PYSAT: Python Satellite Data Analysis Toolkit

R. A. Stoneback , A. G. Burrell , J. Klenzing, M. D. Depew

R. A. Stoneback, W. B. Hanson Center for Space Sciences, 800 W. Campbell Rd. WT 15, Richardson, TX 75080, USA. (rstoneba@utdallas.edu)

A. G. Burrell, W. B. Hanson Center for Space Sciences, 800 W. Campbell Rd. WT 15, Richardson, TX 75080, USA.

J. Klenzing, Space Weather Lab / Code 674, Goddard Space Flight Center, Greenbelt, MD, USA

M. D. Depew, W. B. Hanson Center for Space Sciences, 800 W. Campbell Rd. WT 15, Richardson, TX 75080, USA.

[1]W. B. Hanson Center for Space Sciences, Physics Department, University of Texas at Dallas, Richardson, Texas.

Abstract. A common problem in space science data analysis is combining complementary data sources that are provided and analyzed in different formats and programming languages. The Python Satellite Data Analysis Toolkit (pysat) addresses this issue by providing an open source toolkit that implements the general process of space science data analysis, from beginning to end, in an instrument independent manner. This toolkit uses an Instrument object that enables systematic analysis of science data from a variety of platforms within a single interface. Basic functions such as downloading, loading, and cleaning are included for all supported instruments. Common analysis routines are also included, which are instrument and data source independent. A nano-kernel is used to provide instrument independence, it is attached to the Instrument object and mediates the systematic and arbitrary modification of loaded data. Pysat uses the nano-kernel to improve the rigor of time series analysis, support on-the-fly orbit determination, and cleanly span file breaks. Pysat's functions and higher level scientific analysis features are validated through the use of unit testing. Further adoption by the community provides a set of scientific results produced by a common core, constituting a distributed heritage that supports the validity of the underlying processing and scientific output. These features are used to demonstrate consistency between derived electron density profiles and measured ion drifts, particularly downward ion drifts in the afternoon hours during extreme solar minimum. Pysat builds upon open source Python software that is freely available and encourages community driven development.

1. Introduction

The study of the geospace environment requires a wide variety of measurement techniques and a large number of measurement platforms. The quantity of data itself can be a problem due to the variety of file formats and the unique characteristics of the underlying data. These practical difficulties hinder scientific advancement and result in duplicated efforts, as individual scientists or research groups create their own tools to solve old problems. The scale and impact of these duplicated efforts has become intolerable now that the geospace community has begun to take a system science approach, which requires integrating measurements from multiple platforms to understand the environment as a whole [*CEDAR*, 2010; *Gil et al*, 2016]. Thus, there is a need for a framework to accommodate these varied data sets in an open and reproducible manner, while enabling versatility to pursue various avenues of scientific investigation.

To support these goals, a variety of open source python packages have been released. PyGlow collects a variety of space science models in one place, simplifies installation, and provides a python interface [*Duly et. al.*, 2013]. Apexpy [*Mereen et al*, 2018] and AACGMv2 [*Burrell and Meeren*, 2018] provide interfaces to magnetic field models. OCBPy is a Python module that converts between AACGM coordinates and a magnetic coordinate system that adjusts latitude and local time relative to the Open Closed field line Boundary (OCB) [*Burrell and Chisham*, 2018]. DaViTPy [*DaViTPy*, 2012] provides a suite of tools designed to support the Super Dual Auroral Radar Network (SuperDARN) [*Greenwald et al.*, 1995; *Chisham et al.*, 2007]. GeoData [*Swoboda et al*, 2016] is an API for obtaining and visualizing space science data, with current support for ground based systems.

The Madrigal database, a repository of many space science measurements, has a python interface [*Rideout*, 2004]. SpacePy [*Morely et al*, 2010] includes a variety of tools to support space science, a partial list includes field line tracing, file format support, coordinate conversions, superposed epoch analysis support, and time support functions. pysatCDF [*Stoneback and Depew*, 2018] provides a python interface to NASA CDF libraries and additional functionality to format this data for coupling with pysat. To simplify installation the NASA CDF source code is included within pysatCDF and compiled automatically using standard community tools. A pure python implementation for CDF reading and writing without the use of the NASA C library is under development [*Harter and Liu*, 2018].

The Python Satellite Data Analysis Toolkit (pysat) presented here is an open source software package that handles the tedious details of file and data handling with a consistent front end, allowing researchers to focus on the unique aspects of their scientific research. Pysat's design evolved through years of data analysis using a variety of space and ground based platforms and data types to enable the versatility required to address scientific questions within a single interface. The generalized treatment of data sets and processing by pysat provides the common ground needed to integrate many python package and sources of data and into a cohesive whole that enables system science.

Pysat support begins with assisting users in obtaining data. Each instrument supported by pysat includes routines to download data from appropriate public locations, organize the files on the local computer, and clean the data.

Pysat handles both data and metadata, data about the loaded data, with support for handling files of differing metadata standards in a consistent and robust manner. Even within the same file standard, differing capitalization (case) may be found across files from different teams. Pysat handles metadata in a case preserving manner that is also case-insensitive, enabling ease of use.

