pysatSeasons Documentation

Release 0.2.0-alpha

Klenzing, Jeffrey, Stoneback, Russell, Spence, Carey, Burrell, Ang

Aug 31, 2022

CONTENTS

This documentation describes the pysatSeasons module, which contains routines to load data via pysat.Instrument objects and perform seasonal analysis routines such as bin averaging, occurrence probabilities, and scatter plots.

CHAPTER

ONE

OVERVIEW

Seasonal analysis is one of the most common analysis performed in space science. This package leverages pysat features to create generalized methods for binning and averaging data, calculating occurrence probabilities (by day or by orbit), or understanding the scatter within a data set.

CHAPTER

TWO

INSTALLATION

The following instructions will allow you to install pysatSeasons.

2.1 Prerequisites

pysatSeasons uses common Python modules, as well as modules developed by and for the Space Physics community. This module officially supports Python 3.6+.

Common modules	Community modules
matplotlib	pysat
numpy	
pandas	
xarray	

2.2 Installation Options

1. Clone the git repository

```
git clone https://github.com/pysat/pysatSeasons.git
```

2. Install pysatSeasons: Change directories into the repository folder and run the setup.py file. There are a few ways you can do this:

   A. Install on the system (root privileges required):

   ```
   sudo python3 setup.py install
   ```

   B. Install at the user level:

```
python3 setup.py install --user
```

C. Install with the intent to develop locally:

```
python3 setup.py develop --user
```

CHAPTER

THREE

CITATION GUIDELINES

When publishing work that uses pysatSeasons, please cite the package and any package it depends on that plays an important role in your analysis. Specifying which version of pysatSeasons used will also improve the reproducibility of your presented results.

3.1 pysatSeasons

- Klenzing, J. H., R. Stoneback, C. Spence, and A. G. Burrell. (2021). pysat/pysatSeasons: Version 0.1.3 (v0.1.3). Zenodo. https://doi.org/10.5281/zenodo.4950172

```
@Misc{pysatSeasons,
      author = {Klenzing, J. H. and Stoneback, R. and Spence, C. and Burrell, A. G.},
      title  = {pysat/pysatSeasons: vX.Y.Z},
      year   = {2022},
      doi    = {10.5281/zenodo.3475493},
      url    = {https://doi.org/10.5281/zenodo.3475493},
      }
```

CHAPTER

FOUR

API

4.1 Bin Averaging

Instrument independent seasonal averaging routine.

Supports bin averaging N-dimensional data over 1D and 2D bin distributions.

pysatSeasons.avg.**mean_by_day**(*inst*, *data_label*)

Calculate mean of *data_label* by day over *Instrument.bounds*.

Parameters

- **inst** (*pysat.Instrument*) – Instrument object to perform mean upon.
- **data_label** (*str*) – Data product label to be averaged.

Returns

mean – Mean of *data_label* indexed by day.

Return type

pandas.Series

Note: The range of dates to be loaded, and the cadence used to load data over that range, is controlled by the *inst.bounds* attribute.

---

pysatSeasons.avg.**mean_by_file**(*inst*, *data_label*)

Calculate mean of *data_label* by orbit over Instrument.bounds.

Parameters

- **inst** (*pysat.Instrument*) – Instrument object to perform mean upon.
- **data_label** (*str*) – Data product label to be averaged.

Returns

mean – Mean of *data_label* indexed by start of each file.

Return type

pandas.Series

Note: The range of dates to be loaded, and the cadence used to load data over that range, is controlled by the *inst.bounds* attribute.

---

pysatSeasons.avg.**mean_by_orbit**(*inst*, *data_label*)

Calculate mean of *data_label* by orbit over Instrument.bounds.

Parameters

- **inst** (*pysat.Instrument*) – Instrument object to perform mean upon.
- **data_label** (*str*) – Data product label to be averaged.

Returns

mean – Mean of *data_label* indexed by start of each orbit.

Return type

pandas.Series

Note: The range of dates to be loaded, and the cadence used to load data over that range, is controlled by the *inst.bounds* attribute.

---

pysatSeasons.avg.**median1D**(*const*, *bin1*, *label1*, *data_label*, *auto_bin=True*, *returnData=None*, *return_data=False*)

Calculate a 1D median of nD *data_label* over time binned by *label1*.

