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

# 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

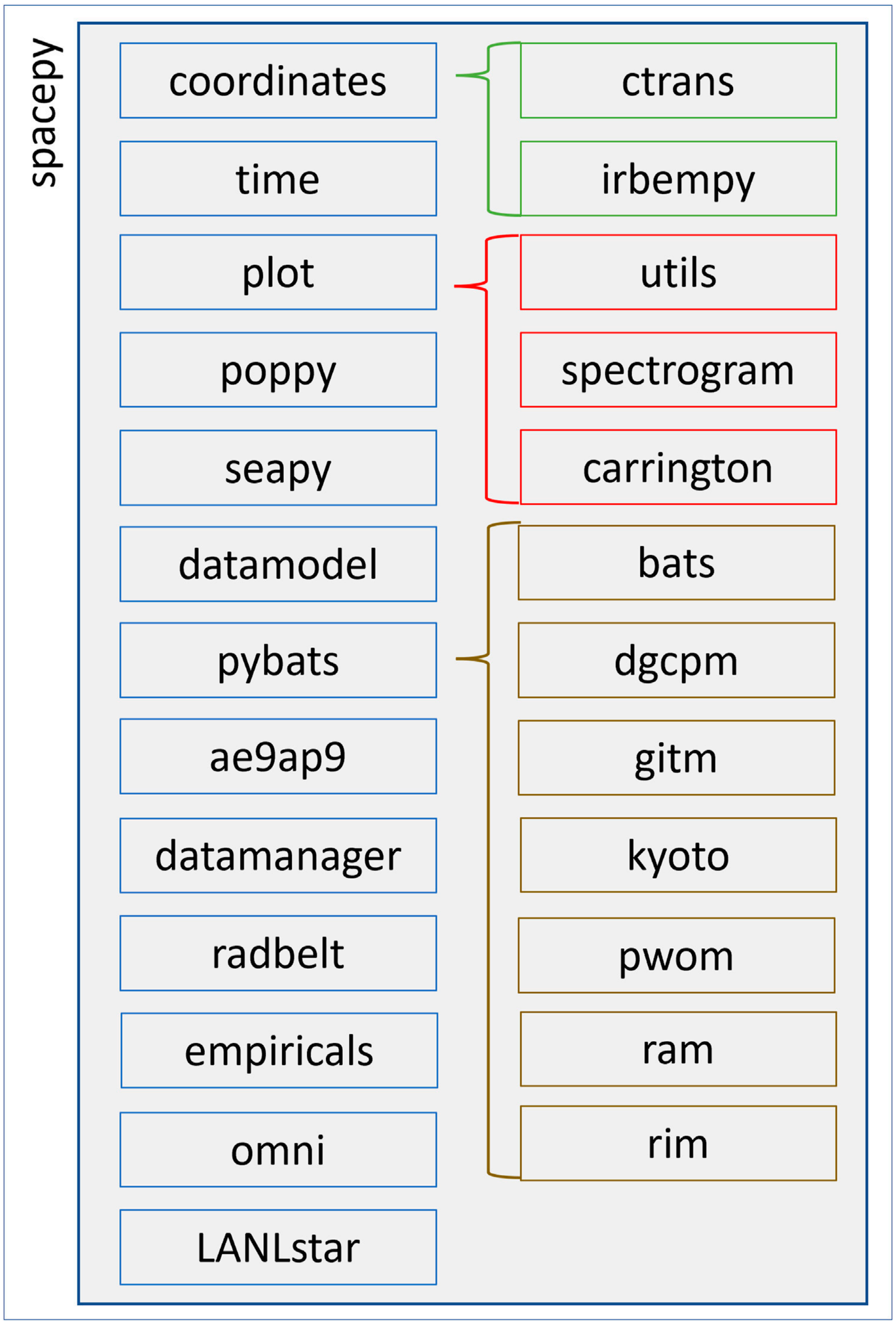

FIGURE 1
Core capabilities organized by namespace in the SpacePy package.

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.

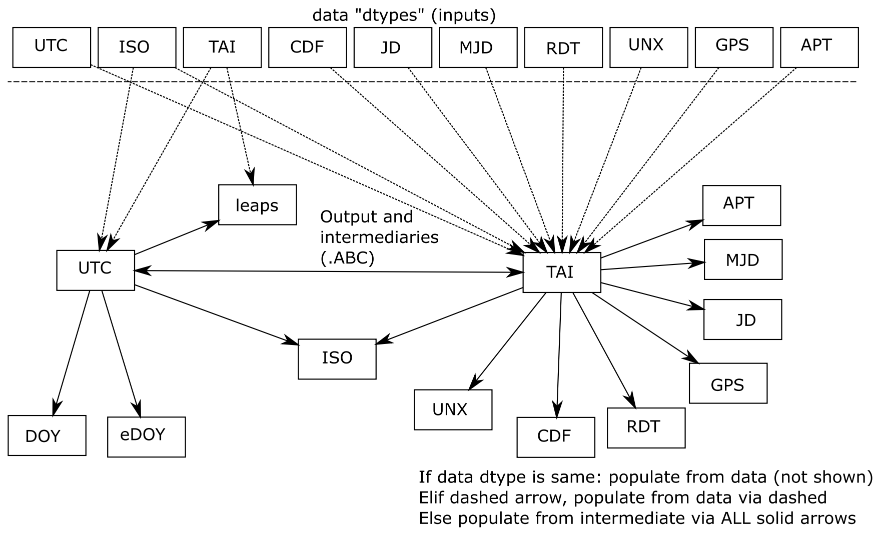
data "dtypes" (inputs)
UTC
ISO
TAI
CDF
JD
MJD
RDT
UNX
GPS
APT
leaps
Output and intermediaries (.ABC)
UTC
TAI
APT
MJD
JD
GPS
ISO
UNX
CDF
RDT
DOY
eDOY
If data dtype is same: populate from data (not shown)
Elif dashed arrow, populate from data via dashed
Else populate from intermediate via ALL solid arrows

FIGURE 2
Workflow for supported input and output types in SpacePy.time.

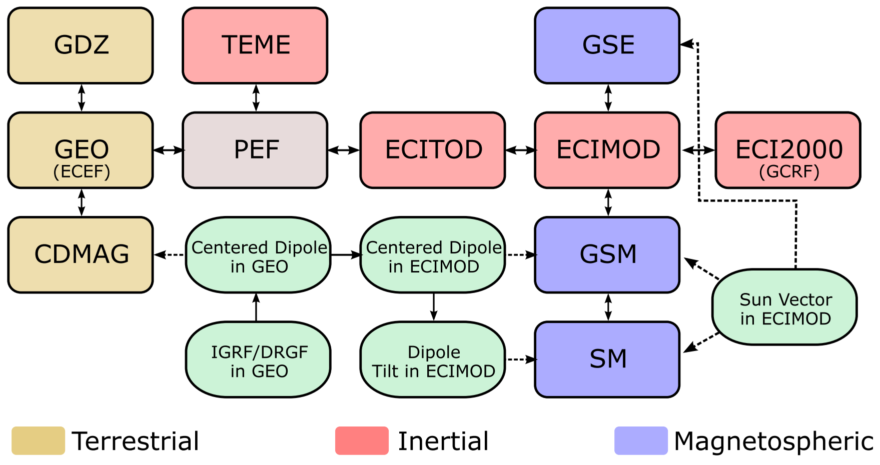

FIGURE 3
Relationships between coordinate systems as implemented in SpacePy.ctrans.

## 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

## 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