To enable the custom processing required by novel scientific investigations pysat includes funcitonality that mediates the application of custom functions upon data as it loaded. This design pattern ensures the availability of the newly processed parameters across all levels of pysat, with no additional effort required by the user.

To ease data distribution, routines have been created that transparently write a pysat Instrument object to disk in a netCDF4 file, as well as load that file and produce the same pysat Instrument object. These routines are written to be consistent with a combined netCDF4 and NASA CDAWeb standard employed by the upcoming NASA ICON mission.

The validity of pysat functions and instrument independent analysis is verified through the use of unit testing. Automated tests have been developed that test instrument support routines, assisting new users in developing new instrument routines, while also ensuring that these routines continue to work. In addition to isolated unit tests that verify specific outputs from isolated functions, simulated instruments have been developed to support the testing of pysat and associated functions as users would interact with the system. As changes are committed to pysat, the test suite is automatically run, ensuring validity and compatibility throughout the development process.

These features support the development and use of instrument independent analysis routines allowing users to focus on the unique aspects of their research project. Pysat's openness to community development also provides a place for researchers to disseminate their analysis routines used in their work. The application of an instrument independent seasonal bin averaging routine is demonstrated here as an example of one such routine, using remote measurements from COSMIC [*Yue et al.*, 2010] and in situ measurements from C/NOFS [*de La Beaujardire et. al.*, 2004]. Despite the large difference in measurement type and data format, the same seasonal routine is used on both platforms. This demonstrates geophysical consistency between the platforms. The use of seasonal bin averaging is widespread in space science data analysis, thus pysat's validated instrument independent implementation of this technique could assist many scientific studies using the same underlying code.

2. Instrument Object

The core functionality of pysat lies in the Instrument object. The intent of the Instrument object is to offer a single interface for interacting with science data that is independent of measurement platform. The layer of abstraction presented by the Instrument object is required for instrument independent analysis procedures, but it can also make science data analysis simpler and more rigorous.

As a simple metaphor, a software object is like a box with buttons. Inside the box the object stores required data and the buttons on the box call methods that understand how to interact with the data and produce the desired products. The pysat Instrument object follows this guideline by storing science data within an object that also includes a number of basic functions designed to load, modify, and analyze the data over arbitrary periods. Data is stored internally in a Python Data Analysis Library (pandas) DataFrame, a format chosen due to its time based array indexing and its ability to align multiple data products. The pandas DataFrame is capable of storing higher dimensional objects, enabling mixed dimensionality data sets [*McKinney*, 2010].

Pysat supports one data set per Instrument object, where a data set is defined as having a single platform, instrument, measurement type, and satellite identifier, as appropriate. Though the particulars of the files and data differ greatly between missions, the interface to the data through the Instrument object remains constant. As an example, consider how to initialize Instrument objects for magnetometer data from the Vector Electric Field Instrument (VEFI) or electron data from the Planar Langmuir Probe (PLP) that flew onboard the Communications/Navigation Outage Forecasting System (C/NOFS), thermal plasma measurement from the Ion Velocity Meter (IVM) also on C/NOFS; Global Positioning Signals (GPS) from the Constellation Observing System for Meteorology, Ionosphere, and Climate (COSMIC) satellites, or high-level, ground-based radar measurements from the northern hemispheric portion of Super Dual Auroral Radar Network (SuperDARN):

```
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b')
ivm = pysat.Instrument(platform='cnofs', name='ivm')
cosmic = pysat.Instrument(platform='cosmic2013', name='gps', tag='ionprf')
darn = pysat.Instrument(platform='superdarn', name='grdex', tag='north')
```

Note how each instrument requires a different level of specificity based on the division of data products within each mission. The full list of supported instruments is available directly in python by inspecting the submodules, as well as within the pysat documentation. Details about the options available for each instrument are stored directly within the code through a python commenting standard called a docstring. These docstrings are automatically collected and presented in the pysat documentation, reflecting the current state of the instrument suite. The pysat documentation is integrated with a continuous documentation service and is automatically generated as versions are released.

For each instrument, pysat looks for supporting routines that understand the unique qualities of the data set and handles the translation into a pysat compatible format. When no existing routines are available, they may be added to pysat. However, if no pysat specific support exists but there are already existing packages to support the loading of a data set, this functionality does not need to be recreated. For example, support for SuperDARN is fundamentally enabled by DaViT Python Project routines, which obtain and load the SuperDARN files.

Pysat support for some public data sources may be generalized. In these cases adding a new instrument to pysat may only involve little effort. Routines have been created for NASA's CDAWeb Common Data Format (CDF) and included with pysat. Pysat's support of C/NOFS' IVM, VEFI, PLP, and NASA's OMNI data sets are all driven by these routines. The only differences in pysat's support for each instrument are the cleaning routines, public data locations, and filename details.

Though the particulars of VEFI magnetometer data, IVM plasma parameters, COSMIC atmospheric measurements, and SuperDARN backscatter measurements are very different, the processes for high level operations on these data are the same. Data for any Instrument may be obtained from data servers intended for public distribution and stored locally by using the 'download' function, and data may be loaded for each instrument using the 'load' function. There are multiple options available when instantiating objects and when loading data that are fully explained in the pysat documentation, but outside the scope of this document.

