Python users.

Data availability statement

The source code used in this study can be found online:

- Python package code in GitHub repository solarmach, archived at https://doi.org/10.5281/zenodo.7311178
- Streamlit code in GitHub repository Solar-MACH, archived at https://doi.org/10.5281/zenodo.7311215
- Extended example Jupyter Notebook in GitHub repository serpentine, archived at https://zenodo.org/record/7104866.

Author contributions

ND proposed the initial idea of Solar-MACH and discussed it with VG, AM, MR, and AV. Based on code by ND, the initial software version was designed by ND and JvF. JG continued the development. JG developed the Python package and the Streamlit version. JG, ND, CP, and RV designed further functionalities. CP and ND developed the pfsspy integration. AK, DP, LR-G, and DT tested the software and provided critical comments for the development. JG and RV prepared the first paper draft, and all authors were involved in the preparation of the final manuscript. All authors revised the manuscript before submission.

Funding

This study has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101004159 (SERPENTINE). JG, CP, RV, and DP acknowledge the support of Academy of Finland (FORESAIL, grants 312357, 336809, and 336807). ND acknowledges the support of Academy of Finland (SHOCKSEE, grant 346902). JvF thanks the German Space Agency (Deutsches Zentrum für Luft- und Raumfahrt e.V., DLR) for their support of his work on the Solar Orbiter EPD team at the University of Kiel under grant 50OT2002. AK acknowledges financial support from NASA's NNN06AA01C (SO-SIS Phase-E) contract. LR-G acknowledges the financial support by the Spanish Ministerio de Ciencia, Innovación y Universidades FEDER/MCIU/AEI Projects ESP2017-88436-R and PID2019-104863RB-I00/AEI/10.13039/501100011033.

Acknowledgments

The authors acknowledge the Heliophysics Archives USer (HAUS) group at ESA, where the idea for this tool was initially discussed. The authors would like to thank everyone who helped to improve Solar-MACH by providing feedback or contributing to the various open-source projects that it utilizes. The authors acknowledge the free cloud hosting provided by Streamlit Inc. The authors thank the two reviewers for their valuable feedback.

Conflict of interest

Author JvF was employed by the company Paradox Cat GmbH since 2021. This article is based on his research conducted prior to that at the University of Kiel and unrelated to his current commercial affiliation.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., et al. (2022). The Astropy Project: Sustaining and growing a community-oriented open-source project and the latest major release (v5.0) of the core package. *ApJ.* 935, 167. doi:10.3847/1538-4357/ac7c74

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., et al. (2018). The Astropy Project: Building an open-science project and status of the v2.0 core package. *AJ* 156, 123. doi:10.3847/1538-3881/aabc4f

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., et al. (2013). Astropy: A community Python package for astronomy. *A&A* 558, A33. doi:10.1051/0004-6361/201322068

Badman, S. T., Carley, E. P., Cañizares, L. A., Dresing, N., Jian, L. K., Lario, D., et al. (2022). Tracking a beam of electrons from the low solar corona into interplanetary space with the Low Frequency Array, Parker Solar Probe and 1 au spacecraft. doi:10.3847/1538-4357/ac90c2

Benkhoff, J., Murakami, G., Baumjohann, W., Besse, S., Bunce, E., Casale, M., et al. (2021). BepiColombo - mission overview and science goals. *Space Sci. Rev.* 217, 90. doi:10.1007/s11214-021-00861-4

Desai, M., and Giacalone, J. (2016). Large gradual solar energetic particle events. *Living Rev. Sol. Phys.* 13, 3. doi:10.1007/s41116-016-0002-5

Dresing, N., Gómez-Herrero, R., Heber, B., Klassen, A., Malandraki, O., Dröge, W., et al. (2014). Statistical survey of widely spread out solar electron events observed with STEREO and ACE with special attention to anisotropies. *A&A* 567, A27. doi:10.1051/0004-6361/201423789

Dresing, N., Gómez-Herrero, R., Klassen, A., Heber, B., Kartavykh, Y., and Dröge, W. (2012). The large longitudinal spread of solar energetic particles during the 17 January 2010 solar event. *Sol. Phys.* 281, 281–300. doi:10.1007/s11207-012-0049-y

Fox, N. J., Velli, M. C., Bale, S. D., Decker, R., Driesman, A., Howard, R. A., et al. (2016). The Solar Probe Plus Mission: Humanity's first visit to our star. *Space Sci. Rev.* 204, 7–48. doi:10.1007/s11214-015-0211-6

Gieseler, J., Dresing, N., and Freiherr von Forstner, J. L. (2022a). jgieseler/Solar-MACH: 0.1.5. doi:10.5281/zenodo.7311215

Gieseler, J., Dresing, N., Freiherr von Forstner, J. L., Palmroos, C., and Vainio, R. (2022b). jgieseler/solarmach: 0.1.5. doi:10.5281/zenodo.7311178

Gieseler, J., Palmroos, C., Dresing, N., Morosan, D., Asvestari, E., Yli-Laurila, A., et al. (2022c). serpentine-h2020/serpentine: v1.1. doi:10.5281/zenodo.7139716

Ginsburg, A., Sipcz, B., Parikh, M., Brasseur, C. E., jcsegovia, Groener, A., et al. (2021). Astropy/astroquery: v0.4.5. doi:10.5281/zenodo.5804082

Hu, Q., He, W., and Chen, Y. (2022). A magnetic flux rope configuration derived by optimization of two-spacecraft *in-situ* measurements. *Front. Phys.* 10, 960315. doi:10.3389/fphy.2022.960315

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.* 9, 90–95. doi:10.1109/MCSE.2007.55

Kouloumvakos, A., Rodríguez-García, L., Gieseler, J., Price, D., Vourlidas, A., and Vainio, R. (2022). PyThea: An open-source software package to perform 3d reconstruction of coronal mass ejections and shock waves. *Front. Astronomy Space Sci.* 9. doi:10.3389/fspas.2022.974137

Lario, D., Kwon, R.-Y., Vourlidas, A., Raouafi, N. E., Haggerty, D. K., Ho, G. C., et al. (2016). Longitudinal properties of a widespread solar energetic particle event on 2014 February 25: Evolution of the associated CME shock. *ApJ.* 819, 72. doi:10.3847/0004-637X/819/1/72

Mackay, D. H., and Yeates, A. R. (2012). The Sun's global photospheric and coronal magnetic fields: Observations and models. *Living Rev. Sol. Phys.* 9, 6. doi:10.12942/lrsp-2012-6

McKinney, W. (2010). "Data structures for statistical computing in Python," in *Proceedings of the 9th Python in science conference*. Editors Stéfan van der Walt, and Jarrod Millman, 56–61. doi:10.25080/Majora-92bf1922-00a

Mierla, M., Zhukov, A. N., Berghmans, D., Parenti, S., Auchère, F., Heinzel, P., et al. (2022). Prominence eruption observed in He II 304 Å up to >6 $R_\odot$ by EUI/FSI aboard Solar Orbiter. *A&A* 662, L5. doi:10.1051/0004-6361/202244020

Müller, D., Cyr, O. C., Zouganelis, I., Gilbert, H. R., Marsden, R., Nieves-Chinchilla, T., et al. (2020). The Solar Orbiter Mission. Science overview. *A&A* 642, A1. doi:10.1051/0004-6361/202038467

Mumford, S. J., Freij, N., Stansby, D., Christe, S., Ireland, J., Mayer, F., et al. (2022). SunPy. doi:10.5281/zenodo.7074315

Palmroos, C., Gieseler, J., Dresing, N., Mororsan, D. E., Asvestari, E., Valkila, S., et al. (2022). Solar energetic particle time series analysis with Python. *Front. Astronomy Space Phys.* doi:10.3389/fspas.2022.1073578

Papaioannou, A., Kouloumvakos, A., Mishev, A., Vainio, R., Usoskin, I., Herbst, K., et al. (2022). The first ground-level enhancement of solar cycle 25 on 28 October 2021. *A&A* 660, L5. doi:10.1051/0004-6361/202142855

Parker, E. N. (1958). Dynamics of the interplanetary gas and magnetic fields. *ApJ.* 128, 664. doi:10.1086/146579

Poljančić Beljan, I., Jurdana-Šepić, R., Brajša, R., Sudar, D., Ružđjak, D., Hržina, D., et al. (2017). Solar differential rotation in the period 1964-2016 determined by the Kanzelhöhe data set. *A&A* 606, A72. doi:10.1051/0004-6361/201731047

Reames, D. V. (1999). Particle acceleration at the Sun and in the heliosphere. *Space Sci. Rev.* 90, 413–491. doi:10.1023/A1005105831781

Reames, D. V. (2013). The two sources of solar energetic particles. *Space Sci. Rev.* 175, 53–92. doi:10.1007/s11214-013-9958-9

Richardson, I. G., von Rosenvinge, T. T., Cane, H. V., Christian, E. R., Cohen, C. M. S., Labrador, A. W., et al. (2014). >25 MeV proton events observed by the high energy telescopes on the STEREO A and B spacecraft and/or at Earth during the first ∼ seven years of the STEREO mission. *Sol. Phys.* 289, 3059–3107. doi:10.1007/s11207-014-0524-8

Rodríguez-García, L., Gómez-Herrero, R., Zouganelis, I., Balmaceda, L., Nieves-Chinchilla, T., Dresing, N., et al. (2021). The unusual widespread solar energetic particle event on 2013 August 19. Solar origin and particle longitudinal distribution. *A&A* 653, A137. doi:10.1051/0004-6361/202039960

Rodríguez-García, L., Gómez-Herrero, R., Dresing, N., Lario, D., Zouganelis, I., Balmaceda, L. A., et al. (2022). Solar energetic electron events measured by MESSENGER and Solar Orbiter. Peak intensity and energy spectrum radial dependences: Statistical analysis. *A&A*. (in press). doi:10.1051/0004-6361/202244553

Stansby, D., Badman, S., Ancellin, M., and Barnes, W. (2022). dstansby/pfsspy: pfsspy 1.1.2. doi:10.5281/zenodo.7025396

Stansby, D., Yeates, A., and Badman, S. T. (2020). pfsspy: A python package for potential field source surface modelling. *J. Open Source Softw.* 5, 2732. doi:10.21105/joss.02732

The SunPy Community Barnes, W. T., Bobra, M. G., Christe, S. D., Freij, N., Hayes, L. A., et al. (2020). The SunPy project: Open source development and status of the version 1.0 core package. *ApJ.* 890, 68. doi:10.3847/1538-4357/ab4f7a

Thompson, W. T. (2006). Coordinate systems for solar image data. *A&A* 449, 791–803. doi:10.1051/0004-6361:20054262

Trotta, D., Vuorinen, L., Hietala, H., Horbury, T. S., Dresing, N., Gieseler, J., et al. (2022). Single-spacecraft techniques for shock parameters estimation: A systematic approach. *Front. Astronomy Space Phys.* (accepted). doi:10.3389/fspas.2022.1005672

Vainio, R., Desorgher, L., Heynderickx, D., Storini, M., Flückiger, E., Horne, R. B., et al. (2009). Dynamics of the Earth's particle radiation environment. *Space Sci. Rev.* 147, 187–231. doi:10.1007/s11214-009-9496-7

van Haarlem, M. P., Wise, M. W., Gunst, A. W., Heald, G., McKean, J. P., Hessels, J. W. T., et al. (2013). Lofar: The LOw-Frequency ARray. *A&A* 556, A2. doi:10.1051/0004-6361/201220873

Van Rossum, G., and Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 13 March 2023
DOI 10.3389/fspas.2022.1077845

Check for updates

OPEN ACCESS

EDITED BY
John Coxon,
Northumbria University, United Kingdom

REVIEWED BY
Ashley Smith,
University of Edinburgh, United Kingdom
Agnit Mukhopadhyay,
National Aeronautics and Space Administration, United States

*CORRESPONDENCE
Adrian Blagau,
blagau@spacescience.ro

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 23 October 2022
ACCEPTED 05 December 2022
PUBLISHED 13 March 2023

CITATION
Blagau A and Vogt J (2023), SwarmFACE: A Python package for field-aligned currents exploration with Swarm.
Front. Astron. Space Sci. 9:1077845.
doi: 10.3389/fspas.2022.1077845

COPYRIGHT
© 2023 Blagau and Vogt. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

SwarmFACE: A Python package for field-aligned currents exploration with Swarm

Adrian Blagau[1,2]* and Joachim Vogt[2]

[1]Institute of Space Science, Magurele, Ilfov, Romania, [2]Jacobs University Bremen, Bremen, Germany

The SwarmFACE package utilizes magnetic field measurements by the Swarm satellites to study systems of field-aligned currents (FACs). Improvements of well-established techniques as well as novel single- and multi-satellite methods or satellite configurations are implemented to extend the characterization of FAC systems beyond the Swarm official Level-2 FAC product. Specifically, the included single-satellite algorithm allows to consider the FAC sheet inclination with respect to the satellite orbit and can work with low- or high-resolution data. For dual-satellite FAC estimation the package provides three algorithms, based on the least-squares, on the singular value decomposition, and on the Cartesian boundary-integral methods. These algorithms offer advantages over the corresponding Level-2 algorithm by providing more stable solutions for 'extreme' configurations, e.g. close to the orbital cross-point, and by allowing for a more general geometry of the spacecraft configuration. In addition, the singular value decomposition algorithm adapts itself to the spacecraft configuration, allowing for continuous, dual-satellite based FAC solutions over the entire polar region. Similarly, when Swarm forms a close configuration, the package offers the possibility to estimate the FAC density with a three-satellite method, obtaining additional information, associated to a different (larger) scale. All these algorithms are incorporating a robust framework for FAC error assessment. The SwarmFACE package further provides useful utilities to automatically estimate the auroral oval location or the intervals when Swarm forms a close configuration above the auroral oval. In addition, for each auroral oval crossing, a series of FAC quality indicators, related to the FAC methods' underlying assumptions, can be estimated, like the current sheet inclination and planarity or the degree of current sheet stationarity.

KEYWORDS

field-aligned currents, Swarm mission, Python, auroral oval, magnetosphere-ionosphere coupling, single and multi satellite gradient methods, minimum variance analysis, space science

1 Introduction

The ESA's Swarm three-satellite mission Friis-Christensen et al. (2006) has as its primary objective the characterization of geomagnetic field and its temporal variations, providing the basis for the development of improved geomagnetic reference models. At the same time, by monitoring the ionosphere region, the mission offers new scientific insights into the thermosphere–ionosphere - magnetosphere system, with application in related areas (ionospheric models, space weather etc). Half a year after the launch of the satellites in 2013, the Swarm operational phase began in April 2014, producing data of excellent quality ever since. Each Swarm satellite hosts (among other experiments) a Vector Field Magnetometer that provides high-accuracy magnetic field measurements. These data are made available though the so called Level-1b magnetic products, both at high-resolution (HR), i.e. 50 Hz, and low-resolution (LR), i.e. 1 Hz, cadence.

Two Swarm satellites, i.e. A and C, are traveling side by side at an altitude of about 460 km, with standard angular separation between their orbital planes of 1.4°. The third satellite, Swarm B, revolves at an altitude of around 50 km higher, on an orbit having a slightly different inclination, setting thus a relative drift between its orbital planes and the orbital planes of the lower satellites of about 22.5°/year. In October 2019 the orbit inclination of Swarm A has been slightly changed so that the angular separation between the lower satellites started to decrease. As a consequence, around October 2021, the orbital planes of all three satellites were very close one to another (close orbit configuration), with Swarm B rotating in the opposite sense (i.e. angular separation of 180°).

Magnetic field-aligned currents (FACs) in the auroral zone are ultimately generated by interactions of the Earth's magnetosphere with the magnetized plasma of the solar wind that build up stresses causing FACs to flow from the magnetopause and remote magnetospheric regions towards the ionosphere at high latitudes (see e.g. Vogt et al., 1999). FACs thus act as coupling agents in the solar wind-magnetosphere-ionosphere system. At the altitudes of Low Earth Orbiting (LEO) satellites, FACs are often organized in planar current sheets, roughly oriented along the geomagnetic East-West direction. Their distribution and dynamics are usually inferred from the associated magnetic signature, detected by space-borne magnetometers (see, e.g., the early work of Zmuda and Armstrong (1974) and Iijima and Potemra (1976) or, more recent one by He et al. (2012)).

Currently, the Swarm mission provides the following FAC Level-2 products:

- Product FAC$\times$TMS_2F with $\times$ = A, B or C, respectively for Swarm A, Swarm B or Swarm C. This provides the single-satellite FAC data sets, estimated from the LR magnetic field measurements. The data cover the mid- and high-latitude parts of the orbit (i.e. magnetic latitude > 30° in either hemisphere).
- Product FAC_TMS_2F, for the dual-satellite FAC data set, based on the (filtered) LR magnetic field observations from Swarm lower satellites pair. The data are provided as well for magnetic latitude > 30° with the exception of a ±4° region close to the geographic poles (the so-called Exclusion Zone, EZ, see details in **Section 2.2.1**), where the cross-track separation between Swarm A and Swarm C becomes too small.

These data sets are produced with algorithms based on the methods introduced in Ritter et al. (2013) (see also Ritter and Lühr (2006)). The algorithms provide reliable estimates but, being developed before the launch of Swarm, could not take into account all the aspects of FAC density estimation and have not benefited from the expertise acquired while working with real measurements. In addition, the data sets have the shortcoming of being static data products, rather than generated with an interactive toolbox.

The SwarmFACE package provides *Python* programs to generate novel or refined FAC estimates based on Swarm observations, together with appropriate FAC quality indicators. The algorithms are based on novel methods (Vogt et al., 2009, 2013, 2020) or are adapted versions of well-established ones (for example the dual-satellite Boundary Integral method from Ritter et al. (2013), the single-satellite Finite Differencing from e.g. Luhr et al. (1996), or the Minimum Variance Analysis from Sonnerup and Cahill (1967)). We briefly mention here the advantages brought by the new algorithms, and give more details in **Section 2**:

- The single-satellite algorithm can estimate FAC density based on both LR or HR magnetic field data. Input data can be filtered for tuning the analysis at small/large scales, and the current sheet inclination with respect to the satellite orbit can be taken into account.
- The dual-satellite algorithms based on the Least Squares and on the adapted Boundary Integral methods yield more stable solutions, which imply smaller EZ extension (details are provided in **Section 4**).
- The dual-satellite method based on Singular Value Decomposition adapts it self to regions with degenerate configurations (i.e. when the satellite cross-track separation is much smaller than the along-track distance, resulting in a four-point configuration close to a line; see **Section 2.2**), being thus able to provide continuous, dual-satellite based, FAC solutions, i.e. no data gaps near to the orbital cross-points (Vogt et al., 2020).
- All dual-satellite algorithms can work on irregular configuration geometries (i.e. non-parallel orbits or skewed

configurations). For example, they can be applied not just to the lower satellite pair but also by combining Swarm B and Swarm A/C data, when Swarm B is sufficiently close.

- The three-satellite method, meant to be applied when Swarm forms a close configuration, offers additional information on the FAC system at different (larger) scale (Blagau and Vogt, 2019). Unlike the single- and dual-satellite methods that require a certain degree of time stationarity in the observed structures, the three-satellite method can provide instantaneous gradient and *curl* estimates under suitable geometric conditions (Vogt et al., 2009).
- All the dual- and the three-satellite methods rely on a robust way to infer the level of errors in FAC estimations as a function of constellation geometry and magnetic field uncertainties.

The SwarmFACE routines implicitly run on pre-defined settings but, optionally, this can be changed by the users according to their needs. For example, in the dual-satellite methods, the same satellite configuration used to generate FAC Level-2 product is implicitly considered (i.e. one sensor is shifted in time to obtain alignment in orbital phase and an along track separation of 5 s is used; see **Section 2.2**) but that can be modified. Also, in case of the single-satellite algorithm, normal (with respect to satellite velocity) orientation of current sheets is implicitly assumed, but the user can provide a different current sheet normal, when such information is available (e.g. from Minimum Variance Analysis).

In addition, the package provides useful routines that (i) automatically identifies the auroral oval (AO) location, (ii) finds Swarm conjunctions above the AO, and (iii) computes the FAC quality indices to help the user assessing the quality of current density estimations.

The magnetic field measurements and the auxiliary parameters/data-sets needed to run SwarmFACE are available on the VirES (Virtual environments for Earth Scientists) for Swarm platform (https://vires.services) which holds a database with the latest versions of Swarm data products. There are several advantages of using VirES platform instead of working directly with the Swarm files available on the ESA database, like avoiding the handling of geomagnetic field models, the access to additional auxiliary data (e.g. quasi-dipole coordinates, magnetic local time), and a smoother portability of the code. The actual data retrieval is done by invoking `viresclient` (Smith et al., 2022b), a *Python* client that manages the communication with the database and makes the data available as *Python* data types. For more information about VirES and `viresclient` the reader is referred to Smith et al. (2022a), this issue. The SwarmFACE package relies as well on other more general *Python* packages like `NumPy` (Harris et al., 2020), `pandas` (pandas development team, 2020), and `SciPy` (Virtanen et al., 2020).

Some of the algorithms now part of the SwarmFACE package have been used right from the beginning of the mission, when the authors have been involved in the Swarm data calibration/validation task called by ESA. Those programs were refined and new ones were developed during the implementation of the ESA SIFACIT project, that aimed to tap the full potential offered by the Swarm measurements for FAC exploration. Originally written in IDL as part of a larger package designed to analyze both Cluster and Swarm multi-satellite/multi-instrument measurements, the routines have been extensively tested and their predictions compared with the values of the FAC Level-2 products (Blagau and Vogt, 2019; Vogt et al., 2020).

The SwarmFACE code is available on GitHub (https://github.com/ablagau/SwarmFACE) and Zenodo (https://doi.org/10.5281/zenodo.7361438) open repositories. Jupyter notebooks that illustrates the application of each high-level routine are available in the `notebooks` subdirectory of the distribution. The package documentation is available at https://swarmface.readthedocs.io/en/latest/. Before the development of SwarmFACE package, a Swarm FAC analysis toolbox, consisting of self-contained Jupyter notebooks that rely on the earlier versions of SwarmFACE low-level routines, has been made available on the Virtual Research Environment accessible through the VirES platform (see also the GitHub repository at https://github.com/Swarm-DISC/FAC_exploration).

The next Section provides details about the underlying methods used by the SwarmFACE routines to estimate the FAC density and the associated quality indices. **Section 3** describes the calling sequence for each higher level routine and presents example of code usage. The last Section discusses various aspects like testing, limitations, result validation and ways for improvements.

2 Methods

In all the methods presented below, the high-latitude magnetic field perturbations **b**, computed by subtracting a magnetic field model from the actual measurements **B**, are considered to be produced by the FACs and therefore used to estimate FACs density (the influence of auroral electrojet on FAC computation is briefly discussed in **Section 4**). Currently, SwarmFACE applies the CHAOS magnetic field model [see e.g. Finlay et al. (2020)], but any model available on VirES platform can be used, with basically the same outcome on the FAC estimation (see **Section 4**).

The algorithms are designed to work with vectors represented in the Cartesian (rectangular) geographic reference frame (indicated as GEO below), more precisely the International Terrestrial Reference Frame (ITRF) also used in Level-1b files to specify the position of Swarm satellites. Since in the same files, the magnetic field measurements are provided in the local North East

Center (NEC) frame, one of the first task is to convert them to GEO. For completeness, ITRF is the Earth-centered, Earth-fixed reference with the z axes oriented towards the North pole, i.e. along the Earth rotation axis, and the x axis passing through the Greenwich meridian. The NEC frame is centered at the satellite location, with the z axis pointing towards the Earth center and the y axis obtained as the cross-product of the z axis and the direction of Earth rotation axis.

For data filtering, SwarmFACE uses a 20 s low-pass Butterworth filter of order 4. This type of filter, implemented in the *Python* SciPy library (module scipy.signal.butter), provides the flattest frequency response within the pass band. The choice of a 20 s bandwidth (like in the FAC dual-satellite Level-2 product) follows from the same reasons presented in Ritter et al. (2013) i.e. to reduce the influence of local small-scale fluctuations and concentrate the analysis on the large-scale FACs (scale length larger than $\sim$ 150 km, corresponding to $\sim$ 20 s of Swarm orbit section). Similarly, in the dual-satellite algorithm the bandwidth ensures meaningful gradient estimations, considering that in the nominal orbital separation phase, the satellite cross-track distance decreases from $\sim$ 160 km at the equator to $\sim$ 60 km at auroral latitudes.

2.1 Single-satellite FAC estimation

The SwarmFACE single-s/c algorithm to estimate the FAC and ionospheric radial current (IRC) densities offers some advantages over the one used to generate the Swarm FAC Level-2 product (Ritter et al., 2013) since (i) both low (LR) and high resolution (HR) Level-1b magnetic field data can be used, (ii) input data can be filtered by the user, e. g for making analysis at small/large scale, and (iii) the inclination of FAC sheet can be taken into account provided that this information is known (e.g. as a result of applying the Minimum Variance Analysis—see **Section 2.4**).

The algorithm works by computing central finite differences of magnetic field perturbation along the orbital track. The current sheet probed by the satellite is assumed to be a planar magnetic field oriented structure, where all the variations in magnetic field perturbations occur only along its normal. This is illustrated on **Figure 1A**, where $\mathbf{B}$ and $\mathbf{N}_{3d}$ indicate the local magnetic field and, respectively, the current sheet normal direction. From the magnetic field perturbations sampled in the tangential plane, i.e. the plane perpendicular to the radial direction $\mathbf{C}$ (see **Figure 1B**) one can compute the IRC density by

$$j_{\mathrm{IRC}} = \frac{1}{\mu_0}\frac{b_y(t+dt) - b_y(t)}{x(t+dt) - x(t)} \tag{1}$$

where $\mathbf{x}$ is along $\mathbf{N}_{2d}$, i.e. the projection of $\mathbf{N}_{3d}$ on the tangential plane, and $\mathbf{y}$ designates the direction tangent to the current sheet, such that $\mathbf{C} = \mathbf{x} \times \mathbf{y}$. Assuming a level δb for the magnetic field errors, the corresponding current density error is

$$\delta j_{\mathrm{IRC}} = \frac{1}{\mu_o}\frac{\delta b}{L} \tag{2}$$

where L is the distance between the points of measurements along $\mathbf{N}_{2d}$. In case of Swarm, the overall estimated noise level in the magnetic Level-1b data is $\lesssim$ 0.5 nT rms (see Tøffner-Clausen et al., 2016).

The FAC density j_{FAC} is simply obtained as

$$j_{\mathrm{FAC}} = \frac{j_{IRC}}{cos\beta} \tag{3}$$

where β designates the inclination of magnetic field, i.e. the angle between $\mathbf{C}$ and $\mathbf{B}$. Note that for Eq. 3 to be used in practice, one needs β values not too close to 90°, a condition always satisfied at auroral latitude but not near the equatorial region.

When no information on current sheet orientation is provided (the default running option), the algorithm assumes normal satellite incidence, i.e. $\mathbf{N}_{2d}$ along the satellite velocity vector $\mathbf{v}$. Otherwise, this information can be supplied as the $\mathbf{N}_{3d}$ vector or, similarly, in the form of $\mathbf{N}_{2d}$ or α (the angle between $\mathbf{N}_{2d}$ and $\mathbf{v}$). All the computations are performed in the GEO frame.

2.2 Dual-satellite FAC estimation

SwarmFACE provides three high-level function to estimate the FAC densities from dual-satellite measurements. All these algorithms follow a similar procedure that differs only in the last step, i.e. point (3) below:

1. Four-point configurations (quads) are formed along the orbit by combining two virtual positions from each satellite. To ensure, as much as possible, formation of regular quads, data from one satellite is time-shifted in the first place;
2. The magnetic field perturbations recorded by the two satellites are low-pass filtered, to minimize the influence of small-scale fluctuations at each satellite and to match the scale of the magnetic perturbations with the configuration scale;
3. The discretized form of the Ampère's law is used to infer the current flowing through the quad and the FAC density is estimated by considering the orientation of local magnetic field.

Below we provide details and underline the characteristics of each algorithm available in SwarmFACE package. In this process we take as reference the algorithm used to generate the official Level-2 product (see Ritter et al., 2013).

2.2.1 Dual-satellite FAC estimation by Least Squares

The algorithm based on the Least Squares method, developed in Vogt et al. (2013), offers a series of advantages over the corresponding Level-2 FAC algorithm, listed below. For a

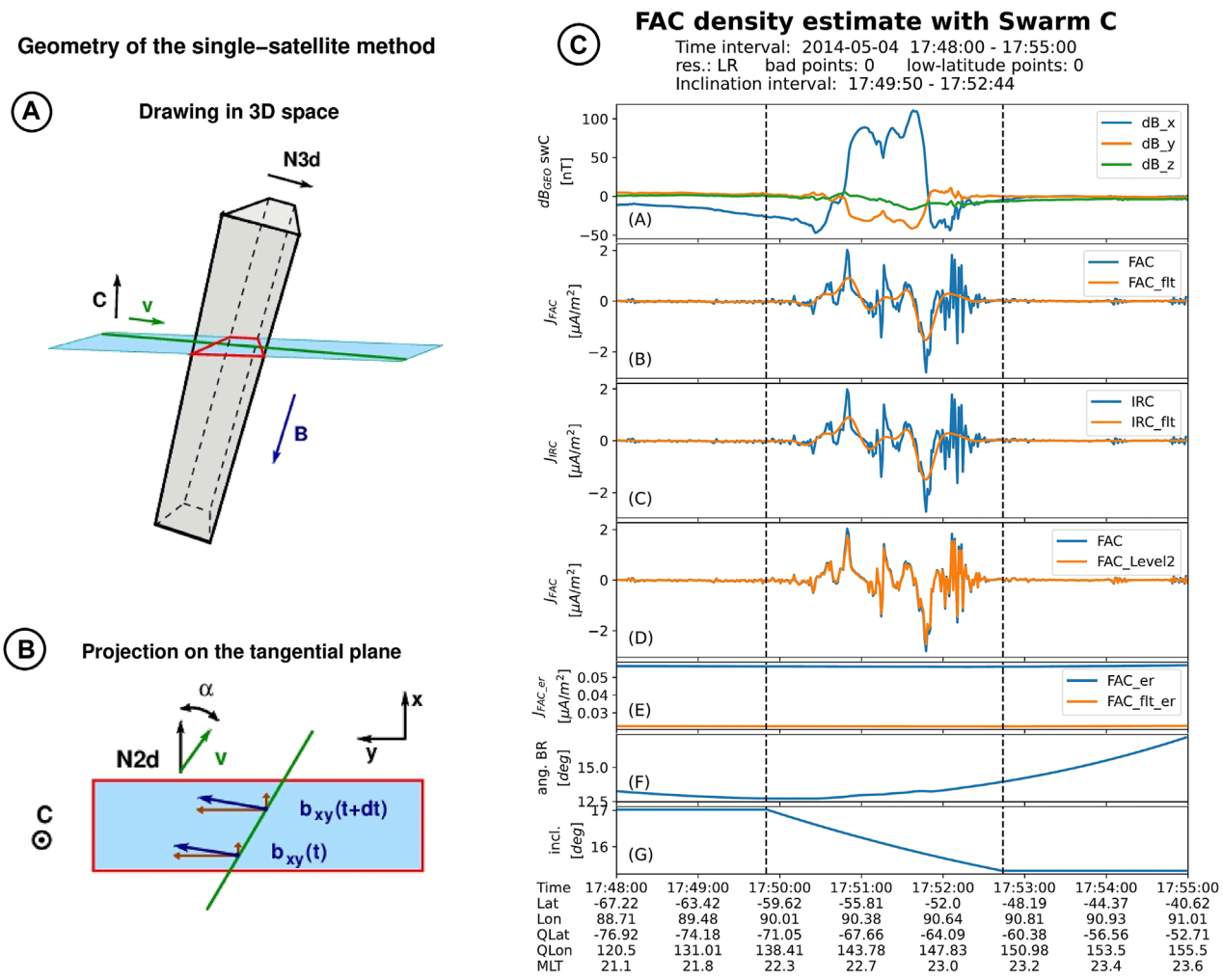


FIGURE 1
FAC density estimate by single-satellite method. **(A)**: Schematic of a current sheet oriented along the local magnetic field *B* and normal direction along **N3d** sampled by a satellite moving with velocity **v**. **(B)**: Schematic projection on the plane perpendicular to the local radial direction *C*. Here **b** designates the magnetic field perturbation. **(C)**: Standard plot generated by the `j1sat` algorithm. From top to bottom the figure shows (A) the Swarm magnetic field perturbation in GEO frame, (B) the un-filtered and filtered FAC densities, (C) the un-filtered and filtered IRC densities, (D) a comparison between FAC estimated with `j1sat` and the Level-2 product, (E) the errors in un-filtered and filtered FAC densities, (F) the angle between the local magnetic field and the radial direction, and (G) the angle α in the tangential plane between the current sheet normal and the satellite velocity. The labels at the bottom presents time in UTC, the satellite geographic and quasi-dipole latitude and longitude, as well as the magnetic local time.

detailed analysis on these aspects, the reader is referred to Blagau and Vogt (2019), illustrated with suggestive Swarm events.

Firstly, the method yields more stable solutions for "extreme" configurations, when the influence of local magnetic field perturbations is highly amplified (e.g. the very elongated quads near the orbital cross-points or the skewed quads encounter during the more recent Swarm phase, with closer orbital planes for the lower satellites). The explanation resides on the fact that one deals with an over-determined problem (i.e. observations from three, instead of four points in a plane are in principle enough to estimate the normal component of *curl* operator) and the LS approach accommodates the contributions of all points by minimizing the estimation errors. The more stable solution of the LS algorithm implies a reduction in the so-called Exclusion Zone (EZ), i.e. the regions near the cross-orbit points where the quad configurations becomes too elongated and the current density estimate is not provided due to its artificially high values.

Then, the configuration geometry considered in the algorithm is more general, see **Figure 2A** [reproduced from Vogt et al. (2013)]. As a result, deviations from parallel satellite orbits or skewed configurations are properly taken into account. This allows, in principle, to apply the algorithm not only to the

pair of lower Swarm satellites but, occasionally, to combine data from Swarm B and Swarm A or Swarm C, provided that all three satellites form a close configuration.

In addition, the method provides a robust way to infer the level of errors in FAC estimations as a function of quad geometry and magnetic field uncertainties. The error estimation scheme comes naturally from the formalism and has been endorsed by Monte-Carlo simulations.

Below, one reproduces a short description of the LS algorithm; for details the reader is referred to the original paper of Vogt et al. (2013). If f designates one component of the magnetic field perturbation in the plane of the quad (the (u, v) plane in **Figure 2A**), with $f_\sigma = f^{+,-}_{a,b}$ the values at the leading and trailing apex for satellite a and b, and with $\boldsymbol{r}_\sigma = \boldsymbol{r}^{+,-}_{a,b}$ the corresponding position vectors, then the components of its planar gradient $\nabla_p f$ can be expressed as

$$\nabla_p f = \sum_{\sigma=1}^{4} f_\sigma \boldsymbol{q}_\sigma = f^+_a \boldsymbol{q}^+_a + f^-_a \boldsymbol{q}^-_a + f^+_b \boldsymbol{q}^+_b + f^-_b \boldsymbol{q}^-_b \tag{4}$$

Here $\boldsymbol{q}_\sigma$ are the so called canonical base vectors; to find them, one has to solve the equations

$$\mathbf{R}_{pos} \boldsymbol{q}_\sigma = \boldsymbol{r}_\sigma , \tag{5}$$

where $\mathbf{R}_{pos}$ is the position tensor $\mathbf{R}_{pos} = \sum_{\sigma=1}^{4} \boldsymbol{r}_\sigma \boldsymbol{r}_\sigma^t$, represented by a 2 × 2 matrix in the (u, v) coordinates. One critical aspect of the algorithm is related to this inversion problem. The stability of the solution to Eq. **5** is characterized by the so called condition number (a factor that enters in the error estimation scheme) $CN = \lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are the eigenvalues of $\mathbf{R}_{pos}$ arranged in descending order.

Assuming mutually uncorrelated and isotropic magnetic field errors at the points of measurements, i.e. $<\delta\mathbf{b}_\sigma \delta\mathbf{b}_\tau> = \delta_{\sigma\tau}(\delta b)^2\mathbf{I}$, with σ, τ sensor indexes, $\delta_{\sigma\tau}$ the Kronecker symbol, and $\mathbf{I}$ the unit matrix, then the mean square error of the radial current j_n through the spacecraft plane is provided by

$$\langle|\delta j_n|^2\rangle = \frac{(\delta b)^2}{\mu_0^2} \sum_\sigma |\mathbf{q}_\sigma|^2 \tag{6}$$

Note that in Vogt et al. (2013) the symbol $\mathbf{R}$ is used for the position tensor. In this paper we adhere to the notations from Vogt et al. (2020), where $\mathbf{R}$ designates the position matrix and the position tensor is indicated by $\mathbf{R}_{pos}$ (see below).

2.2.2 Dual-satellite FAC estimation by Singular Value Decomposition

The dual-satellite SVD algorithm from SwarmFACE represents an adaptation of the more general RASADA (from Robust Adaptive Spacecraft Array Derivative Analysis) method and code developed in Vogt et al. (2020). RASADA allows to estimate the spatial derivative of physical quantities and their corresponding errors from an array of arbitrary numbers of satellites/observation points. Remarkably, the method automatically adapts to possible degenerate geometries of the array configuration, that could be a source of large estimation errors, by identifying and keeping only non-degenerate directions in the analysis.

The general RASADA algorithm is clearly described in Vogt et al. (2020), Section 2. Below we present the computation steps adapted to the context of Swarm dual-satellite FAC estimation:

1. The quad's position vectors $\mathbf{r}_\sigma$, computed with respect to the quad center (i.e. in the mesocentric frame, where $\sum_{\sigma=1}^{4} \mathbf{r}_\sigma = 0$) are stored as rows in the (4×3) position matrix $\mathbf{R}$. The position matrix $\mathbf{R}$ is related to the position tensor $\mathbf{R}_{pos} = \sum_{\sigma=1}^{4} \boldsymbol{r}_\sigma \boldsymbol{r}_\sigma^t$ by $\mathbf{R}_{pos} = \mathbf{R}^t\mathbf{R}$.
2. The SVD of the position matrix is performed, $\mathbf{R} = \mathbf{U}\mathbf{S}\mathbf{V}^t$, where $\mathbf{U}$ is a (4×3) matrix satisfying $\mathbf{U}\mathbf{U}^t = \mathbf{I}$ (with $\mathbf{I}$ the unit matrix) and $\mathbf{V}$ is a (3×3) orthogonal matrix satisfying $\mathbf{V}\mathbf{V}^t = \mathbf{V}^t\mathbf{V} = \mathbf{I}$. The (3×3) matrix $\mathbf{S}$ is diagonal; since in our case the configuration is degenerated to a planar geometry, $\mathbf{S}$ has only two non-zero elements, $\mathbf{S} = \mathrm{diag}(R^{(1)}, R^{(2)}, 0)$. Here $R^{(1)} \geq R^{(2)} \geq 0$.
3. Form a stable pseudoinverse of $\mathbf{S}$. For that purpose, compute $\tau = R^{(2)}/R^{(1)}$; if τ is below a set threshold τ_0, meaning that the quad's configuration is too elongated along the direction associated with $R^{(1)}$ (i.e. close to linear degeneracy), consider only the first eigenvalue during the inversion:

$$\mathbf{S}^- = \begin{cases} \mathrm{diag}\left(1/R^{(1)}, 1/R^{(2)}, 0\right) & \text{if} \quad \tau \geq \tau_0 \\ \mathrm{diag}\left(1/R^{(1)}, 0, 0\right) & \text{if} \quad \tau < \tau_0 \end{cases} \tag{7}$$

4. Compute the (4×3) matrix $\mathbf{Q} = \mathbf{U}\mathbf{S}^-\mathbf{V}^t$. The rows of $\mathbf{Q}$ are the (generalized) reciprocal vectors $\mathbf{q}_\sigma, \sigma = 1, \ldots 4$
5. Considering the magnetic field perturbation at the quad's vertices $\mathbf{b}_\sigma, \sigma = 1, \ldots 4$, the *curl* $\mathbf{c} = \nabla \times \mathbf{b}$ is estimated through $\mathbf{c} \simeq \sum_\sigma \mathbf{q}_\sigma \times \mathbf{b}_\sigma$
6. Assuming isotropic and mutually uncorrelated instrumental errors δb for each component of $\mathbf{b}$, the mean square error of the radial current j_n through the spacecraft plane is provided by $\langle|\delta j_n|^2\rangle = \mathrm{trace}(\mathbf{Q}^t\mathbf{Q})(\delta b)^2/\mu_0^2$

According to third step above, when $\tau \geq \tau_0$ the quad configuration is considered non-degenerated (to a linear geometry) and the LS and SVD provides exactly the same solution. Near the orbital cross-points, when $\tau < \tau_0$, only the eigenvector associated with the biggest eigenvalue (i.e. the along-track direction) is used in the analysis and the algorithm yields estimates that are close to the average (over the satellites) single-satellite estimates. This provides the notable advantage to have dual-satellite based FAC solutions with no data gaps in the region close to the orbital cross-points.