As mentioned previously, pysat uses the pandas DataFrame to store information internally. The DataFrame is similar to a spreadsheet, possessing labeled columns and rows.

Pysat labels labels columns by the data name and rows by date and time. When operations are performed on the underlying data, row indices are aligned before performing the operation. The loaded data may be accessed at the object level using strings. Support for slicing and other operations is included.

For the one dimensional measurements in time, each column in the pandas DataFrame is a simple indexed array of numbers. However, the pandas DataFrames also support general collections of objects, used here to support higher dimensional data structures, such as the two dimensional electron density profiles from COSMIC. This is shown below for the first four elements of a COSMIC electron density profile. Note that the profile for this single time is also indexed by altitude.

```
In[]: cosmic[0,'profiles']
Out[]:
                 ELEC_dens     GEO_lat
MSL_alt
59.592628   34801.515625   49.295200
62.223614   31884.595703   49.303425
64.850388   32775.335938   49.311638
67.472939   29608.988281   49.319843
```

2.1. MetaData

Maintaining information about the data set is important. Pysat has built-in support to keep track of metadata, stored in a Meta object attached to the Instrument object. Metadata may be accessed by name at the object level, similar to standard data. Metadata may be assigned when data is assigned, or as needed. By default, units, name, notes, description, plot label, axis label, fill value, and plot scaling (linear vs log) are always tracked by the Instrument object, though arbitrary additional parameters may be added. When writing to a file, these metadata parameters are translated into a mixed standard spanning file requirements for the netCDF4 files as well as the International Solar Terrestrial Physics (ISTP) standard employed by NASA's CDAWeb. Parameters that may be determined through simple inspection of the data are not tracked.

To help maintain compatibility with multiple standards, the pysat Meta object allows for user specified string labels to identify particular metdata types (fill, units, notes). As an example for fill values, netCDF4 files should use '_FillValue' while ISTP specifies 'FillVal'. Case is preserved for these labels however data access is case-insensitive, thus 'units' works in code even if the label is strictly 'Units'. Label independent access is also provided, thus users can use attributes attached to the pysat Meta object to access the desired metadata type without specifying the string used to label those values.

2.2. Modifying Data

Frequently, data sets need to be modified before a larger analysis may be completed. Instrument specific modifications are handled in pysat by a nano-kernel with a custom processing queue. Functions may be added to the queue as needed, and whenever new data is loaded the nano-kernel will apply the ordered functions before making the data available to the user. This configuration ensures that the newly calculated data has the same properties and availability as parameters that are native to the file. A data cleaning example is shown below. This code segment selects only VEFI magnetometer measurements made at times when the magnetic torque rods on the spacecraft, used for momentum control, were not contaminating the magnetic field environment.

```
# define function to remove flagged values
def filter_vefi(vefi):
    idx, = np.where(vefi['B_flag']==0)
    vefi.data = vefi[idx, :]
    return
# add custom function to instrument
vefi.custom.add(filter_vefi, 'modify')
```

Once added, this function will filter the available VEFI data, modifying it in place, every time vefi.load is called.

2.3. Data Flow

The full data flow through the Instrument object when a load call is invoked is shown in Figure 1. Instrument specific functions (purple) translate the specifics of the given data set into a format suitable for pysat. Options and other parameters provided by the user are supplied as needed, shown in green. Pysat invokes the instrument specific functions as needed to provide the user with data in the desired form. Functions handled by pysat are shown in blue.

3. Generalized Space Science Solutions

Pysat builds upon the consistent object interface across data sets to enable generalized solutions for space science data analysis and visualization. At a basic level, all analysis routines that use the pysat Instrument object gain some independence from the particulars of the analyzed data. For example, string based data access makes it easy to support programmatic use of data. This and other pysat features allow many analytical processes to be generalized. Several examples of pysat generalized solutions to common space science data problems are discussed in this section.

3.1. Recreating Continuous Data from Files

Measurements of continuous processes by scientific instruments are eventually divided into chunks and stored separately on a file system. These file boundaries can interfere with calculations, particularly for those times near the file edges. For example, consider a simple centered smoothing filter that averages a set number of measurements in time. The start and end of the time series will not have enough samples to obtain the result. To handle this problem, as well as the possibility of data gaps, a user must choose how to balance the quality and coverage of the output.

Pysat offers a solution to the problem of file breaks in a data set that requires no specific support by any user supplied routine. When activated, pysat maintains an internal data buffer that spans three files/days, depending upon user selected parameters. Each time a user loads data, pysat centers the data buffer on the requested time, downselects from this full buffer the requested data plus a user specified amount of data padding, applies any user directed custom functions, and then removes the padded data before making the full results available to the user (see Figure 1). This solution does not fix the calculation everywhere, but rather pushes the boundaries where the calculation degrades outside the desired time range and then removes the degraded calculations.

The resulting output is equivalent to a continuous data set, barring measurement gaps. The time period for this padding is arbitrary up to a maximum additional file or day. While this limits the maximum continuous data period available for a time based calculation, shorter period calculations may be applied without error over an effectively infinite time series while only using a small amount of computer memory. Applying this feature over N days only requires N+2 loads from the filesystem. Custom functions applied by the nano-kernel when data padding is enabled do not need to explicitly support the feature, as the data padding is removed after the custom functions are applied.

3.2. Iterating over Time Periods and Orbits

Seasons are one of the natural temporal divisions of geophysical data. To assist the production of seasonally averaged pictures of the upper atmosphere, the temporal analysis loop can be used to load data for a specified range of dates, one file at a time, and operated on as needed for the desired analysis. The temporal analysis loop is a special case of the iteration that is built into the pysat Instrument object. A simple pair of dates may be set for a single season, or a range of dates may be provided for a more distributed temporal analysis. The iteration is activated through standard Python functionality, using the same mechanism employed when iterating over Python list elements. Each loop triggers a load data call on the pysat Instrument object for the next day of data within the desired date range.

This basic data iteration support is sufficient for daily or orbit-based analysis of science data sets. Since not all data sets are stored by day, pysat includes functionality to parse from multiple files the data that corresponds to the requested day. Similarly, pysat is designed to support real-time determination of orbit breaks from the data set, and then iterate over these orbits as desired. Orbits that cross file boundaries are handled using the pysat Instrument's iterative functions, moving forward or backward within the data to determine if the desired orbit begins or ends across one of these filebreaks and then includes the appropriate data.

This combination of features makes it straightforward to make an orbit by orbit plot for any of the satellite missions supported by pysat. A simple code example for plotting the entirety of the VEFI data set by orbit is shown below, while the results are shown in Figure 2. For this particular example the orbits are set to begin and end at 0° geographic longitude.

```
# instantiate instrument with desired orbit breakdown
orbit_info = {'index':'longitude', 'kind':'longitude'}
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b',
                        clean_level=None, orbit_info=orbit_info)
# iterate over dataset, orbit by orbit
for count, vefi in enumerate(vefi.orbits):
    # One orbit of data is now accessible via vefi.data
    # To ensure data gaps do not have a line drawn across the gap,
    # resample data onto constant 1 second cadence, filling in gaps with NaN
    vefi.data = vefi.data.resample('1S', fill_method='ffill',
                                   limit=1, label='left' )
    # data is ready to be plotted
```

3.3. Instrument Independent Seasonal Analysis

Pysat functionality has been used to develop several seasonal analysis routines that are instrument and iteration independent. An example using a pysat occurrence probability routine is shown, reproducing the fundamental processing used to obtain published results by *Stoneback and Heelis* [2014]. Note that the pysat analysis covers all of the data loading, iteration, and analysis. No specific support for VEFI was included in the routine. The routine calculates the number of times a given value exceeds a supplied threshold at least once per temporal period (day, file, or orbit), divided by the number of times a given spatial bin is visited per temporal period. As a demonstration, the probability of a positive perturbation in the meridional component of the geomagnetic field by orbit is shown over a week for VEFI in Figure 3.

To help ensure the plotted data is geophysical, the VEFI torque rod exclusion function introduced earlier is attached to a VEFI pysat object. This function selects data when magnetic torquers on C/NOFS were idle. The torque rod firings interfered with the electromagnetic measurements and are generally located near the magnetic equator. The reduction in counts in Figure 3(b) along the magnetic equator demonstrates that the custom function is properly selecting data. The code to produce Figure 3 follows:

```
# instantiate instrument with desired orbit breakdown
orbit_info = {'index':'longitude', 'kind':'longitude'}
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b',
                        clean_level=None, orbit_info=orbit_info)
# add custom torque rod filter function to instrument
vefi.custom.add(filter_vefi, 'modify')
# set time limits on data analysis
start = pysat.datetime(2010,5,9)
stop = pysat.datetime(2010,5,15)
# download data for specified time period
vefi.download(start, stop)
# perform probability calculation. Any data added by custom functions is
```

```
# available within routine below
ans = ssnl.occur_prob.by_orbit2D(vefi, [0,360,144], 'longitude',
        [-13,13,104], 'latitude', ['dB_mer'], [0.], returnBins=True)
# a dictionary object with keys corresponding
# to data labels is returned
# results are ready for plotting
```

Pysat also includes generalized seasonal analysis routines that support averaging multiple instrument parameters of various dimensionality over a season. Here we use this functionality to average both IVM and COSMIC data, enabling comparisons between the average distributions of ion density and ion drift. The same general process used to obtain the VEFI occurrence probability is used; both IVM and COSMIC pysat objects are instantiated and passed along to the seasonal pysat routines for analysis. The COSMIC data set does not come with location information in geomagnetic coordinates, or with information on the topside scale height, so these parameters are calculated using custom functions and applied to the data set automatically using the nano-kernel functionality. The nano-kernel functionality ensures that the custom COSMIC parameters are available for averaging within the seasonal bin averaging routine.

Figure 4 includes ion drift measurements from IVM and electron density profile parameters from COSMIC, seasonally averaged over apex longitude and local time. The use of apex longitude organizes the data based upon the apex location of the geomagnetic field line at the measurement location. IVM derived vertical ion drifts are at the top followed by the COSMIC derived ion density maximum, height of the density maximum, and the thickness of the density distribution. This ion drift average displays downward afternoon ion drifts, a characteristic of the ionosphere during very low solar activity levels *Stoneback et al* [2011]. These ion drifts employ a geophysically motivated calibration to appropriately set the zero ion drift level used when translating raw IVM measurements to geophysical ion drifts [*Stoneback et al*, 2011].