Nevertheless, the transitions between different gradient estimator at the point where the singular ratio threshold τ_0 is crossed leads to discontinuous current density solutions. In

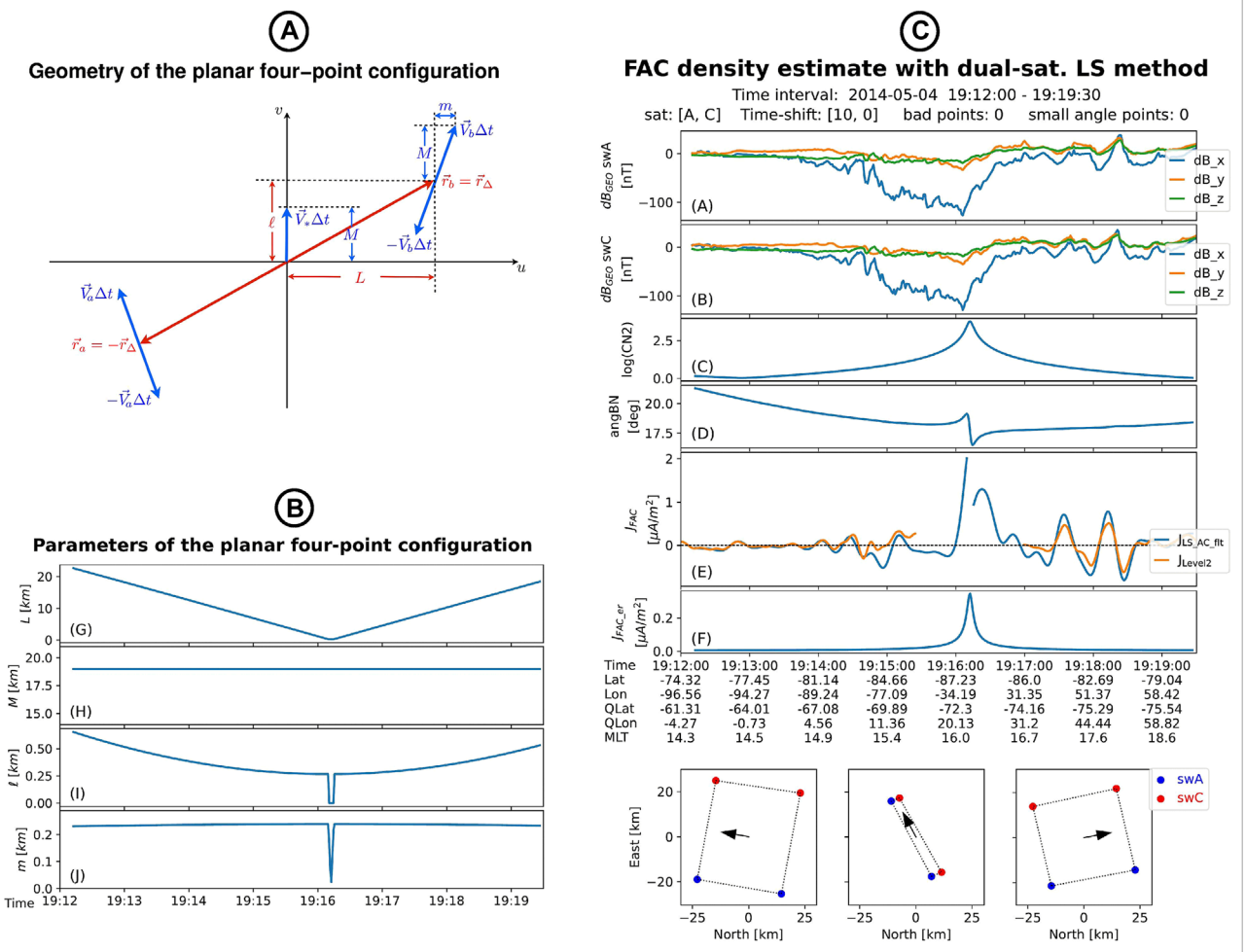


FIGURE 2
FAC density estimate by dual-satellite LS method. **(A)**: Geometry of the planar four-point configuration in the dual-satellite methods. The different parameters that characterizes the configurations are L (half cross track separation), M (half along track separation), ℓ (half orbital lag) and m, with $\epsilon = m/M$ describing the deviation of satellite trajectories from parallel orbits. Taken from Vogt et al. (2013). **(B)**: Parameters of the planar four-point configuration for the event shown on the right. The evolution along the orbit of L, M, ℓ, and m are presented in panels G–J. **(C)**: Standard plot generated by the `j2satLS` algorithm. From top to bottom the figure shows (A and B) the Swarm magnetic field perturbation in the GEO frame, (C) the logarithm of the condition number, (D) the angle between B and the direction normal to the quad, (E) comparison between the dual-satellite LS FAC estimation (blue) and the Level-2 product (orange), and (F) the error level in FAC density estimation. At the bottom, the quad configuration at three instances (i.e. start time, stop time, and at the middle of the interval) is presented as projection on the North-East plane of the local NEC frame.

the SwarmFACE implementation of the SVD algorithm, when such discontinuous behavior is undesired, the transitions may be smoothed by using a pseudoinverse matrix of the form $\mathbf{S}^{-} = \mathrm{diag}(1/R^{(1)}, f(\tau)/R^{(2)}, 0)$, with the function $f = f(\tau)$ defined in a stepwise manner:

$$f(\tau) = \begin{cases} 1 & \text{if} \quad \tau \geq \tau^{\star} \,, \\ g(\tau) & \text{if} \quad \tau^{\star} > \tau \geq \tau_0 \,, \\ 0 & \text{if} \quad \tau_0 > \tau \,. \end{cases} \tag{8}$$

A second threshold parameter, i.e. $\tau^{\star} \geq \tau_0$, is thus introduced. The function $g = g(\tau)$ allows for continuous connection between the first ($\tau \geq \tau^{\star}$) and third ($\tau < \tau_0$) branches of f. Two options are implemented for g, i.e. the linear interpolation

$$g_{\mathrm{lin}}(\tau) = \frac{\tau - \tau_0}{\tau^{\star} - \tau_0} \tag{9}$$

and cubic (spline) interpolation, which allows for smooth (differentiable) continuation

$$g_{\mathrm{cub}}(\tau) = \left(3 - 2\frac{\tau - \tau_0}{\tau^{\star} - \tau_0}\right)\left(\frac{\tau - \tau_0}{\tau^{\star} - \tau_0}\right)^2 . \tag{10}$$

Note that the original behavior (no interpolation) is recovered with $\tau^{\star} = \tau_0$.

As in the case of the LS algorithm, the method provides a robust framework for error assessment, based on the

local quantities (e.g. array configuration and magnetic field measurement uncertainties), as indicated in the sixth step above. This is valid also for the region with $\tau < \tau_0$, where the SVD algorithm predicts the same error level as in the case of the single-satellite method (Eq. (2)) with a sampling distance L given by the along-track separation.

Besides the error assessment, two additional parameters are useful to monitor the SVD solution. One is the condition number, used also in the LS algorithm to characterize the solution stability, i.e. $CN = \lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are the eigenvalues of the position tensor $\mathbf{R}_{pos}$. In the SVD case one recovers the same parameter by using $CN = 1/\tau^2$. A second parameter indicates the configuration/array degeneracy AD, which in the standard RASADA dual-satellite algorithm has two values, i.e. AD = 2 for $\tau \geq \tau_0$ and AD = 1 otherwise. In SwarmFACE implementation of SVD algorithm one uses AD = 1 + $f(\tau)$, with $f(\tau)$ provided in Eq. (8).

2.2.3 Dual-satellite FAC estimation by Boundary Integral

The Level-2 FAC product is based on the BI approach to FAC estimation introduced by Ritter and Lühr (2006) and further elaborated in Ritter et al. (2013). Alternatively, Shen et al. (2012) introduced the Finite Differencing (FD) method to estimate the full magnetic field gradient with a Swarm-like mission. The equivalence of FD and BI *curl* estimation techniques using planar four-point configurations has been demonstrated in Vogt et al. (2013) where the estimators are also brought to a canonical form (see Section 3.3 in that paper), suitable to be used in an error estimation scheme.

SwarmFACE package provides one function based on BI approach that differs in two important aspects from the Level-2 algorithm: (i) equations based on rectangular (instead of spherical) geometry are used, which adds considerable stability to the solutions (i.e. a significant reduction of the EZ) and (ii) the error analysis is based on the canonical base vectors formalism, that has been endorsed by Monte-Carlo simulations (Blagau and Vogt, 2019).

We reproduce from Vogt et al. (2013) the important equations used in the algorithm. The contour integral over the sides of the quad shown in **Figure 2A** can be evaluated with the trapezoidal rule yielding (after some arrangements):

$$\oint \mathbf{b} \cdot d\mathbf{s} \approx \frac{1}{2}\left[\left(\mathbf{b}_b^+ - \mathbf{b}_a^-\right)\cdot\left(\mathbf{r}_a^+ - \mathbf{r}_b^-\right) - \left(\mathbf{b}_a^+ - \mathbf{b}_b^-\right)\cdot\left(\mathbf{r}_b^+ - \mathbf{r}_a^-\right)\right] \quad (11)$$

while the area of the quad is

$$A = \frac{1}{2}\left(\mathbf{r}_a^- - \mathbf{r}_a^+\right)\cdot\left(\mathbf{r}_b^+ - \mathbf{r}_a^+\right) + \frac{1}{2}\left(\mathbf{r}_b^+ - \mathbf{r}_b^-\right)\cdot\left(\mathbf{r}_a^- - \mathbf{r}_b^-\right) \quad (12)$$

which leads to the current density estimation along the normal direction

$$j_n = \frac{1}{\mu_0 A}\oint \mathbf{b} \cdot d\mathbf{s} \quad (13)$$

The mean square error of the radial current j_n through the spacecraft plane is provided by

$$\langle |\delta j_n|^2 \rangle = \frac{(\delta b)^2}{\mu_0^2}\frac{1}{4L^2\mu^2}\left(1 + \mu^2 + \lambda^2\right) \quad (14)$$

The significance of parameters L, μ, and λ is closely related to the configuration shown in **Figure 2A**: (i) the quantity $2L$ is the spatial separation between the satellite orbits, (ii) if one designates by 2ℓ the satellites' separation in orbital phase, then $\lambda = \ell/L$ measures the configuration skewness (deviation from an equal-sided trapezoid), (iii) if $2M$ describes the along track separation, then $\mu = M/L$ measures the quad elongation, i.e. the along-track separation relative to the cross-track distance. The parameter m in **Figure 2A** [not entering in Eq. (14)] indicates the deviation from parallelism of the satellite tracks.

Eq. (6), providing the mean square error of the radial current in the LS method, can also be expressed as a function of configuration parameters using only a slightly more complicated form than Eq. (14) (Vogt et al., 2013). However, there we used an expression that involves the canonical base vectors $\mathbf{q}_\sigma$, obtained directly from the inversion problem in Eq. (5). Contrary to that, in the case of the BI approach, one has to compute the configuration parameters in every point where the FAC density is estimated.

2.3 Three-satellite FAC estimation

Since the upper and the lower Swarm satellites have different orbital velocities, they are periodically aligned in latitude in the auroral zone. When, in addition, a close constellation is formed, the FAC density can be estimated by applying the three-s/c method developed in Vogt et al. (2009). The latter condition is important since the orbital plane of Swarm B has a relative drift (of ~22.5°/year) with respect of the orbital planes of the lower satellites that could lead to a configuration too stretched in longitude when compared with the longitudinal characteristic scale of the FAC sheets. Suitable events are therefore expected at the beginning of the mission (orbital planes roughly aligned), during the more recent Swarm phase when Swarm B and Swarm A/C are orbiting the Earth in opposite directions (orbital planes of ~180° at the beginning of October 2021) or, occasionally, at very high latitude. SwarmFACE provides a tool to find Swarm conjunctions above the auroral oval, see **Sections 3.4**.

When applicable, the three-satellite method provides the unique opportunity of estimating the FAC density at a different scale, larger than in the dual-satellite case. In addition, since the method does not assume time-stationary current structures (unless the user prefers to time-shift the data, e.g. to obtain a more favorable geometry), one can in principle use data with higher temporal resolution as input. For a more detailed discussion of the three-satellite method in the context of Swarm, the reader is referred to Blagau and Vogt (2019).

Below, the recipe for inferring the current density along the direction normal to the spacecraft plane is provided [from Vogt et al. (2009)]. If $\boldsymbol{r}_\alpha$, $\alpha = 1,2,3$ are the satellite's position vectors in the mesocentric frame (the frame where $\sum_{\alpha=1}^{3}\boldsymbol{r}_\alpha = 0$), and $\boldsymbol{r}_{\alpha\beta} = \boldsymbol{r}_\beta - \boldsymbol{r}_\alpha$ are the relative position vectors, the planar reciprocal vectors $\boldsymbol{q}_\alpha$ are formed as

$$\boldsymbol{q}_\alpha = \frac{\boldsymbol{n}\times\boldsymbol{r}_{\beta\gamma}}{|\boldsymbol{n}|^2},\ \alpha = 1,2,3 \tag{15}$$

where (α,β,γ) are cyclic permutations of $(1,2,3)$, and $\boldsymbol{n}$ is the vector normal to the spacecraft plane defined as $\boldsymbol{n} = \boldsymbol{r}_{12}\times\boldsymbol{r}_{13}$. The *curl* of magnetic field perturbations along the normal direction is then provided by

$$(\nabla\times\boldsymbol{b})_n \approx \left(\sum_{\alpha=1}^{3}\boldsymbol{q}_\alpha\times\boldsymbol{b}_\alpha\right)\cdot\boldsymbol{n} \tag{16}$$

with $\boldsymbol{b}_\alpha$ being the magnetic field perturbation recorded by satellite α. With similar assumptions as in the case of dual-satellite LS method, Eq. (6) can be used to estimate the mean square error of the current density along the normal direction. Alternatively, the quality of the planar gradient estimate can be assessed from the position tensor $\mathbf{R}_{pos} = \sum_{\alpha=1}^{3}\boldsymbol{r}_\alpha\boldsymbol{r}_\alpha^t$ (represented by a singular matrix in this case) through the so-called condition number, i.e. the ratio of its (non-zero) eigenvalues.

2.4 Minimum Variance Analysis and correlation analysis on Swarm data

For judging the quality of current density estimations it is important to evaluate to what degree the sampled current structure satisfies the assumptions on which the FAC method(s) rely. For example, the standard single-satellite FAC algorithm assumes one-dimensional/planar magnetic field structure and normal satellite incidence. A certain degree of time stationarity is required in the dual-s/c algorithm as well as in the three-s/c method when virtual (i.e. not instantaneous) satellite positions are employed. SwarmFACE provides functions that automatically estimate the so-called quality indices, like current sheet planarity, inclination and correlation of magnetic perturbations recorded by the lower satellite.

One way to assess the degree of planarity and the inclination of a current structure is by applying the MVA analysis, introduced in (Sonnerup and Cahill, 1967). A short description of the algorithm used in the SwarmFACE package is provided below; for an extensive derivation the reader is referred to Sonnerup and Scheible (1998) and Sonnerup et al. (2006).

Given the magnetic field perturbation recorded during the current sheet crossing, $\mathbf{b}^{(k)}$, $k = 1,2\ldots K$, the standard version of MVA searches for the direction $\mathbf{n}$ that minimizes the quantity (magnetic field variance)

$$\sigma^2 = \frac{1}{K}\sum_{k=1}^{K}\left|\left(\mathbf{b}^{(k)} - \langle\mathbf{b}\rangle\right)\cdot\mathbf{n}\right|^2 \tag{17}$$

subject to the condition that $|\mathbf{n}|^2 = 1$, and associates it with the current sheet normal. Here by $\langle\cdots\rangle$ the mean over the set of K points of measurements is indicated. The minimization leads to an eigenvalue problem for the (symmetric) 3 × 3 magnetic variance matrix $\mathbf{M}^B$, defined as

$$\mathbf{M}^B = \left\langle(\mathbf{b} - \langle\mathbf{b}\rangle)(\mathbf{b} - \langle\mathbf{b}\rangle)^t\right\rangle = \left\langle\mathbf{b}\mathbf{b}^t\right\rangle - \langle\mathbf{b}\rangle\langle\mathbf{b}\rangle^t \tag{18}$$

with the eigenvector corresponding to the smallest eigenvalue of $\mathbf{M}^B$ providing the $\mathbf{n}$ direction. In case of a FAC structure the magnetic perturbations roughly occur only perpendicular to the ambient magnetic field $\mathbf{B}$. That can be taken into account if one replaces the matrix $\mathbf{M}^B$ by the matrix product $\mathbf{P}\mathbf{M}^B\mathbf{P}$, where p represents the so-called projection matrix, given by

$$\mathbf{P} = \mathbf{I} - \mathbf{m}\mathbf{m}^t \tag{19}$$

with $\mathbf{m} = \langle\mathbf{B}\rangle/|\langle\mathbf{B}\rangle|$. This way of performing the analysis, implemented in SwarmFACE, is know as the constrained (to average ambient magnetic field direction) MVA.

The vector $\mathbf{n}$ identified by MVA corresponds to $\mathbf{N}_{3d}$ indicated in **Figure 1A**, from which the vector $\mathbf{N}_{2d}$ and the inclination angle α in the tangential plane can be obtained. A measure of the current sheet planarity is provided by the ratio $\lambda_{max}/\lambda_{min}$, with λ_{max} and λ_{min} the two eigenvalues of the (constrained) magnetic variance matrix.

For case studies, SwarmFACE allows the user to interactively select the analysis interval that enters MVA (see **Section 3.5**). To automatically estimate the planarity and inclination for a number of consecutive orbits (see **Section 3.6**), SwarmFACE performs MVA on the auroral oval (AO) intervals estimated with the lower-level function `find_ao_margins`. This function works on quarter-orbit sectors and therefore in the automatic procedure the initial data interval is first split in sections using as delimiters the moments when the satellite is either above the Equator or when the highest/lowest quasi-dipole magnetic latitude (QDLat) is reached. Note that for each such sector the evolution of QDLat is monotonic.

The following steps are pursued by `find_ao_margins` to find the AO location on a quarter-orbit sector:

1. The single-satellite FAC density estimate is considered and the cumulative sum (integral) of unsigned (absolute value) FAC density is computed. The integration is performed as a function of QDLat (not time), to correct for the non-linear changes in QDLat at the highest/lowest portion of the orbit. Since in the process of current integration, small FAC densities could badly affect the good identification of auroral oval, only current densities above a certain value are considered;
2. The points d_1, d_2, and d_3 when the integral reaches a fraction of 1/4, 1/2, and 3/4 from its total height are identified;
3. The timestamps corresponding to $d_1 - (d_3 - d_1)$, d_2, and $d_3 + (d_3 - d_1)$ are associated with the beginning, the center,

and the end of the AO interval. If the beginning/end timestamp falls outside the quarter-orbit section, its beginning/end time is used.

A hint on the current-sheet time-stationarity or its compliance to the one dimensional model (the two aspects are difficult to dis-entangle) could be obtained with the correlation analysis. SwarmFACE provides a function that estimates the correlation of magnetic field perturbations recorded by the lower Swarm satellites. To that end.

1. The MVA analysis is performed on Swarm A and Swarm C data and the magnetic perturbations along the maximum variance directions are further used;
2. Data from the satellite with the shorter MVA interval, trimmed to this interval, form the reference signal;
3. A running Pearson's correlation coefficient is computed by comparing the time-shifted versions of the reference signal (typically in the ± 30 s range) with data intervals of the same length from the other satellite;
4. The highest correlation coefficient is returned, together with the identified optimum time-lag.

3 Code usage and results

3.1 Single satellite FAC estimation

To estimate the FAC density with one satellite, SwarmFACE provides the `j1sat` high-level routine:

Listing 1. The full calling sequence for `j1sat` high-level routine

```
j_df, input_df, param = j1sat(dtime_beg, dtime_end, sat,
        res='LR', use_filter=True, alpha=None, N3d=None,
        N2d=None, tincl=None, er_db=0.5, angTHR=30.,
        savedata=True, saveplot=True)
```

The mandatory parameters specify the analysis interval start and end time (parameters `dtime_beg` and `dtime_end`), and the Swarm satellite `sat`. With only these parameters provided, the program computes the low-resolution FAC density assuming normal current sheet orientation.

The actual current orientation should be inferred separately, e.g. by applying MVA on the auroral oval interval (see **Section 2.4** and **Section 3.5**), and can be passed to `j1sat` by one of the `N3d`, `N2d`, or `alpha` parameters. Here `N3d` and `N2d` designate (in geographic frame) the full current sheet normal vector and, respectively, its projection on the tangential plane, while `alpha` refer to the angle α between the $\mathbf{N}_{2d}$ and the satellite velocity, see **Figure 1B**. Internally, when α is not provided by outside, the algorithm uses $\mathbf{N}_{3d}$ or $\mathbf{N}_{2d}$ to compute its values along the orbit and further work with these in the calculations. Note that, since the standard MVA provides the constant vector $\mathbf{N}_{3d}$ (assumed to characterize globally the auroral crossing), α is (moderately) varying in time and therefore the user can provide `alpha` either as an average value or as an array of successive values. The parameter `tincl` allows to specify the time interval when the information on current sheet inclination is valid, typically the interval used in MVA. When not specified, the whole analysis interval, i.e. [`dtime_beg`, `dtime_end`] is considered valid, which could lead to undesired results since, e.g. the angle between $\mathbf{N}_{3d}$ and satellite velocity $\mathbf{v}$ could then vary substantially. When `tincl` is provided, the algorithm will use the first/last value of α to calculate the inclination before/after its margins.

With the parameter `use_filter`, the users specify whether or not they want to pre-filter the input magnetic field perturbations that enters the FAC estimation, its default value being `True` (i.e. to use the filter). The parameters `er_db` and `angTHR` specify the level of uncertainties in magnetic field measurements (considered in the current density error calculation) and, respectively, the minimum accepted angle of the magnetic field vector from the tangential plane, needed to estimate FAC from IRC density with Eq. (3). The last two parameters, i. e `savedata` and `saveplot`, are for saving the results as an ASCII file and, respectively, in a plotted format.

The first two objects from the `j1sat` output are pandas DataFrame structures. `j_df` contain all relevant results, i.e. satellite position at the times of FAC estimations, the un-filtered and (if requested so) filtered FAC and IRC densities together with the corresponding errors, the angle between **B** and **C** and the α angle. Similarly, `input_df` essentially contains the input data, as retrieved from ESA database, like satellite position in GEO frame, the magnetic field measurements and the model data in NEC frame. `param` is a dictionary object with all the parameters used in the analysis. It also contains a field for missing or bad Swarm measurements.

Figure 1C presents the standard plot generated by `j1sat` when `saveplot = True`. After the MVA has been applied for this event, the FAC normal `N3d = [0.227, 0.692, 0.684]`, valid for the time indicated by dashed vertical lines, has been supplied to `j1sat`. From top to bottom, the figure shows (A) the Swarm magnetic field perturbation in GEO frame, (B) the un-filtered and filtered FAC densities, (C) the un-filtered and filtered IRC densities, (D) a comparison between FAC estimated with `j1sat` and the Level-2 product, (E) the errors in un-filtered and filtered FAC densities, (F) the angle between **B** and **C** vectors, and (G) the α angle. The labels at the bottom indicate time in UTC, the satellite geographic and quasi-dipole latitude and longitude, as well as the magnetic local time.

3.2 Dual-satellite FAC estimation

The SwarmFACE dual-satellite routines implicitly run on the same configurations as the official Level-2 algorithm, i.e. the sensors are (as much as possible) aligned in orbital phase and a 5 s travel distance for the along track separation is chosen to construct the quads. However, the user can take advantage

of the flexible geometry shown in **Figure 2A** and work e.g. with smaller along-track separation distance to investigate FAC structures at higher spatial resolution or the orbital time lag between the satellites such that the quad configuration is tuned to better characterize current sheets inclined with respect to spacecraft orbit (see Section 3.4 in Blagau and Vogt, 2019). To reduce the influence of local magnetic fluctuations, the magnetic perturbation is pre-filtered with a 20 s cut-off low-pass Butterworth filter (see also the discussion at the beginning of **Section 2**).

3.2.1 Dual-satellite FAC estimation by Least Squares

The dual-satellite LS algorithm is implemented with the `j2satLS` high-level routine:

Listing 2. The full calling sequence for `j2satLS` high-level routine

```
j_df, input_df, param = j2satLS(dtime_beg, dtime_end, sats,
      tshift=None, dt_along = 5, use_filter=True, er_db=0.5,
      angTHR = 30., errTHR=0.1, saveconf=False,
      savedata=True, saveplot=True)
```

Here `sats` is a 2 element array of strings that designates the satellites entering in the analysis, usually the lower Swarm pair, i.e. `sats` = ['A', 'C']. The parameters `dt_along` and `tshift` specify the quad's length in the along-track direction (in seconds of satellite travel distance) and, respectively, the time shift (in seconds) to be introduced in satellite data in order to achieve the desired quad configuration. When tshift (a 2 element array of integer numbers) is not specified, the program uses the lower level `find_tshift2sat` routine to find the optimal time-shift that ensures formation of rectangular quads. Since `find_tshift2sat` firstly estimates the orientation of orbital planes from the downloaded satellite positions, it is advisable to avoid using small intervals of analysis. For the nominal separation of 1.4° between the orbital planes of Swarm lower satellites, an interval of analysis of at least 2 min provides good estimates; however, this could be insufficient during the close orbit configuration campaign centered around October 2021. Note that in the standard plots generated by `j2satLS`, the quad configurations are plotted at the bottom, so that the user can judge whether or not the parameter `tshift` has been correctly computed.

For a regular analysis the parameter `use_filter` should be left unchanged to its implicit value, i.e. `True`, meaning to pre-filter the magnetic field perturbation. For a discussion on working with un-filtered data in dual-satellite FAC method, the reader is referred to Blagau and Vogt (2019), Section 3.4.2. The parameter `errTHR` specifies the accepted error estimation level for the IRC density, the prime quantity evaluated by the algorithm, see Eq. **(6)**; whenever j_n is below this threshold, IRC and FAC densities are set to `NaN`.

When knowing the parameters that describe the planar four-point configuration is desired, the users could set `saveconf = True`, which will add new columns in the `j_df` DataFrame. These parameters, shown in **Figure 2A** and introduced in **Section 2.2.3**, need not to be computed in the LS method [though, they are used to estimate the errors in the BI method, see Eq. **(14)**]. Nevertheless, the evolution of quad parameters along the orbit could be instructive, as shown in **Figure 2B**, which refers to the event presented in **Figure 2C**. The first two panels, (G) and (H), indicate how the (half) cross-track separation L acquires a minimum value at the orbital cross-point, while the (half) along-track separation M is constant. The other two parameters, i.e. ℓ and m, that describe the (un-compensated) orbital lag and the non-parallelism of the satellite tracks, are shown in panels (I) and, respectively, (J). Their values drop around the orbital cross-point because the algorithm takes care of assigning to each vertex the correct satellite positions in order to maintain a convex configuration (as needed in the BI method).

The output from `j2satLS` is structured in a way similar to the `j1sat` output: `j_df` and `input_df` are pandas DataFrames that hold the result and the input data, respectively, while `param` is a dictionary with all the parameters used in the analysis, including a field for missing or bad Swarm measurements.

Figure 2C presents the standard plot generated by `j2satLS` when `saveplot = True`. From top to bottom the figure shows (A, B) the Swarm magnetic field perturbation recorded by the two satellites in GEO frame, (C) the logarithm of the condition number, as defined in **Section 2.2.1**, (D) the angle between the local magnetic vector and the direction normal to the quad, (E) comparison between the dual-satellite LS FAC estimation (blue) and the Level-2 product (orange), and (F) the error level in FAC density estimation. At the bottom, the quad configuration at three instances (i.e. start time, stop time, and at the middle of the interval) is presented as projection on the North-East plane of the local NEC frame. Note in panel (E) the different EZ extensions for the Level-2 solution [±4° co-latitude, according to Ritter et al. (2013)] and in the LS solution, set by the value specified in `errTHR` (here we chose a level of 0.25 μA/m^2 for errTHR, to make evident the differences with respect to the BI algorithm, see **Section 3.2.3**).

3.2.2 Dual-satellite FAC estimation by Singular Value Decomposition

The dual-satellite SVD algorithm is implemented with the `j2satSVD` high-level routine:

Listing 3. The full calling sequence for `j2satSVD` high-level routine

```
j_df, input_df, param =j2satSVD(dtime_beg, dtime_end, sats,
                  tshift=None, dt_along=5, use_filter=True
                  er_db=0.5, tauast=0.13, taunul=0.07,
                  intpol='Linear', angTHR=30., saveconf=False,
                  savedata=True, saveplot=True)
```

Here the characteristic parameters are `tauast`, `taunul`, and `intpol`; the first two specify the threshold

values τ^* and τ_0, whereas the last one indicates the interpolation method to be used for matching the different gradient estimators, i.e. the branches for $\tau \geq \tau_*$ and $\tau < \tau_0$, respectively (see Section 2.2.2). Values for `intpol` should be 'Linear' (for linear interpolation; the default value), 'Cubic' (for cubic spline interpolation), or None (when no interpolation is desired; in this case the `tauast` parameter is ignored and the standard RASADA algorithm is applied with the threshold value specified by `taunul`).

Figure 3A presents the standard plot generated by `j2satSVD` when `saveplot = True`. Comparing with the standard plot produced by the LS routine, an additional panel (the new panel E) has been introduced that presents the evolution of two parameters, i.e. τ (matrix **S** eigenvalues ratio) and the array degeneracy AD, as defined in Section 2.2.2. Far enough from the orbital cross-point AD = 2 and the SVD and LS solution are identical. When τ is below the threshold set by `taunul`, AD = 1 and the SVD algorithm provides a solution close to the average (over the satellites) single-satellite estimates. In that region also the magnetic field angle (panel D) is calculated with respect to the radial direction (not with respect to the quad normal).

The error in FAC estimation (panel G) are increasing as τ decreases toward the τ^* level. For even smaller values of τ, the error starts to decrease up to the level predicted by the single-satellite method for a sampling distance given by the along-track separation (region where $\tau < \tau_0$). As one can see, the implicit values used in the SVD algorithm (0.13 for `tauast` and 0.07 for `taunul`) provide a more conservative current density estimate than in the LS case, since the error level remains below 0.05 μA/m^2, i.e. the implicitly accepted threshold used to control the LS solution (see the parameter `errTHR` from Section 3.2.1).

3.2.3 Dual-satellite FAC estimation by Boundary Integral

The dual-satellite BI algorithm, implemented with the `j2satBI` high-level routine, requires the same parameters as in the LS case:

Listing 4. The full calling sequence for `j2satBI` high-level routine

```
j_df, input_df, param = j2satBI(dtime_beg, dtime_end, sats,
                tshift=None, dt_along=5, use_filter=True,
                er_db=0.5, angTHR =30., errTHR=0.1,
                saveconf=False, savedata=True, saveplot=True)
```

Figure 3B presents the standard plot generated by `j2satBI` for the same event analyzed with the LS algorithm (see Figure 2C). Only five panels are needed since no condition number is computed. We used the same threshold value for `errTHR` to put in evidence higher values/lower stability for the BI FAC estimate close to the orbital cross-point (compare panel E in Figure 2C with panel D in Figure 3B) as well as broader evolution of FAC errors in that region (last panels in the same plots). When compared with the Level-2 product (panel D in Figure 3B), the `j2satBI` solution shows greater stability, although it relies on the same, i.e. BI, method the only difference being that `j2satBI` uses equations based on rectangular (instead of spherical) geometry.

3.3 Three-satellite FAC estimation

The three-satellite algorithm, intended to be applied when the Swarm satellites form a close configuration, is implemented with the `j3sat` high-level routine:

Listing 5. The full calling sequence for `j3sat` high-level routine

```
j_df, input_df, param = j3sat(dtime_beg, dtime_end,
            tshift=[0,0,0], use_filter=True, er_db=0.5,
            angTHR = 30., savedata=True, saveplot=True)
```

A parameter that specifies the satellite (like `sats` in the single- and dual-satellite routines) is not needed in this case. However, if the user decides to perform the analysis with shifted satellite position (e.g. to achieve a more favorable spacecraft constellation), the `tshift` parameter (now a three element array of integer numbers) should specify the time shifts in seconds keeping the internal order of satellites, i.e. Swarm A, Swarm B, and Swarm C.

Figure 4A presents the standard plot generated by `j3sat` when `saveplot = True`. From top to bottom, the figure shows (A–C) the Swarm magnetic field perturbation recorded by the Swarm satellites in GEO frame, (D) the logarithm of the condition number, as defined in Section 2.3, (E) the angle between the local magnetic field vector **B** and the direction normal to the spacecraft plane, (F) comparison between the un-filtered and (if `use_filter = True`) filtered three-satellite FAC estimations (blue and orange, respectively) and the Level-2 dual-satellite product (green), and (G) the error level in FAC density estimation(s). At the bottom, the spacecraft configuration at three instances (i.e. start time, stop time, and at the middle of the interval) is presented as projection on the North-East plane of the local NEC frame.

To interpret the differences between dual- and three-satellite solutions (panel F), one should keep in mind that (i) these estimations refer to different points, i.e. the corresponding mesocenters have different latitude and longitude and (ii) the scales involved are different. A detailed discussion about the differences between dual- and three-satellite FAC estimations on the event presented in Figure 4A has been provided in Section 4.2 of Blagau and Vogt (2019). In the same paper, other critical aspects related to the application of three-sat method in the Swarm context (longitudinal separation, linear field variation, orientation of the spacecraft plane etc) are discussed.

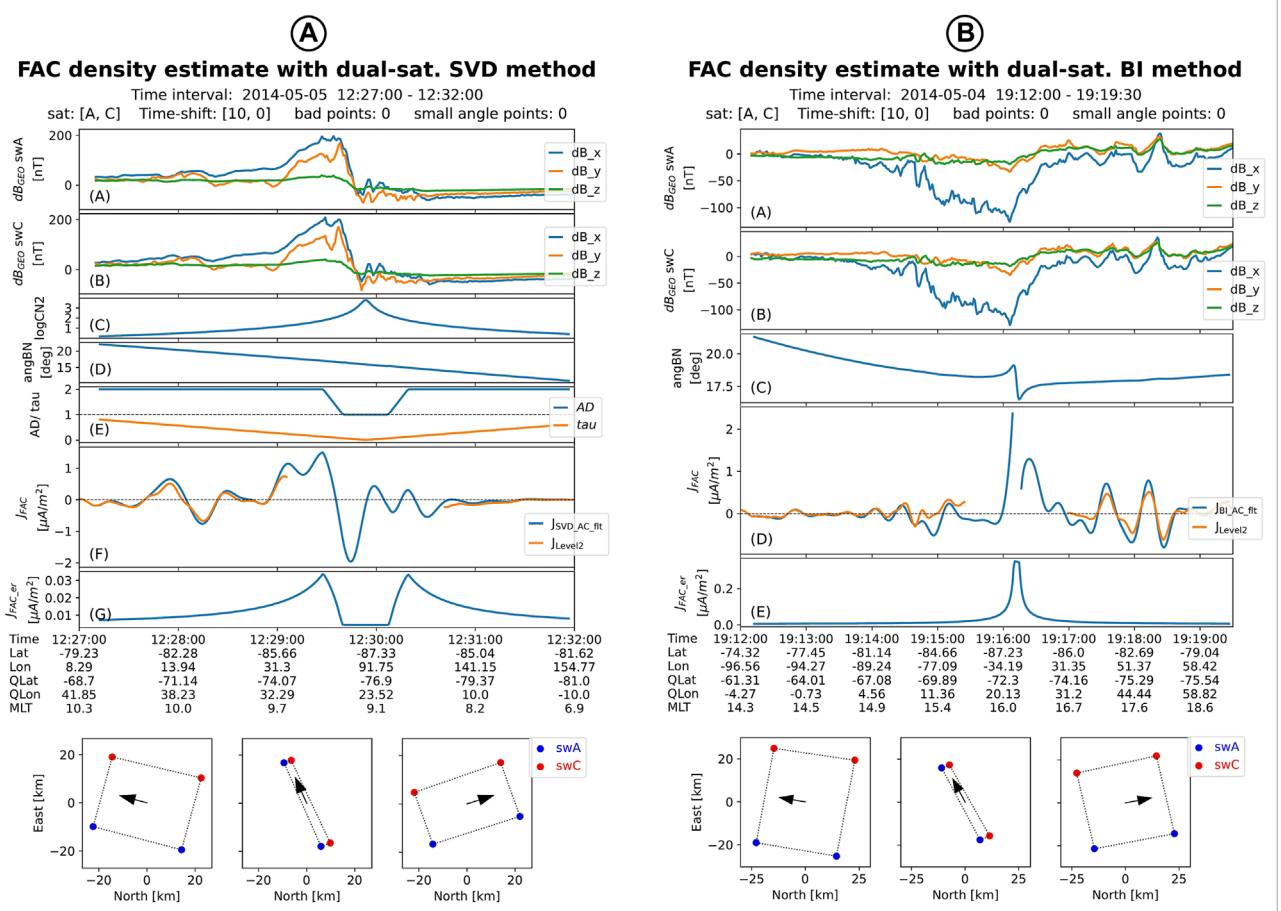


FIGURE 3
(A): Standard plot generated by the `j2satSVD` algorithm. The figure is similar to **Figure 2C**, with an additional panel (the new panel E) that presents the evolution of eigenvalue ratio τ (orange line) and the array degeneracy AD (blue line). **(B)**: Standard plot generated by the `j2satBI` algorithm. The same quantities as in **Figure 2C** are plotted, with the exception of the condition number, which is not computed by the algorithm.

3.4 Identification of Swarm conjunctions above the auroral oval

SwarmFACE package provides a tool to find Swarm conjunctions above the auroral oval. These events would then qualify to investigate the FAC structure with the three-satellite method or with the dual-satellite method when Swarm B data are combined with data from one lower Swarm satellite. The algorithm is implemented with the `find_3sat_conj` high-level routine:

Listing 6. The full calling sequence for `find_3sat_conj` high-level routine

```
conj_df, param = find_3sat_conj(dtime_beg, dtime_end,
   delT=120, delN=300, delE=1200, jTHR=0.05, saveplot=True)
```

The routine relies on the lower-level function `find_ao_margins` (see its description in **Section 2.4**) that automatically identifies the auroral oval location for each satellite. The following processing steps are performed:

1. For each satellite, the single-sat Level-2 FAC data corresponding to the full consecutive orbits that completely cover the time-interval provided by the user (i.e. a larger interval than `[dtime_beg, dtime_end]`) is downloaded with *viresclient*. In order to work with smaller arrays, only orbital sections where the quasi-dipole latitude is >45° or < −45° are retrieved. The FAC data are also filtered using a low-pass Butterworth filter
2. Data are split in quarter-orbits, as described in **Section 2.4**, and for each section `find_ao_margins` is called to estimate the central position of the auroral oval
3. Conjunctions are found by imposing temporal and spatial conditions, i.e. that Swarm B and Swarm A/C auroral oval central points to be encountered within a certain time-

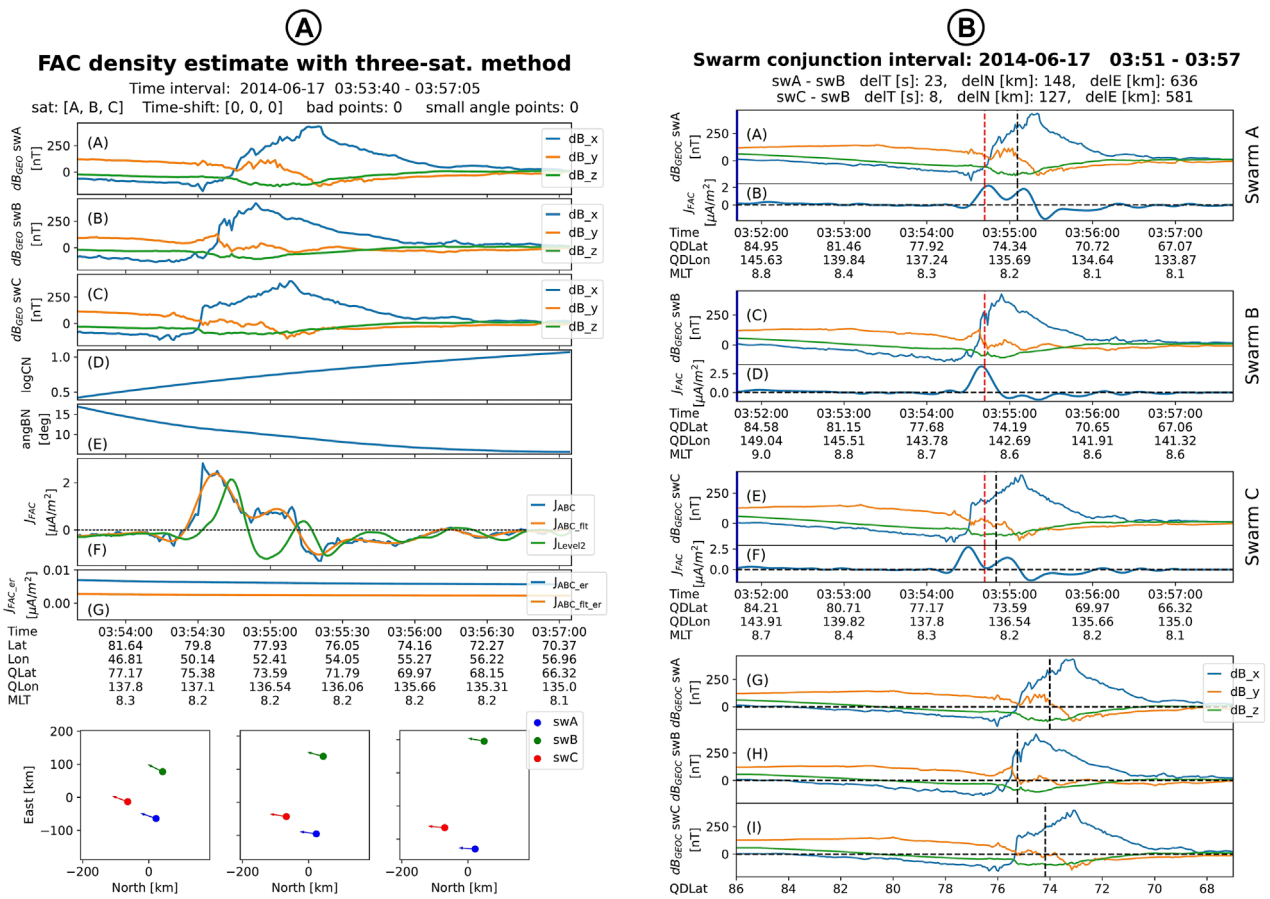


FIGURE 4
(A): Standard plot generated by the `j3sat` algorithm. From top to bottom the figure shows (A–C) the magnetic field perturbation recorded by the Swarm satellites in GEO frame, (D) the logarithm of the condition number, (E) the angle between *B* and the direction normal to the spacecraft plane, (F) comparison between the un-filtered and filtered three-satellite FAC estimations (blue and orange, respectively) and the Level-2 dual-satellite product (green), and (G) the error level in FAC density estimation(s). At the bottom, the spacecraft configuration at start time, stop time, and at the middle of the interval is presented as projection on the North-East plane of the local NEC frame. **(B)**: Standard plot generated by the `find_3sat_conj` algorithm. The magnetic field perturbation in GEO panels (A, C, and E) and the low-pass filtered single-satellite Level-2 FAC density data (panels B, D, and F) are plotted for each satellite. The panels (G, H, and I) present the magnetic field perturbation as a function of QDLat. The auroral central times/location are shown as vertical black dotted lines. In panels (A–F) the auroral central time for Swarm B is indicated as reference (the red doted lines).

window (specified, in seconds, with the parameter `delT`), and within a certain spatial range along the North-South and East-West direction (specified, in km, with the parameter `delN` and, respectively, `delE`). The parameter `jTHR` (value in μA/m^2) is used by `find_ao_margins` as a threshold to neglect small FAC densities that could badly affect the good identification of auroral oval (see **Section 2.4**)

The output from `find_3sat_conj` are `the conj_df` DataFrame object, with details on the identified conjunctions (e.g. conjunction time and location, time difference between the AO central times, spatial difference between the AO central locations), and `param`, a dictionary object of parameters used in the computation. The content of `conj_df` is always saved in an ASCII file. Since the automatic identification of AO intervals might not work accurately in all cases, it is advisable for the user to let the program generate standard plots, designed to help in (visually) assessing the quality of each conjunction (i.e. keep `saveplot = True`). Nevertheless, the generation of plots could be time consuming, mainly because the magnetic field data have to retrieved for each conjunction (5 min Before/after the central AO detected by Swarm B).