In the late afternoon and evening sector, longitudinal and local time variations in the meridional ion drift recorded by IVM have equivalent variations in the altitude of the density maximum recorded by COSMIC. A strong correlation between drifts and density is not expected during the morning through afternoon as plasma production from sunlight is a dominant driver of density. In the late afternoon and evening hours, when plasma production and loss processes are small or nearly equal, redistribution of the plasma to different altitudes through transport by ion drifts are expected to have a measurable impact upon the ionosphere. The results in Figure 4 between 15 - 24 local time have a strong apparent correlation between areas with upward (downward) ion drifts and an increase (decrease) in the height of the density maximum across all longitudes.

The full electron density profiles from COSMIC are shown in Figure 5 and correspond to the first four longitude sectors ($0° - 60°$) in Figure 4. The first two longitude sectors (top panel) have upward slants in the bottomside density distribution at night, consistent with the upward drifts after sunset in Figure 4. In contrast, the bottom two panels, show longitudes associated with downward drifts in the evening and have flat bottomside ion distributions at night. These changes in the bottomside density profiles are consistent with the meridional plasma drift, because a negative drift moves plasma to lower altitude field lines with higher neutral densities, where loss processes rise exponentially. This effectively produces a minimum viable altitude for the nighttime ionosphere. The consistency demonstrated between IVM and COSMIC measurements provides supporting evidence that both platforms are reporting measurements with geophysical significance that have been analyzed in a consistent and appropriate manner. The same generalized seasonal analysis code was used for both IVM and COSMIC. The complete code to produce these figures is included in the pysat repository under demos.

3.4. Validating Results

To validate space science results, both the code and the underlying data must be tested. A suite of unit tests have been developed to help ensure robust performance of pysat and its features. These tests initialize the system in a known state, perform a limited set of operations, and then compare the result of those operations against a known output. The pysat development repository is connected to a continuous integration service, which runs the test suite after every change to the codebase. Currently, 460 unit tests cover 82% of pysat's code, as determined using standard community tools. Basic tests cover options for instantiating the pysat Instrument object and its handling of data, metadata, and files.

To facilitate the testing of pysat features that require science data, such as the nano-kernel support, orbit-by-orbit iteration, and instrument independent analysis functions, testing instrument platforms are also included. These pysat test instruments operate like a normal pysat Instrument object. However, the typical load routines that read science data from the filesystem are replaced with a basic simulation of satellite motion. Signals representing the large scale periodic features of local time, longitude, latitude, altitude, etc. are generated. These routines produce infinite continuous streams of reproducible data that may be used as known inputs in a unit testing framework.

The general process of determining orbits from a random science data set faces a number of data and file issues. Accounting for these variables, along with the various input options that can be selected, required a significant testing suite. Iterating by orbit requires, in part, determining where orbit breaks occur, completing orbits across file breaks, accounting for data gaps, and ensuring consistent orbit numbering. To cover all of these options a general class of orbit tests were created that produced a wide range of data and file gaps. Each problem type is expanded upon to ensure coverage for edge and pathological cases. This whole suite of tests is then run using each type of orbit pysat supports (local time, longitude, latitude, orbit number), ensuring that loading or processing data by orbit will not affect the scientific analysis.

Unit tests have also been developed to monitor the instrument specific routines that download, load, and clean science data as part of the pysat process. Each run of the unit testing suite downloads, loads, and cleans test days for pysat supported instruments using data obtained from the appropriate public data source. General pysat compliance is also checked, assisting users developing code to support new instruments. Some instruments have to be excluded from parts of the testing process, as access to the data sources requires authentication.

3.5. Creating Data for Distribution

Creating data sets suitable for distribution has remained a challenge. Files that are easy to put together lack the metadata for a self-supporting specification of the data. Formats that are capable of storing a wide variety of data, formats, and metadata generally require significant effort to provide this information. Pysat approaches this issue on two fronts. Pysat includes metadata by default. Thus, as a routine to create a dataset is written, both the data and metadata may be specified naturally. Following best coding practices, the data specification work is distributed across the whole development effort.

When a pysat Instrument object loads data both the instrument data and metadata are pulled from the file and attached to the pysat Instrument object. File routines have been created to reverse this process and transparently store a pysat object to disk as a netCDF. As many different data schemes may be stored within pysat, a translation layer has been developed that stores the data in the netCDF in a format intuitive to humans. A complementary netCDF load routine is also included with pysat, making it possible to recover the original pysat Instrument object state without any additional processing. Recovery back to the original pysat object relies upon a variable naming pattern, and thus is not guaranteed for non-pysat netCDFs.

4. Future Possibilities

Pysat provides a systematic and versatile framework for the arbitrary modification and analysis of data. A selection of instruments and analyses are included that currently reflect the research interests of the authors. The list is not exhaustive. Since instrument data types ranging from in situ satellite data, satellite based remote sensing data, and ground based data have already been successfully integrated into pysat, a wide range of instruments are expected to be supported without significant changes to pysat's structure. With community support, the full range of space science data sets could be available from a single, consistent interface. Additional analysis types, such as superposed epoch analysis, can also be added to pysat.