In **Figure 4B** one standard plot generated by `find_3sat_conj` is presented. The magnetic field perturbation in GEO (panels A, C, and E) and the low-pass filtered single-satellite Level-2 FAC density data (panels B, D,

and F) are plotted for each satellite. The last three panels (i.e. G, H, and I) present the magnetic field perturbation as a function of QDLat. The auroral central times/locations are shown as vertical black dotted lines. In panels A–F the auroral central time for Swarm B is indicated as reference (the red doted lines). The plot subtitle provides details about the conjunction, e.g. the time and spatial difference between the central AO encountered by the satellites.

3.5 Interactive Minimum Variance Analysis on Swarm events

Besides the MVA on automatically identified AO intervals, SwarmFACE provides the user with the possibility to interactively select the interval for MVA on individual events. The necessary lines of code, to be run from a 2-cell *jupyter notebook* are presented below; the code could be adapted to other environments provided that an interactive backend for *matplotlib* is available. The first cell specifies the input parameters and prepares the necessary data for analysis.

Listing 7. First part of interactive Minimum Variance Analysis

```
%matplotlib notebook
from SwarmFACE.mva1sat import *
dtime_beg = '2015-03-17T08:51:54'
dtime_end = '2015-03-17T08:57:11'
sat = ['C']
use_filter = True

j_df, input_df, param, span_sel, span = get_data_mva1sat
                                        (dtime_beg,
                                        dtime_end, sat,
                                        use_filter=True)
```

Note the `%matplotlib notebook` magic command at the beginning, that ensures the production of interactive plot(s) embedded within the notebook. The `get_data_mva1sat` routine retrieves (through *viresclient*) the 1 Hz resolution magnetic field data, computes the standard (i.e. assuming normal FAC inclination) single-satellite FAC density estimation and produces a plot of these variables on the output cell. The parameters `dtime_beg` and `dtime_end` can be only roughly specified; the algorithm actually downloads Swarm measurements on a larger interval, i.e. half-orbit time span, and offers the possibility to pan/zoom on the plotted data. If the users want to change the initial time-interval, i.e. [`dtime_beg`, `dtime_end`], they can do so by mouse clicking and adjusting, while vertical lines will indicate the new interval of analysis. This is achieved by calling (in the background) the matplotlib `SpanSelector` widget. The output variable `span_sel` holds the new interval of analysis, while `span` is a reference to `SpanSelector` that prevents it to be garbage collected, making it thus available outside `get_data_mva1sat`.

The second cell performs the MVA on the (up-dated, if necessary) analysis interval:

Listing 8. The second part of interactive Minimum Variance Analysis

```
span_sel              # to update the MVA interval
jcorr_df, dBmva_df, mva_df, param = perform_mva1sat
                                    (j_df, input_df, param,
                                    span_sel, savedata=True,
                                    saveplot=True)
```

The `perform_mva1sat` routine takes as input the variables produced by `get_data_mva1sat`, i.e. `j_df` and `input_df` which are DataFrames that hold FAC density data and, respectively, the data retrieved from the ESA database. The output variables `jcorr_df`, `dBmva_df`, and `mva_df` are DataFrames with the corrected (for current sheet inclination) FAC density, magnetic field perturbation in MVA frame (the frame aligned along the MVA eigenvectors' directions), and the result from the MVA analysis, respectively.

Figure 5A presents the plot generated by `perform_mva1sat` when `saveplot = True`. From top to bottom one has (A) the magnetic field perturbation in GEO, (B) the magnetic field perturbation in the MVA frame, (C) a comparison between the standard and corrected for inclination FAC density estimates, and (D) the angle in the tangential plane between the FAC sheet normal and the satellite velocity vector (angle α in **Figure 1B**). The hodograph of magnetic field perturbation in the plane perpendicular to the average magnetic field direction is presented in panel E.

3.6 Quality indicators

To automatically estimate the quality indices of FAC structures, SwarmFACE provides the `fac_qi` high-level routine:

Listing 9. The full calling sequence for `fac_qi` high-level routine

```
input_df, RBdBAng_df, fac_df, qimva_df, qicc_df, param =
      fac_qi(dtime_beg, dtime_end, swB=True, saveplot=True)
```

The full set of quality indices, i.e. the FAC sheet planarity, inclination, and correlation of individual magnetic field perturbations is provided for the Swarm lower satellite pair. In addition, the MVA based quality indices (i.e. FAC sheet planarity and inclination) for Swarm B could be estimated as well if the user explicitly requests so by setting `swB = True`. Similar to the routine that finds Swarm conjunctions above the AO, the interval of analysis for each satellite is represented by the full consecutive orbits that completely cover the time-interval provided by the user (i.e. an interval larger than [`dtime_beg`, `dtime_end`]).

The following processing steps are performed for each satellite:

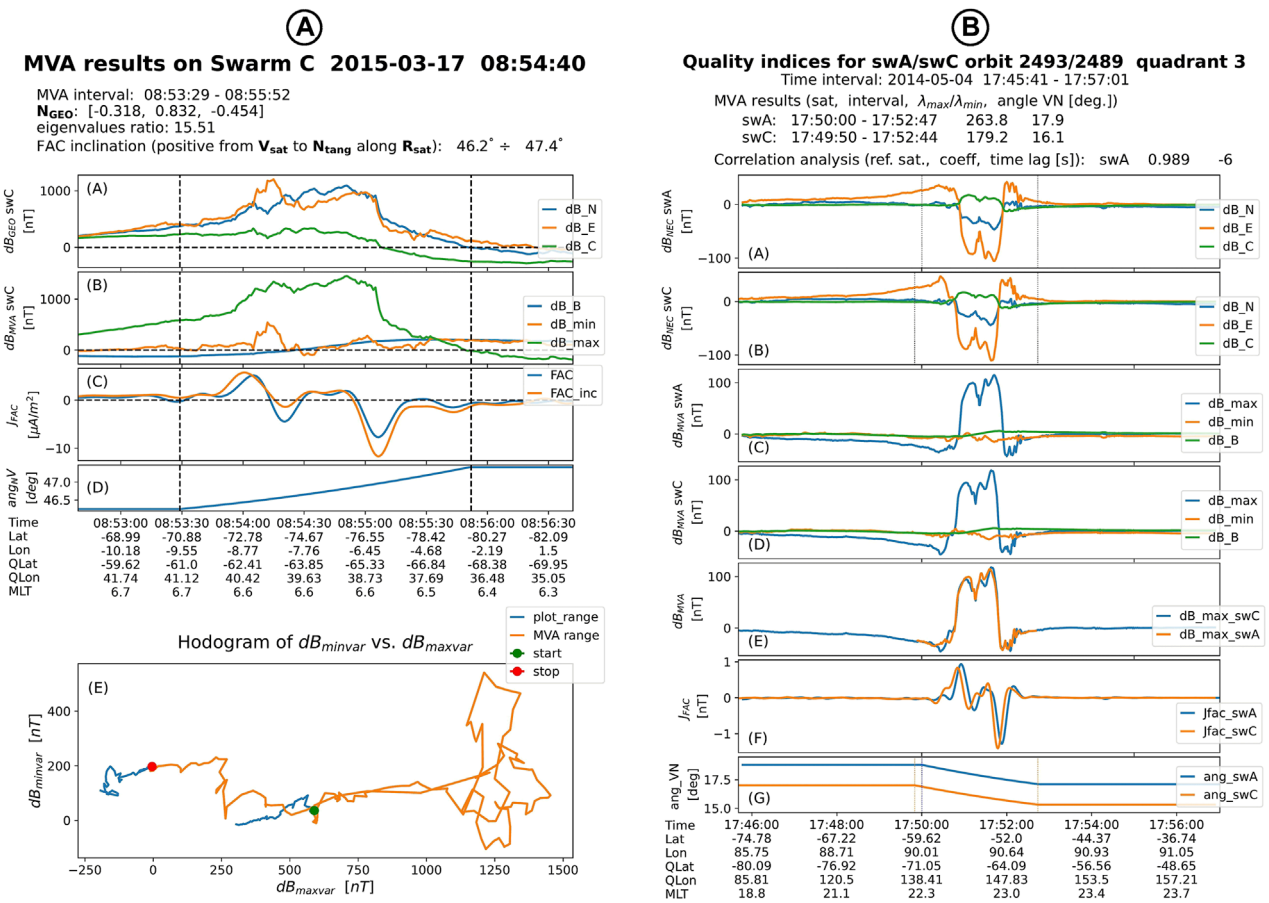


FIGURE 5
(A): Standard plot generated by `perform_mva1sat`. From top to bottom the figure shows (A and B) the magnetic field perturbation in GEO frame and MVA frame, respectively, (C) a comparison between the standard and corrected for inclination FAC density estimates, and (D) the angle in the tangential plane between the FAC sheet normal and the satellite velocity vector (angle α in **Figure 1B**). The hodograph of magnetic field perturbation in the plane perpendicular to the average magnetic field direction is presented in panel (E). **(B)**: Standard plot generated by `fac_qi` algorithm (data only from the lower Swarm satellites). From top to bottom the figure shows the magnetic field perturbation in NEC panels (A,B) and in MVA frames panels (C,D). The vertical dashed lines in panels (A and B) indicate the AO intervals that enter in the MVA analysis. In panel (E) the magnetic perturbations along the maximum variance directions of both sensors are plotted with the proper lag applied to the reference satellite. Panel (F) shows the (filtered) FAC densities, while panel (G) presents the current sheet inclination in the tangential plane.

1. The Level-1b magnetic field data and auxiliary data are downloaded with *viresclient*; only relevant orbital sections, where the quasi-dipole latitude is >45° or < − 45°, are retrieved in this process
2. Data are split in quarter-orbits, as described in **Section 2.4**, and for each sector the single-satellite FAC density is estimated from the magnetic field perturbation
3. The AO margins are estimated by calling the `find_ao_margins` function and the MVA analysis is performed on the identified AO interval
4. For the magnetic field perturbations recorded by the lower satellite pair, the correlation coefficient and the optimum time lag are computed as described in the last paragraph of **Section 2.4**

The output from `fac_qi` consists of pandas DataFrames, lists of DataFrames (one DataFrame for each satellite and quarter-orbit sector) and the dictionary object `param`, with parameters used in the analysis. The output variable `input_df` is a list of DataFrames with the input data retrieved from ESA database. Similarly, `RBdBAng_df` is a list of DataFrames that contains intermediate variables computed by the routine, i.e. the satellite position, magnetic field, and magnetic field perturbation in GEO frame, the magnetic field perturbation in the MVA frame (the frame oriented along the MVA eigenvectors), and the angle between the current sheet normal direction and the satellite velocity in the tangential plane. The single-satellite FAC density time-series are provided in `fac_df` (list of DataFrames). The output variables `qimva_df` and `qicc_df` are DataFrames

with the full MVA results and related quality indices (planarity and inclination) and, respectively, results from the correlation analysis between magnetic field perturbation on Swarm A and Swarm C (i.e. correlation coefficient and the optimum time-lag). The information from these two variables is automatically saved as ASCII files.

Figure 5B presents the standard plots (one per quarter orbit section) generated by `fac_qi` when `saveplot = True`. Only data and results that applies to the lower Swarm satellites are plotted. The first panels show, for both satellites, the magnetic field perturbation in NEC (panels A, B) and in MVA frames (panels C, D). The vertical dashed lines in panels A and B indicate the (automatically estimated) AO intervals that enter in the MVA analysis. To illustrate the correlation between the magnetic perturbations recorded by the two sensors, in panel E the magnetic perturbations along the maximum variance directions of both sensors are plotted with the proper lag applied to the reference satellite. Panel F shows the FAC densities obtained from filtered magnetic field data, while panel G presents, for each satellite, the current sheet inclination with respect to the satellite velocity in the tangential plane. The results of MVA and correlation analysis are indicated in a concise form in the plot subtitle.

4 Discussions and future work

The SwarmFACE package integrates algorithms based on single- and multi-satellite methods to characterize the FAC system with Swarm beyond the official FAC Level-2 products. The individual routines have been extensively tested on synthetic as well as on real data. In the same time, using a poll of test-events, we compared the single- and multi-satellite FAC density estimations with the output of the original IDL routines, recovering essentially the same results, down to rounding errors.

The dual-satellite estimations based on the LS and Cartesian Boundary Integral methods have been compared with the Level-2 estimates on $\sim$ 1500 randomly selected AO crossings that occurred during the nominal (i.e. 1.4°) orbit separation of Swarm A and Swarm C (Blagau and Vogt, 2019). Considering magnetic measurements affected by instrumental noise of 0.5 nT and using a threshold in FAC estimation error of 0.1 μA/m^2 (i.e. the default values for parameters `er_db` and `errTHR` in **Sections 3.2.1** and **Section 3.2.3**) a conservative assessment of the results indicates that roughly 3 times less FAC data have to be discarded due to the smaller (than in the Level-2 case) EZ extension. During the more recent phase of the mission, when the cross-track separation between Swarm A and Swarm C initially decreased (zero value reached at the beginning of October 2021) and then start to increase again, the dual-satellite estimations are expected to become less stable close to the orbital cross-point and new values for the processing parameters have to be used.

Blagau and Vogt (2019) and Vogt et al. (2020) have analyzed the capabilities of the methods used in the SwarmFACE package and how the methods perform in various Swarm contexts (e.g. solution stability, use of different configuration geometries, influence of the local magnetic fluctuations etc) based on synthetic data. Two additional aspects were also investigated: firstly, it has been shown that the dual- and three-satellite FAC density estimations depend only marginally (tens of nA/m^2 at most) on the magnetic field model used to compute the magnetic field perturbation. The single-satellite method is somewhat more sensitive, since an outdated magnetic model causes an artificial trend in the density solution. The second aspect refers to the influence the ionospheric electrojet could have on the FAC density estimations. Using synthetic data it has been concluded that in the dual- and three-satellite methods, where one effectively integrates the magnetic perturbation on a closed path, this influence is small (few nA/m^2 in the dual-satellite methods and tens of nA/m^2 in the three-satellite case). The single-satellite method is again more influenced (up to a fraction of μA/m^2) by the presence of the electrojet.

One could think of several possible directions to develop the SwarmFACE package and improve thus on the characterization of the FAC current system using Swarm data:

- Integrate more filtering routines and make access available to more magnetic field models for preparing the magnetic perturbation within SwarmFACE; allow the users to easily change these parameters. For example, using the same magnetic field model and data filtering as in the Level-2 algorithms would ensure a precise comparison with the Swarm FAC products (according to our interpretation, much of the differences between the Level-2 and the SwarmFACE dual-satellite estimations, sufficiently far from the EZ, come from the use of different filters). Similarly, during the more recent close orbit configuration different (smaller) quad scales can be used in the dual-satellite algorithm. In such cases working with a slightly higher cutoff frequency would still provide meaningful estimates, while offering at the same time a more detailed description of the FAC structures.
- Improve the way SwarmFACE package deals with missing or bad quality data points in the input files. Currently, warning messages are issued and results are not provided for timestamps when magnetic information is not available (i.e. records with null magnetic field intensity). A more flexible approach would be to use the quality flags available within the magnetic Level-1b files.
- Increase the flexibility with regard to how the results and plot outputs are produced by the routines, e.g. as objects returned from a function, with file paths provided by the user for saving them.
- Separate the tasks of download/pre-processing and data analysis, adding thus flexibility in the processing system,

e.g. for batch processing or parallel processing. At present, the SwarmFACE higher-level routines that calculate FAC density are mainly designed for event analysis. While in principle a mass processing of the data could be performed by repeatedly calling the existing scripts, downloading small data intervals from VirES platform is inefficient. Similarly, a higher integration of the routines, that avoids inefficient re-download of the data, would be desirable for some analysis, e.g. when a comparison between different FAC density estimations is intended, or when the quality indicators are needed to interpret the FAC density estimation in a three-satellite event.

- Other routines could be added to SwarmFACE package to better characterize the FAC system by using, e.g. the multi scale MVA method introduced in Bunescu et al. (2015) or the technique from Forsyth et al. (2017) to analyze magnetic correlations between Swarm satellites. The magnetic influence of the auroral electrojet at Swarm altitude could be estimated (and corrected for) by the line currents method (e.g. Aakjær et al., 2016) or taking advantage of the Swarm Level 2 Auroral Electrojet products, based on the Spherical Elementary Current Systems method (Amm and Viljanen, 1999).
- Integrate SwarmFACE or parts of it in other *Python* packages where the characterization of FAC system is needed or beneficial. Three such packages are discussed in this Frontiers special issue: the *SwarmPAL* package (see Smith et al., 2022a) aimed to provide a range of analysis and visualization tools for Swarm data product, the *DaedalusMASE* package (see Sarris and Tourgaidis, 2022) developed for the study of the Lower Thermosphere-Ionosphere region, and the *Lompe* (Local Mapping of Polar Ionospheric Electrodynamics) package (see Hovland and Laundal, 2022) where FAC currents, derived from Swarm data as well as from platform magnetometers data like Cryosat, GOCE and GRACE, could be used as additional input. In principle, adjusting the SwarmFACE single-satellite routine for processing other data sets is not expected to be difficult.

5 Resource identification initiative

SwarmFACE relies on *viresclient* to access ESA's Swarm database, has been developed in *Python* Programming Language (RRID:SCR_008394) and makes use of the following *Python* packages: NumPy (RRID:SCR_008633), Pandas (RRID:SCR_018214), MatPlotLib (RRID:SCR_008624), Jupyter Notebook (RRID:SCR_018315), SciPy (RRID:SCR_008058).

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://earth.esa.int/eogateway/missions/swarm/data.

Author contributions

JV lead the development of dual-satellite, three-satellite, and RASADA methods and developed the IDL core codes (dual-satellite, three-satellite) and *Python* code (RASADA). AB developed the rest of the code, adapted the routines to the Swarm context, tested them on synthetic and real data, and provide the *Python* translation of the integrated package. AB wrote the first draft of the manuscript and both authors contributed to manuscript revision and approved the submitted version.

Funding

The work in Romania was supported by the ESA project MAGICS, PRODEX contract No. 4000127660.

Acknowledgments

The authors want to thank Ina Vornsand for testing the SwarmFACE package.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Aakjær, C. D., Olsen, N., and Finlay, C. C. (2016). Determining polar ionospheric electrojet currents from Swarm satellite constellation magnetic data. *Earth, Planets Space* 68, 140. doi:10.1186/s40623-016-0509-y

Amm, O., and Viljanen, A. (1999). Ionospheric disturbance magnetic field continuation from the ground to the ionosphere using spherical elementary current systems. *Earth, Planets Space* 51, 431–440. doi:10.1186/BF03352247

Blagau, A., and Vogt, J. (2019). Multipoint field-aligned current estimates with Swarm. *J. Geophys. Res. (Space Phys.* 124, 6869–6895. doi:10.1029/2018JA026439

Bunescu, C., Marghitu, O., Constantinescu, D., Narita, Y., Vogt, J., and Blăgău, A. (2015). Multiscale field-aligned current analyzer. *J. Geophys. Res. (Space Phys.* 120, 9563–9577. doi:10.1002/2015JA021670

Finlay, C. C., Kloss, C., Olsen, N., Hammer, M. D., Tøffner-Clausen, L., Grayver, A., et al. (2020). The CHAOS-7 geomagnetic field model and observed changes in the South Atlantic Anomaly. *Earth, Planets Space* 72, 156. doi:10.1186/s40623-020-01252-9

Forsyth, C., Rae, I. J., Mann, I. R., and Pakhotin, I. P. (2017). Identifying intervals of temporally invariant field-aligned currents from Swarm: Assessing the validity of single-spacecraft methods. *J. Geophys. Res. (Space Phys.* 122, 3411–3419. doi:10.1002/2016JA023708

Friis-Christensen, E., Lühr, H., and Hulot, G. (2006). Swarm: A constellation to study the Earth's magnetic field. *Earth, Planets, Space* 58, 351–358. doi:10.1186/bf03351933

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

He, M., Vogt, J., Lühr, H., Sorbalo, E., Blagau, A., Le, G., et al. (2012). A high-resolution model of field-aligned currents through empirical orthogonal functions analysis (MFACE). *Geophys. Res. Lett.* 39, L18105. doi:10.1029/2012GL053168

Hovland, A. Ø., Laundal, K. L., Reistad, J. P., Hatch, S. M., Walker, S. J., and Madelaire, M. (2022). The Lompe code: A Python toolbox for ionospheric data analysis. *Front. Astronomy Space Sci* 9, 1025823. doi:10.3389/fspas.2022.1025823

Iijima, T., and Potemra, T. A. (1976). The amplitude distribution of field-aligned currents at northern high latitudes observed by Triad. *J. Geophys. Res.* 81, 2165–2174. doi:10.1029/JA081i013p02165

Luhr, H., Warnecke, J. F., and Rother, M. K. A. (1996). An algorithm for estimating field-aligned currents from single spacecraft magnetic field measurements: A diagnostic tool applied to Freja satellite data. *IEEE Trans. Geoscience Remote Sens.* 34, 1369–1376. doi:10.1109/36.544560

The Pandas Development Team (2020). *pandas-dev/pandas*. Pandas. doi:10.5281/zenodo.7344967

Ritter, P., and Lühr, H. (2006). Curl-B technique applied to Swarm constellation for determining field-aligned currents. *Earth, Planets, Space* 58, 463–476. doi:10.1186/bf03351942

Ritter, P., Lühr, H., and Rauberg, J. (2013). Determining field-aligned currents with the Swarm constellation mission. *Earth, Planets, Space* 65, 1285–1294. doi:10.5047/eps.2013.09.006

Sarris, T. E., Tourgaidis, S., Pirnaris, P., Baloukidis, D., Papadakis, K., Psychalas, C., et al. (2022). Daedalus MASE (Mission Assessment through Simulation Exercise): a toolset for analysis of in situ missions and for processing global circulation model outputs in the Lower Thermosphere-Ionosphere. *Front. Astronomy Space Sci.* 9, 1048318. doi:10.3389/fspas.2022.1048318

Shen, C., Rong, Z. J., and Dunlop, M. (2012). Determining the full magnetic field gradient from two spacecraft measurements under special constraints. *J. Geophys. Res. (Space Phys.* 117, A10217. doi:10.1029/2012JA018063

Smith, A. R. A., and Pačes, M., and Swarm DISC (2022). Python tools for ESA's Swarm mission: VirES for Swarm and surrounding ecosystem. *Front. Astronomy Space Sci.* 9. doi:10.3389/fspas.2022.1002697

Smith, A. R. A., Pačes, M., and Santillan, D. (2022b). ESA-VirES/VirES-Python-Client. doi:10.5281/zenodo.6557779

Sonnerup, B. U. O., and Cahill, L. J., Jr. (1967). Magnetopause structure and attitude from Explorer 12 observations. *J. Geophys. Res.* 72, 171. doi:10.1029/jz072i001p00171

Sonnerup, B. U. Ö., Haaland, S., Paschmann, G., Dunlop, M. W., Rème, H., and Balogh, A. (2006). Orientation and motion of a plasma discontinuity from single-spacecraft measurements: Generic residue analysis of Cluster data. *J. Geophys. Res.* 111, A05203. doi:10.1029/2005JA011538

Sonnerup, B. U. Ö., and Scheible, M. (1998). "Minimum and maximum variance analysis," in *Analysis methods for multi-spacecraft data*. Editors G. Paschmann, and P. W. Daly (ESA Publications Division), ISSI Scientific Reports Series), 185–220.

Tøffner-Clausen, L., Lesur, V., Olsen, N., and Finlay, C. C. (2016). In-flight scalar calibration and characterisation of the Swarm magnetometry package. *Earth, Planets, Space* 68, 129. doi:10.1186/s40623-016-0501-6

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2

Vogt, J., Albert, A., and Marghitu, O. (2009). Analysis of three-spacecraft data using planar reciprocal vectors: Methodological framework and spatial gradient estimation. *Ann. Geophys.* 27, 3249–3273. doi:10.5194/angeo-27-3249-2009

Vogt, J., Blagau, A., and Pick, L. (2020). Robust adaptive spacecraft array derivative analysis. *Earth Space Sci.* 7, e00953. doi:10.1029/2019EA000953

Vogt, J., Haerendel, G., and Glassmeier, K. H. (1999). A model for the reflection of Alfvén waves at the source region of the Birkeland current system: The tau generator. *J. Geophys. Res. (Space Phys.* 104, 269–278. doi:10.1029/1998JA900048

Vogt, J., Sorbalo, E., He, M., and Blagau, A. (2013). Gradient estimation using configurations of two or three spacecraft. *Ann. Geophys.* 31, 1913–1927. doi:10.5194/angeo-31-1913-2013

Zmuda, A. J., and Armstrong, J. C. (1974). The diurnal flow pattern of field-aligned currents. *J. Geophys. Res.* 79, 4611–4619. doi:10.1029/JA079i031p04611

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Methods
PUBLISHED 23 March 2023
DOI 10.3389/fspas.2022.977781

Check for updates

OPEN ACCESS

EDITED BY
K.-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Arnaud Masson,
European Space Astronomy Centre (ESAC), Spain
Gabriele Pierantoni,
University of Westminster, United Kingdom

*CORRESPONDENCE
Shawn Polson,
shawn.polson@lasp.colorado.edu

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 24 June 2022
ACCEPTED 30 November 2022
PUBLISHED 23 March 2023

CITATION
Polson S, Ringuette R, Rastaetter L, Grimes E, Niehof J, Murphy NA and Zheng Y (2023), Making an executable paper with the Python in Heliophysics Community to foster open science and improve reproducibility.
Front. Astron. Space Sci. 9:977781.
doi: 10.3389/fspas.2022.977781

COPYRIGHT
© 2023 Polson, Ringuette, Rastaetter, Grimes, Niehof, Murphy and Zheng. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Making an executable paper with the Python in Heliophysics Community to foster open science and improve reproducibility

Shawn Polson[1]*, Rebecca Ringuette[2,3], Lutz Rastaetter[3], Eric Grimes[4], Jonathan Niehof[5], Nicholas A. Murphy[6] and Yihua Zheng[3]

[1]Laboratory for Atmospheric and Space Physics (LASP), University of Colorado, Boulder, CO, United States, [2]ADNET Systems Inc, Bethesda, MD, United States, [3]Community Coordinated Modeling Center (CCMC), NASA Goddard Space Flight Center, Greenbelt, MD, United States, [4]Institute of Geophysics and Planetary Physics, University of California, Los Angeles, Los Angeles, CA, United States, [5]Space Science Center, University of New Hampshire, Durham, NH, United States, [6]Center for Astrophysics | Harvard and Smithsonian, Cambridge, MA, United States

We share the story of how we made this paper, the first executable paper in Heliophysics, through cross-disciplinary collaboration to highlight the benefits of our process. Executable papers are interactive documents that put a publication's text inline with the code used in the research in a containerized environment with the data and dependencies needed to run the code. This approach enables readers to reproduce every step taken to arrive at the publication's conclusions and to easily build upon and extend the work—all important components of open science. Open science is, broadly speaking, transparent and accessible knowledge that is shared and developed through collaborative networks. In this work, we present an adaptable workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. Most of the authors are members of the Python in Heliophysics Community (PyHC), an international, multi-organizational community that serves as a knowledge base for performing Heliophysics research in the Python programming language. PyHC promotes the executable paper format as a supplemental tool to improve the reproducibility of publications and support open science. A key takeaway is that our collaboration made such a complex task an easy feat in the end. Additionally, the executable version of our paper makes it trivial for others to reproduce our work, and it gives them a better launching point to extend it. These facts underscore the success of our approach. In highlighting this new open science approach, we hope to be an example to our field and encourage this way of doing science.

KEYWORDS

Python in Heliophysics Community, PyHC, executable paper, open science, improving reproducibility, magnetosphere models, cross-disciplinary collaboration, deepnote

1 Introduction

We recount how we made this paper, the first executable paper in Heliophysics, to highlight the benefits of our process. Executable papers are a new kind of paper written in software that combines text, data, and code to enable readers to reproduce every step taken to arrive at the paper's conclusions (Lasser 2020). Our executable paper is centered around an adaptable Python workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. We detail how our process was facilitated by such a collaboration and guided by open science principles.

The following subsections provide the background to understand this work. We set the scene in Section 1.1 by discussing open science and how it relates to issues with reproducibility. Then we describe the organization from which our team originates, the Python in Heliophysics Community (PyHC), and how our goals align with this work in Section 1.2. Then we fully explain the concept of executable papers in Section 1.3—what they are, why they benefit reproducibility, and how they compare to traditional papers. Next, we discuss the cross-disciplinary collaboration underpinning our work and why it was crucial to our success in Section 1.4. Then we give the science background to follow the workflow presented in our executable paper before actually presenting it in Section 1.5. The following "Method" section contains the workflow (Section 2). We will showcase it fully, from the underlying concepts to the implementation using our PyHC packages. Section 3 covers the results and outcomes from our work. Finally, we end the paper with our closing remarks in Section 4.

1.1 Open science and reproducibility

Open science is a disruptive new approach to the scientific process based on cooperative work and new ways of diffusing knowledge by using digital technologies and new collaborative tools (FOSTER 2022). It is about extending the principles of openness to the whole research cycle, fostering sharing and collaboration as early as possible. These principles are at the heart of this work, but it may be intriguing to note that this is our interpretation of open science. It is a new enough concept that it does not yet have an exact formal definition. One systematic literature review came up with the concise definition: "Open science is transparent and accessible knowledge that is shared and developed through collaborative networks" (Vicente-Saez and Martinez-Fuentes 2018). However it is interpreted, we aim to make it a pivotal conversation within the field of Heliophysics. The increased ease of reproducing research gained from the sharing of raw data and research materials is a strong motivator for our work.

It has been six years since a Nature survey "lifted the lid on the reproducibility crisis" (Baker 2016) and scientific communities are still grappling with the issue. The frequency of crisis narratives in publications has skyrocketed (Fanelli 2018; Nelson et al., 2021), but progress has been hampered by a lack of technical computer skills needed to make the results and supporting work in publications reproducible (Bajpai et al., 2017; Cacho and Taghva 2020) along with disagreements on what "reproducibility" means (Baker 2016b). The severity of the crisis (or whether there is one at all) is still debated, but it spawned a growing push to bring reproducible, open science practices into the mainstream. A significant outcome of this push has been the NASA TOPS (Transform to OPen Science) initiative (Gentemann et al., 2021). This five-year initiative designates 2023 as the "Year of Open Science," and it will accelerate the engagement of the scientific community in open science practices through coordinated events and activities.

PyHC embraces open science, and as members of PyHC, we support this push by encouraging the Heliophysics community to create executable papers. We describe how we developed our executable paper by coordinating efforts between scientists and software developers. The purpose of this work is to present our methodology as an example to the community of a more sustainable approach to reproducibility, particularly lessening the increasing demand on the scientists involved.

1.2 The Python in Heliophysics Community

PyHC is an international, multi-organizational community that serves as a knowledge base for performing Heliophysics research in the Python programming language (Ware and Roberts 2019). Our mission is to facilitate scientific discovery by promoting the use and development of sustainable open-source Python software across the solar and space physics community; to improve communication and collaboration between disciplines, developers, and users; to establish and maintain development standards (PyHC 2022d); and to foster interoperability and reproducibility.

Recent years have seen an incredible boom in both the amount of space science data and the software tools used to dissect and investigate it. The problem of today is not "does a tool exist to do this research," but rather "how does one *quickly find which tool* is used to do this research?" PyHC was formed in 2018 to help answer this problem for the Heliophysics and space weather communities. PyHC has done the work of bringing together key members in these communities, organizing community activities, and identifying potential areas for collaboration and integration. This information is housed in a centralized location, the PyHC website (PyHC 2022c), which has over 60 Heliophysics Python packages (PyHC 2022a) and a gallery of examples (PyHC 2022b) about how to use them.

The idea behind this project came from a PyHC telecon during which we were brainstorming new examples for the PyHC Gallery. We wanted an example that showed multiple packages chained together in a unified workflow, which led us to the workflow we demonstrate in the "Method" section below. As we worked on this, however, we came to realize that others could benefit from learning

how we conducted our work. So, the idea grew into this paper at hand. Comparing magnetosphere models to spacecraft observations is an interesting scientific use case, and in chaining together our packages we made a novel workflow that is easier than previous ones, but we are putting the spotlight on our process rather than the science. We could have easily written about any number of other use cases and still made the same point.

1.3 Executable papers

Traditional publications face a variety of issues in the context of reproducibility: missing raw or original data, lack of a tidied-up version of the data, no source code available, or lacking the software to run the experiment. Furthermore, even when all these tools are available, a lack of documentation or deprecated dependencies can make replicating the research a non-trivial task (Cacho and Taghva 2020). In contrast, executable papers include all components of the research. As such, developing one is a significant effort worthy of recognition independent of the typical publication. A current lack of recognition discourages significant effort in this area, however, as indicated by the publication history described in Dr. Jana Lasser's article "Creating an executable paper is a journey through Open Science" (2020):

> There was a short outburst of activity back in 2011, driven by Elsevier (Elsevier n.d.), but other than that little turns up when searching for the term online. Other than a couple of templates (Akhlaghi n.d.), very short guidelines (Reproducible Research n.d.), or very detailed but not very broadly applicable how-to's (CodaLab 2020), there is currently not much to build on.

In that spirit, this work aims to provide a broadly applicable example of how to build an executable paper in Heliophysics—the first of its kind.

The idea of an executable paper is to enable readers to reproduce the steps taken to arrive at a publication's conclusions, from the raw data to polished figures and everything in-between, and to build directly upon the work. It is an interactive document that puts the publication's text inline with the code used in the research, in an environment with the data and dependencies needed to run the code. The environment holding these components is an internal container such as a Docker container (Docker 2022). This containerization means there is never a need to find and download data nor install any software. It also means the executable paper is isolated from any problems that may arise with future versions of the installed software packages, because the current package versions are preserved in working order inside the container.

Executable papers are a fairly new concept with little standardization, but have been gaining traction, particularly with Python-related research, due to the growing popularity of Jupyter notebooks (Jupyter 2022). At present, executable papers *are* Jupyter notebooks—for Python code, at least. It does not *have* to be this way; executable papers are a concept more abstract than Jupyter notebooks. There just does not yet exist an alternate implementation that satisfies the requirements. Because readers can copy a Jupyter notebook and modify it without changing (and potentially breaking) it for others, the executable paper format gives readers a better launching point than traditional papers to produce new research based on the same or similar processes. In contrast, when a paper's methods are only described generically in prose, one might easily come up with different "false cognate" solutions that do not directly replicate the original work. This is especially true when multiple packages are used. Executable papers prevent this by unambiguously showing what code to run when—typically referred to as "direct replication" (American Society for Cell Biology 2015)—and by enabling users to directly build upon the work.

We encourage the Heliophysics community to make executable papers, although we recognize that most impactful journals do not yet accept executable papers as formal submissions. They therefore fulfill the role of an extended preprint that exists next to the publication in the journal (Lasser 2020). Most journals do not typically accept executable papers because there are unsolved problems with them. Chief among them is the maintenance and background support required to keep them working for long periods of time. For example, it is not unheard of to reference papers that are, say, 80–100 years old. It is hard to fathom that level of support being supplied to executable papers (or any software, for that matter) for such a long period of time. Containerization adds stability to executable papers, but only when 100% of all the research's components are inside the container. If, for example, the data were stored on a separate server because it did not fit in the container, then that executable paper's reproducibility would only be valid for as long as the linked data was available. Additionally, making executable papers takes extra work and a certain level of technical proficiency. We recognize the stability problem as well outside the scope of this paper. However, we address the technical proficiency problem by forming a collaboration between a collection of scientists and software developers, distributing the work according to their expertise. We pause here to briefly describe the notebook platform we chose for this project, Deepnote, because it uniquely facilitated this collaborative development.

1.3.1 Deepnote

Deepnote is a cloud-based notebook platform with all the standard Jupyter functionalities as well as real-time collaboration (Deepnote 2022). It is newer than its well-established competitors (which we discuss in "Supplementary Appendix B") and, in fact, just came out of beta in May 2022. Despite its newness, Deepnote's real-time collaborative editing capability swayed us into choosing it. Users of Google Docs will be familiar with this feature; everyone works on the same document simultaneously, and all edits are immediately reflected in real-time for everyone.

In addition to this feature, Deepnote supports everything necessary to make an executable paper. It offers fully-customizable environments for each notebook to handle software dependencies (through Docker images). It offers a persistent file system within these environments to store data, other media, and the notebooks themselves. It offers integrations with third-party large-file storage services

to manage data files that exceed the default storage limits. It even has a feature to publish and host notebooks as publicly-available "articles", which is particularly well-suited for sharing executable papers. We shared ours using that feature.[1] Although, in recognition that platforms can change and links can break, we also uploaded our Docker image files to Zenodo in order to obtain a DOI.[2]

1.4 Collaboration

Our team consists of scientists and PyHC members, most serving a software development role, some serving as topic experts to help with the science. Some have backgrounds in both roles but served in only one role for this project. This cross-disciplinary collaboration made us more capable. A single team member aiming to complete the entire task would have been faced with either a lack of scientific, software, or technical expertise, depending on the team member's background, which typically makes such a project too time consuming. Our developers did not know enough about the science of magnetospheres and our scientists did not know enough about the software. But, in distributing work according to our expertise, we completed the task with relative ease. We hope this is encouragement for readers finding themselves daunted by a similar task.

We developed this executable paper together in a single shared Jupyter notebook. Deepnote's real-time collaboration made this simple and convenient. Effort was taken from the beginning by everyone to document and explain the steps of their work as comments in the notebook, instead of separately elsewhere. This allowed us to understand each other's work and made it so that much of the paper was already written by the time development ended. To give two examples of task distribution: Eric Grimes wrote the portions of this paper that use pySPEDAS because he leads development of the pySPEDAS project, and Yihua Zheng provided guidance about the magnetosphere models we used because she is one of our topic experts. We note that more scientists could easily be included in the group, and the number of software developers in the group could easily be fewer, all depending on the complexity of the science question being addressed and the software needed. See "Supplementary Appendix A" for descriptions of our team members, including their backgrounds and contributions to this project.

1.5 Comparing magnetosphere models to spacecraft observations

We present the current executable workflow as an example of the level of open science and reproducibility possible through this approach. We have covered our process for developing the workflow, so, let us finally turn to the science of its contents: comparing magnetosphere models to spacecraft observations. The magnetosphere is the region in which Earth's magnetic field is the predominant effective magnetic field (NASA 2022). The magnetopause is the magnetosphere's outer limit (Merriam-Webster 2022). In this simple workflow, we can detect where the magnetopause is by observing magnetopause crossings in spacecraft readings, test the accuracy of empirical magnetopause models by comparing the observed locations to modeled ones, and validate physics-based magnetosphere models by virtually flying spacecraft through them and comparing the modeled readings to the observed data. This is made possible by five PyHC packages: pySPEDAS, SpacePy, PlasmaPy, PyTplot, and Kamodo. We describe these packages in "Supplementary Appendix C."

Empirical magnetopause models, such as the one created by Shue et al., estimate where the magnetopause is by deriving a functional form that depends on solar wind parameters. The Shue et al. function represents the shape of the magnetopause and its scale (represented by the standoff distance on the Sun-Earth line) depending on the solar wind parameters (Shue et al., 1998). Physics-based models, such as the Open Geospace General Circulation Model (OpenGGCM), model the magnetosphere by deriving its properties from physical first principles, and provide the global state of the entire geospace (which includes the magnetosphere) (Community Coordinated Modeling Center, 2022). We compare these modeled data to data from NASA's Magnetospheric Multiscale Mission (MMS), which orbit Earth and regularly cross the magnetopause boundary.

We demonstrate how to use SpacePy functions to calculate metrics from MMS data to detect where these crossings happen and compare the calculated locations to those estimated by the Shue et al. model to test the model's accuracy. MMS also takes other magnetosphere readings as it orbits, such as magnetic field components, which can be used to calculate plasma parameters. We show how to use PlasmaPy to calculate plasma parameters from these readings to augment our detections of magnetopause crossings. We also show how to virtually fly MMS through the OpenGGCM model and compare the modeled readings to the observed data using Kamodo's flythrough capability, again to test the model's accuracy. This workflow is depicted in Figure 1. Although we use MMS data with the Shue et al. and OpenGGCM models in this example, the workflow can be easily adapted to numerous datasets and models available through these and other packages.

1 Link to the executable version of this paper on Deepnote: https://deepnote.com/@shawn-polson/PyHC-Paper-101b9646-3fd0-4978-a48e-a4f3e708a0ac.