Pysat's support for test instruments and inclusion of unit testing provides a mechanism to validate analysis code. Details of exhaustive test procedures are not typically included in scientific publications, limiting the ability of the audience to audit the analysis. Adoption of open source analyses such as pysat by the community provides a verifiable code standard that minimizes both the effort required by the author, as well as the innate level of trust required by the reader. The sum total of publications based upon pysat code provides a heritage base that supports future publications. This can be of particular importance for analyses that produce controversial results.

Pysat's structure enables a common ground and a single interface for all space science data sets. This does not preclude the development and use of instrument specific packages, as desired by the community. In these situations pysat can and will make use of the instrument specific tools when adding support for that instrument. In most cases, a thin translation layer from the native data format to the pandas DataFrame will provide the majority of the required functionality. A pair of functions that translate the data back and forth between standards would even enable the use of instrument specific processing functions from within pysat. While the instrument specific package may be optimal for primary instrument users, outside users could utilize the standard interface provided by pysat and still benefit from the creation of the instrument specific tools.

The range of file management features required to support pysat also provide an underlying basis for a CubeSat data processing system. While Explorer-level missions supported by NASA typically have enough funds to produce a dedicated software ecosystem to support the processing of data, funding levels typically employed for CubeSats are insufficient for this level of software development. Pysat provides a foundation for file and data processing management that reduces the workload required to create a system capable of delivering upon the science goals of the mission. If leading CubeSat missions are willing to use pysat for this purpose as well as contribute code back to the repository, this community resource could increase both the dollar and science efficiency of future CubeSat missions.

The functions provided by pysat constitute the underlying functionality needed to drive a Graphical User Interface (GUI) for easy visualization of data. In this scenario if a user finds something interesting visually and wanted to complete a more rigorous analysis the exact same tools would be available at the command line, providing continuity for scientific analysis. Given that user interface requirements can differ significantly based upon the analysis or instrument type, a range of specialized GUIs all powered by the same underlying pysat code would be ideal.

Functionality provided by pysat also supports the creation of a Constellation object, a heterogenous collection of pysat Instrument objects. This abstraction will allow custom collections of instruments to be operated upon as a whole. As the processing required for each instrument within the constellation could be unique, custom functions may be attached to the Constellation object and applied to individual instruments automatically. Analysis functions and orbit determination on the constellation level are also planned.

5. Conclusion

Pysat provides a systematic process for custom analysis of science data sets. The pysat Instrument object enables a complex flow for each user request of data, providing for an arbitrary relationships between the requested and archived data. This processing flow is used to solve problems associated with multiple data sets, data distribution in files, accurate time-series calculations, orbit determination, data modification, and the calculation of new scientific products. The combination of the pysat Instrument object, pandas DataFrame, and this computational versatility enables instrument independent analysis and simplifies the comparison of results across data sets. These features are expected to be sufficient to enable integration of data sets across space science into a single common platform.

The adoption of unit testing across the package provides a verification chain to ensure results are robust. Tests are applied to the Instrument object as well as the higher order analysis routines (seasonal bin averaging, etc.) The public availability of both the code and the tests provides a mechanism for verifiable and reproducible science. Should pysat be adopted by the wider community, additional validation is gained as scientists use and individually verify the tools as part of their own research. Thus, scientific papers that incorporate pysat not only benefit from the heritage established by previous use, each new use of pysat also provides validation that the outputs provided by pysat are scientifically valid.

Pysat is being used as a foundational framework for ground station processing of IVM measurements for the upcoming ICON and COSMIC-2 missions. While work is still underway, pysat has been integrated by both CDAAC and the Berkeley ground software system in anticipation of these missions. The data flow generated by these missions will provide a strong heritage that future missions and science data analyses can build upon.

Acknowledgments. Supported by NSF grant 125908 and NASA grant NNX10AT02G.
The data used in this article may be obtained at:

CDAWeb: http://cdaweb.gsfc.nasa.gov

CDF Library: http://cdf.gsfc.nasa.gov.

CDAAC: http://cosmic-io.cosmic.ucar.edu/cdaac/index.html pysat : https://github.com/rstoneback/pysat

pysatCDF : https://github.com/rstoneback/pysatCDF

DaViTpy : https://github.com/vtsuperdarn/davitpy

References

Burrell, A., Meeren, C., Laundal, K. (2018, April 5). aacgmv2: v2.4.1 (Version 2.4.1). Zenodo. http://doi.org/10.5281/zenodo.1212695

Burrell, A., Chisham, G.. (2018, February 19). aburrell/ocbpy: Beta Release (Version 0.2b1). Zenodo. http://doi.org/10.5281/zenodo.1217177

CEDAR: The New Dimension

Chisham, G. et al. (2007), A decade of the Super Dual Auroral Radar Network (SuperDARN): scientific achievements, new techniques and future directions, Surv. Geophys., 28(1), 33?109, doi:10.1007/s10712-007-9017-8.

(2012) Data and Visualization Toolkit-Python for SuperDARN, https://github.com/vtsuperdarn/davitpy

de La Beaujardire, O. et. al. (2004), C/NOFS: a mission to forecast scintillations, Journal of Atmospheric and Solar-Terrestrial Physics ,66(17):1573?1591, doi: 10.1016/j.jastp.2004.07.030

Duly, T., et al (2013), PyGlow: Upper atmosphere climatological models in Python, https://github.com/timduly4/pyglow

Gil, Y., et al. (2016), Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance, *Earth and*

Space Science, 3, 388?415, doi:10.1002/2015EA000136.