2 DOI to the files of the executable version of this paper on Zenodo: https://doi.org/10.5281/zenodo.7412347.

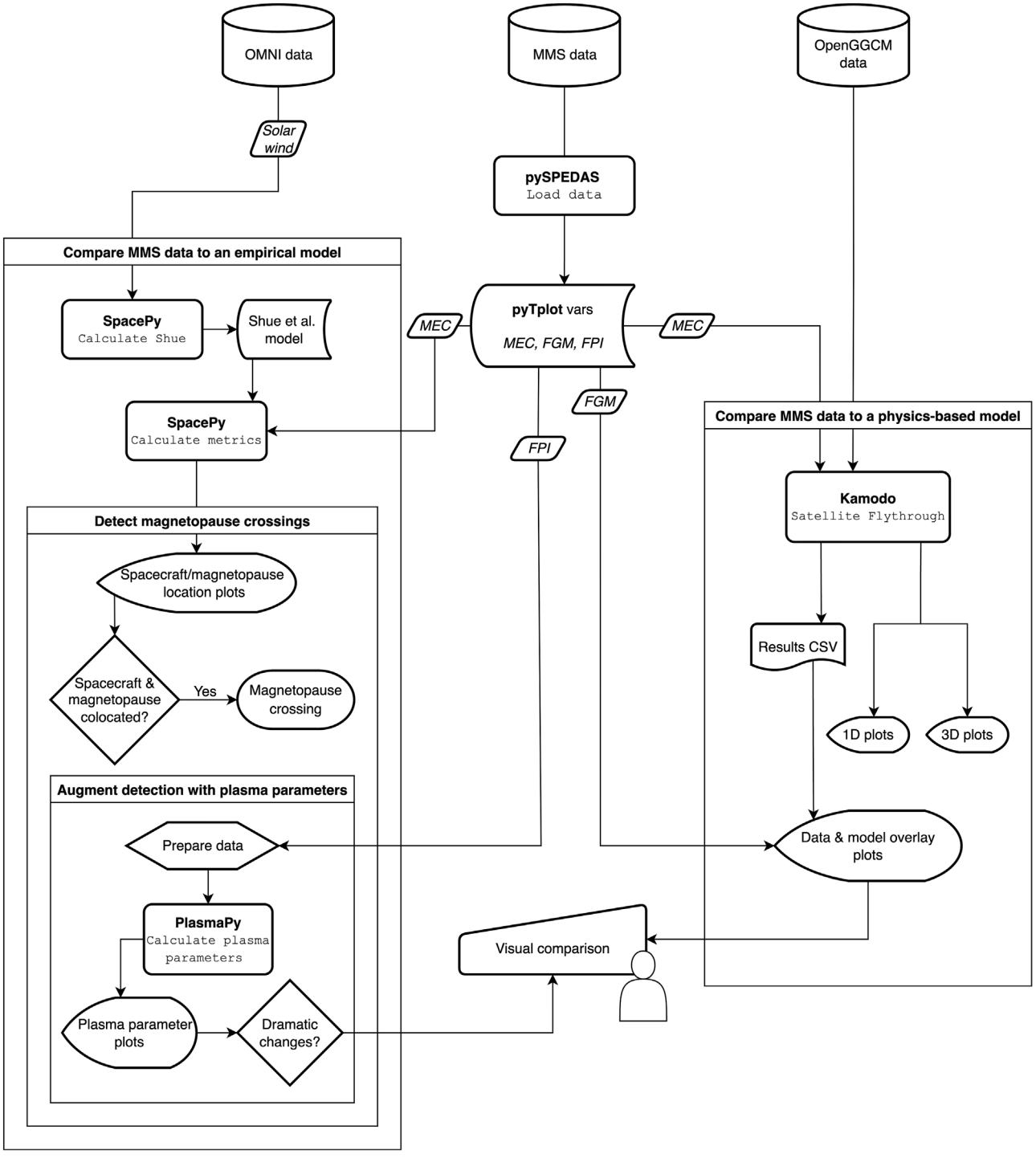


FIGURE 1
A flowchart depicting our workflow to compare magnetosphere models to spacecraft observations.

1.5.1 Data specifics

We use science-quality level 2 fast survey data from MMS probe 1 that is preloaded into our container's file system. Specifically, we use: MMS Ephemeris and Coordinates (MEC) data to retrieve the spacecraft's location, Fluxgate Magnetometer (FGM) data to measure the magnetic field strength experienced by the spacecraft, and Ion moments from the Fast Plasma Investigation (FPI) data to show how the ion population changes as the spacecraft crosses the magnetopause. We look at data from 16 October 2015 because data from that date is used in several papers to compare observational data to model data (Burch et al., 2016; Alm et al., 2017; Egedal et al., 2017).

The OpenGGCM model data we use was generated by CCMC for this work and is also stored in our file system. Other model data outputs are freely available to the public by request (https://ccmc.gsfc.nasa.gov/). Our model (ID Yihua_Zheng_040122_1) uses default settings with OpenGGCM 5.0, Rice Convection Model (RCM) 1.0, and OMNI data (Qin et al., 2007) as solar wind input. Specific input parameters are given as a footnote.[3] To save storage space, our model was intentionally generated at such a coarse resolution that it is not useful for deep scientific analysis.[4]

2 Methods

We turn to our workflow to compare magnetosphere models to spacecraft observations.

2.1 Imports

First, we need to import everything. We import `pyspedas` to load the MMS data and `tplot` from `pytplot` to plot it. Then we configure the necessary CDF library environment variables and make sure we have the solar wind data updated in our `".spacepy/"` directory before importing the `spacepy` modules relevant to our magnetopause location calculations (plus `numpy` and `matplotlib`). We import `plasmapy` (plus `astropy` units) to augment our detections of magnetopause crossings. Finally, we import the `kamodo` modules relevant to performing our model flythrough.

3 OpenGGCM input parameters:
- run ID: Yihua_Zheng_031022_1
- model: OpenGGCM
- version: 5.0
- IM_model: RCM
- IM_version: 1.0
- cs_input: GSM
- cs_output: GSE
- event_date: 16 October 2015
- start_time: 2015/10/16 11:30
- end_time: 2015/10/16 17:00
- run_type: event
- solarwind: var
- sw_source: OMNI
- despike solar wind data: 0
- despike: threshold (sigmas): 3
- despike: number of samples: 5
- setting option for Bx: user
- constant-Bx: 0
- By-coefficient: 0
- Bz-coefficient: 0
- b_abs: 2.19
- b_angle: 123.13
- iono_conductance: auroral
- Pedersen Conductance: default
- Hall Conductance: default
- f10.7: 108.4.

4 The OpenGGCM model was run with a low-resolution grid ("dayside emphasis grid with 3,500,000 cells" with 355 × 100 × 100 cells) in the simulation domain extending from -350 to 33 R_E in X_{GSM}, and up to 49 R_E in $|Y_{GSE}|$ and $|Z_{GSE}|$ with a 0.3 R_E resolution around the Earth. During the packaging of magnetosphere model outputs into Kamodo, NetCDF files were reduced to only include magnetic field components (B_x, B_y, B_z) at a 600-s time cadence to limit model outputs to 1.6 GB. The near-Earth boundary conditions at 2.5 R_E distance includes 100 particles/cm^3, a temperature of 100 eV, and a shielding latitude of 45° in the ionosphere electric field potential solver.

```
# pySPEDAS imports
import pyspedas

# PyTplot imports
import pytplot
from pytplot import tplot

# SpacePy imports
import matplotlib.pyplot as plt
import numpy as np
import spacepy.coordinates
import spacepy.empiricals
import spacepy.time
import spacepy.toolbox

# PlasmaPy imports
import plasmapy
from astropy import units as u

# Kamodo imports
from kamodo_ccmc.flythrough import SatelliteFlythrough as S
import kamodo_ccmc.flythrough.model_wrapper as MW
```

2.2 Get MMS data

We retrieve MMS data using pySPEDAS for the time range specified in the `trange` variable. Note that pySPEDAS can retrieve data from any supported project (pySPEDAS 2022b) and that `trange` can be adjusted to any time range. The data gets downloaded into the `pydata` directory. No download is necessary if the data is already in the directory, such as ours is.

2.2.1 Set the time range for loading the data

```
trange = ['2015-10-16/11:30', '2015-10-16/17:00']
```

2.2.2 Load the MEC, FGM and DIS data from pySPEDAS

Data can be loaded by calling `pyspedas.mission.instrument()` with options set via keyword arguments. For example, MEC data can be loaded by calling `pyspedas.mms.mec()`. There is, by default, a prompt to enter a science data center (SDC) username (which could be blank) before getting the MMS data. We avoid the prompt by creating an `mms_auth_info.pkl` file during our initialization. Data automatically gets loaded into Tplot variables upon loading (PyTplot 2022b).

MMS ephemeris and coordinates (MEC) data

These data show the spacecraft location at the times of the magnetopause crossings.

Fluxgate magnetometer (FGM) data

These data show the magnetic field as the spacecraft crosses the magnetopause.

Ion moments from the fast plasma investigation (FPI) data

These data show how the ion population changes as the spacecraft crosses the magnetopause.

```
def load_mms_data(time_range):
    """Loads MMS MEC, FGM, and FPI data."""
    mec_vars = pyspedas.mms.mec(trange=time_range, time_clip=True, probe=1, level='l2',
                                data_rate='srvy', varformat='*_r_gsm')  # Cartesian GSM coords

    fgm_vars = pyspedas.mms.fgm(trange=time_range, time_clip=True, probe=1, level='l2',
                                data_rate='srvy')

    ion_vars = pyspedas.mms.fpi(trange=time_range, datatype=['dis-moms', 'des-moms'], level='l2',
                                data_rate='fast', time_clip=True, center_measurement=True)

    return mec_vars, fgm_vars, ion_vars
```

2.3 Compare MMS data to an empirical model

We use SpacePy to calculate where our MMS spacecraft is in relation to the magnetopause. SpacePy implements the empirical magnetopause model of Shue et al. (1998). A simple function supports calculating positions in the equatorial plane. For the out-of-plane calculations used here, more flexible functions are available. Upstream solar wind outputs, which are required for the model, are supported via SpacePy's `omni` module using the OMNI data stored in our "`.spacepy/`" directory. We first perform the calculations then plot the results to visualize them.

SpacePy can work with other empirical models which also offer plasmapause distance and last closed drift shell as outputs.

2.3.1 Perform SpacePy calculations with MMS data

```
def spacecraft_magnetopause_calculations(mms_mec_vars):
    """Returns epoch,
               pos,
               distance between spacecraft and magnetopause,
               magnetopause's distance from Earth,
               spacecraft's distance from Earth,
               and solar zenith angle.
    """
    data = pytplot.get_data(mms_mec_vars[0])
    pos_gsm = data.y
    ticks = spacepy.time.Ticktock(data.times, dtype='UNX')
    epoch = ticks.UTC
    c = spacepy.coordinates.Coords(pos_gsm, 'GSM', 'car', units='km', ticks=ticks)
    pos = c.convert('GSE', 'car').data

    # Get the Shue coefficients
    alpha = []  # Shue flaring angle
    standoff = spacepy.empiricals.getMPstandoff(ticks, alpha=alpha)  # Shue subsolar standoff
    alpha = np.array(alpha)

    # Solar zenith angle of s/c position (angle with GSE +x)
    sza = np.arctan2(pos[:, 0], np.sqrt((pos[:, :2] ** 2).sum(axis=1)))
    # Radial distance to MP along Earth-SC line (application of Shue coefficients)
    mp_dist = standoff * (2. / (1 + np.cos(sza))) ** alpha
    # Radial distance to SC
    sc_dist = np.sqrt((pos ** 2).sum(axis=1)) / 6378
    # How far is SC outside of MP?
    sc_to_mp = sc_dist - mp_dist

    return epoch, pos, sc_to_mp, mp_dist, sc_dist, sza
```

2.3.2 Detect magnetopause crossings

Since we can calculate the distance between the spacecraft and the magnetopause, we can find crossings by collecting the indices where that distance crosses 0 (i.e., changes sign):

```
def find_magnetopause_crossings(sc_to_mp):
    """Returns the indices of elements before which zero crossings occur."""
    return np.where(np.diff(np.sign(sc_to_mp)))[0]
```

2.3.3 Plot distance outside magnetopause, spacecraft and magnetopause locations, and solar zenith angle

The following blocks of code plot each for our time range, using the helper functions we have so far defined.

```
# loaded Tplot variables are printed as output
mec_vars, fgm_vars, ion_vars = load_mms_data(trange)
epoch, pos, sc_to_mp, mp_dist, sc_dist, sza = spacecraft_magnetopause_calculations(mec_vars)
```

```
%matplotlib inline

fig1, axs1 = plt.subplots(1, 1, figsize=(8, 4), sharey=True)
fig2, axs2 = plt.subplots(1, 1, figsize=(8, 4), sharey=True)
fig3, axs3 = plt.subplots(1, 1, figsize=(8, 4), sharey=True)
fig4, axs4 = plt.subplots(1, 1, figsize=(8, 4), sharey=True)

# Spacecraft distance outside magnetopause
fig1.suptitle('(A) Distance Outside Magnetopause')
axs1.plot(epoch, sc_to_mp)
axs1.set(xlabel='Time', ylabel='Re')
crossings = find_magnetopause_crossings(sc_to_mp)
for x in crossings:
    axs1.plot(epoch[x], sc_to_mp[x], 'o', label='Crossing')
axs1.legend()
fig1.autofmt_xdate()

# Spacecraft and magnetopause locations
fig2.suptitle('(B) Distance From Earth')
l_mp, = axs2.plot(epoch, mp_dist, label='Magnetopause')
l_sc, = axs2.plot(epoch, sc_dist, label='Spacecraft')
axs2.set(xlabel='Time', ylabel='Re')
fig2.autofmt_xdate()
fig2.legend([l_mp, l_sc], [l_mp.get_label(), l_sc.get_label()], loc='right')

# Solar zenith angle
fig3.suptitle('(C) Solar Zenith Angle')
axs3.plot(epoch, np.degrees(sza))
axs3.set(xlabel='Time')
fig3.autofmt_xdate()

# Spacecraft coordinates
fig4.suptitle('(D) Spacecraft Coordinates')
l_x, = axs4.plot(epoch, pos[:, 0] / 6378, label='x')
l_y, = axs4.plot(epoch, pos[:, 1] / 6378, label='y')
l_z, = axs4.plot(epoch, pos[:, 2] / 6378, label='z')
fig4.autofmt_xdate()
fig4.legend([l_x, l_y, l_z], [l_x.get_label(), l_y.get_label(), l_z.get_label()], loc='right')
```

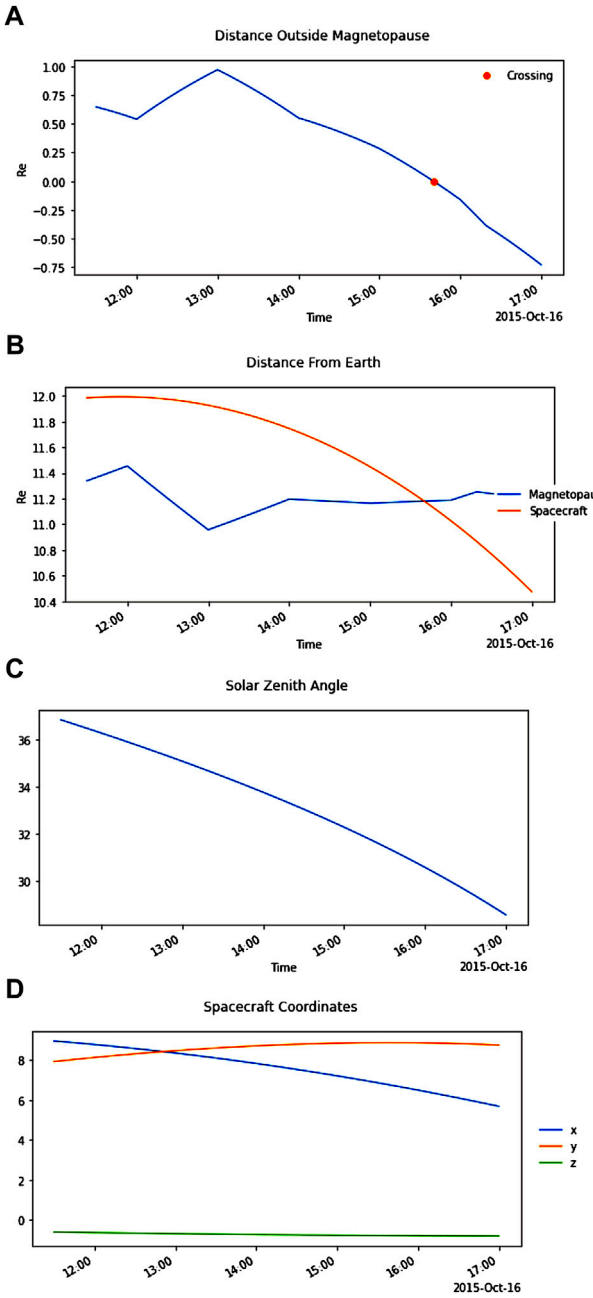


FIGURE 2
Time series plots showing **(A)** the distance between the MMS spacecraft and the modeled magnetopause, **(B)** the distances between the Earth and both the MMS spacecraft and the modeled magnetopause, **(C)** the solar zenith angle, and **(D)** the X, Y, and Z Cartesian coordinates of the MMS spacecraft. Note how plot **(A)** shows a predicted magnetopause crossing slightly before 16:00.

2.4 Augment comparison with plasma parameters

We use PlasmaPy to calculate various plasma parameters from the MMS data to help detect when MMS crosses the magnetopause boundary. Note that we calculate only a subset of the parameters PlasmaPy supports (PlasmaPy 2022a). The MMS data is retrieved from its Tplot variables and goes through some preparation before PlasmaPy can use it. We interpolate to a common set of times using the `tinterpol` function from pySPEDAS. We calculate the total temperature from the two temperature components given to us. And, lastly, we add Astropy units to the data (Astropy, 2022). Once we have these plasma parameters, we can plot them and visually compare them to plots of the MMS data. If the plots show sudden dramatic changes around a particular time, we can be reasonably confident that a magnetopause crossing happened then.

2.4.1 Prepare data for plasmaPy

2.4.1.1 Interpolate to a common set of times

We need to interpolate the B-field and DES (electron) data to the DIS (ion) time stamps. Note that `tinterpol` creates a new Tplot variable containing the interpolated output with the suffix '-itrp'.

```
from pyspedas import tinterpol
tinterpol('mms1_fgm_b_gse_srvy_l2_btot', 'mms1_dis_numberdensity_fast')
tinterpol('mms1_des_numberdensity_fast', 'mms1_dis_numberdensity_fast')
tinterpol('mms1_des_temppara_fast', 'mms1_dis_numberdensity_fast')
tinterpol('mms1_des_tempperp_fast', 'mms1_dis_numberdensity_fast')
```

2.4.1.2 Extract the data values

```
from pytplot import get_data
fgm_b = get_data('mms1_fgm_b_gse_srvy_l2_btot-itrp')
dis_n = get_data('mms1_dis_numberdensity_fast')
dis_Tpara = get_data('mms1_dis_temppara_fast')
dis_Tperp = get_data('mms1_dis_tempperp_fast')
des_n = get_data('mms1_des_numberdensity_fast-itrp')
des_Tpara = get_data('mms1_des_temppara_fast-itrp')
des_Tperp = get_data('mms1_des_tempperp_fast-itrp')
```

2.4.1.3 Calculate temperature

Temperature data released by the FPI team come as parallel and perpendicular components, but we need the total temperature. For details on this calculation, see the FPI Data Product Guide (LASP, 2022).

```
dis_T = (dis_Tpara.y + 2*dis_Tperp.y) / 3.0
des_T = (des_Tpara.y + 2*des_Tperp.y) / 3.0
```

2.4.1.4 Add units to the data

PlasmaPy requires us to specify the units of the data using Astropy units.

```
B = fgm_b.y * u.nT
n_i = dis_n.y * u.cm**-3
n_e = des_n.y * u.cm**-3
T_i = dis_T * u.eV
T_e = des_T * u.eV
```

2.4.2 Calculate plasma parameters

2.4.2.1 Alfvén speed

```
Va = plasmapy.formulary.parameters.Alfven_speed(B, n_i, 'p')

# convert to km / s
Va = Va.to(u.km / u.s)
```

2.4.2.2 Plasma beta

```
# ions
beta_i = plasmapy.formulary.dimensionless.beta(T_i, n_i, B)

# electrons
beta_e = plasmapy.formulary.dimensionless.beta(T_e, n_e, B)

# combined
beta = beta_i + beta_e
```

2.4.2.3 Ion inertial length

```
n_i = np.float64(n_i.value) * u.cm ** -3  # convert values from float32 to float64
d_i = plasmapy.formulary.parameters.inertial_length(n_i, 'p+')
```

2.4.2.4 Debye length

```
lamda_d = plasmapy.formulary.parameters.Debye_length(T_e, n_e)
```

2.4.2.5 Ion gyrofrequency

```
omega_ci = plasmapy.formulary.parameters.gyrofrequency(B, 'p', to_hz=True)
```

2.4.2.6 Ion gyroradius

```
r_i = plasmapy.formulary.parameters.gyroradius(B, 'p', T=T_i)

# convert to km
r_i = r_i.to(u.km)
```

2.4.2.7 Bohm diffusion coefficient

```
DB = plasmapy.formulary.parameters.Bohm_diffusion(T_e, B)
```

2.4.2.8 Lower hybrid frequency

```
omega_lh = plasmapy.formulary.parameters.lower_hybrid_frequency(B, n_i, 'p', to_hz=True)
```

2.4.2.9 Upper hybrid frequency

```
omega_uh = plasmapy.formulary.parameters.upper_hybrid_frequency(B, n_e, to_hz=True)
```

2.4.3 Generate plots with PyTplot

We can plot the calculated the plasma parameters with PyTplot.

2.4.3.1 Save the data in tplot variables

```
from pytplot import store_data
store_data('alfven_speed', data={'x': fgm_b.times, 'y': Va})
store_data('plasma_beta', data={'x': fgm_b.times, 'y': beta})
store_data('ion_inertial_length', data={'x': fgm_b.times, 'y': d_i})
store_data('debye_length', data={'x': fgm_b.times, 'y': lamda_d})
store_data('omega_ci', data={'x': fgm_b.times, 'y': omega_ci})
store_data('ion_gyroradius', data={'x': fgm_b.times, 'y': r_i})
store_data('bohm_diffusion_coeff', data={'x': fgm_b.times, 'y': DB})
store_data('omega_lh', data={'x': fgm_b.times, 'y': omega_lh})
store_data('omega_uh', data={'x': fgm_b.times, 'y': omega_uh})
```

2.4.3.2 Set some plot metadata

```
from pytplot import options
options('alfven_speed', 'ytitle', 'Va \\ (' + str(Va.unit) + ')')
options('alfven_speed', 'legend_names', 'Alfvén speed')
options('plasma_beta', 'ytitle', 'Beta')
options('plasma_beta', 'legend_names', 'Plasma Beta')
options('ion_inertial_length', 'ytitle', 'd_i \\ (' + str(d_i.unit) + ')')
options('ion_inertial_length', 'legend_names', 'Ion inertial length')
options('debye_length', 'ytitle', 'Lamda_d \\ (' + str(lamda_d.unit) + ')')
options('debye_length', 'legend_names', 'Debye length')
options('omega_ci', 'ytitle', 'omega_ci \\ (' + str(omega_ci.unit) + ')')
options('omega_ci', 'legend_names', 'Ion gyrofrequency')
options('ion_gyroradius', 'ytitle', 'r_i \\ (' + str(r_i.unit) + ')')
options('ion_gyroradius', 'legend_names', 'Ion gyroradius')
options('bohm_diffusion_coeff', 'ytitle', 'DB \\ (' + str(DB.unit) + ')')
options('bohm_diffusion_coeff', 'legend_names', 'Bohm diffusion coefficient')
options('omega_uh', 'ytitle', 'omega_uh \\ (' + str(omega_uh.unit) + ')')
options('omega_uh', 'legend_names', 'Upper hybrid frequency')
options('omega_lh', 'ytitle', 'omega_lh \\ (' + str(omega_lh.unit) + ')')
options('omega_lh', 'legend_names', 'Lower hybrid frequency')
```

2.4.3.3 Plot the plasma parameters

```
tplot(['alfven_speed',
       'plasma_beta',
       'ion_inertial_length',
       'debye_length',
       'omega_ci',
       'ion_gyroradius',
       'bohm_diffusion_coeff',
       'omega_uh',
       'omega_lh'])
```

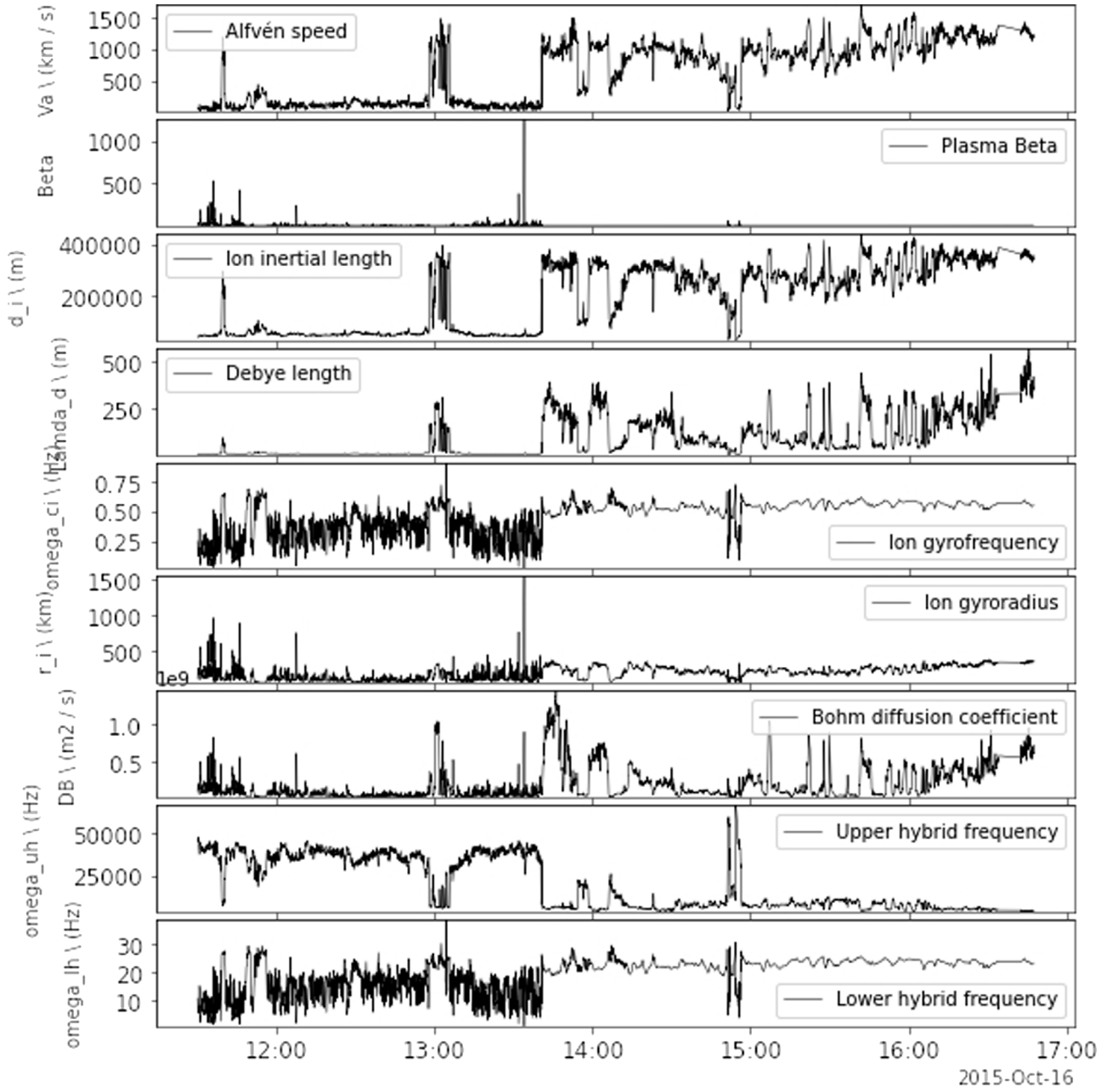


FIGURE 3
Time series plots showing the calculated plasma parameters. Note the dramatic visual changes between 13:00–14:00 and the spikes just before 15:00.

2.4.3.4 Plot the MMS data

```
tplot(['mms1_dis_energyspectr_omni_fast',
       'mms1_dis_numberdensity_fast',
       'mms1_fgm_b_gsm_srvy_l2_bvec',
       'mms1_mec_r_gsm'])  # Plot a multi-instrument figure
```

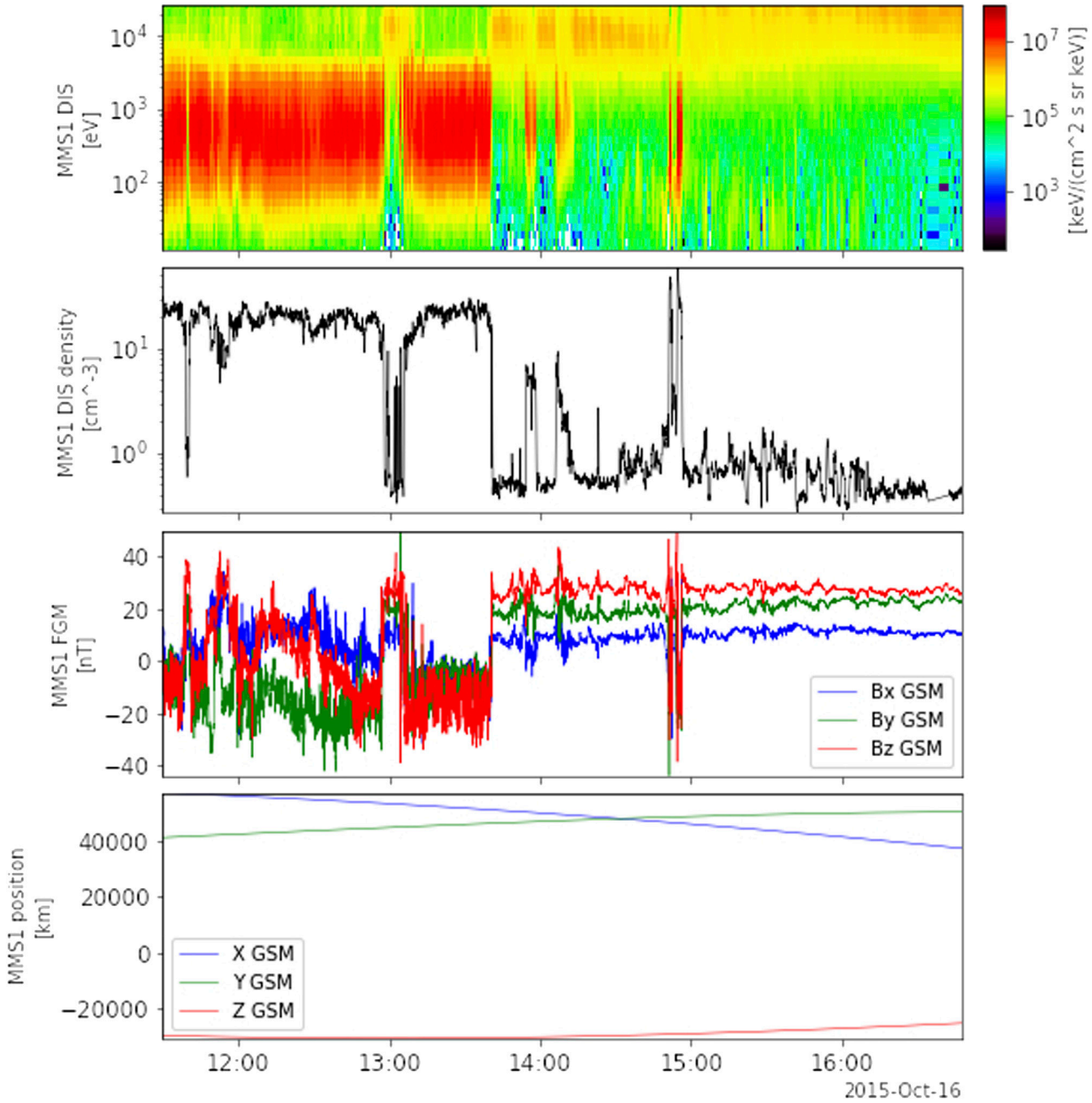


FIGURE 4
Time series plots showing the observed MMS data. Note the dramatic visual changes between 13:00–14:00 and the spikes just before 15:00.

2.4.4 Visual comparison

We can identify magnetopause boundary crossings in our data by looking for sudden changes around particular times. Both Figure 2 and Figure 3 show dramatic visual changes between 13:00–14:00 and spikes just before 15:00. These indicate at least two magnetopause crossings. If we compare these times with the earlier plot of a predicted crossing, Figure 4, we find a disagreement. The Shue et al. model predicted a crossing would have happened close to 16:00. *Therefore, we can conclude the magnetopause was farther from Earth than the Shue et al. model predicted.*

2.5 Compare MMS data to a physics-based model

We use Kamodo to virtually fly the MMS spacecraft through the OpenGGCM magnetosphere model (Ringuette et al., 2022). Note that Kamodo's flythrough capability provides access to various physics-based models and works on a variety of model types, not just magnetospheric models. See Kamodo's flythrough documentation for more information (Kamodo 2022e). The choice of model is specified with a String variable (Kamodo 2022d). Run `MW.Choose_Model('')` to see which Strings are allowed. A writable path to the model data files must be given to the `ModelFlythrough` function. The path must be writable because, upon running the `ModelFlythrough` function, Kamodo adds a CSV file to that directory which will have all available time ranges covered by the model's data files.

2.5.1 Perform model flythrough with kamodo

2.5.1.1 Specify the model

```
model = 'OpenGGCM_GM'
```

2.5.1.2 See what variable names are available from the model

```
MW.Model_Variables(model)
```

```
The model accepts the standardized variable names listed below.
--------------------------------------------------------------------------------
B_x : '['x component of magnetic field', 0, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'nT']'
B_y : '['y component of magnetic field', 1, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'nT']'
B_z : '['z component of magnetic field', 2, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'nT']'
E_x : '['x component of electric field', 6, 'GSE', 'car', ['time', 'x', 'x', 'x'], 'mV/m']'
E_y : '['y component of electric field', 7, 'GSE', 'car', ['time', 'y', 'y', 'y'], 'mV/m']'
E_z : '['z component of electric field', 8, 'GSE', 'car', ['time', 'z', 'z', 'z'], 'mV/m']'
N_plasma : '['number density of plasma (hydrogen equivalent)', 12, 'GSE', 'car', ['time', 'x', 'y',
 'z'], '1/cm**3']'
P_plasma : '['plasma pressure', 14, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'pPa']'
eta : '['resistivity', 13, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'm**2/s']'
j_x : '['current density, x component', 15, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'muA/m**2']'
j_y : '['current density, y component', 16, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'muA/m**2']'
j_z : '['current density, z component', 17, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'muA/m**2']'
v_plasmax : '['x component of plasma velocity', 9, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'km/s']'
v_plasmay : '['y component of plasma velocity', 10, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'km/s']'
v_plasmaz : '['z component of plasma velocity', 11, 'GSE', 'car', ['time', 'x', 'y', 'z'], 'km/s']'
```

Note: although the model accepts all the variable names above, we have only populated our model files with data for `B_x`, `B_y`, and `B_z` to conserve space.

2.5.1.3 Prepare input variables

```
# Choose input values for ModelFlythrough function call
file_dir = '/root/work/pydata/kamodo/'             # full file path to where the model output data is stored
variable_list = ['B_x','B_y', 'B_z']               # list of desired variable names
coord_sys = "GSM-car"                              # GSM and cartesian because we're using x,y,z from above
output_dir = '/root/work/output/ModelFlythrough/'  # directory where the user wants the output stored
file_name = 'OpenGGCMFlythrough'                   # what the user wants the output files to be named
output_name = output_dir + file_name + ".csv"      # output dir plus output file name without extension
plot_coord = 'GSM'                                 # coordinate system chosen for output plots

# Convert to UTC timestamps
from datetime import datetime, timezone
sat_time = [time.replace(tzinfo=timezone.utc).timestamp() for time in epoch]  # times should be in the model

# Use MMS trajectory acquired from PySPEDAS/SpacePy
sat_x = pos[:, 0] / 6378
sat_y = pos[:, 1] / 6378
sat_z = pos[:, 2] / 6378
```

2.5.1.4 Perform the flythrough

The numeric results of this flythrough are exported to a file we named with the `output_name` variable. If a file by that name already exists, the `ModelFlythrough` function tells the user through an error message in order to prevent them from accidentally overwriting previous results. If the results have already been exported, they can be read from the file with a reader function as shown below.

Run `ModelFlythrough` with user-supplied trajectory to generate results:

```
results = S.ModelFlythrough(model, file_dir, variable_list, sat_time, sat_x, sat_y, sat_z,
                            coord_sys, output_name=output_name, plot_coord=plot_coord)
```

If results have been previously generated, use the data import function to reload them:

```
results = S.WO.SF_read(output_name)
```

2.5.1.5 Interact with flythrough plots

After the flythrough completes, the output plot files are exported to the directory `output_dir`; open them in a separate browser window for interactivity—nothing will open here. Both 3D and 1D plots are generated (Figures 5, 6).

3D plots will look like:

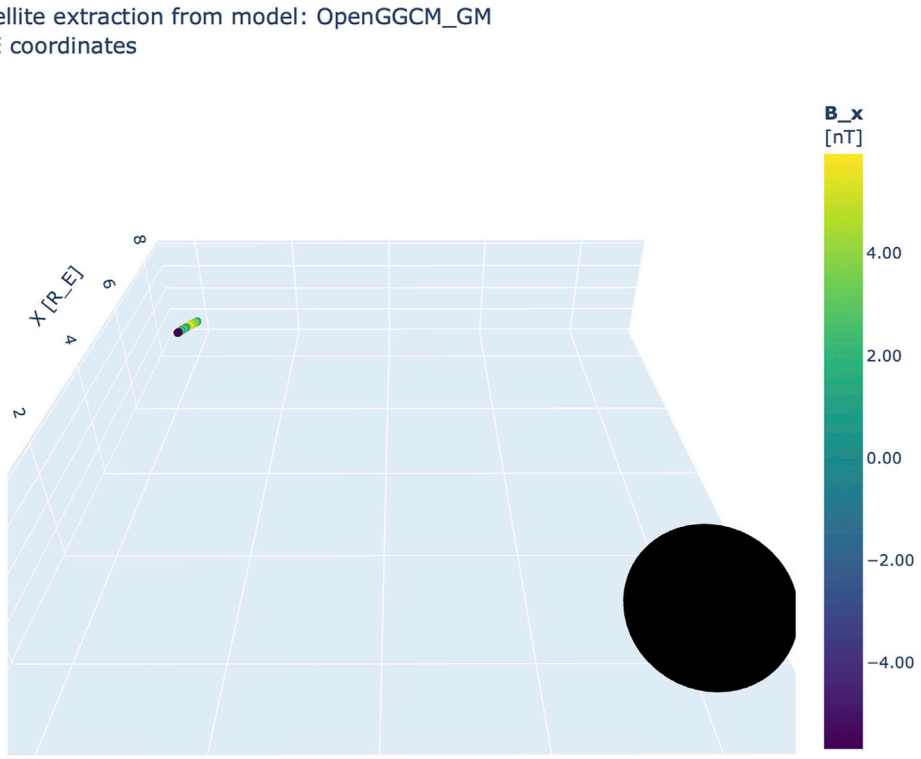


FIGURE 5
A screenshot of a 3D plot produced by the Kamodo flythrough.

1D plots will look like:

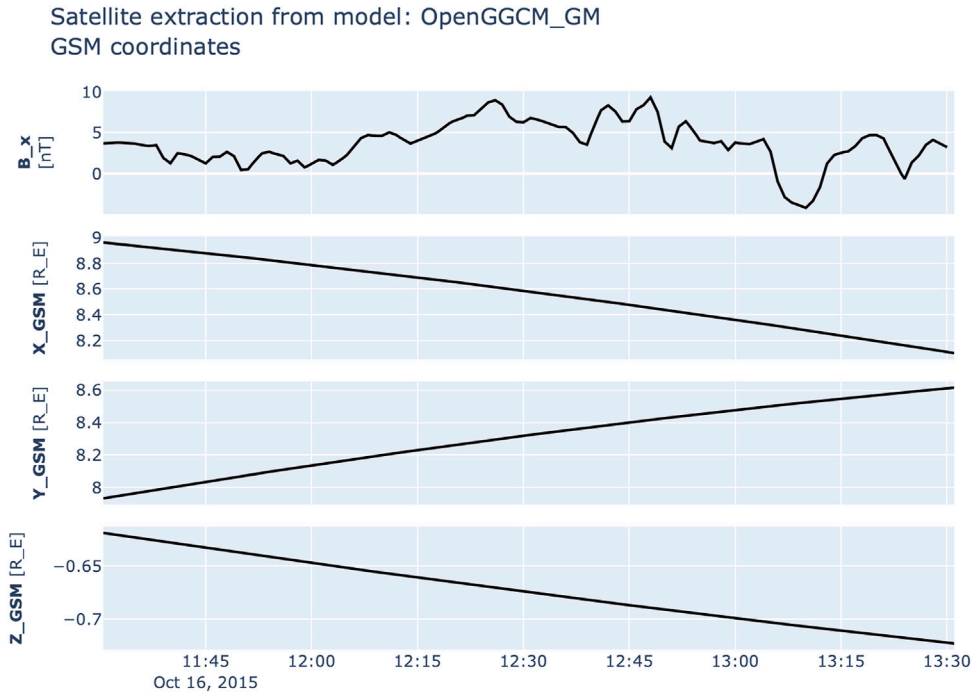


FIGURE 6
A screenshot of a 1D plot produced by the Kamodo flythrough.

2.5.2 Visual comparison

We now have values from a model of the magnetic field components measured by our MMS spacecraft. We can plot the modeled and observed values together to see how accurate the model is.

We do so using Kamodo's built-in plotting capability, rather than `matplotlib`, because it produces interactive plots with very little code. We "Kamodofy" the results before we can plot them. "Kamodofication" is a concept Kamodo uses to "functionalize" callable functions, something that allows many problems in scientific data analysis to be posed in terms of function composition and evaluation. See the Kamodofication documentation for details about this concept (Kamodo 2022c).

Note that the model data files we use contain only the variables output by `MW.File_Variables(model, file_dir)` as compared to the full list of variables listed by `MW.Model_Variables(model)`. We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the modeled and observed magnetic field components

```
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data("mms1_fgm_b_gsm_srvy_l2_bvec")
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the modeled and observed magnetic field components

```
kamodo_object.plot('B_xMMS', 'B_x')
```

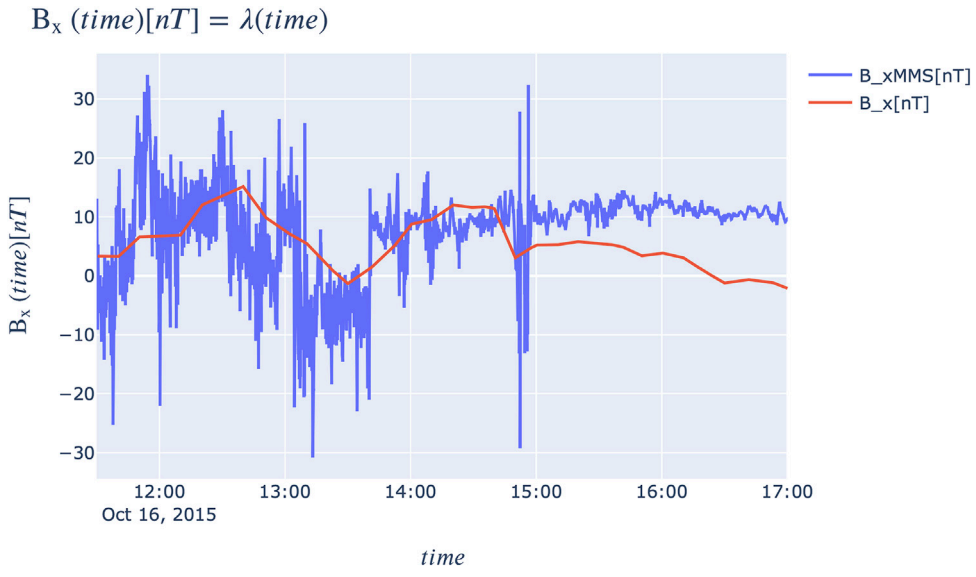


FIGURE 7
Time series plot of the observed B_x magnetic field component (blue) with the modeled one (red).

```
kamodo_object.plot('B_yMMS', 'B_y')
```

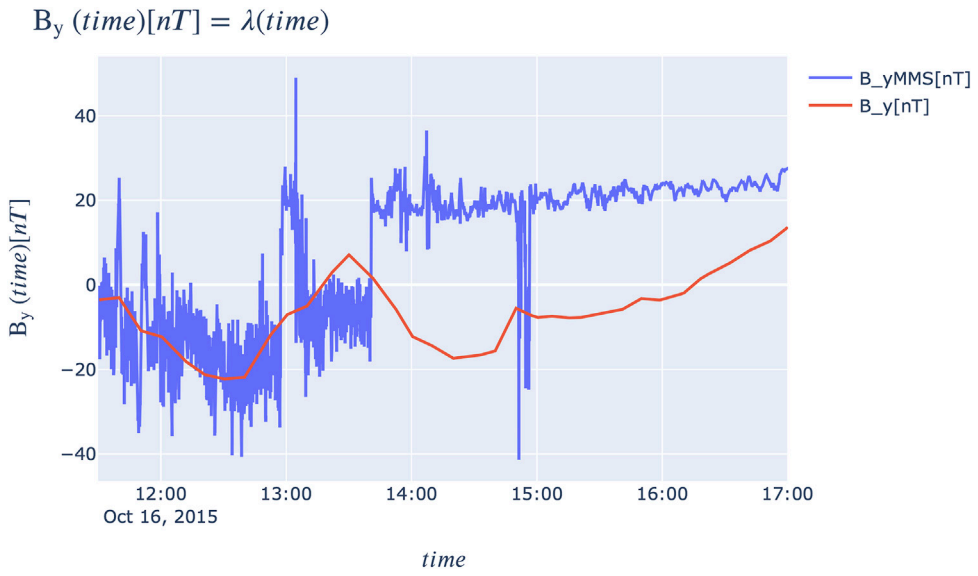


FIGURE 8
Time series plot of the observed B_y magnetic field component (blue) with the modeled one (red).

```
kamodo_object.plot('B_zMMS', 'B_z')
```

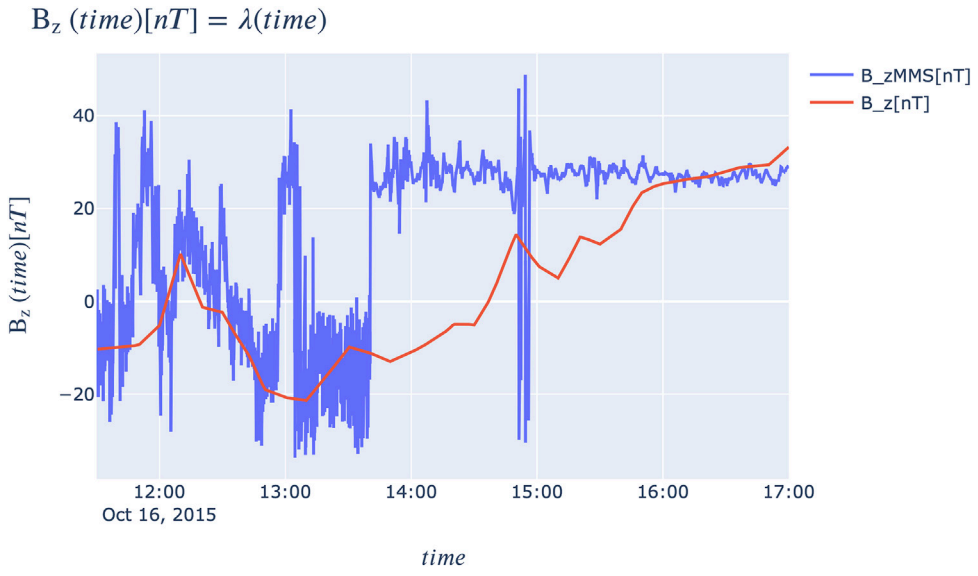


FIGURE 9
Time series plot of the observed B_z magnetic field component (blue) with the modeled one (red).

The three plots in Figure 7, Figure 8, and Figure 9 show how well the OpenGGCM model simulates our real-world data. Recall that our model was intentionally generated at a coarse resolution to save storage space, so the simulation is only a rough approximation.

The sudden increases in `B_x`, `B_y`, and `B_z` in blue indicate that the MMS spacecraft transitioned from the magnetosheath (high plasma density, lower magnetic field strength) into the magnetosphere (low density, high magnetic field strength). The 10-min samples from the OpenGGCM model roughly follow the trend of the magnetosheath magnetic field. After about 13:40, when the satellite is in the magnetosphere, the model seems to predict that the satellite is still in the magnetosheath until about 14:50. In Figure 8, we see that the satellite moves about 0.25 R_E/hour in X towards the Earth. Encountering the magnetosphere in the sampling of the model outputs translates to an under-estimation of the magnetopause position in X by about 0.3 R_E. Confirmation of such an estimate, however, would require considerably higher temporal and spatial model resolution.

3 Results

In this work, we have described how open science principles guided the process of making this first executable paper in Heliophysics. The effort required for this process was sustainable and demonstrated how scientists and software developers can work together to more easily produce research of this quality and reproducibility. In presenting this example of our workflow, we showed that, during the time range used, the magnetopause was farther from Earth than the Shue et al. model predicted. We also showed how well the OpenGGCM model simulated the magnetic field components measured by MMS. By building this work into an executable paper, we have enabled not only direct reproducibility, but also made it so that anyone can easily build upon these results. We encourage others in our community to use this work as a launching point for their own research.

The open-source Python packages in our workflow (pySPEDAS, SpacePy, PlasmaPy, PyTplot, and Kamodo) made the analysis simple and guarded against reinventing the wheel. They constitute a great improvement over the outdated paradigm of siloed scientists writing scripts that ultimately never leave their machines. By using these packages in this format, we have lowered the barriers for others to reproduce our work. All they must do is run the same functions from the same package versions with the same data. The executable version of this paper makes that task trivially easy (all it takes is a few button clicks) due to the containerization provided by the Deepnote platform. Moreover, because our paper unambiguously showed what code to run when—something a generalized description of our method would have lacked—we prevented others from accidentally coming up with "false cognate" solutions that would not directly replicate our work. We promote this novel paper format as a vital supplemental tool to improve the reproducibility of traditional papers.

4 Discussion

There is some poetry in a newer organization like PyHC using a new platform like Deepnote to make a new kind of paper. But there are already well-established alternatives to Deepnote that may be more appropriate for other kinds of work. We discuss these alternatives in "Supplementary Appendix B" to guide the reader in their choice of platform. We note it is possible to create and share executable papers with any of these platforms. The decision of which platform to use may ultimately come down to the level of effort the authors are willing to put in. For example, the platform's file size limits will be a crucial consideration for most projects since most platforms do not offer enough persistent storage space to hold hundreds of gigabytes. If the research requires a large amount of data, the authors might want to use a platform like JupyterLab where the executable paper could be shared from their own servers, rather than a platform like Deepnote which does not come with that much storage space. The trade-off would be the effort required to maintain the servers.

Now that we have published this executable paper, it is our responsibility to maintain it indefinitely. If anything breaks, we have to fix it and then update the DOI if necessary. We have no way of knowing how much work this will turn out to be. However, the likelihood of anything breaking was dramatically reduced by putting 100% of our research materials in our container. We predict the main thing to watch out for will be problems with future versions of the underlying containerization software. Still, this is a drawback to this paper format and a leading reason why it cannot yet replace traditional papers. Significant new infrastructure will be needed if the scientific community ever decides to dedicate themselves to executable papers. But the format offers too many reproducibility benefits to ignore it.

We encourage all researchers to make executable versions of their papers, but especially Heliophysics researchers who use Python. PyHC's goal, after all, is to bring such people into our community and unite them. Imagine with us: a beautiful world where all Python-using Heliophysicists use the same Python packages when doing the same work to answer the same questions. Furthermore, they share executable versions of their resulting research papers, giving them all a faster launching point to reproduce and use each other's work. At the same time, they are satisfying the necessary requirements to do responsible open science. If you like this vision, join us! [See our contact page (Polson and Barnum 2022)]. Our field would be more on the same page, making it easier to answer our questions about the Universe. And is that not the whole point of why we do this research in the first place?

4.1 Future work

At present, our OpenGGCM data is too coarse for deep scientific analysis. We would like to improve our model data to better capture nuances at the magnetopause boundary. For example, we could increase the 10-min cadence to, say, one second. Or, to avoid increasing the file sizes excessively, the model output conversion step to NetCDF files could include the possibility to extract a subset of the model data in space around the satellite location.

Such improvements could potentially make the data files too large to fit in this container despite the mentioned possibilities to decrease it, but if it does, we could start storing everything in a container that we host ourselves. Once we have this better model data, we will shift to focus on scientific applications of our workflow and compare more magnetosphere models to spacecraft data. We plan to publish an in-depth comparison that will examine more time ranges, spacecrafts, and models and delve deeper into the relevant science questions.

The approach used to create this paper was valuable and sustainable. We will do more work in the future to share this approach in an effort to convince our field to adopt our practices. This will include presenting talks about our approach at conferences and organizing PyHC activities around the matter (such as telecon presentations and hackathons during our biannual community meetings). We will also investigate matters of interoperability with other executable papers and how to describe our executable papers with metadata. Finally, we will offer direct support to any researchers who contact us for guidance.

Data availability statement

All data used in this work are stored inside the Docker container that houses this executable paper (https://doi.org/10.5281/zenodo.7412347). Further inquiries can be directed to the corresponding author.

Author contributions

All authors contributed to manuscript revision, read, and approved the submitted version. Individual contributions are described in Supplementary Appendix A.

Funding

This work is supported by NASA under award 80NSSC22K0326, managed by the Goddard Space Flight Center.

Acknowledgments

We gratefully acknowledge Darren De Zeeuw for valuable discussions and code contributions to the Kamodo project in support of this work. We gratefully acknowledge Bryan Harter for technical assistance and code contributions to the PyTplot project in support of this work. Simulation results have been provided by the Community Coordinated Modeling Center at Goddard Space Flight Center through their publicly available simulation services (https://ccmc.gsfc.nasa.gov). The OpenGGCM model was developed by J. Raeder at the University of California, Los Angeles, and T. Fuller-Rowell at NOAA/SEC. The RCM model was developed by R. K. Jaggi, R. A. Wolf, R. W. Spiro, and F. J. Rich at Rice University and at the Phillips Laboratory of the United States Air Force (Wolf et al., 2016). The Shue et al. model was developed by J.-H. Shue et al. at Nagoya University in Toyokawa, Japan. We thank the Magnetospheric Multiscale Mission team for the use of data from their instruments, the OMNI team for their assimilative solar wind datasets, and J. Gannon for USGS Dst data. We also thank contributors to the NumPy, SciPy, and matplotlib tools, and our SpacePy co-authors at Los Alamos National Laboratory. And lastly, we thank the Frontiers Design Team for the special treatment and extra effort needed to typeset this paper in its novel format.

Conflict of interest

RR was employed by ADNET Systems Inc.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fspas.2022.977781/full#supplementary-material

References

Akhlaghi, M. *reproducible-paper*. GitLab. Available at: https://gitlab.com/makhlaghi/reproducible-paper.

Alm, L., Argall, M. R., Torbert, R. B., Farrugia, C. J., Burch, J. L., Ergun, R. E., et al. (2017). *EDR signatures observed by MMS in the october 16 event presented in a 2D parametric space*. American Geophysical Union.

American Society for Cell Biology (2015). *Task force on scientific reproducibility calls for action, reform: Public concern about reproducibility is legitimate, scientists must respond effectively, say experts*. Science Daily. Available at: https://www.sciencedaily.com/releases/2015/07/150715090508.htm.

Astropy (2022). Units and quantities. Available at: https://docs.astropy.org/en/stable/units/index.html (Accessed April 8).

Bajpai, V., et al. (2017). Challenges with reproducibility." in *Proceedings of the reproducibility workshop*.

Baker, M. (2016a). 1, 500 scientists lift the lid on reproducibility. *Nature* 533, 452–454. doi:10.1038/533452a

Baker, M. (2016b). Stat-checking software stirs up psychology. *Nature* 540, 151–152. doi:10.1038/540151a

Binder (2022). *Binder documentation*. Binder User Guide. Available at: https://mybinder.readthedocs.io/en/latest/.

Burch, J. L., Torbert, R. B., Phan, T. D., Chen, L.-J., Moore, T. E., Ergun, R. E., et al. (2016). Electron-scale measurements of magnetic reconnection in space. *Science* 352 (6290), aaf2939. doi:10.1126/science.aaf2939

Cacho, J. R., and Taghva, K. (2020). The state of reproducible research in computer science." in 17th International Conference on Information Technology–New Generations (ITNG 2020). Springer.

CodaLab (2020). *Executable papers*. CodaLab Worksheets Documentation. Available at: https://codalab-worksheets.readthedocs.io/en/latest/Executable-Papers/.

Community Coordinated Modeling Center (2022). *OpenGGCM model information*. NASA GSFC. Available at: https://ccmc.gsfc.nasa.gov/models/modelinfo.php?model=OpenGGCM.

Deepnote (2022). *Data science notebook for teams*. Deepnote. Available at: https://deepnote.com.

Docker (2022). *What is a container?* Docker. Available at: https://www.docker.com/resources/what-container/. (Accessed April 8).

Egedal, J., Le, A., William, S., and Daughton, S. (2017). *The kinetic structure of the electron diffusion region observed by MMS during asymmetric reconnection*. AGU Fall Meeting, SM22B–01.

Elsevier *Executable Papers - improving the article format in computer science*. Elsevier News. Available at: https://www.journals.elsevier.com/the-journal-of-logic-and-algebraic-programming/news/introducing-executable-papers.

Fanelli, D. (2018). Opinion: Is science really facing a reproducibility crisis, and do we need it to? *Proc. Natl. Acad. Sci. U. S. A.* 11511, 2628–2631. doi:10.1073/pnas.1708272114

Foster (2022). *What is open science? Introduction*. FOSTER Open Science. Available at: https://www.fosteropenscience.eu/content/what-open-science-introduction.

Gentemann, C., Ivey, Y., Crawford, S., Baynes, K., Murphy, K., Ward, K., et al. (2021). *NASA open-source science initiative: Transform to open science (TOPS)*. *Zenodo*. October 29. doi:10.5281/zenodo.5621674

Google Research (2022a). Colaboratory frequently asked questions." Google Colab. Google. Available at: https://research.google.com/colaboratory/faq.html.

Google Research (2022b). Welcome to colaboratory. *Google Colab*. Google. Available at: https://colab.research.google.com/ (Accessed April 8).

Jupyter (2022). Jupyter NotebookProject Jupyter. Available at: https://jupyter.org/.

JupyterLab (2022). *JupyterLab documentation*. Jupyter. Available at: https://jupyterlab.readthedocs.io/en/stable/ (Accessed April 8).

Kamodo (2022a). *kamodo-core*." GitHub. Available at: https://github.com/EnsembleGovServices/kamodo-core (Accessed April 8).

Kamodo (2022b). GitHub. Available at: https://ccmc.gsfc.nasa.gov/Kamodo (Accessed April 8).

Kamodo (2022c). Kamodofication tutorial." CCMC. Available at: https://ccmc.gsfc.nasa.gov/Kamodo/notebooks/Kamodofying_Models/ (Accessed April 8).

Kamodo (2022d). *notebooks*. GitHub. Available at: https://github.com/nasa/Kamodo/tree/master/notebooks (Accessed April 8).

Kamodo (2022e). *model_wrapper.py*." GitHub. Available at: https://github.com/nasa/Kamodo/blob/b8823e5e37875f4b8aae473551b76a4d9c8aef7a/kamodo_ccmc/flythrough/model_wrapper.py#L50 (Accessed April 8).

Lasp (2022). FPI data product guide. Available at: https://lasp.colorado.edu/galaxy/pages/viewpage.action?pageId=37618954 (Accessed April 8).

Lasser, J. (2020). Creating an executable paper is a journey through Open Science. *Commun. Phys.* 31, 143. doi:10.1038/s42005-020-00403-4

Merriam-Webster (2022). *Magnetopause definition & meaning*. *Merriam-Webster*. Available at: https://www.merriam-webster.com/dictionary/magnetopause (Accessed April 8).

Morley, S. K., et al. (2011). "Spacepy - a Python-based library of tools for the space sciences," in *Proceedings of the 9th Python in science conference*. SciPy 2010). doi:10.25080/Majora-92bf1922-00c

NASA (2022). Magnetospheres." *NASA science*. NASA. Available at: https://science.nasa.gov/heliophysics/focus-areas/magnetosphere-ionosphere (Accessed April 8).

Nelson, N. C., Ichikawa, K., Chung, J., and Malik, M. M. (2021). Mapping the discursive dimensions of the reproducibility crisis: A mixed methods analysis. *Plos one* 167 (2021), e0254090. doi:10.1371/journal.pone.0254090

Niehof, J. T., et al. (2022). *SpacePy: Tools for space science applications*. doi:10.5281/zenodo.3252523

PlasmaPy (2022a). Plasma parameters. Available at: https://docs.plasmapy.org/en/stable/formulary/parameters.html (Accessed April 8).

PlasmaPy (2022b). *PlasmaPy*." GitHub. Available at: https://github.com/PlasmaPy/plasmapy (Accessed April 8, 2022).

Polson, S., and Barnum, J. (2022). Heliopython. Available at: https://heliopython.org/contact/ (Accessed April 8, 2022).

PyHC (2022a). Projects. Available at: https://heliopython.org/projects/ (Accessed April 8).

PyHC (2022b). Python in Heliophysics community gallery. Available at: https://heliopython.org/gallery/generated/gallery/index.html (Accessed April 8).

PyHC (2022c). Python in Heliophysics community (PyHC). Available at: https://heliopython.org (Accessed April 8).

PyHC. (2022d). Python in Heliophysics community (PyHC) standards. (Accessed April 8). doi:10.5281/zenodo.2529131

pySPEDAS (2022a). pySPEDAS. GitHub. Available at: https://github.com/spedas/pyspedas (Accessed April 8).

pySPEDAS (2022b). Load routines. Available at: https://pyspedas.readthedocs.io/en/latest/projects.html (Accessed April 8, 2022).

PyTplot (2022a). PyTplot. GitHubAvailable at: https://github.com/MAVENSDC/pytplot (Accessed April 8).

PyTplot (2022b). Tplot variables. Available at: https://pytplot.readthedocs.io/en/latest/tplot_variables.html (Accessed April 8).

Qin, Z., Denton, R. E., Tsyganenko, N. A., and Wolf, S. (2007). Solar wind parameters for magnetospheric magnetic field modeling. *Space weather.* 5, 11. doi:10.1029/2006SW000296

Ragan-Kelley, B., and Willing, C. (2018). "Binder 2.0-Reproducible, interactive, sharable environments for science at scale." in *Proceedings of the 17th Python in Science conference* Editors F. Akici, D. Lippa, D. Niederhut, and M. Pacer 113–120.

Reproducible Research How to make a paper reproducible? Available at: https://reproducibleresearch.net/how-to-make-a-paper-reproducible/.

Ringuette, R., De Zeeuw, D., Rastaetter, L., and Pembroke, A. (2022). Kamodo's model-agnostic satellite flythrough: Lowering the utilization barrier for Heliophysics model outputs. *Front. Astron. Space Sci.* 9, 1005977. doi:10.3389/fspas.2022.1005977

Shue, J.-H., Song, P., Russell, C. T., Steinberg, J. T., Chao, J. K., Zastenker, G., et al. (1998). Magnetopause location under extreme solar wind conditions. *J. Geophys. Res.* 103 (A8), 17691–17700. doi:10.1029/98ja01103

SpacePy GitHub. Available at: https://spacepy.github.io (Accessed April 8, 2022).

Vicente-Saez, R., and Martinez-Fuentes, C. (2018). Open science now: A systematic literature review for an integrated definition. *J. Bus. Res.* 88, 428–436. doi:10.1016/j.jbusres.2017.12.043

Ware, A., and Roberts, D. A. (2019). The Python in Heliophysics community. *Geophys. Res. Abstr.* 21.

Wolf, R. A., Spiro, R. W., Sazykin, S., Toffoletto, F. R., and Yang, J. (2016). *Forty-seven years of the Rice convection model*. doi:10.1002/9781119066880.ch17

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 14 April 2023
DOI 10.3389/fspas.2023.1023550

Check for updates

OPEN ACCESS

EDITED BY
K-Michael Aye,
Freie Universität Berlin, Germany

REVIEWED BY
Nickolay Ivchenko,
Royal Institute of Technology, Sweden
Pablo Gutiérrez-Marqués,
Max Planck Institute for Solar System Research, Germany

*CORRESPONDENCE
Carl Björn Kjellstrand,
bkjell@gmail.com

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 19 August 2022
ACCEPTED 28 February 2023
PUBLISHED 14 April 2023

CITATION
Kjellstrand CB and Williams BP (2023), SkyWinder: A Python package for flight control, telemetry, and image analysis. *Front. Astron. Space Sci.* 10:1023550. doi: 10.3389/fspas.2023.1023550

COPYRIGHT
© 2023 Kjellstrand and Williams. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

SkyWinder: A Python package for flight control, telemetry, and image analysis

Carl Björn Kjellstrand[1]* and Bifford P. Williams[2]

[1]School of Earth and Space Exploration, Arizona State University, Tempe, AZ, United States, [2]Global Atmospheric Technologies and Sciences, Boulder, CO, United States

SkyWinder is an open-source Python package useful for instrument control, telemetry, and image analysis. It is adapted from software that successfully managed flight control, telemetry, preliminary image analysis, and data visualization for a balloon-borne mission and it has broad uses for mid- and upper-atmosphere science instrumentation including aurora, cloud, and airglow imagers. SkyWinder will save future aeronomy experiments significant time and money, and lowers the barrier to entry in analyzing data hosted in public available repositories. Our software consists of two distinct parts: the flight and analysis modules. The SkyWinder flight package includes modular distributed flight control including telemetry and subsystem coordination for running mobile aeronomy experiments such as balloon-borne payloads, airplanes, sounding rockets, and suborbital reusable launch vehicles (sRLVs), as well as isolated semi-autonomous ground instruments. The SkyWinder analysis software provides functionality more broadly useful in neutral upper atmosphere dynamics, such as pointing reconstruction, image projection, preliminary image processing, and various image analysis techniques.

KEYWORDS

flight control, Python, software, PMC-turbo, telemetry, ballooning, image projection

1 Introduction

NASA Heliophysics research extends from the upper atmosphere to the sun. The Python in Heliophysics Community (PyHC) codebase is currently better developed for the solar, plasma, and magnetospheric physics than for the upper neutral atmosphere. To fill this need, we developed a Python Value Added Enhancement in coordination with the PyHC project. We named it "SkyWinder" to reference the aeronomy uses and reference the "sidewinder" snake species that have an unusual method of locomotion since snake motifs are commonly used in Python packages. The sidewinder snake's body motion is wavelike and has the phase and group velocity mostly perpendicular like GWs. Also, many types of GWs "wind" up into the sky. We refined and published existing heliophysics mission and analysis code from the successful Polar Mesospheric Cloud Turbulence (PMC-Turbo) experiment. This package is comprised of two distinct parts: the SkyWinder flight and SkyWinder analysis submodules.

The SkyWinder flight software provides modular flight control functionality useful for a variety of balloon-borne and other suborbital experiments, including telemetry and subsystem coordination. This includes functionality for running mobile aeronomy experiments such as balloon-borne payloads, airplanes, sounding rockets, and suborbital reusable launch vehicles (sRLVs). These features will also be useful for remote off-the-grid

ground stations. We designed the software specifically to work with balloon platforms and telemetry channels provided by the Columbia Science Balloon Facility (CSBF), which is a NASA facility that launches and manages large unmanned balloons. However, much of the functionality is broadly useful to semi-autonomous experiments. The SkyWinder software is a refinement of the flight and analysis software for the PMC-Turbo experiment. The software ran during a successful science flight July 2018 and a successful secondary ("piggyback") payload December-January 2019-2020 described in Kjellstrand et al. (2020) and Kjellstrand (2021).

The SkyWinder flight package provides redundant and distributed functionality for flight control, instrument control and coordination, and telemetry. SkyWinder allows for a framework for a distributed communication system in which several networked computers can each assume flight control duties in the event of any individual hardware failure. It communicates to and from the ground control station in a channel-agnostic manner that has been used with Tracking and Data Relay Satellite System (TDRSS) and Iridium satellite constellations. The package runs at a low level to provide commands to the hardware and exposes the handles of that interface on the experiment network. It selects, compresses, and packages data for downlink in such a way that the ground station can locate lost data packets, identify corrupted data, and estimate downlink times. It provides a convenient graphical user interface (GUI) for the ground station to monitor a remote experiment status, look at downlinked data, and send and track commands to the remote experiment.

The SkyWinder analysis software provides functionality more broadly useful in neutral upper atmosphere dynamics, such as pointing reconstruction, image projection, preliminary image processing, and various image analysis techniques. The SkyWinder analysis package includes procedures for flat-fielding and removing time-varying stray light from images, star identification and pointing reconstruction from the stars, projection from rectilinear projection to accurate on-the-sky geometry. Data visualization tools such as stitching image arrays together and movie-making (both projected and unaltered) are included. The SkyWinder analysis code will also include data visualization tools to combine imaging and lidar data. This has proven to be critical in analysis for PMC-Turbo and we anticipate many mobile aeronomy platforms will have multiple instruments. For example, the National Science Foundation (NSF) DeepWave mission described in Fritts et al. (2016) included a sodium lidar, Rayleigh lidar, advanced mesospheric temperature mapper (AMTM), two OH imagers, dropsondes, and many other supporting instruments. This was analyzed largely using commercial languages such as Interactive Data Language (IDL) and Matlab, making sharing code with outside data users more difficult.

2 Methods—Flight software

PMC-Turbo imaged dynamics located about 80 km above the Earth surface, while itself lofted aboard a balloon-borne payload at nearly 40 km altitude. It included 8 science instruments on board: seven optical cameras and a Rayleigh lidar. Each of these cameras collected data at a high rate—a burst of four 20 MB images every 2 s. Balloon-borne payloads include experimental risks that can result in the loss of collected data. PMC-Turbo was the first experiment of its type, so our predictions accounted for a large uncertainty in sky conditions and optimal viewing strategy. Therefore, the science objectives of the PMC-Turbo experiment imposed the following requirements on our software design.

1. The software needed to manage the data throughput from cameras to the hard drives.
2. The software needed to be robust to failure of individual computers, communication components, or other subsystems.
3. We needed to have the ability to command the payload from the ground, monitor each subsystem through aggregated housekeeping, and sample imager data.

Our requirement for data handling drove our selection of computer hardware in turn since the large data volume generated by our cameras required a server-grade motherboard to store to disk. Since we required powerful computers for data storage, we found no downside to including flight control capabilities in each individual computer, as opposed to the more common design of using a dedicated flight control computer. This allowed us to greatly improve the resilience of our system to single-point failures since any instrument computer could assume responsibility of the communication between local systems and our telemetry connections. To reduce the impact of an isolated disk failure, each computer ran software to grab data from the connected instrument and distribute the images to the four hard drives connected to the computer.

To reduce the risk inherent in balloon-borne platforms and to allow for adaption of our observation strategies in real time, we developed a robust telemetry system. We continuously monitored our instrument statuses and we downlinked as much data as possible to retain some scientifically useful data in the event our payload could not be recovered. We used the telemetry channels made available by NASA to send compressed and packetized science and housekeeping data to the ground. To monitor the data received on the ground, we developed software to track data sent down and send commands to the payload.

While we wanted to monitor and command the payload as much as possible, we needed to account for expected communication outages and the resulting absence of commands. We implemented these procedures to ensure that our camera systems captured and stored images to disk even if we lost contact with them. Our cameras had no real-time control requirements, although we did have the ability to control them at times we could communicate with the experiment from the ground. The operating system uses a Linux program (supervisor) to automatically start the data acquisition and communication software in a useful state and when the instrument module receives power, the motherboard automatically boots the operating system. The data acquisition software includes a watchdog that restarts the operating system after 10 min without new images. The default camera settings included an auto-exposure algorithm to maintain useful exposure times and a nominal focus sufficient to capture good data.

We wrote the bulk of our software in the Python programming language. This language allows for speedy development and is widespread in the scientific programming community. We developed the software on the Linux operating system since it

is commonly used for scientific instrument applications and our developers were experienced with it. We used industry standard continuous integration tools to ensure reliable and speedy software development.

2.1 Network

The distributed flight control structure of the SkyWinder flight control software requires consistent communication between the individual computers capturing experiment data. We use the Python Remote Object (Pyro) package to abstract much of this communication. This package allows programs running on one computer to seamlessly call classes and functions running on another. Pyro enables each process to use a Uniform Resource Identifier (URI) on the local network. After registering a URI, one process (such as the `Controller`, described in this section) can call another (such as the `Pipeline`) as a Python object. Not only does this facilitate communication between processes running on one machine, but our network structure allows each registered process to be visible and accessible to the entire network. **Section 2.2**, **Section 2.3**, and **Section 2.4** describe the programs running on our computers and which were accessible to each other on our local network.

Figure 1 shows an overview of the final PMC-Turbo network as an example for how one can structure a distributed flight control hardware network to work with SkyWinder. Seven pressure vessels contain primary science instruments and the associated computers. The two RS-232 Ethernet converters, two Ethernet switches, and a direct Ethernet connection to Iridium Pilot ensure that no single point of failure removes communication to all the channels. Our telemetry box includes two Ethernet switches connected to each other. Three or four of the pressure vessels, one of the two power boxes, and one RS232-Ethernet converter connect to each Ethernet switch. This ensures that if either Ethernet switch failed or one SIP connection failed, we would not lose communication to the entire payload. We distribute the computers interfacing with our science instruments across the two Ethernet switches to avoid losing communication with all instruments if either Ethernet switch failed. The expanded view labelled "Pressure Vessel Interior" in **Figure 1** shows the primary software processes of our software architecture described in this section. While PMC-Turbo included seven computers running the precursor to the SkyWinder software, we have also used the software to run a single instrument (during the piggyback flight). As far as we are aware, the upper limit for the number of instances of SkyWinder running on a local network is set by the hardware of that network and the computational resources available, rather than the SkyWinder software.

2.2 Communicator and leader assignment

The `Communicator` class coordinates between the instrument computers contained within the pressure vessels and communicates with ground software. It aggregates housekeeping data, provides status reports, and receives, relays, and responds to commands. Each camera computer runs a `Communicator` instance. While all `Communicator` instances listen for commands from the ground, one `Communicator` is designated the "leader" and it assumes the bulk of the communication duties. One specific camera computer is designated leader by default, but the ground operators can change which camera computer assumes leader duties remotely since each `Communicator` instance includes the leader functionality.

Upon booting up, each `Communicator` reads a configuration file to set operational parameters. These parameters include the initial assignment of leader duties to one `Communicator`, the instrument hardware associated with each computer, other instrument statuses monitored by individual communications (such as power system metrics and exterior temperature monitors), and the peer polling order prescribing the order and frequency subsystems are polled. Each `Communicator` also instantiates SkyWinder `HirateDownlink`, `LowrateDownlink`, and `Uplink` objects. These objects include the IP and ports used for the experiment's telemetry connections. The `LowrateDownlink` is responsible for the low-bandwidth connections made available by the ballooning hardware and only sends messages 255 bytes and less. The HirateDownlink is used for all other downlinks. It packetizes data and sends an appropriate number of packets given the configured bandwidth when prompted. The Uplink receives packets sent from the ground. All of the links use Python `socket` objects for communication and use UDP packets due to a potentially unstable connection to ground.

The configuration file should be tailored to each experiment's needs. Experiments will use difference telemetry links, numbers of computers, and include distinct hardware to monitor. Operational parameters can be changed during operation. For example, if a group lost contact over the Iridium Pilot channel, they may want to decrease the downlink bandwidth on that link to avoid sending data on that link that they will never receive. Alternately, they may want to change the peer polling order if an interesting feature were observed in one specific instrument but not others.

Listing 1 shows a simplified version of the `main_loop` method of the `Communicator` class. While the `main_loop` includes other functionality in the PMC-Turbo, including monitoring specific hardware (e.g., charge controllers or batteries) and weaving status updates into several downlink streams, at it's core the `Communicator` looks for data uplinked to the experiment network, responds to them, and sends data on the available downlinks. Recall that every computer on the network runs a `Communicator` instance, but only one `Communicator` instance is assigned as leader.

2.2.1 Receiving and processing data from uplinks

The `get_and_process_sip_bytes` looks for data on every `Uplink` and executes the packets. The CSBF Support Instrumentation Package (SIP) sends three types of message to the network: science data request messages at a regular cadence, science command messages relayed from the ground, and location information (e.g., GPS data).

The PMC-Turbo experiment responded to regular science data requests by aggregating a short (255 byte) status and sending it on the `LowrateDownlink`. SkyWinder includes this procedure in the `Communicator` `get_next_status_summary` method. The

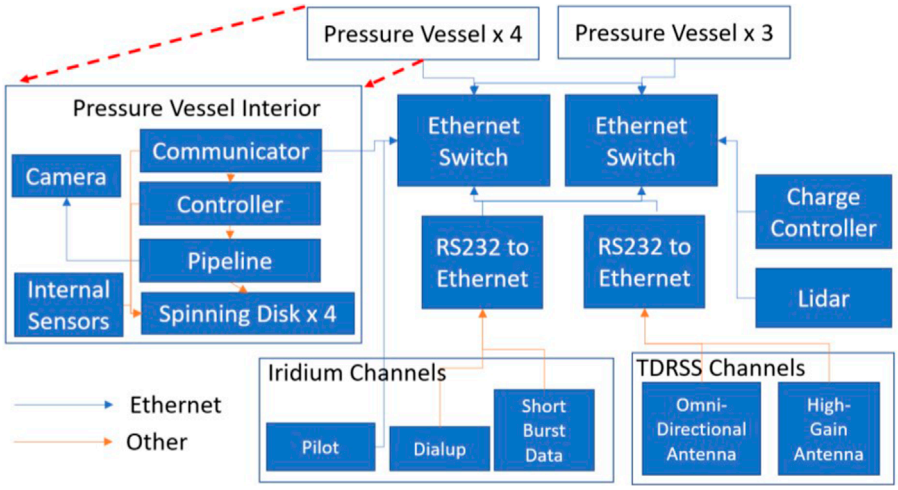


FIGURE 1
An overview of the PMC-Turbo network. The instrument suites within the pressure vessels, charge controllers, lidar, and telemetry antenna primary communicate over a local ethernet network. The expanded box labelled "Pressure Vessel Interior" shows the communication links between the subsystems associated with each computer. Figure originally published in Kjellstrand et al. (2020).

```
def main_loop(self):
    while not self.end_loop:
        self.get_and_process_sip_bytes()
        if self.leader:
            self.send_data_on_downlinks()
        time.sleep(self.loop_interval)
```

Listing 1. Simplified version of the main communicator loop.

leader kept track of which computer to query, and each computer pulled metrics from saved logs and packaged them when prompted. While this code is included in SkyWinder, the metrics for each experiment will be unique to that experiment's hardware, so each science group will need to craft such aggregation to their own needs.

SkyWinder includes functionality to package, interpret, and execute command messages in the `Command` object. Commands include changing downlink attributes such as downlink bandwidth and peer polling order, changing hardware settings such as exposure time and aperture width, and requests for a specific file or files or instructions about how to prepare files for downlink generally (level of downsampling, whether to downlink a full image or selection, etc.).

Finally, any other data that is received on the uplink (most likely position data) is logged. One notable feature of the uplink monitoring functionality of the `Communicator` is that every computer on the network receives every uplinked message. However, generally only the assigned leader responds to commands and science data requests. The exceptions are when the command is flagged as a "super command". This behavior is intended to allow for switching which `Communicator` assumes leader duties, even (and especially) when the previous leader has ceased to communicate successfully.