Greenwald, R. A. et al. (1995), DARN/SUPERDARN, Space Sci Rev, 71(1-4), 761?796, doi:10.1007/BF00751350.

Harter, B. and Liu, M., https://github.com/MAVENSDC/cdflib

Hunter, J. (2007), Matplotlib: A 2d graphics environment, *Computing in Science Engineering*, *9*(3), 90–95, doi:10.1109/MCSE.2007.55.

Jones, E., T. Oliphant, P. Peterson, et al. (2001–), SciPy: Open source scientific tools for Python, [Online; accessed 2016-02-19].

McKinney, W. (2010), Data structures for statistical computing in python, *Proceedings of the 9th Python in Science Conference*, edited by S. van der Walt and J. Millman, 51 – 56.

Meeren, C., Burrell, A. G., and Laundal, K. (2018, April 6). apexpy: ApexPy Version 1.0.3 (Version 1.0.3). Zenodo. http://doi.org/10.5281/zenodo.1214207

Morley, S., Welling, D., Koller, J., Larsen, B., Henderson, M., Niehof, J., SpacePy - A Python-based Library of Tools for the Space Sciences, *Proceedings of the 9th Python in Science Conference*, edited by S. van der Walt and J. Millman, 39 – 45.

Prez, F., and B. Granger (2007), iPython: A system for interactive scientific computing, *Computing in Science Engineering*, *9*(3), 21–29, doi:10.1109/MCSE.2007.53.

Rideout, B. (2004), Open Madrigal Initiative, http://madrigal.haystack.edu/madrigal/madDownload.html

Stoneback, R. A., R. A. Heelis, A. G. Burrell, W. R. Coley, B. G. Fejer, and E. Pacheco (2011), Observations of quiet time vertical ion drift in the equatorial ionosphere during the solar minimum period of 2009, *J. Geophys. Res.*, 116, A12327, doi: 10.1029/2011JA016712.

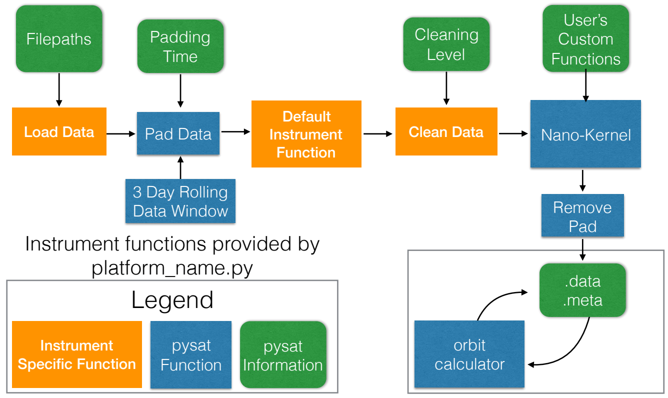

Figure 1. Pysat program flow when an pysat.Instrument.load() routine is called.

Stoneback, R. A. and Heelis, R. A. (2007) Identifying equatorial ionospheric irregularities using in situ ion drifts, *Ann. Geophys.*, doi:https://doi.org/10.5194/angeo-32-421-2014.

Stoneback, R. A. and Depew, M. (2018). rstoneback/pysatCDF: Windows compatibility and improved pysat meta handling (Version 0.3.0). Zenodo. doi: http://doi.org/10.5281/zenodo.1217181

Swoboda, J., Hirsch, M., Stuhlmacher, A., Starr, G., Semeter, J. (2016). jswoboda/GeoDataPython: ISR Sim Paper (Version v0.1). Zenodo. doi: http://doi.org/10.5281/zenodo.154533

van der Walt, S., S. Colbert, and G. Varoquaux (2011), The numpy array: A structure for efficient numerical computation, *Computing in Science Engineering*, *13*(2), 22–30, doi:10.1109/MCSE.2011.37.

Yue, X., W. S. Schreiner, J. Lei, S. V. Sokolovskiy, C. Rocken, D. C. Hunt, and Y.-H. Kuo (2010), Error analysis of Abel retrieved electron density profiles from radio occultation measurements, Annales Geophysicae, 28(1), 217?222.

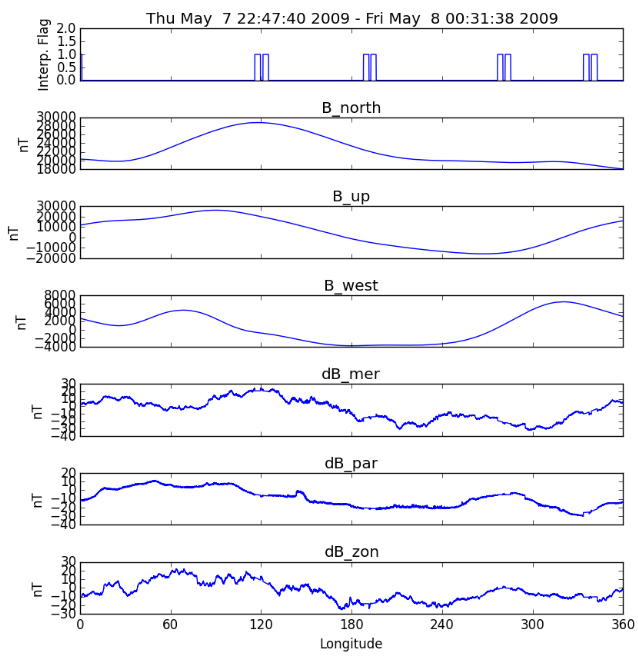

Figure 2. Sample orbit figure using VEFI data, where the interp flag reflects times when spacecraft operations could interfere with measurements.