```
def send_data_on_downlinks(self):
    for link in list(self.downlinks.values()):
        if link.has_bandwidth():
            peer_id = self.peer_polling_order[self.peer_polling_order_idx]
            logger.debug('getting next data from camera %d' % peer_id)
            next_data = None
            active_peer = self.peers[peer_id]
            if self.check_peer_connection(active_peer):
                next_data = active_peer.get_next_data()  # pyro call
            if not next_data:
                logger.debug('no data was obtained.')
            else:
                link.put_data_into_queue(next_data, self.file_id)
                self.file_id += 1
            self.peer_polling_order_idx = (self.peer_polling_order_idx + 1)
                                          % len(self.
peer_polling_order)
        else:
            if link.enabled:
                link.send_data()
```

Listing 2. Streamlined data sending method with error catching removed for clarity.

2.2.2 Sending data on downlinks

Listing 2 shows a simplified version of the sending process. The leader goes through each of its downlinks and checks whether it has bandwidth. If the `HirateDownlink` currently has packets queued, the leader simply tells it to send some more packets. The number of packets is calculated by the `HirateDownlink`, which keeps track of the assigned downlink speed, time it sent data, and amount of sent data. If the `HirateDownlink` does not have packets to send, the leader pings the next peer in the list `peer_polling_order`. The `Communicator` attribute `peers` is a dictionary of the peer Pyro URIs. The leader finds the URI using the next entry in `peer_polling_order` as the key and tries pinging the peer to ensure it is responsive. If it is, the leader requests data from the peer and puts that data into the `HirateDownlink` data queue. Since each experiment ground has a different system for saving data from their experiment, each experiment will need

a custom `Communicator` method for retrieving data. This may be as simple as finding the newest file in a specific directory. The PMC-Turbo `Communicator` calls the `Controller`, which compresses the raw image data with configurable settings and keeps track of a queue of files to downlink. This is typically the latest file, but can include a series of requested images.

2.2.3 Packet structure

The `HirateDownlink` packetized data passed to it in chunks of 1,000 bytes. We developed a packet format to maximize the bandwidth available to us on multiple communication channels with distinct communication protocols and track missing or incomplete files with the ground-side software.

Our custom packet format identifies itself with respect to all data we have downlinked. While the SkyWinder software can use many different telemetry channels, the PMC-Turbo experiment received data from Iridium Pilot at a fixed IP address on Columbia University campus in Manhattan, data from the LOS link at our launch location in Sweden, and the TDRSS data at computers in Texas. During flight, we could access all these geographically separated computers remotely, but we needed metadata generated from the communicator to identify the files. For example, the Nth file could be sent over the Iridium Pilot link to New York while the N+1st file could have arrived in Texas over the TDRSS link.

We needed our software to account for missing packets and incomplete files. We expected regular and sometimes a significant percentage, of missing packets and early on we had decided to use UDP packets. Unlike TCP packets, UDP packets include no handshaking to determine whether data has arrived at the destination, so we had no built-in method of knowing whether data was missing. However, the absence of arrival verification meant UDP packets made for more robust communications. In the event of low quality connections, we prioritized maximizing the chance that packets would get through, rather than jeopardize the communication by requiring a TCP response. Instead of using built-in packet verification, when the communicator prepares data for downlink, it includes metadata with the packetized chunks indicating the communication channel and the packet number, along with the total packet numbers. This allows the ground-side software to track missing packets and aggregate packetized data back into complete files. We also include a checksum to verify that individual bytes for packets had not been corrupted or lost (for example, one packet losing its "tail" and another losing its "head").

When a `Communicator` retrieves a file using the `get_next_data` method, it requests data from the `Controller` (described in the next section) running on the same machine. While each `Controller` is callable to each `Communicator` on the local network *via* Pyro, a `Communicator` only interfaces with the `Controller` running on the same computer in the functions we have implemented so far. Unlike the leader assignment of the `Communicators`, `Controllers` have identical responsibilities. After the `Controller` selects and prepares a file (typically with compression), it wraps it using one of the classes inheriting from the `FileBase` object in the `file_format_classes` module. While each of the inheritors are somewhat different, they essentially all wrap the file in a Python class that includes useful metadata. This object is then converted to a buffer of bytes and passed by the `Communication` to one of its `HirateDownlink` instances. This buffer is split into chunks of 1,000 bytes, which are then wrapped using the `FilePacket` objects from the `packet_classes` module and sent to the `HirateDownlink` `socket` to be sent as UDP packets.

On the ground, the packets can arrive wrapped in different states, depending on the link. The ground software (described in **Section 2.6**) processes and metadata from the telemetry link, tracks and reassembles the FilePacket objects, and writes both the FileBase object and the payload file in that object to disk.

2.3 Controller

The `Controller` class interprets commands from the communicator and relays them to our hardware interface (the pipeline, described in **Section 2.4**) in order to control camera settings and retrieves images from the data disks. It also handles grabbing files from the hard drives and compressing these files to prepare them for downlinking. While simple experiments may be able to skip using the `Controller`, the image preparation, file compression, and command interpretation functionality of the `Controller` will be useful to many groups. However, the SkyWinder software can work without using a `Controller`, and some science groups may chose to use only the `Communicator`.

As one monitors the experiment from the ground, they often want to review specific images and files. The `Controller` can also prepare specific images upon request or raw files (such as a raw housekeeping log or uncompressed image). Since the PMC-Turbo experiment dealt primarily with images, we also included the functionality of requesting specific regions of the image. A ground-based user could request the default downlinked image to be an arbitrary pixel area (such as 512×512) selected from any location in the image at any possible downsample resolution (that is a resolution reachable *via* integer division of the original resolution of the image).

The `Controller` can also interpret and run multi-step processes, such as a focus sweep where the camera cycles through a range of focus steps, capturing an image at each step, and downlinks the images for ground review. The `Controller` interprets the focus sweep command from the communicator and automatically prepares and relays commands to adjust focus step settings and retrieve images taken at each focus step for evaluation on the ground.

While the `Controller` primarily acts as an interface between the communicator and the pipeline, it has some automated duties as well. On a periodic interval, it checks for completed commands, executes steps in multi-step commands (such as focus sweeps or downlinking a series of images), and updates the merged index of images and commands. It can also run a custom configurable auto-exposure algorithm if required.

2.4 Pipeline

We anticipate that most science groups using the SkyWinder flight control software will need to write their own hardware-software interface due to unique science instruments. However,

we have included the PMC-Turbo pipeline in SkyWinder as an example and for our own use. It sends commands to our instrument hardware, arms the camera to capture an image, receives the image data, and writes the image to one of the four data disks. The `Pipeline` classes manages these process

The primary responsibility of the `Pipeline` is moving data from the camera buffers to the hard drives. Upon initialization, the `Pipeline` creates a configurable number of raw image buffers and puts them into an input queue. The `Pipeline` also instantiates an output queue, where the buffers will be moved when filled with image data. The `Pipeline` starts one `AcquireImageProcess` class and several `WriteImageProcess` classes which watch the input and output queues instantiated by the `Pipeline`. The `AcquireImageProcess` passes the buffers of the input queue to camera *via* the SDK. It periodically checks whether these buffers have been filled and it moves them to the output queue once they have been filled. The `WriteImageProcess` instances periodically check the output queue and write these buffers to disk. The `WriteImageProcess` then move the now empty buffer back to the input queue. The `Pipeline` performs lossless compression on the images when it stores them on the spinning disks using the Blosc library.

Due to the structure of the Vimba software development kit (SDK) released for our cameras, the `Pipeline` also acts as our interface with our specific camera hardware. The `Pipeline` acts as our software-hardware interface with the cameras *via* the SDK. Allied Vision publishes the Vimba SDK with functionality including image capture, setting parameters for the camera (exposure time) and lens (aperture, focus), and capturing bursts of images. While the SDK is written in C++, our own software includes a Cython wrapper to allow our Python code to interface with the SDK.

The `Pipeline` instantiates a command queue and acts as the final destination for commands received by the communicator and interpreted by the `Controller`. When a command is put into that queue by the `Controller`, the `Pipeline` process will execute the command by calling the SDK function that directly commands the camera.

2.5 Commanding

SkyWinder includes software to command the payload from the ground in nearly real time. The `CommandSender` class in the `commanding` module provides the central commanding functionality. It sends commands to the appropriate `socket` or `Serial` instance and translates human-readable commands into a compressed telemetry-hardware-appropriate format.

Since the uplink bandwidth is very low, the SkyWinder commanding packages commands as few bytes as possible—typically less than twenty. The format for each command is defined in the `command_table` module. The `CommandManager` object reads these definitions and constructs `Command` objects from them. When called with the appropriate arguments, each `Command` object will return a byte-string readable by the `Communicator`. Each `Command` object also describes the arguments and types it expects for this call using the `Command.__doc__` method. The user-interface

```
payload = my_command_sender.example_command(args)
my_command_sender.send(payload, destination=computer_number)
```

Listing 3. Pseudo-code showing a command sending example.

`CommandSender` adds each `Command` as an attribute for convenience. The user can then send a `Command` simply with as shown in the pseudo-code in **Listing 3**.

We designed this software to operate with the CSBF telemtery hardware, but we believe our software is general enough to easily adapt to the telemetry hardware science groups will use. The `CommandSender` packages the command bytes appropriately for the various telemetry uplink channels (wrapping the bytes in a `GSECommandPacket` or `CommandPacket`) and logs the commands sent. The `Communicator` also keeps track of received commands and will send the logs to the ground upon request. With PMC-Turbo, we also included the latest executed command in our short status summary.

2.6 Ground receiving

We developed custom ground software to display status updates and files received from our payload. The software reassembles the packetized files prepared by the communicator and displays included metadata. The telemetry software on both the ground and payload side worked with all of our telemetry channels. The ground side software also communicated with the NASA SIP module for an additional channel for basic housekeeping metrics.

The core part of the SkyWinder ground software is the `GSEReceiver`. `GSEReceiver` receives data from a `socket` or `Serial` instance. It sorts through the raw data buffer to identify packets and their origin. It uses packet metadata to reassemble files from the packet payload and verify that no data has been corrupted in the downlinking process. The `GSEReceiver` logs the received packets, their origin, writes status packets to disk, and writes the reassembled `FileBase` inheriting objects to disk for review of the attached metadata. It also writes the payload of the `FileBase` objects—the original file—for quick survey of the downlinked files.

We wrote quality of life software that interfaces with the `GSEReceiver`. The `lowrate_monitor_gui` module launches a GUI that displays the status updates from all computers received by the `GSEReceiver` in an organized table that updates every few seconds. The `gse_receiver_monitor` module shows individual packets as the `GSEReceiver` identifies them and shows how many packets out of the total have arrived of each new file that the `GSEReceiver` identifies.

The `pmc_viewer` module displays JPEGs saved by the `GSEReceiver` along with image metadata. We built this software specifically for PMC-Turbo's image viewing needs. However, experiments that want to view image data as it arrives on the telemetry downlink will likely find this software valuable. **Figure 2** displays a screenshot of the `pmc_viewer` showing an image including interesting science features. This screenshot was taken during the 2018 PMC-Turbo flight. The dynamic range has been adjusted using the GUI software to emphasize the PMCs, visible as

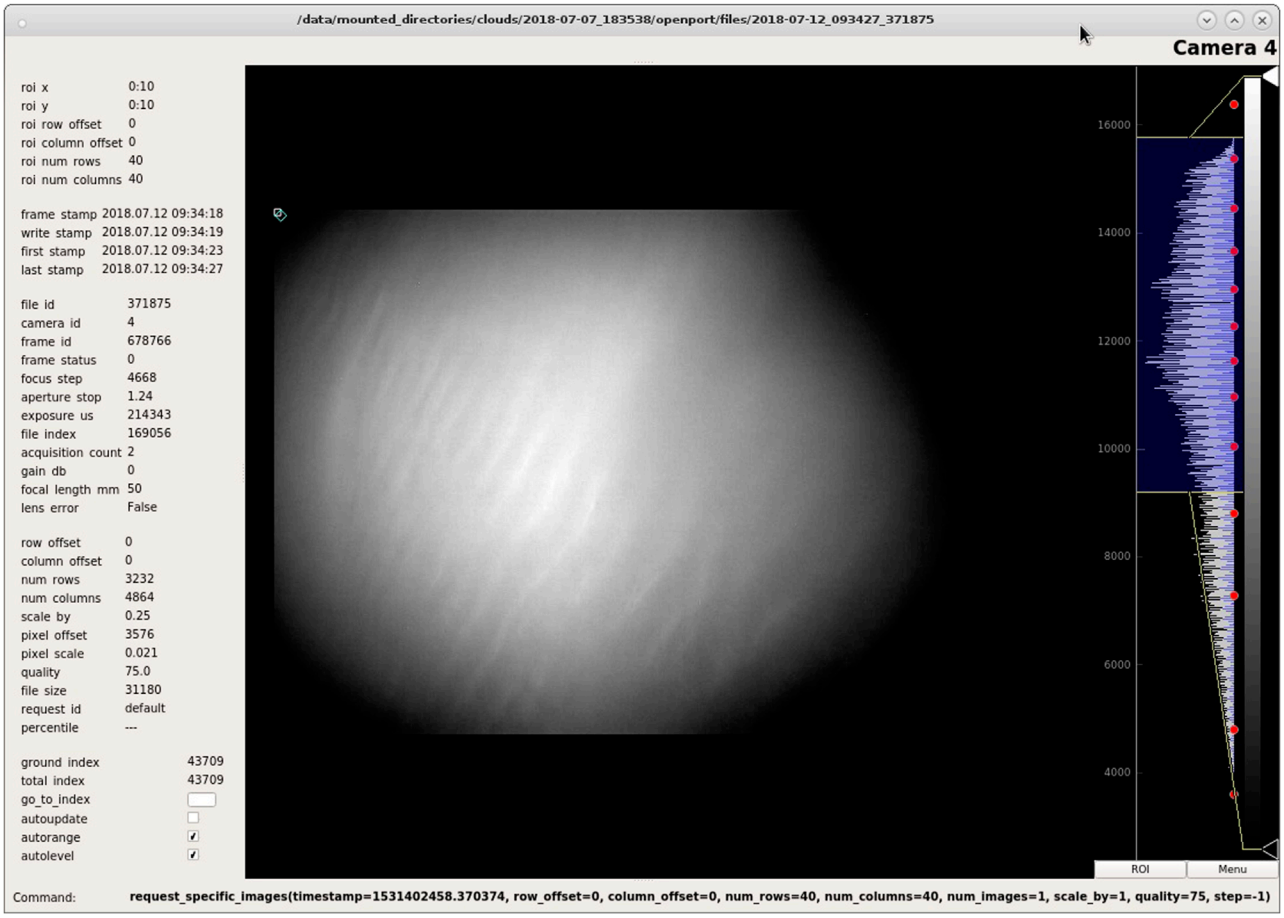


FIGURE 2
The pmc_viewer software during the 2018 PMC-Turbo flight. The diagonal striations in the image are Kelvin-Helmholtz Instabilities traced by PMCs. Figure originally published in Kjellstrand et al. (2020).

bright stripes oriented diagonally within the image. The left column displays metadata, including exposure time, aperture stop, file id, and focus step. The metadata also informs the viewer of the level of downsampling of the image and the time stamp of the image. In the upper left corner of the image within the GUI is a selection tool, which can be dragged and re-sized. The text below the image changes as the user adjusts the selection and gives a command argument to request the selected region of the image. To the right a histogram of the image allows a viewer to change the display grey scale range live and see image statistics.

3 Methods—Analysis software

3.1 Analysis software structure

The LDB EBEX Analysis Pipeline (LEAP) [described in Chapman (2015)] successfully analyzed PMC images from the balloon-borne E and B experiment (EBEX). While the primary science of the experiment was to detect signatures in the cosmic microwave background, its star cameras serendipitously recorded images of PMCs during its 2012-2013 Antarctic flight. These images provided the motivation and basic design for the PMC-Turbo experiment. We used the same analysis structure for our analysis software. The software is structured such that "apps" generate specific data products while importing generalized procedures from shared libraries of code. This structure is standard for the development of software projects, which makes publishing and maintaining this code as an open-source package straightforward. The consistent structure helps when a user wants to generate data products while using previous products as a reference.

The libraries include functions we frequently use in analysis programs. The intention behind this structure is that these tools will continue to be refined and developed, rather than duplicated during individual analysis projects. Libraries include broadly useful functions performing image processing and projection, movie making, power spectrum calculation, and data handling and manipulation.

While the libraries included code designed to perform frequently used tasks, each app generated a specific data product. For example, I wrote separate apps to generate image series, power spectra statistics, and Lomb-Scargle periodograms. All these example apps drew from the shared image processing and projection libraries.

Each app inherited a framework and associated functionalities from a parent class. This class defined a common template imposing requirements and providing features for each app. This included receiving input parameters in a dedicated settings file. For example, the image series app settings included timestamps for images, which camera data to include, which flat-field procedures to use, which region of the sky to show, which color scale to use, whether to show timestamps, and whether to output a series of individual images or a movie file. Another important feature of the app parent class was generation of an output file. This file included the data product generated by the app, logs from running the app, and the settings used.

Each app includes "default_settings" file included in its directory. This file includes all the configurable settings for any analysis procedure. The user writes a separate "custom_settings" to input any settings other than the default. For example, an image-generating app takes timestamps, resolution, and selection criteria from the settings files.

Along with the specialized data product (images, movies, etc.), each app writes the default_settings and custom_settings files to disk. It also writes a text file including the settings used and a log file from the run.

3.2 Data reduction

We implemented image correction techniques to examine our data, and these became the first entries to our software libraries described in **Section 3.1**. Furthermore, our cameras captured a rectilinear projection of the sky due to their geometry, so we needed to convert this projection back to the physical plane of the PMCs in order to correctly analyze scales and morphology of observed dynamics. These data reduction procedures are standard for imaging of the sky using standard lenses, and we anticipate that they will be broadly useful to many atmospheric science groups.

3.2.1 Image calibration and stray light removal

We calibrate a raw image R with a flat-field image F, dark image D, and simulated sky brightness image S. In this process we use the following calibration data.

- A dark image D defined before flight to account for the dark current in the CCD.
- A simulation of sky brightness S accounting for brightness due to air mass and the scattering angle between each pixel's view and the sun.The model for the sky brightness is geometric based on the position of the sun and the viewing angle of the camera, the derivation of which is outlined in Geach (2020). To quote the result derived there, the sky brightness $S_{unscaled}$ is

$$S_{unscaled}(\theta,\phi,\theta_S,\phi_S) = \frac{1}{\cos\phi}1 + (\cos(\theta-\theta_S)\sin\phi\sin\phi_S + \cos\phi\cos\phi_S)^2 \quad (1)$$

where θ and ϕ are the azimuth angle and off-zenith angle of a pixel and θ_S and ϕ_S are the azimuth angle and off-zenith angle of the sun. As discussed in the next section, the location of each pixel is determined with imaged star fields, and the GPS included on the payload allowed us to know the position of the sun in the sky when our cameras recorded an image.

Since the signal of a raw image is dominated by sky brightness, to account for pixel sensitivity and exposure time $S_{unscaled}$ was scaled to S as

$$S = \frac{< R-D >}{< S_{unscaled} >} S_{unscaled} \quad (2)$$

where brackets note scalar mean values.

- A flat field F defined before flight from twilight images with varying brightnesses to account for vignetting, illumination differences due to the lens hardware, and variations in pixel sensitivity. Since we did not have access to a uniformly illuminated flat-field, this flat-field was corrected not by removing the dark image D, but by removing sky brightness S. In an ideal case, the flat field image F is found by imaging a uniformly illuminated field and normalizing that image after subtracting D. However, F can be found with any known illumination field instead if a uniform field is not available. We used the dusk sky for this purpose since we needed to use it for our pre-flight pointing verification. At dusk, the sun is below the horizon, but light still reaches the Earth's surface because air masses at higher altitudes have a further horizon than the ground and reflect incident sunlight. The integrated brightness scales with atmospheric density. Since the density falls off exponentially with altitude, the amount of light scattered to the ground can be well-approximated as proportional to the lowest altitude illuminated by the sun. When we know pointing and sun coordinates, the lowest illuminated altitude can be calculated for each pixel. Therefore, multiplying the sky brightness model S by a value proportional to the atmospheric density at the lowest illuminated altitude for each pixel provides a known illumination field which we use to calculate F for each camera. This procedure was repeated for several hundred images taken at different times during dusk and the results were co-added and normalized to find the final flat field F. This procedure is described in detail in Geach (2020).

During out image calibration process, we first apply a flat field correction using the standard method (using the corrected flat field described above):

$$I_{corrected} = < F > \frac{R-D}{F} \quad (3)$$

where $<F>$ is the scalar mean of F to relevel the image. We then remove the sky background signal using S and normalize by exposure time according to

$$I_{calibrated} = \frac{1}{t_{exposure}}\left(I_{corrected} - S\frac{< I_{corrected} >}{< S >}\right) \quad (4)$$

where $t_{exposure}$ is the exposure time and brackets denote scalar means.

During the PMC-Turbo flight, insufficient baffling between the cameras and balloon allowed stray light varying with timescales of minutes to hours into our cameras' fields of view as the relative position of the sun and balloon-borne platform changed. To correct for this stray light contamination, we subtract a moving average of the calibrated images from our final image. This step may not

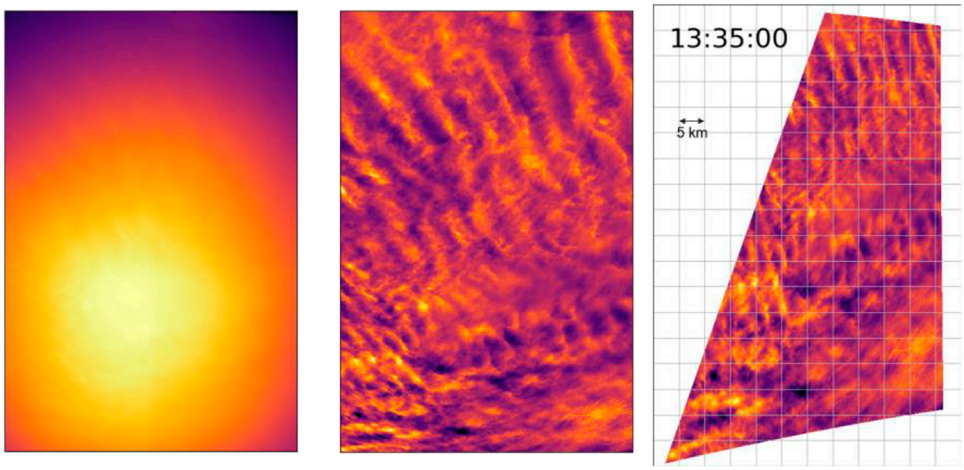


FIGURE 3
From left to right: raw false-colored image of one of the wide-field cameras containing PMCs; the same image after flat-fielding; and the same image after flat-fielding and projection based on the pointing solution. Up orientation in the first two images corresponds to the zenith, and up orientation in the projected image corresponds to the longitude of the sun. We projected the image looking from below. Figure originally published in Kjellstrand et al. (2020).

be necessary for all experiments, but stray light contamination is a common challenge in ballooning, so we have included these procedures in the SkyWinder image correction procedures. We subtract a moving average image I_{stray} accounting for stray light from our final image I_{final}:

$$I_{final} = I_{calibrated} - I_{stray} \quad (5)$$

We construct I_{stray} by taking a pixelwise average of calibrated images in a window of time centered around the image to be corrected (typically 10 min). The *ith*, *jth* pixel value in I_{stray} is the mean value of the *ith*, *jth* pixel in each calibrated image within that window. To minimize loading time and memory usage during this process, we save calculated I_{stray} instances at a 1-min cadence throughout flight and use these saved images to construct the I_{stray} for any specific time rather than finding it from scratch each time we apply the image processing procedure. This allows us to load about 10 files per stray-light-removal application rather than 1,000. The pre-constructed I_{stray} files reduce the time resolution available for our window, so we use interpolation to construct the subtracted I_{stray} from them.

Figure 3 shows a raw image to the left and an image corrected using the steps above in the center. Of these corrections, the removal of the moving average is by far the most significant and simply removing the average can deliver sufficient signal-to-noise ratio to quickly survey data.

3.2.2 Pointing

Balloon-borne experiments must record data at some off-zenith angle to avoid the highly-reflective balloon. Pointing off-zenith with a rectangular CCD and a standard lens introduces rectilinear distortion. While some lenses (such as fisheye lenses) will require different projection techniques, many images can use the projection techniques included with SkyWinder.

We remove rectilinear distortion by finding the pointing of the images using background star fields. For each camera we found the sky location of an evenly-spaced grid of pixels (spaced every quarter of the CCD, including the edges). We interpolated over this grid to map each pixel in the CCD to the proper location on the sky.

We found that we could use the tracking software package Astrometry to get the right-ascension and declination (ra/dec) of an evenly spaced grid of pixel locations. From right ascension and declination we converted to azimuth and elevation of the grid using GPS timestamps and locations from the flight. However, this software required the pixel locations of stars. Since the PMC Turbo pressure vessel design was based on the star camera design from the EBEX mission, we used a procedure based on the star finding procedure used in EBEX described in depth in Chapman (2015) and used the image manipulation python package CV2 python, which wraps the well-known library OpenCV. The star identification algorithm operates as follows.

1. We mask previously identified "hot pixels".
2. We convolve the image with a Gaussian kernel of configurable size and sigma in order to smooth the image.
3. We divide the image into sections - or "cells" - of configurable size.
4. In each cell we look for values above a configurable threshold level.
5. We compare the pixel brightness values at above-threshold levels to the pixel brightness values in a dilated version of the same image. If the values are the same we have found a local maximum.
6. We select a configurable number (typically 1) of local maxima for each cell and return their coordinates.
7. As an optional additional step, we fit 2-dimensional Gaussian distributions to each blob. We can filter by the sigma of the Gaussian fit to filter out unphysical results caused by CCD artifacts that have slipped through our other filters.

Figure 4A shows stars identified following these steps. **Figure 4B** shows the location of these stars in the corrected image before projection.

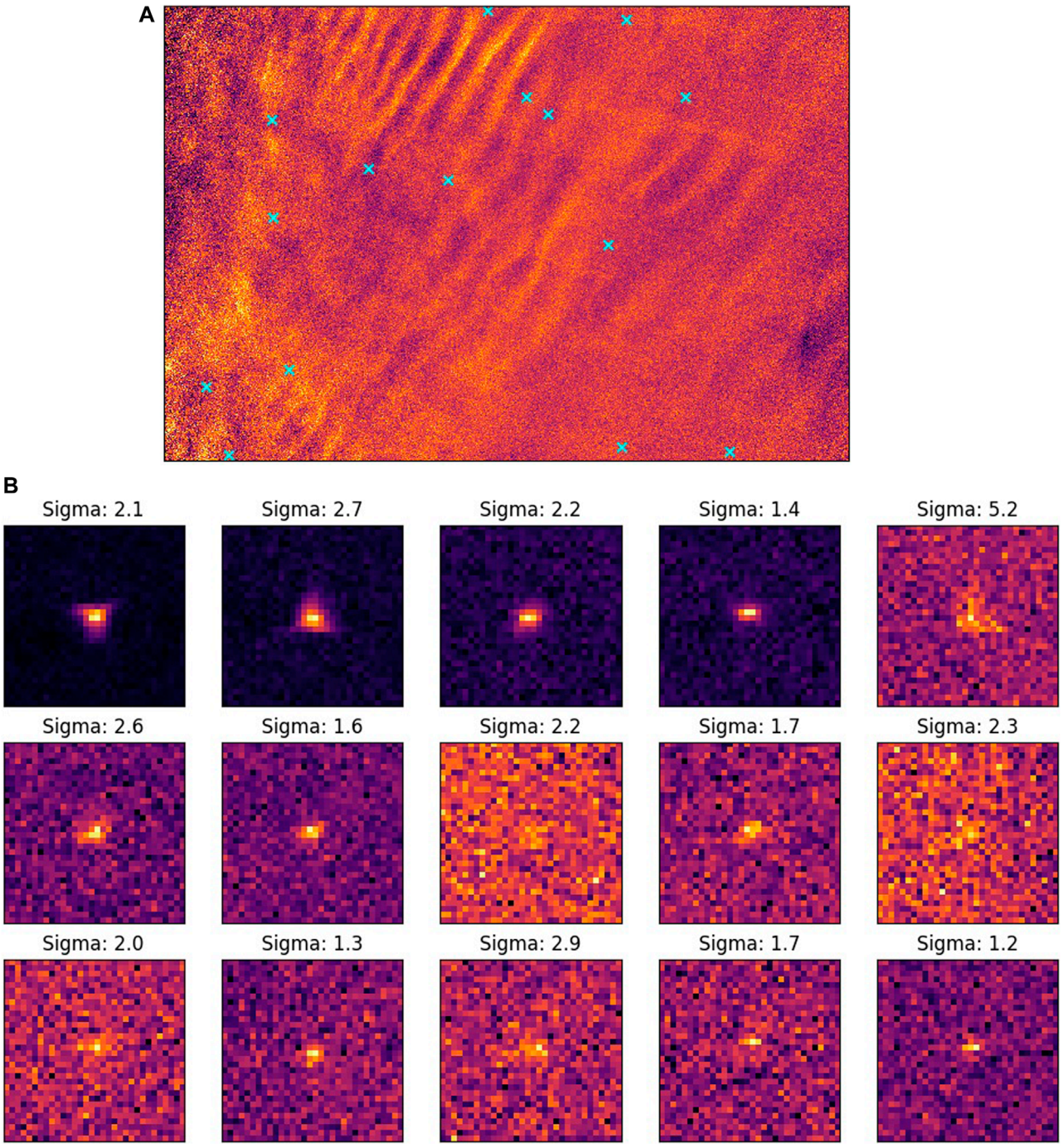


FIGURE 4
An example of star identification. Figure originally published in Kjellstrand (2021). **(A)** Cyan X's mark the location of stars found with the SkyWinder star identification algorithm. The image shown has been calibrated and stray light have been removed, but not projected. **(B)** Close up view of fifteen of the stars identified in **(A)**.

3.2.3 Projecting

Once we have a pointing solution for the evenly-spaced grid of pixels for each image, we project the raw image onto the plane of the PMCs to reconstruct their true scale and morphology. From the grid we interpolate the location of every pixel in the raw image. SkyWinder includes two processes to use these pixel locations to project the image. Each has unique benefits and drawbacks.

One technique is pixel projection. This technique maps the pixel brightness value to the location of the pixel on an x-y plane at the PMC altitude determined by the pointing solution. Compared to the method described below, this method is fast. However, this

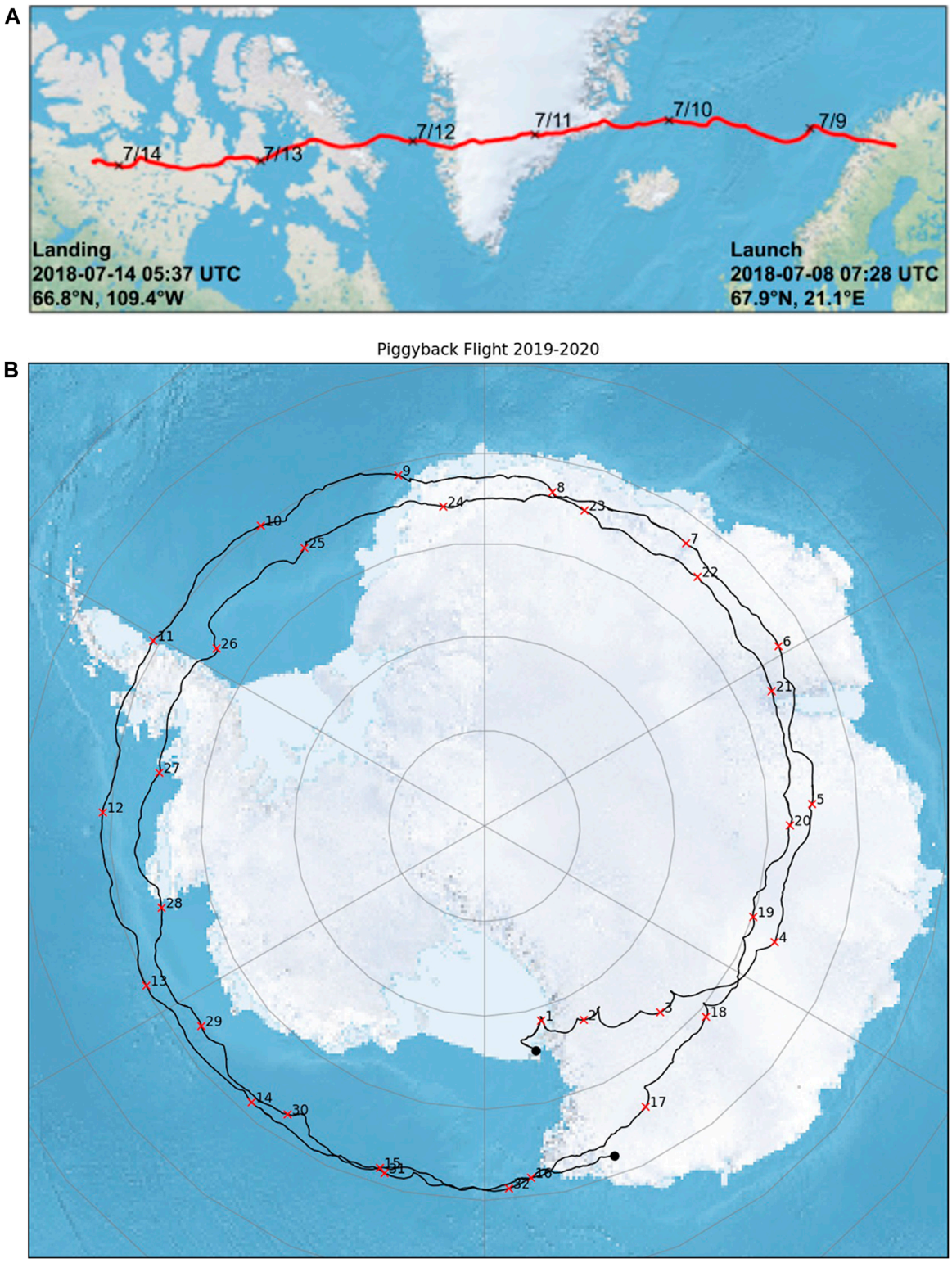


FIGURE 5
PMC-Turbo flight trajectories using SkyWinder flight control software. **(A)** The PMC-Turbo flight trajectory from Kjellstrand (2021). **(B)** Piggyback flight trajectory with marked days of flight from Kjellstrand (2021).

method did not generate evenly-sampled data as pixels at large off-zenith angles gathered light from a larger area than those at small off-zenith angles. This method can more easily rotate the projected FOV with changing pointing due to the rotation of the sun through coordinate transforms—an often useful feature in ballooning since instruments typically have active anti-sun pointing.

The second technique developed is grid mapping. In this method, an evenly spaced grid representing the sky at the PMC altitude is filled in the pixel brightness values. These values are found from the source image by mapping each grid pixel to the corresponding image pixel. A Cartesian grid on the PMC plane is chosen, with the origin chosen to be the location of the gondola and the -Y direction corresponding to the anti-sun direction. The grid can be instantiated at any resolution, so the spatial resolution of the final data product is limited by the raw image spatial resolution. While this process has been optimized to use standard look-up tables rather than calculating the pixel corresponding to each grid, the pixel-by-pixel fill still takes longer than the pixel projection. As a result, we tend to use this method during short duration events where the sun's rotation does not have a large effect or when we do not care about the cardinal direction of the dynamics. The evenly spaced grid facilitates many quantitative analysis techniques such as 2D power spectra.

3.3 Basic data visualization

The SkyWinder analysis code includes a variety of data visualization procedures useful for imager data sets. While specific science groups will want to develop their own analysis procedures, SkyWinder includes some generally useful apps. Images can be stitched together in projected and unprojected states so dynamics can be viewed over a large composite field of view. These images can be aggregated into movies and image series at configurable cadence and pixel-binning. We also include power spectrum-per-image frame generation, keogram creation, and basic statistic reporting.

4 Results

The SkyWinder flight software has been used in the field three times. It ran during the 6-day PMC-Turbo flight during 2018 and again for about a month on a single PMC-Turbo instrument suite contained within a pressure vessel that flew as a secondary ("piggyback") experiment on the 2019-2020 Super Trans-Iron Galactic Element Recorder (SuperTIGER) flight. The trajectories of these flights are shown in **Figure 5**. Additionally, the flight software has been integrated with the BALloon Based Observations for Sunlit Aurora (BALBOA), which had a North American test flight September 2022 and is scheduled for an Antarctic science flight 2023-2024.

During the primary science flight we successfully sent commands in real time in response to the changing sky conditions, we observed PMCs even in low quality downlinked images, as shown in **Figure 2**, and we used the communication channels to run several live tests including using stars to dial in our focus settings with focus sweeps and measuring the sky brightness as a function of sun angle. During flight we downlinked over 37,000 files over the LOS link, 245,000 files over Iridium Pilot, and 257,000 files over TDRSS. Of these 540,000 files, about a third were compressed images.

The piggyback also successfully downlinked images for the first half of flight. While issues with the downlink channel prevented us from receiving files or communicating with the piggyback for the second half of flight, the instrument was retrieved and found to have recorded data as designed in spite of the communication blackout.

The data from the PMC-Turbo primary science flight and piggyback were successfully retrieved. Analysis is continuing and has already resulted in notable science results in gravity wave breaking, vortex ring formation, mesospheric bores, and the role of turbulence in gravity wave momentum deposition in the middle atmosphere [see Fritts et al. (2019); Geach et al. (2020); Fritts et al. (2020)].

5 Discussion

The SkyWinder flight software is suitable for experiments which have relatively low performance requirements for communication speed and which therefore prioritize ease of implementation and testing more highly. The use of Python rather than a compiled language limits the ability of the user to optimize for computational efficiency, but greatly increases implementation speed. Many balloon-borne experiments will match this characterization. The downlink speeds currently available to balloon-borne experiments are hundreds of kilobits per second, so data throughput does not exceed the rates possible with Python. Furthermore, experimental complexity is limited by the power, weight, and volume requirements of the balloon-borne platform. We expect many of these limitation to complexity to be shared with other mobile experiment platforms.

We designed and tested the SkyWinder flight software on the Linux operating system. Python will work on other operating systems, but in our experience the science group using SkyWinder needs to have in-depth knowledge of the alternate operating system to successfully implement the SkyWinder flight control that relies heavily on Pyro. Fortunately, Linux is commonly used for scientific computing.

We are distributing and managing the package with the Python in Heliophysics Community (PyHC) project. PyHC is a project consisting of scientists developing a comprehensive set of Python packages useful in Heliophysics and establishing a set of agreed standards and best practices for the quality and preparation of open-source Python packages. We are maintaining our code to meet Heliophysics code development standards.

The PyHC group has assembled Heliophysics python packages on their website in an effort to provide a common hub for Python-based heliophysics modules. We will add the SkyWinder repository link to that hub. We will also provide the appropriate documentation and installation information for other scientific groups to use the code at the PyHC hub.

Our instrument data will also be publicly available. Both the data reduction software and the raw data will be easily accessible

to interested scientists. This will greatly facilitate collaborations and future research, and ensure continued value from the NASA data set. The PMC-Turbo data will be hosted in the NASA Space Physics Data Facility (SPDF). The SkyWinder analysis code will include methods to pull the PMC-Turbo data from the SPDF. The SkyWinder package can also be used with the extensive imaging and lidar data on the NSF Madrigal database, which the PI has contributed to and used extensively.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

CK wrote the manuscript, was a member of the team that wrote the PMC-Turbo software, and adapted that legacy code to the SkyWinder project. BW acted as the principal investigator of SkyWinder and provided feedback on the manuscript.

Funding

NASA Grant 80NSSC20K0178 funded the PyHC SkyWinder development. NASA Contract 80NSSC18K0050 funded the PMC-turbo mission from which the SkyWinder code was developed.

Acknowledgments

We would like to acknowledge the contributions of our PMC-Turbo team members, especially those who made substantial contributions to the experiment software included in the SkyWinder modules. Glenn Jones played a central role in the development of the PMC-Turbo flight software. Christopher Geach developed analysis procedures including much of our image correction method.

Conflict of interest

Authors BW was employed by GATS.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Chapman, D. (2015). *Ebex: A balloon-borne telescope for measuring cosmic microwave background polarization*. Ph.D. thesis (Columbia University).

Fritts, D. C., Kaifler, N., Kaifler, B., Geach, C., Kjellstrand, C. B., Williams, B. P., et al. (2020). Mesospheric bore evolution and instability dynamics observed in pmc turbo imaging and Rayleigh lidar profiling over northeastern Canada on 13 july 2018. *J. Geophys. Res. Atmos.* 125, e2019JD032037. doi:10.1029/2019jd032037

Fritts, D. C., Miller, A. D., Kjellstrand, C. B., Geach, C., Williams, B. P., Kaifler, B., et al. (2019). Pmc turbo: Studying gravity wave and instability dynamics in the summer mesosphere using polar mesospheric cloud imaging and profiling from a stratospheric balloon. *J. Geophys. Res. Atmos.* 124, 6423–6443. doi:10.1029/2019jd030298

Fritts, D. C., Smith, R. B., Taylor, M. J., Doyle, J. D., Eckermann, S. D., Dörnbrack, A., et al. (2016). The deep propagating gravity wave experiment (deepwave): An airborne and ground-based exploration of gravity wave propagation and effects from their sources throughout the lower and middle atmosphere. *Bull. Am. Meteorological Soc.* 97, 425–453. doi:10.1175/bams-d-14-00269.1

Geach, C., Hanany, S., Fritts, D., Kaifler, B., Kaifler, N., Kjellstrand, C., et al. (2020). Gravity wave breaking and vortex ring formation observed by pmc turbo. *J. Geophys. Res. Atmos.* 125, e2020JD033038. doi:10.1029/2020jd033038

Geach, C. P. (2020). *Gravitational and gravity waves: The EBEX and PMC turbo balloon experiments*. Ph.D. thesis (The University of Minnesota, Minneapolis).

Kjellstrand, C. B., Jones, G., Geach, C., Williams, B. P., Fritts, D. C., Miller, A., et al. (2020). The pmc turbo balloon mission to measure gravity waves and turbulence in polar mesospheric clouds: Camera, telemetry, and software performance. *Earth Space Sci.* 7, e2020EA001238. doi:10.1029/2020ea001238

Kjellstrand, C. B. (2021). *The PMC turbo experiment: Design, development, and results*. Ph.D. thesis (Columbia University).

frontiers | Frontiers in Astronomy and Space Sciences

TYPE Technology and Code
PUBLISHED 27 April 2023
DOI 10.3389/fspas.2023.1119775

Check for updates

OPEN ACCESS

EDITED BY
Jonathan Eastwood,
Imperial College London, United Kingdom

REVIEWED BY
Arnaud Masson,
European Space Astronomy Centre (ESAC), Spain
Peter Chi,
University of California, Los Angeles, United States

*CORRESPONDENCE
R. A. Stoneback,
contact@stoneris.com

SPECIALTY SECTION
This article was submitted to Space Physics, a section of the journal Frontiers in Astronomy and Space Sciences

RECEIVED 09 December 2022
ACCEPTED 28 February 2023
PUBLISHED 27 April 2023

CITATION
Stoneback RA, Burrell AG, Klenzing J and Smith J (2023), The pysat ecosystem. *Front. Astron. Space Sci.* 10:1119775. doi: 10.3389/fspas.2023.1119775

COPYRIGHT
© 2023 Stoneback, Burrell, Klenzing and Smith. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

The pysat ecosystem

R. A. Stoneback[1]*, A. G. Burrell[2], J. Klenzing[3] and J. Smith[3,4]

[1]Stoneris, Plano, TX, United States, [2]Naval Research Laboratory, Space Science Division, Washington, DC, United States, [3]NASA Goddard Space Flight Center, ITM Physics Laboratory Greenbelt, Greenbelt, MD, United States, [4]Catholic University of America, Washington, DC, United States

The Python Satellite Data Analysis Toolkit (pysat) is an open source package that implements a general data analysis workflow for arbitrary data sets, providing a consistent manner for obtaining, managing, analysing, and processing data, including modelled and observational ground and space-based data sets for the space sciences. Pysat enables systematic and individual treatment of data as well as simplifies rigorous data access and use, allowing larger-scale scientific efforts including machine learning, data assimilation, and constellation instrumentation processing. Since the start of its development pysat has evolved into an ecosystem, separating general file and data handling functionality from both individual data set support and generalized data analysis. This design choice ensures that the core pysat package has only the necessary functionality required to provide data management services for the wider development community. The shift of data and analysis support to ecosystem packages makes it easier for the community to contribute to, as well as use, the full array of features and data sources enabled by pysat. Pysat's ease of use, and generality, supports adoption outside of professional science to include industry, citizen science, and education.

KEYWORDS

pysat, data analysis, data management, space-based data, ground-based data, heliophysics, space physics

1 Introduction

The future expansion of scientific instrumentation will exacerbate the difficulties in finding, downloading, accessing, and utilizing space science data. Currently, data is stored and distributed by a variety of government, academic, and commercial agencies through a variety of mechanisms and file formats. While these specific mechanisms and formats may reflect the specific needs of a particular community it produces an additional challenge for scientists seeking to work with data across various sub-fields. Many problems require the integration and use of ground-based, space-based, and even modelled data to understand or answer science questions.

There are emerging solutions for cloud computing where the data and processing capabilities are hosted on remote servers that scientists can access. These services can be extremely useful, particularly within a community that has similar needs but differing funding levels. While this technique can make it easier for scientists to access and load data, the available data is limited to sets installed on the server. Thus, interdisciplinary users are likely to encounter situations where desired data sets may be distributed across multiple servers, similar to the current data distribution challenges. While users may be able to download data from different services on a computer they control, it is less likely that users will be able to upload their own data sets.

A shift to cloud computing also has negative implications for equitable and long term accessibility of data. Use of cloud computing has ongoing non-trivial costs and excels for situations where significant resources are needed for only a limited time. Users without funding, or additionally without access, like the public, would not be able to utilize those services for their own research. The ongoing costs for local hardware, by virtue of not including hardware costs as done in cloud computing, are lower. Further, administrative approval is not required for users to operate personal computers. Thus, software that supports scientific processing, including data downloads, on a user's local machine (as well as *via* the cloud) provides the greatest accessibility and access for the largest number of people.

One approach for solving the data distribution and format issues is to mandate a single file format across a scientific discipline and create a super server, or single access point, for all data. The challenge with this type of solution is it ignores the historical reasons that created multiple file formats and access mechanisms in the first place. Typically speaking, each community within space science has its own requirements, based upon science focus, data type, and distribution mechanisms that evolved over time consistent with that particular communities needs. Mandating a single method or format across space science ignores these sub-field requirements. Working with a single general format everywhere is thus likely not possible, and at best sub-optimal, increasing the cost of working with data for everyone. A change in file format could also break existing systems. Further, unless all data in space science, as well as all disciplines that can impact space science, are all integrated into the same format and server, there will always be scientists that must work across formats and data sources.

While data discovery, download, and loading are significant challenges, scientific data also generally requires additional processing to be useful. Typically, at minimum, data must be selected for high quality observations (or cleaned). Depending upon the data set there may be flags included with the data. Alternately, the appropriate conditions for selecting from all available data may only be available in a scientific publication. Though there is a move towards greater access to publications the historical record is still typically confined behind a paywall. Even when readily accessible, the current situation requires researchers to construct code to properly clean data for robust scientific analysis.

The scientific community has been shifting from analyzing data from one or few platforms to working with multiple platforms, particularly for data assimilation or machine learning models. Working with multiple data sets involves additional challenges. While some analyses can work with multiple data sets individually, this is not always the case. When an individual data set approach is not possible, multiple data sets need to be loaded at once, where data from one source is used directly in the selection or processing of other data. Without a systematic framework to work with multiple data sources the practical challenges are likely to be solved sub-optimally, as pressures to produce results and publish can limit the quality, extensibility, and maintainability of the code produced.

The traditional instrument mission in space science required the concerted effort of multiple space scientists per instrument. This level of personnel is supportable when there are few spacecraft and the ground-based instrumentation is easily maintained. The rise of massive constellations with thousands of satellites, such as Starlink, cannot maintain the same staffing levels (Moigne, 2018). Further, one of the challenges when working with multiple instruments in a constellation is each instrument may have unique and unexpected characteristics that requires additional specialized processing. To enable science to make use of large constellations there is a need for software that can effectively scale the efforts of few scientists to many instruments, while accommodating the potentially unique processing requirements for individual instruments in the constellation.

The Python Satellite Data Analysis Toolkit (pysat) provides a community wide solution for these problems. The core pysat package provides a data-independent user interface that abstracts away the tedious details of file, data, and metadata handling so that scientists can focus on science. Pysat is designed as a data plug-in system where support for each data set includes functionality to download, load, process, and clean data. Plug-ins differ from modules in that plug-ins are not functional on their own and depend upon a host program to plug into. Further, users don't generally interact with the plug-ins directly but through the host program, or pysat. This plug-in configuration supports any file format or data source. Further, it makes it possible for community members to distill their data knowledge into working code so that the community can automatically work with the best interpretation of data. In addition, the core pysat package includes a variety of standards and functionality tests so that developers of these plug-ins get direct feedback on standards compliance.

Support for a wide variety of data sets in pysat has been grouped by data provider and released as independent software packages. This configuration reduces the requirements for any given user as they only need to install the packages they need while offering the broadest array of instrument support in the open source community. The versatility and generality of these features makes it possible for pysat to support data from any provider and in any format. Rather than require all funding agencies to adopt the same standard, pysat is designed to interact with a variety of data sources and file formats but still provide a common and consistent interface for scientists. Thus, communities may maintain their specialized approaches as desired, while those that work across or within domains can do so through a consistent interface.

The array of pysat features are well suited for instrumentation processing. The data and metadata handling features reduces requirements on developers while maintaining the customizability needed for large constellations. Pysat is in use for Ion Velocity Meter (IVM) processing on both the NASA Ionospheric Connections (ICON) Explorer as well as the National Oceanic and Atmospheric Administration (NOAA)/National Space Organization (NSPO) Constellation Observing System for Meteorology Ionosphere and Climate-2 (COSMIC-2) constellation. Pysat features are also used to create the publicly distributed files for the missions.

Pysat is an open source package that builds upon the existing community of tools and is designed to interact with others (e.g., Burrell et al., 2018; Pembroke et al., 2022). In particular, pysat builds upon the pandas (pandas development team, 2022) and xarray (Hoyer et al., 2022) packages so that the general and scientific community use the same tools. This commonality reduces the barrier for the public to interact with scientific data, as well as

making it more likely for pysat to expand beyond the currently supported community.

Pysat's generalized features makes it well suited for any data set available from any source. The core pysat team are ionospheric scientists thus the data sets already supported by pysat reflect this scientific focus. However, pysat itself is not limited to ionospheric data, and pysat's plug-in design makes it possible for users to add their own data to pysat. As such, pysat is suitable for use in industry, professional science, citizen science, and the classroom. To support this wider perspective pysat joined NumFOCUS as an affiliated package.

Since its original publication (Stoneback et al., 2018), pysat has broken out data support and analysis packages into an ecosystem. Features within pysat have also been expanded and generalized as needed. For clarity, this manuscript covers the advances within the core pysat package as part of a description of pysat's larger feature set. Examples from each of the pysat ecosystem packages (collectively referred to as the pysat Penumbra) are also included. Analysis functionality has also been broken out into individual packages. This reconfiguration of pysat helps to minimize requirements on users and developers, while maintaining the same level of functionality. The ecosystem makes it easier for the wider community to contribute instrument support code as each pysat Penumbra package is focused and contains functions designed to work with a particular data source or perform a particular type of analysis. Pysat is a core package within the Python in Heliophysics Community (Burrell et al., 2018; Barnum et al., 2022) and is a community developed package.

2 Methods

2.1 Pysat

The main user interface is the pysat Instrument object. The Instrument object provides a consistent interface for working with data independent of source, abstracting away tedious file and data handling details. The Instrument object incorporates a data processing workflow to accommodate all of the versatility required for research data analysis within a systematic framework. Data can be loaded by users either by file or by specifying a time range in increments of days, independent of the time stored in a given file. The features in the Instrument object enables the construction of instrument independent analysis procedures that work independent of the data dimensions or source, and provides a foundation to transition to the analysis of many data sources with limited personnel.

A canonical example for working with any data set supported by pysat is included below.

First, relevant packages are imported and then a directory for pysat to store data is assigned. The pysat directory, or directories, only needs to be assigned once per installation. For this example we will work with the Solar Wind Electron Proton Alpha Monitor (SWEPAM) instrument on the Advanced Composition Explorer (ACE) spacecraft. Support for this data set is provided in the pysatSpaceWeather (Burrell et al., 2022b) package and must be registered once before use.

```
import datetime as dt
import matplotlib.pyplot as plt

import pysat
import pysatSpaceWeather

# Set data directory if user hasn't already set one
if len(pysat.params['data_dirs']) == 0:
    pysat.params['data_dirs'] = '~/pysatDemo'

# Register instruments with pysat. Only needed once per install.
pysat.utils.registry.register_by_module(pysatSpaceWeather.instruments)

# Instantiate a pysat Instrument that will use support in
# pysatSpaceWeather for ACE Space Weather data
ace = pysat.Instrument('ace', 'swepam', tag='historic')

# Download data for July, 2021
ace.download(dt.datetime(2021, 7, 1), dt.datetime(2021, 7, 31))

# Load all data
ace.load(use_header=True)

# Make plot of all variables for part of July, 2021
sdate = dt.datetime(2021, 7, 5)
edate = dt.datetime(2021, 7, 27)
for var in ace.variables:
    plt.figure()
    ylabel = "".join([ace.meta[var, "long_name"], " (",
                      ace.meta[var, "units"], ")"])
    title = " ".join(["Platform:", ace.platform,
                      "Name:", ace.name,
                      "Tag:", ace.tag,
                      "Clean Level:", ace.clean_level])

    ace[sdate:edate, var].plot(title=title, ylabel=ylabel,
                               xlabel="Date",
                               marker=".", linestyle="",
                               xlim=[sdate, edate])
    plt.tight_layout()
```

To accommodate the wide variety of data sources and file and metadata formats pysat is designed as a plug-in system. Support for a particular data set is enabled by an external module that implements a variety of methods required by pysat, including support for downloading, loading, and cleaning data. These modules 'plug-in' to specific interfaces within pysat, are controlled by pysat, and aren't functional on their own. As directed by a user, pysat will invoke these supporting functions as needed to provide data and metadata to pysat in appropriate formats. The configuration provides a consistent interface for the user while accommodating a wide variety of technical solutions for any particular data set.

A pysat instrument object for the ACE SWEPAM data is instantiated after the pysat directory is assigned. Data sets are labeled by up to four parameters: 'platform', 'name', 'tag', and 'inst_id'. 'Platform' and 'name' are required and refer to the measurement platform and instrument name. 'Tag' and 'inst_id' may be used to further distinguish between multiple outputs from an instrument, or perhaps multiple data products from a given measurement platform (e.g., spacecraft, constellation, or observatory) and instrument combination.

Using the Instrument object the full life cycle for data analysis is enabled through class methods. Data is retrieved from the remote repository using the 'ace.download' command. The user specifies a range of dates and pysat calls the ACE SWEPAM support in pysatSpaceWeather to access the remote repository, retrieve the data, and store it locally. The files are organized under the pysat data directory assigned by the user. By default, data is organized under this user-specified directory using 'platform', 'name', 'tag', and 'inst_id' sub-directories, allowing simple machine and user file navigation.

All of the ACE SWEPAM data on a user's system is loaded using the pysat Instrument class method 'load'. A load command with no date restrictions will load all data on the user's system. Note that no memory checks are performed before loading. This means that if a '.load()' command is issued for a very large data set, pysat will attempt to load all of the data, even if it exceeds local memory. Alternately, a single date (specified by the year and day of year or a Python datetime object) or a range of dates may be provided.

To continue the canonical pysat code example, a range of dates is defined and each ACE SWEPAM variable is plotted as a function of time. Since files from different sources do not typically have the same metadata standards, pysat automatically translates loaded metadata into a set of labels that may be controlled by the user. Default metadata labels of 'units' and 'name' are used in this example. A selection of the plots produced from the code above are in Figure 1.

To support an even more generalized approach for cases where users assign non-default metadata labels developers may use the 'meta.labels' instance to access metadata as below. This ensures that regardless of the string values assigned to the Instrument object to identify 'units' or 'name' the code will continue to work. We emphasize the code below is not needed to account for different labels used within files as metadata labels are automatically translated to the standard assigned in the Instrument object.

```
# Make plot of all variables for part of July, 2021
sdate = dt.datetime(2021, 7, 5)
edate = dt.datetime(2021, 7, 27)
for var in ace.variables:
    plt.figure()
    units = "".join([ace.meta[var, ace.meta.labels.name], " (",
                     ace.meta[var, ace.meta.labels.units], ")"])
    title = " ".join(["Platform:", ace.platform,
                      "Name:", ace.name,
                      "Tag:", ace.tag,
                      "Clean Level:", ace.clean_level])

    ace[sdate:edate, var].plot(title=title, ylabel=units,
                               xlabel="Date",
                               marker=".", linestyle="",
                               xlim=[sdate, edate])
    plt.tight_layout()
```

The only portions of the canonical example code above that are specific to SWEPAM on ACE are the particular values of 'platform', 'name', and 'tag' when instantiating the Instrument. Updating those values for any of the other instruments within pysat will similarly download, load, and plot data in early July 2021, provided the data is available.

A listing of all data sets registered in pysat is available to the user with the 'pysat.utils.display_available_instruments()' function which prints the corresponding 'platform', 'name', 'tag', 'inst_id', and a short description for all registered plug-ins. Further, additional information may be obtained from each data plug-in module using the 'help' command. Similarly, help may be invoked on the 'inst_module' attribute attached to the pysat Instrument class for expanded information on the instantiated data set.

2.1.1 Data

Pysat also includes generalized data access at the Instrument level for ease of use and to support the construction of the generalized processing functions. Some instruments produce a collection of one dimensional signals that depend soley on time, while others require higher-dimensional structures. To support the widest variety of data sets pysat provides support for Instruments to utilize a pandas DataFrame or an xarray Dataset as the underlying data format. The DataFrame excels at tabular data, or a collection of one dimensional data in time. Higher dimensional data is better served using xarray, which builds upon pandas indexing to support higher dimensional data sets.

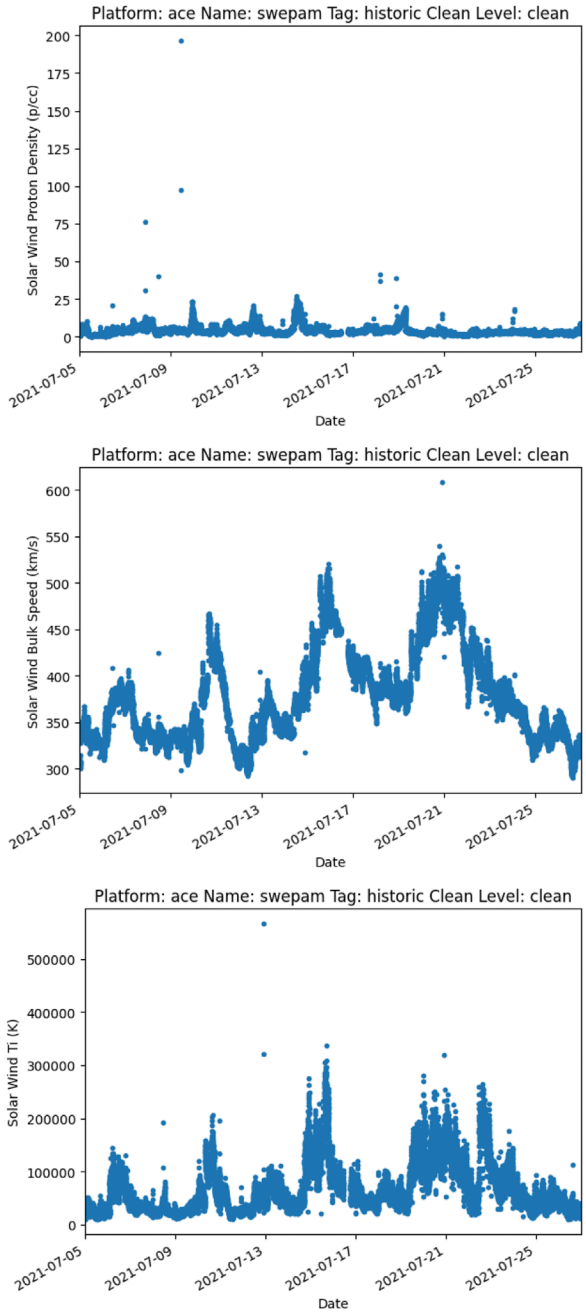


FIGURE 1
Selected output plots from ACE SWEPAM example.

Similar to numpy (Harris et al., 2020), pandas, and xarray, the pysat Instrument supports a variety of indexing techniques for data access and assignment. These methods are shown below.

```
import datetime as dt
import numpy as np

import pysat

# Create test Instrument
inst = pysat.Instrument("pysat", "testing_xarray")

# Load data
inst.load(date=inst.files.start_date, use_header=True)

# Data assignment examples

# Simple assignment, default metadata assigned
inst["name"] = new_data

# Assignment with Metadata
inst["name"] = {"data": new_data,
                "long_name": long_name,
                "units": units}

# Conditionally change data
inst[inst["mlt"] < 12., "mlt"] += 24.

# Partially assign data
idx, = np.where(inst["mlt"] > 23.)
inst[idx, "mlt"] = np.arange(len(idx))
# Access data

# By name
inst["name"]

# By list of names
inst[["name1", "name2"]]

# By position
inst[row_index, ["name1", "name2"]]

# Slicing by row
inst[row1:row2, "name"]

# By Date
inst[datetime, "name"]

# Slicing by date, inclusive.
inst[datetime1:datetime2, "name"]
```

Though a simulated data set loaded into xarray is used here, the same commands also work for Instrument objects with pandas DataFrames as the underlying format. Variables can be selected singly or through a list of strings. Data may be down-selected using Boolean or integer indexing. Indexing functionality supports both one and higher dimensional data. While the access and assignment functionality should serve all developer needs the full data set is always available to users and developers at the 'data' Instrument attribute. Users and developers may access or modify the loaded data directly as needed.

The data access provided by pysat is primarily enabled by the underlying data formats. Pysat is simply mapping the inputs to the relevant underlying format. This simplifies the user's experience, since these methods are not always treated identically by pandas and xarray. This general data support is a core functionality of pysat, and will be maintained if other internal data formats are introduced.

2.1.2 Metadata

Pysat also includes support for tracking metadata, so users can easily understand a data set they are working with and by accessing the data documentation provided in the original files. While xarray is equipped to track metadata for both individual variables and an entire data set, pandas is not equipped to do so. To ensure the most consistent user interface, pysat tracks metadata using its own class and is thus independent of the underlying data format.

Since file standards typically document metadata in different ways pysat builds in functionality to automatically translate metadata in the file to a user provided consistent standard. Pysat defaults to tracking seven different metadata parameters: Units, long name, notes, description, minimum, maximum, and fill value. The categories of metadata are used to automatically translate metadata as directly stored in a file to a standard set by the user. Users can track any number of desired parameters for individual variables. File-level metadata is stored using the file-specified attributes in the 'meta.header' class. This ensures all metadata provided within a file is accessible within the pysat Instrument without duplicating any data.

For ease of use, the metadata access is case preserving but case insensitive. Thus, using 'units' or 'Units' or any other variation in case returns or assigns the same data. This feature is intended to support broader compatibility of code written to a particular metadata standard.

Pysat's metadata support is intended to support coupling metadata from any standard for use within pysat. As such, adopting a particular metadata file format for pysat compatibility is not strictly required. Maximum compatibility is achieved when the file metadata standard includes information in the seven categories that pysat tracks by default, as noted above. However, pysat supports additional categories as needed.

2.1.3 File handling and organization

Pysat's Instrument class is intended to free users from specific knowledge about a data set's files. Enabling this abstraction requires that pysat has knowledge about the files on the local system, obtained by parsing filenames. The file list is used by pysat when loading data, either through the 'load' class method, or through pysat's built in iteration features. Information about local files is also used when updating a local data set for consistency with the most recent files at the data source.

Pysat requires that all of the data be placed in one or more user-specified high level directories. Nominally data sets are organized under the top-level directories using the corresponding values of 'platform', 'name', 'tag', and 'inst_id'. Users can set their own preferred schema and utilities are included to move files from one schema to another. This preference may be assigned pysat wide using the parameters class, or an a per Instrument basis using the 'directory_format' Instrument keyword.

Pysat has file parsing utilities to properly categorize the date of the file, as well as parameters such as version, revision, or cycle. Users can direct pysat to parse out custom information as well. The informational structure of a filename is typically specified and

used internally by a developer when constructing a pysat data set plug-in. Users can load data with a different filename structure, e.g., for files obtained from a non-default data provider, by setting the same information at Instrument instantiation with the 'file_format' keyword.

Pysat provides two functions for parsing filenames using delimited and fixed width standards. The fixed length parser uses the fixed location of information within a filename to extract information. The delimited parser uses the presence of a supplied character to locate information. In practice, released data sets may employ a combination of techniques when encoding filenames. There is a large degree of overlap between the two functions: The delimiter parser works on fixed length filenames without a single delimiter, while the fixed width parser works on some filenames with a variable width. Different approaches are taken within each function, particularly in areas with feature overlap, to help ensure that any unexpected edge cases remain parsable.

The 'download_updated_files' Instrument class method will keep a local machine up to date with respect to files on servers. Pysat will identify the files on the local machine, as well as those on the data source server. Any dates on the server but not present locally are downloaded, and server files with a newer version, revision, or cycle compared to local files are also downloaded. These features makes it trivial for a user to ensure their local machine is current with the most recent server products.

2.1.4 Loading and customizing data

Pysat's features make it possible to construct instrument independent analysis functions and scale scientific analysis from few to many data sources. The processing cycle described in this section is a fundamental enabling technology for those features. The internal data flow makes it possible for users to interact with data in increments different than stored in a file. The availability of programmatic hooks at multiple locations within the data flow makes it possible for developers and users to easily configure a data flow that satisfies a broad range of technical and processing requirements. These features provide a foundation for constructing instrument independent analysis functions, as well as the scaling needed to move from interacting with few to many data sources in scientific analysis.

When a user invokes the 'load' Instrument class method a chain of function calls begins as in **Figure 2**. First, a list of files is returned from pysat corresponding to the date range provided by the user. The list of files is passed along to supporting data set plug-in functions that perform the actual loading. Those functions return data and metadata loaded from the supplied filenames. The data and metadata are attached to the Instrument object which is then passed by pysat to the 'preprocess' and 'clean' functions.

The 'init' method is only called when instantiating the Instrument, and is not shown in the load data flow. The 'init' method is generally useful for setting parameters that don't typically change, such as the data set's acknowledgements and references. A full Instrument instance is passed to 'init' to ensure users and developers can change any aspect of the Instrument.

The 'preprocess' function allows a data set plug-in developer to automatically modify loaded data, or an Instrument object, just after the data is loaded internally and before any other changes could be made. This feature is one of the ways developers can transfer their practical data set knowledge to users. As an example, the Communications/Navigation Outage Forecasting System (C/NOFS) IVM (Heelis and Hanson, 1998; de La Beaujardière, 2004; Stoneback et al., 2012) data set begins with measurements at a 2-Hz sample rate but later shifts to a 1-Hz sample rate. The C/NOFS IVM pysat plug-in thus assigns an attribute to the Instrument object during loading for the sample rate. Downstream functions intended for C/NOFS IVM can easily refer to that attribute as needed during processing. Alternately, custom C/NOFS functions can use that information to accurately couple into more general community packages.

After pre-processing, the Instrument object is passed to a 'clean' function. This function is written by a developer so that users, by default, operate upon high quality scientific data. While data sets may feature a flag indicating data quality, it is still incumbent upon the user to properly incorporate that flag. For data sets that do not include a quality flag this information may be in a published manuscript, or, in a worst case scenario, not present. Thus the 'clean' function allows knowledgeable developers to construct data filters that correspond to four quality levels, 'clean', 'dusty', 'dirty', or 'none'. By default, data is loaded at the 'clean' level. This setting may be updated by users, either as a general pysat wide parameter, or when setting up the Instrument.

After cleaning, the Instrument object goes through optional data padding that turns disparate files into a computationally continuous data set. Time based calculations can require a minimum number of continuous samples for proper output. Thus, to apply this function to the first sample of a file could require more samples for an accurate calculation than would otherwise be loaded (such as data before and after the desired analysis period). The data padding feature, enabled at the Instrument level, pads the primary loaded time frame with a user specified amount of data before and after the primary data window. After padding, the data is processed by user specified functions, then the additional padded data is removed. The feature thus provides a user transparent spin-up and spin-down data buffer that produces accurate time-based calculations equivalent to loading the full data set. To minimize excess loading, a cache is employed for leading and trailing data.

The 'Custom Functions' section enables users to attach a sequence of user provided functions that will be automatically applied to the Instrument object in order as part of the loading process. This functionality makes it possible to easily modify data as needed for instrument independent analysis code. Suppose there is an analysis package that internally loads one or more days of data. If a user wants that analysis applied to a calculated variable not directly stored in the data set file then without custom functions the user would have to either modify the analysis package to modify the data after it is loaded, or produce a new file that also includes the new data and then use that data set. By including a custom processing queue within the Instrument object data from the file may be arbitrarily modified without requiring any changes within the external analysis code. The only requirements on these functions are that the Instrument object must be the first input argument and that any information to be retained is added directly to the Instrument.

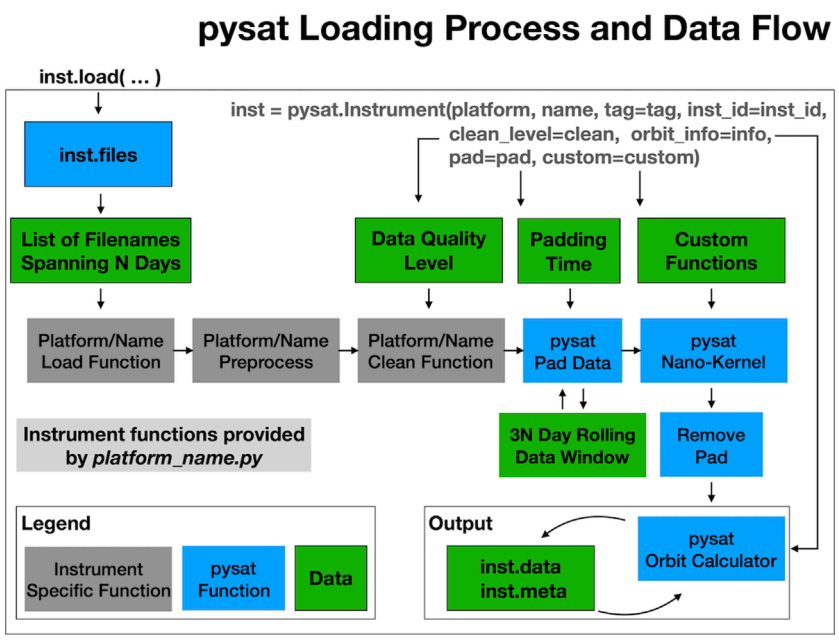


FIGURE 2
Internal pysat data flow when user invokes. Load command.

For a standard load call the pysat data flow is now complete. The loaded data is attached to the Instrument object through a 'data' attribute, readily available to the user for further modification.

The availability of programmatic hooks at multiple locations within the data flow makes it possible for developers and users to easily configure a data flow that satisfies a broad range of technical and processing requirements.

An additional layer of versatility is supported by enabling users to engage options within Instrument plug-ins. Custom keyword arguments provided by users are identified and passed to appropriate plug-in functions. These keywords only need to be defined in the data set plug-in code. Pysat identifies any undefined keywords in relevant method calls, compares these keywords to those defined in the plug-in methods, and passes matching keywords and values to the relevant methods. These keywords may be provided upon instantiation or in a method call. If provided both at instantiation and in a particular call then the value in the method call is used. The value at instantiation is retained and used by default if a value is not provided in the relevant call.

2.1.5 Iterating through data

Analysis of data over time requires loading data over a range of days, files, orbits, or some general condition upon the data. Pysat includes functionality to support this type of loading through iteration independent of the data distribution in the files. Loading data through iteration engages the same process in Figure 2 to ensure a consistent user experience.

Pysat's iteration may be accessed using Python's for loop construction as demonstrated below.

```
import datetime as dt
import pandas as pds

import pysat

# Create a custom range of files supported by test instruments
start_range = dt.datetime(2022, 1, 1)
stop_range = dt.datetime(2022, 12, 31)
file_date_range = pds.date_range(start_range, stop_range, freq="D")


# Basic test instrument, pandas format, non-default file range
inst = pysat.Instrument("pysat", "testing",
                        file_date_range=file_date_range)

# Set bounds for iteration. Accepts information for the
# start, stop, step size, and loaded data width when iterating.
# Users can supply either a single start, stop, or a
# list of starts/stops.

# Iteration works either with datetimes/timestamps or with filenames
start = start_range + pds.DateOffset(weeks=0)
stop = start_range + pds.DateOffset(days=13)

# Demonstrate assigning multiplate date ranges
start = [start, start + pds.DateOffset(months=3)]
stop = [stop, stop + pds.DateOffset(months=3)]

# Number of days to move forward for each subsequent load.
# Uses pandas frequency strings. "3D" is three days.
step = "3D"

# Number of days to load each iteration
width = dt.timedelta(days=3)

# Assign iteration limits
inst.bounds = (start, stop, step, width)

# Loop through data loading `width` days each loop,
# and moving forward `step` for the subsequent load
for loop_inst in inst:
    print("".join(["Time range: ",
          loop_inst.index[0].strftime("%b %d, %Y %H:%M:%S"), " - ",
          loop_inst.index[-1].strftime("%b %d, %Y %H:%M:%S")]))
```

The code produces the following output,

```
Time range: Jan 01, 2022 00:00:00 - Jan 03, 2022 23:59:59
Time range: Jan 04, 2022 00:00:00 - Jan 06, 2022 23:59:59
Time range: Jan 07, 2022 00:00:00 - Jan 09, 2022 23:59:59
Time range: Jan 10, 2022 00:00:00 - Jan 12, 2022 23:59:59
Time range: Apr 01, 2022 00:00:00 - Apr 03, 2022 23:59:59
Time range: Apr 04, 2022 00:00:00 - Apr 06, 2022 23:59:59
Time range: Apr 07, 2022 00:00:00 - Apr 09, 2022 23:59:59
Time range: Apr 10, 2022 00:00:00 - Apr 12, 2022 23:59:59
```

The bounds of the iteration are set through the Instrument 'bounds' attribute, which may contain one or more ranges of date/files that the Instrument will load data for. In addition, users can set the number of days/files to load each iteration, as well as the number of days/files to increment for the next iterative load. The bounds attribute is employed, rather than using a specification in the for loop itself, so that iteration independent code may be constructed by developers.

After imports and definition of time ranges, a Instrument is instantiated with a simulated satellite-like test data set. The default range of 'files' for the simulated data are outside the desired range so the 'file_date_range' custom keyword is used to alter the supported range of dates. Note that the 'file_date_range' keyword is specific to the pysat testing data plug-ins and is not a keyword supported for all Instruments. Next, two example definitions for the starting and ending conditions are shown. The bounds are assigned through 'inst.bounds' and are followed by the iteration commands.

Each iteration of the for loop will load 3 days worth of data using the 'inst' Instrument object. A copy of the 'inst' Instrument object is returned as 'loop_inst'. For performance reasons the underlying data within 'inst' is not actually duplicated but is available through 'loop_inst'. The next loop will step forward 'step' days and load 'width' days, repeating until the final stop bound is reached.

Users may also iterate through data one orbit at a time. This iteration calculates the locations of orbit breaks as part of the data loading and orbit iteration process. The process enables users to employ arbitrary conditions to define an 'orbit'. Complete orbits are returned each loop independent of day or file breaks.

The current code has support for identifying orbits through an orbit number variable, a negative gradient, or a sign change. If an orbit value is already provided pysat will iterate through the data set selecting all times with the next orbit value each iteration. The negative gradient or sign change detectors look for specific data conditions with the user supplied data variable to determine where orbit breaks occur.

The orbit iterator can compare the time of a detected orbit against a user supplied value to limit false positives when working with noisy data. Note that some tolerance is required as not all orbit types have a consistent orbit period. Geophysical variability in the orbit environment can physically change orbit properties. Additionally, orbit periods calculated with respect to magnetic local time aren't consistent orbit to orbit. This variability arises due to the offset of the geomagnetic field with respect to Earth's rotation axis.

In the future the orbits class will be generalized so that users can directly select a wider variety of techniques for calculating iteration breaks in the data. This generalization in user input will enable iterating through the data against arbitrary data conditions, not just against orbit expectations.

2.1.6 Creating files

Support is included for writing Instrument objects to disk as a compliant netCDF4 file with arbitrary metadata standards. By default, pysat will create files with a simplified version of the Space Physics Data Facility (SPDF) International Solar-Terrestrial Physics (ISTP)/Iteragency Consultancy Group (IACG) standard. Instrument objects may be written to disk, then reloaded, without loss of information.

The Ionospheric Connections (ICON) Explorer mission created a new standard for SPDF and netCDF4 files. The original SPDF standard was composed for Common Data Format (CDF) files. ICON chose the netCDF variant for the mission which required some translation from the original standard due to an existing library of netCDF software. To achieve the greatest software compatibility the SPDF netCDF4 format includes some information under multiple names. For example, the standard itself requires the use of '_FillValue' but 'FillVal' is also mandated for community compatibility reasons. Further, maximum and minimum expected data values have multiple names. To support the SPDF netCDF4 and other formats pysat makes it possible to arbitrarily modify stored Instrument metadata as it is being written to the file. This enables developers to work with the minimum unique information throughout processing and then expand as needed for file creation.

The code below maintains metadata compliance with the full SPDF netCDF4 standard.

```
# Define SPDF metadata labels
labels = {"units": ("units", str), "name": ("long_name", str),
          "notes": ("notes", str), "desc": ("desc", str),
          "plot": ("plot_label", str), "axis": ("axis", str),
          "scale": ("scale", str),
          "min_val": ("value_min", np.float64),
          "max_val": ("value_max", np.float64),
          "fill_val": ("fill", np.float64)}

# Instantiate instrument with custom metadata labels
inst = pysat.Instrument("pysat", "testing", labels=labels)

# Define translation of pysat metadata labels to those in the
# netCDF file
meta_translation_table = {inst.meta.labels.fill_val: ["_FillValue",
                                                      "FillVal"],
                          inst.meta.labels.desc: ["CatDesc"],
                          inst.meta.labels.name: ["Long_Name"],
                          inst.meta.labels.units: ["Units"],
                          inst.meta.labels.max_val: ["ValidMax",
                                                     "Valid_Max"],
                          inst.meta.labels.min_val: ["ValidMin",
                                                     "Valid_Min"],
                          inst.meta.labels.notes: ["Var_Notes"],
                          inst.meta.labels.scale: ["ScaleTyp"],
                          inst.meta.labels.plot: ["FieldNam"],
                          inst.meta.labels.axis: ["LablAxis"]}

# Write netCDF file
pysat.utils.io.inst_to_netcdf(inst, fname,
                              meta_translation=meta_translation_table)
```

The labels class provides a mapping from the types of metadata to be stored, the string value used to store the metadata, and the default type for that parameter. The labels dictionary is provided at Instrument instantiation to modify the default metadata labels. While the underlying 'testing' data set is created with the standard pysat metadata labels, and not the expanded labels needed for SPDF, pysat will automatically translate the standard labels in the file to the user provided values as part of loading. New labels without a corresponding entry in the data are left empty.

The 'meta_translation_table' defines a mapping from the metadata labels in the Instrument object to those used when writing a file. Entries may be mapped to one or more labels in the file. The multi-label support is used to convert single entries at the Instrument level, such as the minimum expected data values, to multiple labels in the file, 'Valid_Min' and 'ValidMin'.

Additional parameters required by SPDF but that can be unambiguously determined directly from data are handled by pysat. The time data also has different metadata parameters available and is assigned by pysat. Metadata is constructed in the same way for both xarray and pandas Instrument objects. For pandas objects pysat directly creates the netCDF file. For xarray objects pysat uses xarray's netCDF4 functions when writing the actual file.

Global attributes in the netCDF4 file created in the code above may be set by assigning new attributes to 'inst.meta.header'.

2.1.7 General settings

Pysat has a parameters class to create a central location for users to set pysat or custom settings. The 'pysat.params' class stores defaults for checking file system every instantiation, default clean level, directories for data, whether to ignore empty files, and registered user modules. Users may also assign their own custom entries.

2.1.8 Constellation

To support the use of multiple data sets in concert pysat also offers a Constellation class. Constellations may be defined and distributed by a developer, similar to Instrument data set plug-ins. Alternately, users can collect an arbitrary group of Instruments in a list to instantiate a Constellation object or create a Constellation of all available Instruments that share a set of defining characteristics (e.g., load all historic ACE data or all instruments from an observational platform). There are no restrictions on the types of Instruments object that may be combined in a Constellation.

The intent of the Constellation class is to provide a high degree of compatibility between Constellation and Instrument objects in terms of attributes and methods. As an example, invoking the Constellation load method will trigger a corresponding call to load data for all Instrument objects within the Constellation. However, as the Constellation object is a collection of Instruments, each Instrument may still be manipulated individually. For example, custom functions may be applied to the Constellation object itself (affecting all Instruments) or to single Instruments within the Constellation.

The Constellation class is more than simply a wrapper for a list of Instruments. It contains several attributes designed to improve analysis on multiple Instruments. These include the establishment of a common time-series and attributes ('empty' and 'partial_empty') that define if all, some, or no data was present for the desired time period. The Constellation class is the youngest of the core pysat classes, and is undergoing active testing and development. Future enhancements include a method to convert from a Constellation to an Instrument and allowing Constellation sub-modules to provide 'init' and 'preprocess' methods.

2.2 The penumbra environment

To provide support for the broadest array of data sets and data providers, both in and out of science, pysat is designed to accommodate data sets through a modular system. This system design ensures a consistent user experience without requiring any consistency from data providers or analysis packages. For Python packages built using pysat, the user will (for most processes) call pysat directly. This simplifies the analysis process for scientists performing studies from multiple data sets.

Because the data providers are typically self-consistent, pysat packages that focus on providing data are organized by data provider. pysatNASA (Klenzing et al., 2022b) supports NASA data from Coordinated Data Analysis Web (CDAWeb), pysatMadrigal (Burrell et al., 2021) for National Science Foundation (NSF) data from Madrigal database, and pysatCDAAC (Klenzing et al., 2021) for Cosmic Data Analysis and Archive Center (CDAAC) data. Other pysat packages focus on a particular type of data set that may span multiple data providers: pysatSpaceWeather (Burrell et al., 2022b) focuses on space weather indices and real-time data, while pysatModels (Burrell et al., 2022a) has utilities designed to facilitate model-observation comparisons and loading model files for analysis. Other pysat packages focus on a particular analysis goal, with pysatMissions (Klenzing et al., 2022a) providing tools to simulate and propagate orbits for current or future space missions and pysatSeasons (Klenzing et al., 2022c) providing averaging processes independent of data source and dimensionality.

Template instruments are included in pysat as well as several of the pysat Penumbra packages to make it easier for users and developers to add new Instruments. The templates include calls to pysat provided functions that are generally applicable. Each stage of the template Instrument plug-in is documented with comments and descriptive basic docstrings.

To assist developers in ensuring compliance of data module functions pysat includes a suite of unit tests for external instrument modules. Data modules outside the core package can inherit this core suite of tests in local tests built using pytest (Krekel et al., 2004). These tests cover standards compliance for each module, as well as a test run of a common set of operations: download a sample file, load it with different levels of cleaning, and test remote file listing if available. All test data is downloaded to a temporary directory to avoid altering an end user's working environment should they contribute to the code. The tests are inherited from a top-level class, and are used across the ecosystem to maintain consistent standards. Additional tests for instruments with custom inputs can be added at the package level using the inherited setup.

2.2.1 Currently supported data

2.2.2 Supporting new data sets

Pysat's plugin design enables the wider community to load any data set *via* pysat. Given pysat's plug-in design, pysat does not directly identify, download, load, or clean a data sets files. Rather, pysat directs Instrument plug-ins, or a collection of appropriately written methods, to perform various actions as required to work with a particular data set.

Supported data sets grouped alphabetically by package				
Platform	Name	Tag	Inst_ID	Description
Package: pysatSpaceWeather				
ace	epam	realtime	''	Real-time ACE EPAM data (SWPC)
		historic		Historic ACE EPAM data (SWPC)
	mag	realtime		Real-time ACE Magnetometer data (SWPC)
		historic		Historic ACE Magnetometer data (SWPC)
	sis	realtime		Real-time ACE SIS data (SWPC)
		historic		Historic ACE SIS data (SWPC)
	swepam	realtime		Real-time ACE SWEPAM data (SWPC)
		historic		Historic ACE SWEPAM data (SWPC)
sw	dst	noaa		Historic Dst data (NOAA/NCEI)
		lasp		Predicted Dst (LASP)
	f107	historic		Daily LASP value of F10.7
		prelim		Preliminary SWPC daily solar indices
		daily		Daily SWPC solar indices (contains last 30 days)
		forecast		SWPC Forecast $F_{10.7}$ data (next 3 days)
		45 days		Air Force 45-day Forecast
	kp	''		Deprecated: definitive and nowcast Kp (GFZ)
		def		Definitive Kp (GFZ)
		now		Nowcast Kp (GFZ)
		forecast		SWPC forecast Kp (next 3 days)
		recent		SWPC historic Kp (past 30 days)
	mgii	composite		Composite data set of MgII core-to-wing index
		sorce		SORCE SOLSTICE MgII core-to-wing index
Package: pysatNASA				
cnofs	ivm	''	''	CINDI-C/NOFS Ion Velocity Meter
	plp			C/NOFS Langmuir Probe
	vefi	dc_b		C/NOFS DC Magnetometer data
de2	lang	''		DE-2 Langmuir Probe data
	nacs			DE-2 Neutral Atmosphere Composition Spectrometer data
	rpa			DE-2 RPA data
	wats			DE-2 Wind And Temperature Spectrometer data
icon	euv			ICON EUV Level 2 data
	fuv	day		Level 2 ICON FUV daytime O/N2
		night		ICON FUV Level 2 nighttime O profile
	ivm	''	a	ICON IVMa Level 2 data
			b	ICON IVMb Level 2 data

(Continued on the following page)

(*Continued*)

Supported data sets grouped alphabetically by package				
Platform	Name	Tag	Inst_ID	Description
	mighti	vector_wind_green	vector	ICON MIGHTI green-line vector winds
		vector_wind_red		ICON MIGHTI red-line vector winds
		los_wind_green	a	ICON MIGHTIa green-line LoS winds
			b	ICON MIGHTIb green-line LoS winds
		los_wind_red	a	ICON MIGHTIa red-line LoS winds
			b	ICON MIGHTIb red-line LoS winds
		temperature	a	ICON MIGHTIa neutral temperature data
			b	ICON MIGHTIb neutral temperature data
formosat1	ivm	‘’	‘’	F1/ROCSAT IVM
iss	fpmu			International Space Station FPMU
jpl	gps	roti		Rate of change in TEC
omni	hro	1min		OMNI HRO 1-min time averaged data
		5min		OMNI HRO 5-min time averaged data
ses14	gold	nmax		GOLD Level 2 Nmax data
timed	saber	‘’		
	see			
Package: pysatCDAAC				
cosmic	gps	ionprf	‘’	COSMIC ionospheric profiles
		wetprf		COSMIC atmospheric profiles with moisture
		atmprf		COSMIC atmospheric profiles without moisture
		eraprf		COSMIC ERA-40 interim reanalysis data
		gfsprf		COSMIC NCEP operational analysis data
		ionphs		COSMIC ionospheric excess phase
		podtec		COSMIC absolute TEC and auxiliary data
		scnlv1		COSIC S4 scintillation index and auxiliary data
cosmic2	ivm	‘’	e1	COSMIC2 IVM data
			e2	COSMIC2 IVM data
			e3	
			e4	
			e5	
			e6	
Package: pysatMadrigal				
dmsp	ivm	‘’	f11	DMSP IVM Level 2 data
			f12	
			f13	
			f14	
			f15	
			f16	
			f17	
			f18	

(Continued on the following page)

(*Continued*)

Supported data sets grouped alphabetically by package				
Platform	Name	Tag	Inst_ID	Description
		utd	f11	UTDallas DMSP data processing
			f12	
			f13	
			f14	
			f15	
gnss	tec	vtec	''	Ground-based vertical TEC
jro	isr	drifts		JRO ISR drifts and wind
		drifts_ave		JRO ISR averaged drifts
		oblique_stan		JRO ISR standard Faraday rotation double-pulse
		oblique_rand		JRO ISR randomized Faraday rotation double-pulse
		oblique_long		JRO ISR long pulse Faraday rotation
Package: pysatMissions				
missions	ephem	''	''	Satellite simulation data set
	sgp4			
Package: pysatModels				
pydineof	dineof	''	''	pyDINEOF output file
		test		Standard output of pyDINEOF for benchmarking
sami2py	sami2	''		sami2py output file
		test		Standard output of sami2py for benchmarking
ucar	tiegcm	''		UCAR TIE-GCM file
End of Supported Data Sets				

Pysat has an ionospheric heritage, reflected in the currently supported data sets across the pysat ecosystem, consistent with the scientific focus of the developers. However, pysat itself is not limited to ionospheric or even scientific data. As the pysat team cannot directly develop plug-ins for every data set, pysat is designed so that users can construct their own plug-in modules to support their own particular data. This user available Instrument plug-in creation support is the same used by the pysat team to add the full variety of data sets in **section 2.2.1**.

In this section we provide an overview of Instrument plug-in requirements and demonstrate that any data set that a user can load on their own system is a data set that can be loaded from the pysat interface. An example for an Instrument support plugin is included below. The docstrings and other comments have been significantly reduced here for brevity. The full template is included with the pysat source code under 'pysat/instruments/templates'. The file should be named 'platform_name.py' where 'platform', 'name' are replaced with appropriate values. These identifiers are used by pysat when a user instantiates an Instrument object. The instrument plug-in file must be part of a python module and registered with pysat before it will be available for use.

First, a variety of Instrument attributes are set by the developer, along with a range of testing attributes. Pysat includes a suite of general Instrument plug-in tests that are applied to all registered plug-ins whenever unit tests are run. This is done to make it easy for users to ensure compliance of any plug-in code. The testing attributes enable the developer to specify what types of tests to run and under what conditions.

Following the attribute assignments a variety of required and optional functions are defined. The use of these functions within pysat's loading process is covered in **section 2.1.4**. Minimum required functions are 'list_files', 'load', 'clean' and 'download'. The 'preprocess' is optionally defined as needed by developers to modify data as it is loaded. Finally, the 'list_remote_files' is optional but recommended. Pysat uses this function to keep a local system up to date with respect to the most recent server data.

The 'init' function is run once upon instantiation. As shown, it is typically used to set references, acknowledgements, and other top-level Instrument attributes. The 'clean' and 'preprocess' functions provide opportunities to clean the data, as requested by the user, or otherwise processes or modify the data before it becomes available to the user. These functions are data set specific. The 'init', 'clean', and 'preprocess' functions all receive the pysat.Instrument object as input.