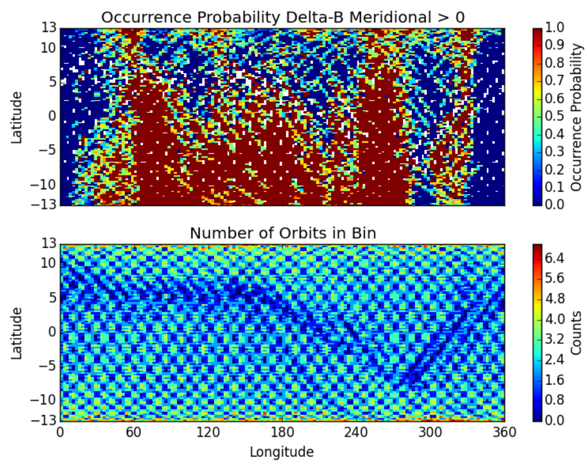

Figure 3. Seasonal occurrence probability demo using VEFI data. The location of the magnetic equator may be seen in the lower data counts for the data distribution.

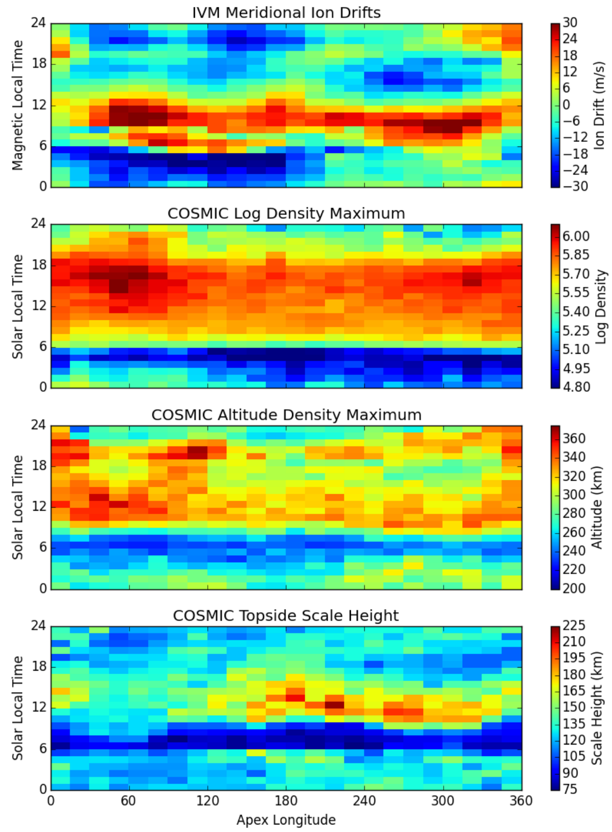

Figure 4. Comparison of seasonal averages of IVM meridional ion drifts (top) and averages of COSMIC profiles covering NmF2, hmF2, and topside scale height.

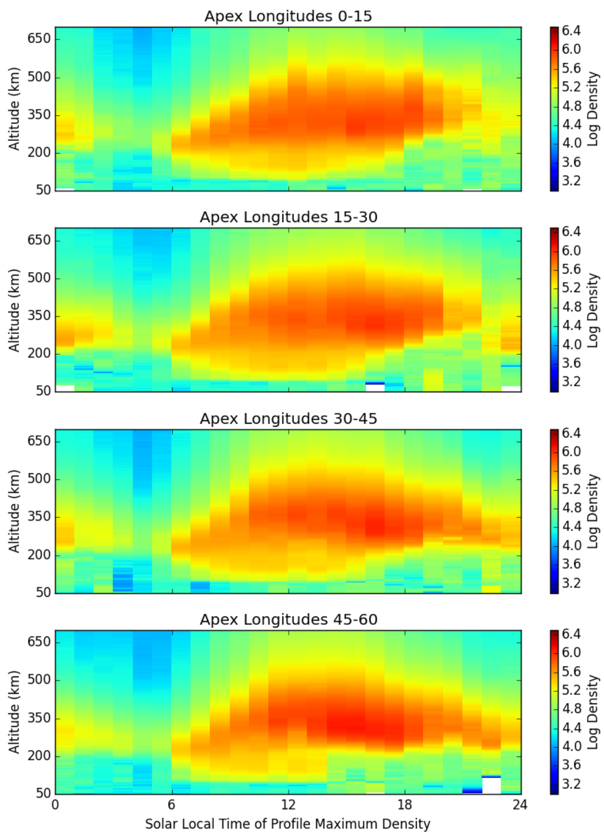

Figure 5. Seasonal average of COSMIC electron density profiles.