```
"""Template for a pysat.Instrument support file."""

import pysat

# ----------------------------------------------------------------------
# Instrument attributes:
# Not shown for brevity

# ----------------------------------------------------------------------
# Instrument testing attributes:
# Not shown for brevity

# ----------------------------------------------------------------------
# Instrument methods: routines that are attached to the pysat.Instrument
# as class methods
# Required method
def init(self):
    """Initialize the Instrument object with instrument specific values.
    """

    # Required attribute: acknowledgements
    self.acknowledgements = "This would go in the Acknowledgements section"

    # Required attribute: references
    self.references = "These are the instrument references"

    # Direct feedback to logging info
    pysat.logger.info(self.acknowledgements)
    return
# Required method
def clean(self):
    """Return `platform_name` data cleaned to the specified level."""
    return


# Optional method
def preprocess(self):
    """Perform standard preprocessing."""
    return

# ----------------------------------------------------------------------
# Instrument functions: routines that are attached to the pysat.Instrument
# as function attributes
# Required function
def list_files(tag="", inst_id="", data_path="", format_str=None):
    """Produce a list of files corresponding to PLATFORM/NAME."""

    if format_str is None:
        # User did not supply an alternative format template string
        format_str = "example-name_{year:04d}_{month:02d}_{day:02d}.nc"

    # Use pysat provided function to identify files.
    file_list = pysat.Files.from_os(data_path=data_path,
                                    format_str=format_str,
                                    delimiter="_")
    return file_list


# Required function
def download(date_array, tag, inst_id, data_path=None, user=None,
             password=None, **kwargs):
    """Download `platform_name` data from the remote repository."""
    return


# Required function
def load(fnames, tag="", inst_id="", custom_keyword=None):
    """Load `platform_name` data and meta data."""
    return data, mdata


# Recommended function
def list_remote_files(tag, inst_id, user=None, password=None):
    """Return a Pandas Series of every file for chosen remote data."""
    return
```

The 'list_files' function provides pysat with information on the files on a user's system. The function as presented is typically sufficient for most data sets, the developer merely needs to provide a filename template in the variable 'format_str' as well as set keywords in the call to 'pysat.Files.from_os'. The 'format_str' variable provides information on which portions of a filename contain relevant information and how to parse that information. The example includes the 'year', 'month', and 'day' keywords, thus the day will be treated as day of month. If only 'year' and 'day' are included then the day is treated as day of year.

The 'download' function is responsible for downloading data for the provided dates in 'date_array' to the local 'data_path' location. User information is provided by the user, as needed, when they invoke the download method. It is up to the developer to provided the underlying functionality to actually download the data. Typically, this function is the same for a given data source provider.

The 'load' method is responsible for loading data from the local system and returning appropriate data and metadata. A list of filenames to be loaded is provided to the developer by pysat. It is up to the developer to provide the required functionality to load the files and suitably format the loaded data and metadata for pysat. Metadata is stored within the pysat.Metadata class to ensure a proper format. Properly formatted data is either a pandas DataFrame or an xarray data set. A given data provider may tend to serve files in a particular format, thus, the load function may be shared across multiple plug-ins from the same source.

Any file format may be loaded with this plug-in design. Note that pysat itself doesn't impose any requirements on the formatting of the data or the metadata to be loaded, only on the data and metadata returned by the function. For simple text or other data files that don't include metadata in the file the developer can define the information in the function and return it as part of the metadata. As metadata in pysat is primarily informational the system still functions even when no metadata is provided by a developer though pysat warns the user that metadata defaults are being applied.

While pysat is file format agnostic when loading data, pysat includes built-in support for writing and loading netCDF files. The netCDF support will, by default, transparently store and load an instantiated pysat.Instrument to and from disk. The functions also include a variety of metadata and other options to store the data using other file standards with user specified properties. Support for loading other file formats, Common Data Format (CDF) and Hierarchical Data Format (HDF), may be found in pysat penumbra packages pysatCDF (Stoneback et al., 2022) and pysatMadrigal, respectively.

3 Penumbra examples

3.1 Instrument independent analysis

The pysat Instrument and Constellation objects make it possible to build analysis software that works with any combination of data sets. This generalization removes the need for repeated development or modification of the same analysis functions but applied to different data. Analysis packages built upon pysat can also utilize the included test instruments to develop rigorous unit tests and enable validated access across the community. These features increase the general trustworthiness of manuscripts while simultaneously reducing the workload upon the scientists.

Each of the following sub-sections covers a single pysat example.

3.1.1 Bin averages in time

PysatSeasons generalizes the commonly employed two dimensional binning of data, such as binning a variable over longitude and local time, to produce maps of geophysical parameters. pysatSeasons is built on pysat's Constellation object to accommodate averaging data from multiple data sets at once. It

supports binning N-dimensional data within each bin as all input data is translated into xarray form.

An example producing the distribution of ion density as functions of magnetic local time and longitude using data from the Constellation Observing System for Meteorology Ionosphere and Climate (COSMIC-2) constellation is in the code below.

```
import datetime as dt
import matplotlib.pyplot as plt
import numpy as np

import pysat
import pysatCDAAC
import pysatSeasons

# Register pysatNASA ICON IVM data plug-in. Required once per install.
pysat.utils.registry.register_by_module(pysatCDAAC.instruments)

# Define a Constellation of all COSMIC-2 IVMs
const = pysat.Constellation(platforms=["cosmic2"], names=["ivm"],
                            clean_level="none")
# Set the range of dates for the analysis
const.bounds = (dt.datetime(2021, 1, 1), dt.datetime(2021, 1, 31))

def add_log_dens(inst):
    """Calculate log of ion density and add to 'inst'.

    Parameters
    ----------
    inst : pysat.Instrument
        Instrument object to operate upon

    """
    # Log of ion density
    inst["log_ion_density"] = np.log10(inst["orig_ion_density"])

    # Assign metadata
    labels = inst.meta.labels
    inst.meta["log_ion_density"] = {
        labels.units: "N/cc", labels.name: "Log Ion Density",
        labels.desc: "Log base 10 of ion density",
        labels.min_val: np.float64(1.),
        labels.max_val: np.float64(9.),
        labels.fill_val: np.float64(np.nan)}
    return

# Attach to the COSMIC-2 IVM constellation
const.custom_attach(add_log_dens)

pysat.logger.setLevel(20)  # Provide feedback at "info" level

# Seasonal bin average of ICON IVM temperature and density data
results = pysatSeasons.avg.median2D(const, [0., 24., 24], "mlt",
       [-180., 180., 30], "lon", ["log_ion_density"], return_data=True)
```

A user-defined constellation is created after the pysat directory assignment by defining a list containing Instrument objects for each satellite within the six satellite constellation. The list of Instruments is provided to the Constellation class at instantiation. The bounds of the analysis are set at the Constellation level which passes these limits to each individual Instrument object within.

While measurements of ion density are directly available in the COSMIC-2 IVM files, given the large range in values for this analysis we want to produce a map of the log of the density. A simple function to calculate the log of ion density, as well as add appropriate metadata, is defined after the bounds assignment. After definition, this custom function is attached to all the Instruments within the Constellation by attaching the function using the Constellation class 'custom_attach' function. Support is also offered for applying custom functions at the Constellation level. Functions must accept a Constellation as the first input rather than an Instrument. Alternately, users may directly attach custom functions to individual Instruments.

Near the end of the example code the pysat logger is updated to provide additional feedback as the pysatSeasons bin averaging process runs. The 'median2D' function will load data for each of the Instruments in the Constellation over the assigned bounds, binning the data as appropriate. After all constellation data is loaded a median is applied to the data within each bin. Though the binning function currently only internally supports calculating the bin median, all of the binned data may be returned to the user with the 'return_data' keyword, enabling application of any statistical analysis.

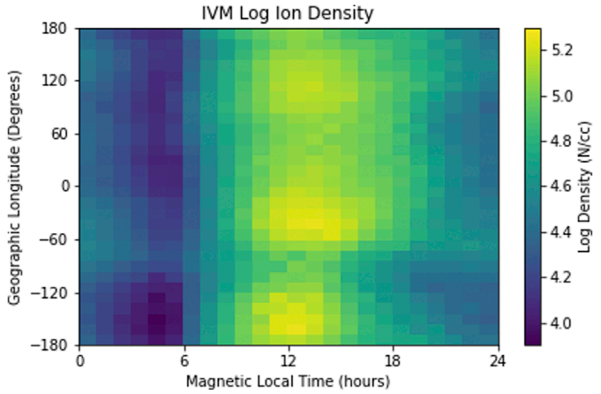


FIGURE 3
Median ion density from COSMIC-2 constellation from January 1—January 31, 2021.

The returned output in the final line of code is stored in 'results', a dictionary, whose values are plotted in **Figure 3**. Raw density values range over multiple orders of magnitude thus the observed range in values between 4-5 is clear evidence that user generated variables added through the custom functions features are supported for averaging. Consistent with general geophysical expectations a clear wave three signature is seen in longitudinal variations in total ion density near local noon.

3.1.2 Occurrence probability

PysatSeasons also includes generalized support for determining how often a user determined condition occurs. The occurrence probability is the number of times the condition occurs at least once per bin per load iteration divided by the number of times the Instrument made at least one observation in a given bin per load iteration. The occurrence probability may be calculated using a daily (or longer) load iteration or using the orbit iterator.

The code below calculates and plots how often plasma bubbles will be detected at ICON's location per orbit.

The first three code groups import required packages as well as register a single NASA data set within pysat. Next, a pysat Instrument is instantiated for the Ion Velocity Meter (IVM) onboard ICON. Two optional features are engaged with the instantiation. Namely, information on identifying orbit breaks in the data, as well as data padding for accurate time-based calculations. Finally, a simple date range is set on the object.

```
import datetime as dt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pds
import warnings

import pysat
import pysatSeasons
# Register pysatNASA ICON IVM data plug-in. Only required once per install.
import pysatNASA
pysat.utils.registry.register(["pysatNASA.instruments.icon_ivm"])

# Orbit breakdown info
orbit_info = {"kind": "lt", "index":"Magnetic_Local_Time"}

# We will employ a centered time-based calculation. Engage pysat's data
   padding.
pad = pds.DateOffset(seconds=30)

# Instantiate IVM instrument data
ivm = pysat.Instrument("icon", "ivm", inst_id="a", orbit_info=orbit_info,
                       pad=pad)

# Set the range of dates for the analysis
ivm.bounds = (dt.datetime(2020, 6, 1), dt.datetime(2020, 7, 31))

# Modify data as it is loaded to add a log ion density measurement and
   shift local
# times for a focus on nighttime observations
def add_std_dens(inst):
    """Calculate Standard Deviation in Ion Density.

    Parameters
    ----------
    inst : pysat.Instrument
        Instrument object to operate upon.

    """
    # Calculate and add ion density standard deviation
    inst["Ion_Density_STD"] = inst["Ion_Density"].rolling("30s",
       center=True).std()

    # Add metadata
    inst.meta["Ion_Density_STD"] = {inst.meta.labels.units: "N/cc",
                                    inst.meta.labels.desc: "30 second
                                       standard deviation",
                                    inst.meta.labels.name: "Ion Density
                                       Std. Dev.",
                                    inst.meta.labels.min_val: 0.,
                                    inst.meta.labels.max_val: np.inf,
                                    inst.meta.labels.fill_val: np.nan}

    return
def shift_local_times(inst):
    """Shift magnetic local times between 0-16 to 24-40."""

    # Shift local times
    inst[inst["Magnetic_Local_Time"] < 16., "Magnetic_Local_Time"] += 24.

    # Update the "notes" metadata parameter
    inst.meta["Magnetic_Local_Time"] = {
        inst.meta.labels.notes: "Times 0-16 shifted by +24 hours."}

    return

# Attach to IVM
ivm.custom_attach(add_std_dens)
ivm.custom_attach(shift_local_times)

# Run Occurrence Probability
answer = pysatSeasons.occur_prob.by_orbit2D(ivm, [0, 360, 24], "Longitude",
                                [16., 30., 14], "Magnetic_Local_Time",
                                ["Ion_Density_STD"], [5.E3], return_bins=True)

# A dict indexed by data_label is returned.
ans = answer["Ion_Density_STD"]

# Plot occurrence probability
f, axarr = plt.subplots(2, 1, sharex=True, sharey=True,
                        figsize=(5.5, 8))

# Mask for locations not observed.
masked = np.ma.array(ans["prob"], mask=np.isnan(ans["prob"]))

# Plot occurrence probability
im = axarr[0].pcolor(ans["bin_x"], ans["bin_y"], masked)
axarr[0].set_title("Occurrence Fraction Delta-N > 5.E3 N/cc")
axarr[0].set_ylabel("Magnetic Local Time (Hours)")
# axarr[0].set_yticks((-20, -15, -10, -5, 0, 5, 10, 15, 20))
axarr[0].set_ylim((ans["bin_y"][0], ans["bin_y"][-1]))
plt.colorbar(im, ax=axarr[0], label="Occurrence Fraction")

# Plot number of orbits per bin.
im = axarr[1].pcolor(ans["bin_x"], ans["bin_y"], ans["count"])
axarr[1].set_title("Number of Orbits in Bin")
axarr[1].set_xlabel("Geographic Longitude (Degrees)")
axarr[1].set_xticks((0, 60, 120, 180, 240, 300, 360))
axarr[1].set_xlim((ans["bin_x"][0], ans["bin_x"][-1]))
axarr[1].set_ylabel("Magnetic Local Time (Hours)")
plt.colorbar(im, ax=axarr[1], label="Counts")

f.tight_layout()
```

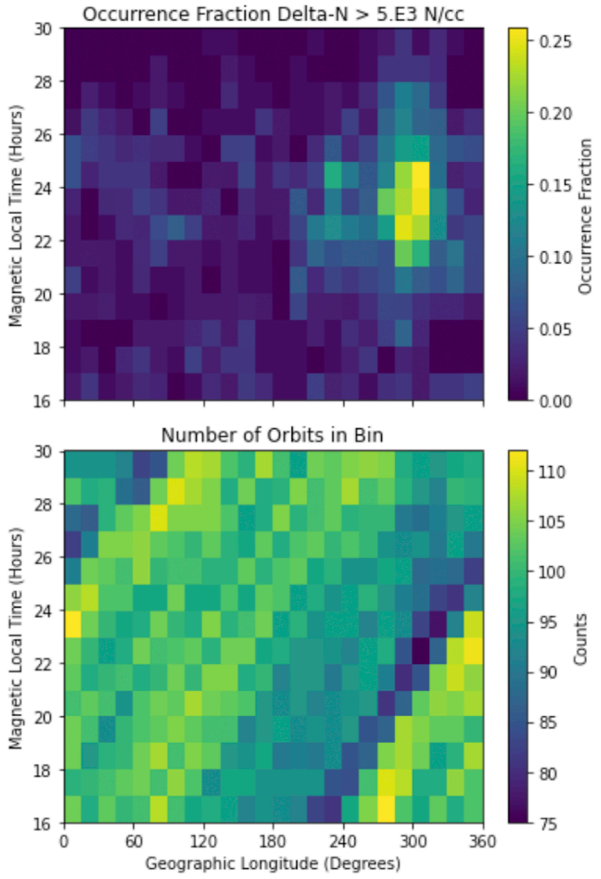


FIGURE 4
Occurrence probability results for ICON IVM between 1 January 2020—31 March 2020.

A pair of custom functions are then defined to modify the ICON data as loaded from files. To identify plasma bubbles a simple running standard deviation is defined in 'add_std_dens'. IVM data is loaded into a pandas DataFrame thus pandas functionality is used to calculate the standard deviation. First, a rolling centered time window of 30 s is defined and that rolling window is used to calculate the standard deviation. Plasma bubbles predominantly occur at night thus the 'shift_local_time' function shifts local times from 0 to 24 to 16-40. Both custom functions are attached to the Instrument object.

The occurrence of standard deviations in plasma density greater than 5E3 N/cc as determined by the 'pysatSeasons.occur_pron.by_orbit2D' is in **Figure 4**. The top figure is the distribution of plasma bubbles as a function of longitude and local time, from late afternoon until dawn. Only the South American sector shows any activity. The 'by_orbit2D' function uses the pysat Instrument object to iterate through ICON data orbit-by-orbit. In this case, orbit breaks are defined using the 'Magnetic_Local_Time' variable and the internally observed locations with significant negative gradients. For complete data sets significant negative gradients would only be observed when local times rolled over from 40 to 16.

3.1.3 Model data comparison

PysatModels includes support for loading model data through pysat as well as functionality for comparing model data with other data sets. The code below will download, load, and compare Jicamarca Radar Observatory Incoherent Scatter Radar (JRO-ISR) observations with a test model data set in pysat.

```
import datetime as dt
import pandas as pds
import pysat
from pysatMadrigal.instruments import jro_isr
import pysatModels

# Initalize the observed data
stime = dt.datetime(2021, 1, 3)
jro = pysat.Instrument(inst_module=jro_isr, tag="drifts",
                       user="Your Name",
                       password="your.email@inst.type")

# Download data if necessary
if stime not in jro.files.files:
    jro.download(start=stime)

# Get fake model data from the pysat model test instrument
mod_drange = pds.date_range(stime, stime + dt.timedelta(days=1),
                            freq="1D")
model = pysat.Instrument("pysat", "testmodel", tag="",
                         file_date_range=mod_drange)
model.load(date=stime)

# Get the model longitude range, and make sure the loaded data
# has the same range
if model["longitude"].min() >= 0 and model["longitude"].max() > 180:
    min_lon = 0.0
    max_lon = 360.0
else:
    min_lon = -180.0
    max_lon = 180.0

jro.custom_attach(pysat.utils.coords.update_longitude,
                  kwargs={"lon_name": "gdlonr", "high": max_lon,
                  "low": min_lon})
# Load data. You may receive a warning for unknown data
# variables (this is ok).
jro.load(date=stime)

# Set the model dummy variable units
model.meta["dummy2"] = {model.meta.labels.units: "m/s"}

# Get the xarray data from the model instrument with metadata attached
model_data = pysatModels.utils.convert.convert_pysat_to_xarray(model)

# Set the extract input parameters
input_args = [jro, model_data, ["gdlonr", "gdlatr", "gdalt"],
              ["longitude", "latitude", "altitude"], "time", "time",
              ["deg", "deg", "km"]]
input_kwargs = {"sel_name": ["dummy2"]}

# Run the extract function
added_vars = pysatModels.utils.extract.extract_modelled_observations(
    *input_args, **input_kwargs)
```

After the imports the JRO instrument is instantiated. Data is downloaded if not already present on the local system. The test model is then instantiated and loaded. The data is simulated thus no download is necessary. To ensure both data sets have the same longitude range, the range from the test model is identified and used as part of the input to a custom function applied to the JRO data, which is then loaded.

Next a comparison between the two data sets is performed. First, for comparison against JRO ion drift measurements a non-specific test variable is modified to have appropriate units. Next, the pysat Instrument object is distilled into an xarray data set. Input arguments identifying equivalent variables between data sets and other needed parameters are constructed and used to extract model-data pairs, locations where both data sets have information. Figure 5 plots the distribution of points for the model, for JRO, and indicates which points are selected as present in both data sets. The values extracted in this case only refer to simulated test data and thus aren't shown.

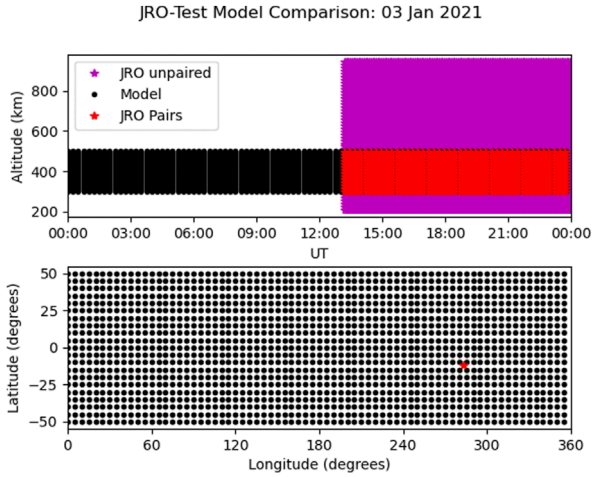


FIGURE 5
Distribution of points identified as part of comparison.

3.1.4 Model data interpolation

PysatModels also includes support for interpolating model data onto another data set. This feature may be used to switch a model to a different grid, or alternately, could be used to project model results onto a satellite orbit.

The code below is an example for interpolating model data from a regular grid onto a satellite orbit.

```
import datetime as dt

import pysat
import pysatModels
from pysatModels.utils import extract

# Load simulated satellite Instrument data set
inst = pysat.Instrument("pysat", "testing", max_latitude=45.)
inst.load(2009, 1, use_header=True)

# Load simulated regular-grid model Instrument
model = pysat.Instrument("pysat", "testmodel")
model.load(2009, 1, use_header=True)

# Linearly interpolate all model data onto inst based
# on time and position
new_data_keys = extract.instrument_view_through_model(
    inst, model.data, ["longitude"], ["longitude"], "time",
    "time", ["deg"], ["mlt"])

# Store results of linear interpolation
inst.rename({new_data_keys[0]: "mlt_linear"})

# Run interpolation using "nearest" model data
new_data_keys = extract.instrument_view_through_model(
    inst, model.data, ["longitude"], ["longitude"], "time",
    "time", ["deg"], ["mlt"], ["nearest"])

# Store results of nearest-neighor interpolation
inst.rename({new_data_keys[0]: "mlt_nearest"})

# Plot a portion of the results
stime = inst.date
etime = inst.date + dt.timedelta(hours=1)
title = "Interpolating MLT Example"
ylabel = inst.meta["mlt", "long_name"]
inst[stime:etime, ["mlt_linear", "mlt_nearest"]].plot(title=title,
                                                      ylabel=ylabel)
```

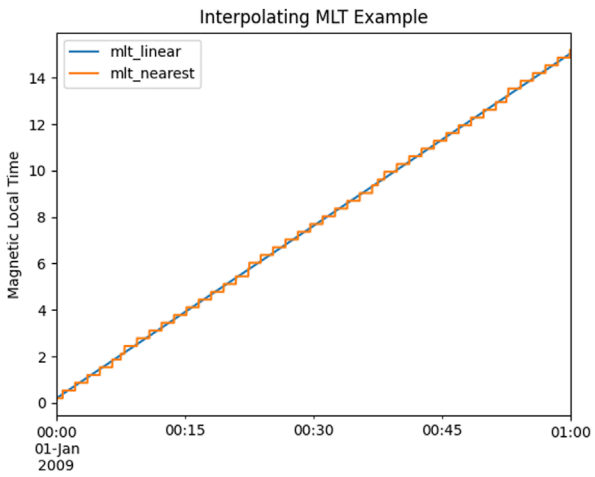


FIGURE 6
One dimensional example for interpolating model data values onto instrument using linear or nearest neighbor interpolation.

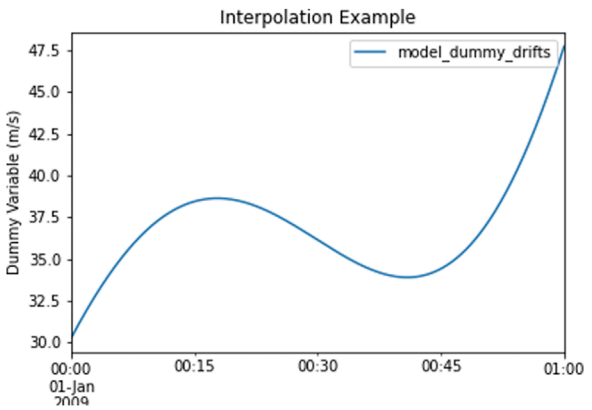


FIGURE 7
Interpolating irregularly gridded data in altitude onto Instrument using intermediate pressure calculation to enable regular interpolation.

After the imports a test satellite and a test model are both instantiated and loaded with data. Next the model results are interpolated onto the satellite track using either the nearest valid value or a linearly interpolated value. Figure 6 shows a results comparison for both settings. As expected, the linear interpolation produces a smooth signal while the nearest neighbor method shifts between discrete values as the satellite moves and the nearest neighbor shifts.

Not all models have a regular distribution of points over the variables of scientific interest. The 'pressure_levels' tag for pysat's testmodel simulates a model that has a regular grid over longitude, latitude, and pressure level, but pressure level has an irregular relationship to altitude. Interpolating from this model onto a satellite data set with altitude, with performance, requires converting the satellite altitude to a model pressure.

The code below covers the full process.

```
import datetime as dt

import pysat
import pysatModels
from pysatModels.utils import extract

# Load simulated satellite Instrument data set
inst = pysat.Instrument("pysat", "testing",
                        max_latitude=45.)
inst.load(2009, 1, use_header=True)

# Load simulated irregular-grid model Instrument
model = pysat.Instrument("pysat", "testmodel",
                         tag="pressure_levels")
model.load(2009, 1, use_header=True)

keys = extract.instrument_altitude_to_model_pressure(
    inst, model.data, ["altitude", "latitude", "longitude"],
    ["ilev", "latitude", "longitude"], "time", "time",
    ["", "deg", "deg"], "altitude", "altitude", "cm")

new_keys = extract.instrument_view_through_model(
    inst, model.data, ["model_pressure", "latitude", "longitude"],
    ["ilev", "latitude", "longitude"], "time", "time",
    ["", "deg", "deg"], ["dummy_drifts"])
stime = inst.date
etime = inst.date + dt.timedelta(hours=1)

ylabel = "Dummy Variable (m/s)"
inst[stime:etime, new_keys].plot(title="Interpolation Example",
                                 ylabel=ylabel)
```

The altitudes and pressure levels in the model are used to generate equivalent pressure levels for the satellite consistent with the satellite altitude through the 'extract.instrument_altitude_to_model_pressure' function. The appropriate satellite pressure levels are generated by guessing initial solutions and then iteratively using a regular grid interpolation on the model to get the equivalent altitudes. The satellite pressure levels are increased/decreased as appropriate and the iteration continues until the difference between the actual satellite altitude and the equivalent altitude are within the specified tolerance. The final pressure levels are stored as 'model_pressure' in the satellite Instrument.

The obtained satellite pressure is used with regular linear interpolation to extract model ion drift values along the satellite track through the 'extract.instrument_view_through_model' call. The resulting interpolated quantities are accurate and generated much faster than using a full irregular interpolation, available through 'extract.interp_inst_w_irregular_model_coord'. The results of the interpolation are in Figure 7.

3.2 Satellite instrumentation processing

The file management, data, and metadata features within pysat are well suited for science instrument data processing. Scientific instrumentation goes through a general data flow. Raw measurements directly from the instrument are converted to physical quantities. These physical measurements are then used to generate geophysical parameters such as ion density or temperature. Finally, the results are stored in standards compliant files and distributed to the public.

While the details of created files, and the instruments themselves, may vary from mission to mission, building instrumentation processing on pysat makes it easy to build long term heritage in processing while still being versatile and adaptable. IVM processing software for ICON and COSMIC-2 is built on pysat. First, A general IVM processing package was developed. The functions within were written using the pysat Instrument object

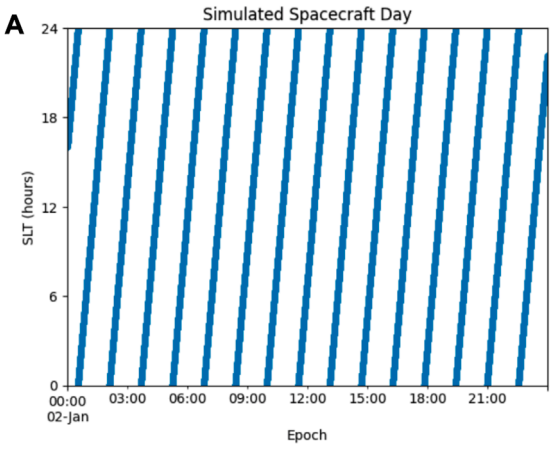


FIGURE 8
(A) Full day of simulated satellite data for January 2 **(B)** A single complete orbit defined as 0-24 MLT produced by pysat's orbit iteration functionality. Note the orbit begins on January 1 and ends on January 2, spanning two files. Replace with fancier ICON summary type plot!.

to provide access to needed data and metadata. A higher-level package was then created for both ICON and COSMIC-2 that simply connects the generalized IVM processing package with the ICON and COSMIC-2 processing environments and file standards. This design configuration ensures that any processing improvements developed or identified within a single mission are automatically available for the other missions. It further allows for significant unanticipated differences between instruments as well as processing environments without modifying the core software.

Unit tests were developed for the generalized IVM package based upon the simulated pysat test instruments. Since the package was built on pysat, it is easy to substitute new data sources into the processing, simulated or measured. These unit tests may thus be reused for future missions, saving future developer time, and ensuring that processing results maintain accuracy across missions.

Using pysat as a foundation makes it easier to deal with unanticipated changes in processing as a mission evolves. This is best achieved by creating a pysat data plug-in that supports each file stage during processing. This enables pysat to mediate loading data as well as provides access to a variety of mechanisms to alter data as needed. The plug-in structure provides multiple functional hooks for working with the data under a variety of conditions. Further, the custom function queue attached to the Instrument makes it easy to change that processing as the mission evolves. For a Constellation object, the functional hooks make it easy to alter the processing of some Instruments within the Constellation. The performance of the same physical instrument may not be the same across an actual constellation of satellites.

3.2.1 Satellite ephemeris

In addition to analysis of existing satellite data, one can also build simulated spacecraft orbits through the pysatMissions package. The core instrument module here simulates a day of orbits using the sgp4 (Rhodes, 2018) package. The spacecraft can be generated from either a pair of Two-Line Elements (TLEs), or from a set of individual orbital elements (orbital inclination, altitude of apoapsis, etc.). When combined with pysatModels simulated data can be added to the simulated orbits. While the data is generated on the fly, operations and analysis are identical to a standard pysat instrument. The current accuracy of the simulated orbit is on the order of 5 – 10 km in altitude for low Earth orbits. Improvements to the accuracy are planned in the future.

The code below uses pysatMissions to simulate a satellite orbit as well as iterate through the simulated data set, orbit-by-orbit.

```
import datetime as dt

import pysat
import pysatMissions

# Register the data plug-ins in pysatMissions.
# Only once per installation.
pysat.utils.registry.register_by_module(pysatMissions.instruments)

# Define on-the-fly orbit breakdown input.
# Note that the solar local time "slt" variable will be added by
# a custom function.
orbit_info = {"kind": "lt", "index": "slt",
              "period": dt.timedelta(minutes=95)}

# Instantiate satellite propagator. Note that 'inclination' and
# 'alt_periapsis' are keywords defined by 'missions_sgp4' module,
# not pysat itself.
inst = pysat.Instrument("missions", "sgp4", orbit_info=orbit_info,
                        inclination=10, alt_periapsis=500.)

# Add calculation of solar local time. Pass in options
# to 'calc_solar_local_time'.
inst.custom_attach(pysat.utils.coords.calc_solar_local_time,
                   kwargs={"lon_name": "longitude"})
# Load a day of data and plot raw 'slt'.
# pysat appplies all custom functions during load process.
inst.load(2019, 2, use_header=True)
inst["slt"].plot(title="Simulated Spacecraft Day",
                 ylabel="SLT (hours)", ylim=[0., 24.],
                 yticks=[0., 6., 12., 18., 24.],
                 marker=".", linestyle="")

# Iterate orbit-by-orbit. Plot the first orbit then stop. To create
# this plot pysat also loads/simulates day previous to ensure that
# the first orbit is complete.
for i, orbit_inst in enumerate(inst.orbits):
    orbit_inst["slt"].plot(ylabel=orbit_inst.meta["slt", "long_name"],
                           title="Single Orbit across Day/File Break",
                           ylim=[0., 24.],
                           yticks=[0., 6., 12., 18., 24.],
                           marker=".", linestyle="")
    if i == 0:
        break
```

After the imports and pysat directory assignment, a dictionary is defined with information needed by the pysat orbit iterator to

determine orbit breaks on the fly. The 'index' identifies the variable for the system to use, while 'kind' selects between several internal orbit break calculations, in this case local time. The period sets the nominally expected orbit period. The subsequent input cell uses this information as part of instantiation. Multiple parameters used internally by the orbit propagator are also set. Internally, these keywords are routed to the appropriate 'missions_sgp4' plug-in function.

A calculation of solar local time within pysat is coupled to the simulated orbit using the 'inst.custom_attach' command. The function itself simply passes user identified variables into the relevant pysat call, then adds the output variables to the Instrument, with metadata. Not all Instrument data sets label quantities like longitude, latitude, or altitude the same, thus string inputs are used to accommodate different naming schemes. Note that only the label to access the data, not the underlying data itself, is passed so that the process works as part of the load process for any day or combination of days.

Data for 2 January 2019 is simulated and loaded into the Instrument object through the standard 'inst.load'. The full data set is plotted in **Figure 8** to demonstrate that the simulated orbit data is treated in the same way as typical satellite data. Note that the first sample is from January 2.

Finally, pysat's orbit iteration is used to break down the daily increment of loaded satellite data into individual orbits. In this case, orbit breaks are determined using solar local time, appropriate for plasma investigations due to the large influences from the Sun and the geomagnetic field. The code is configured to stop the for loop after the first orbit is loaded. The intent of the orbit iteration is to provide full orbits each increment of the iteration loop. As orbit boundaries generally do not respect file or day breaks, pysat uses its internal data cache to load and store data from January 1 as part of its internal calculations. Thus the first orbit sample at 0 Solar Local Time (SLT) occurs on January 1. Note that the load statement from the previous cell is not required for the iteration, orbit or otherwise, to function.

4 Conclusion

The abstractions and functionality provided by pysat enables it to integrate a wide variety of data sets and analysis tools, current or historical, into a cohesive whole. This is particularly important for historical packages that are unlikely to comply with current or future data file standards, since pysat does not impose any requirements upon these external packages. The versatility of pysat's coupling functionality also addresses a fundamental challenge in open source development. Due to the low barrier for open source development there are a wide variety of packages in the scientific community. However, by being open there is no specific requirement that these packages all work together. Of course, with specific effort individual packages may be coupled on a one to one basis. With pysat though, each coupled package coupled is now in an ecosystem where the outputs from the other packages are also available, creating a one to many coupling.

The versatility of pysat's design enables scientists and developers to address the unique aspects of any instrument while retaining a systematic and coherent structure. For each type of change desired by a user, or developer, pysat has built-in functionality to mediate that change. Pysat is thus also well suited as a foundation for instrumentation processing. The plug-in design supports the development of robust and verifiable code for instrumentation processing. Further, the attention to the full data life cycle ensures full support for metadata and the requirements of creating publicly distributed files.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/pysat/.

Author contributions

RS, AB, and JK are core pysat developers. JS is a regular pysat contributor. RS is the primary author of the manuscript. RS, AB, JK, and JS edited and reviewed the manuscript. All authors read and approved the manuscript for submission.

Funding

This work was supported by the DARPA Defense Sciences Office. RS was supported by the Naval Research Laboratory, N00173191G016 and N0017322P0744. AB is supported by the Office of Naval Research. JK was supported through the Space Precipitation Impacts (SPI) project at Goddard Space Flight Center through the Heliophysics Internal Science Funding Model. JS supported through NNH20ZDA001N-LWS and 80NSSC21M0180.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Barnum, J., Masson, A., Friedel, R. H., Roberts, A., and Thomas, B. A. (2022). Python in heliophysics community (pyhc): Current status and future outlook. *Adv. Space Res.* 2022. doi:10.1016/j.asr.2022.10.006

Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—An overview of python in heliophysics. *J. Geophys. Res. Space Phys.* 123, 10384–10402. doi:10.1029/2018JA025877

Burrell, A. G., Klenzing, J., Stoneback, R., and Pembroke, A. (2021). *pysat/pysatmadrigal*. v0.0.4 release. doi:10.5281/zenodo.4927662

Burrell, A. G., Klenzing, J., Stoneback, R., Pembroke, A., Spence, C., and Smith, J. M. (2022b). *pysat/pysatspaceweather*. v0.0.7. doi:10.5281/zenodo.7083718

Burrell, A. G., Klenzing, J., and Stoneback, R. (2022a). *pysat/pysatmodels*. v0.1.0 release. doi:10.5281/zenodo.6567105

de La Beaujardière, O. (2004). C/nofs: A mission to forecast scintillations of equatorial aeronomy sparked by the Jicamarca radio observatory. *J. Atmos. Solar-Terrestrial Phys.* 66, 1573–1591. doi:10.1016/j.jastp.2004.07.030.40

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Heelis, R. A., and Hanson, W. B. (1998). "Measurements of thermal ion drift velocity and temperature using planar sensors," in *Measurement techniques in space plasmas: Particles*. Editors R. F. Pfaff, E. Borovsky, and T. Young (AGU), 61–71. doi:10.1029/GM102

Hoyer, S., Roos, M., Joseph, H., Magin, J., Cherian, D., Fitzgerald, C., et al. (2022). *xarray*. doi:10.5281/zenodo.7195919

Klenzing, J., Stoneback, R., Burrell, A. G., Depew, M., Spence, C., Smith, J. M., et al. (2022a). *pysat/pysatmissions*, 3. Version 0.3. doi:10.5281/zenodo.7055089

Klenzing, J., Stoneback, R., Burrell, A. G., Smith, J., Pembroke, A., and Spence, C. (2022b). *pysat/pysatnasa*. v0.0.4. doi:10.5281/zenodo.7301719

Klenzing, J., Stoneback, R., Pembroke, A., Burrell, A. G., Smith, J. M., and Spence, C. (2021). *pysat/pysatcdaac*. Version 0.0.2. doi:10.5281/zenodo.5081202

Klenzing, J., Stoneback, R., Spence, C., and Burrell, A. G. (2022c). *pysat/pysatseasons*. v0.2.0. doi:10.5281/zenodo.7041465

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., and Bruhin, F. (2004). *pytest* 7, 2.

Moigne, J. L. (2018). Distributed spacecraft missions (dsm) technology development at NASA Goddard Space Flight Center. *IGARSS 2018 - 2018 IEEE Int. Geoscience Remote Sens. Symposium* 293–296. doi:10.1109/IGARSS.2018.8519065

Pandas development team (2022). *pandas-dev/pandas: Pandas*. doi:10.5281/zenodo.7223478

Pembroke, A., DeZeeuw, D., Rastaetter, L., Ringuette, R., Gerland, O., Patel, D., et al. (2022). Kamodo: A functional api for space weather models and data. *J. Open Source Softw.* 7, 4053. doi:10.21105/joss.04053

Rhodes, B. (2018). python-sgp4.

Stoneback, R. A., Burrell, A. G., Klenzing, J., and Depew, M. D. (2018). Pysat: Python satellite data analysis toolkit. *J. Geophys. Res. Space Phys.* 123, 5271–5283. doi:10.1029/2018JA025297

Stoneback, R. A., Davidson, R. L., and Heelis, R. A. (2012). Ion drift meter calibration and photoemission correction for the c/nofs satellite. *J. Geophys. Res. Space Phys.* 117. doi:10.1029/2012JA017636

Stoneback, R. A., Depew, M., Klenzing, J., Iyer, G., Pembroke, A., Starr, G., et al. (2022). pysat/pysatCDF: v0.3.2 (v0.3.2). Zenodo. doi:10.5281/zenodo.1217180

Frontiers in
Astronomy and Space Sciences

Explores planetary science and extragalactic astronomy in all wavelengths

Advances the understanding of our universe - from planetary science to extragalactic astronomy, to high-energy and astroparticle physics.

Discover the latest Research Topics

See more →



Frontiers

Avenue du Tribunal-Fédéral 34
1005 Lausanne, Switzerland
frontiersin.org

Contact us

+41 (0)21 510 17 00
frontiersin.org/about/contact