Parameters

- **const** (*Constellation or Instrument*) – Constellation or Instrument object.
- **bin1** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **label1** (*str*) – Identifies data product for bin1.
- **data_label** (*list-like*) – Strings identifying data product(s) to be averaged.
- **auto_bin** (*bool*) – If True, function will create bins from the min, max and number of bins. If false, bin edges must be manually entered in *bin1*. (default=True)
- **returnData** (*bool or NoneType*) – If True, also return binned data used to calculate the average in the output dictionary as 'data', in addition to the statistical outputs. Deprecated in favor of *return_data*. (default=None)
- **return_data** (*bool*) – If True, also return binned data used to calculate the average in the output dictionary as 'data', in addition to the statistical outputs. (default=False)

Returns

median – 1D median accessed by *data_label* as a function of *label1* over the season delineated by bounds of passed instrument objects. Also includes 'count' and 'avg_abs_dev' as well as the values of the bin edges in 'bin_x'. If returnData True, then binned data stored under 'data' under *data_label*.

Return type

dict

Note: The range of dates to be loaded, and the cadence used to load data over that range, is controlled by the *const.bounds* attribute.

---

pysatSeasons.avg.**median2D**(*const*, *bin1*, *label1*, *bin2*, *label2*, *data_label*, *returnData=None*, *auto_bin=True*, *return_data=False*)

Calculate 2D median of nD *data_label* over time and *label1 label2*.

Parameters

- **const** (*pysat.Constellation or Instrument*) –
- **bin1** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **bin2** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **label1** (*str*) – Identifies data product for binning.
- **label2** (*str*) – Identifies data product for binning.
- **data_label** (*list-like*) – Strings identifying data product(s) to be averaged.
- **returnData** (*bool or NoneType*) – If True, also return binned data used to calculate the average in the output dictionary as 'data', in addition to the statistical outputs. Deprecated in favor of *return_data*. (default=None)
- **auto_bin** (*bool*) – If True, function will create bins from the min, max and number of bins. If false, bin edges must be manually entered in *bin**. (default=True)
- **return_data** (*bool*) – If True, also return binned data used to calculate the average in the output dictionary as 'data', in addition to the statistical outputs. (default=False)

Returns

median – 2D median accessed by data_label as a function of label1 and label2 over the season delineated by bounds of passed instrument objects. Also includes 'count' and 'avg_abs_dev' as well as the values of the bin edges in 'bin_x' and 'bin_y'.

Return type

dict

Note: The range of dates to be loaded, and the cadence used to load data over that range, is controlled by the *const.bounds* attribute.

4.2 Occurrence Probability

Occurrence probability routines, daily or by orbit.

Routines calculate the occurrence of an event greater than a supplied gate occurring at least once per day, or once per orbit. The probability is calculated as the (number of times with at least one hit in bin) / (number of times in the bin). The data used to determine the occurrence must be 1D. If a property of a 2D or higher dataset is needed attach a custom function that performs the check and returns a 1D Series.

Note: The included routines use the bounds attached to the supplied instrument object as the season of interest.

---

pysatSeasons.occur_prob.**by_orbit2D**(*const*, *bin1*, *label1*, *bin2*, *label2*, *data_label*, *gate*, *returnBins=None*, *return_bins=False*)

2D Occurrence Probability of *data_label* orbit-by-orbit over a season.

Deprecated since version 0.2.0: *returnBins* will be removed in v0.3.0, it is replaced by *return_bins* because it has standards compliant case.

If *data_label* is greater than *gate* at least once per orbit, then a 100% occurrence probability results. Season delineated by the bounds attached to Instrument object. Probability = (# of times with at least one hit) / (# of times in bin)

Parameters

- **const** (*pysat.Instrument or pysat.Constellation*) – Instrument/Constellation to use for calculating occurrence probability.
- **bin1** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **bin2** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **label1** (*str*) – Identifies data product for binX.
- **label2** (*str*) – Identifies data product for binX.
- **data_label** (*list of str*) – Data product label(s) to calculate occurrence probability.
- **gate** (*list of values*) – Values that *data_label* must achieve to be counted as an occurrence.
- **returnBins** (*bool or NoneType*) – If True, also return arrays with values of bin edges, useful for pcolor. Deprecated in favor of *return_bins*. (default=None)
- **return_bins** (*bool*) – If True, also return arrays with values of bin edges, useful for pcolor. (default=False)

Returns

occur_prob – A dict of dicts indexed by data_label. Each entry is dict with entries 'prob' for the probability and 'count' for the number of orbits with any data; 'bin_x' and 'bin_y' are also returned if requested. Note that arrays are organized for direct plotting, y values along rows, x along columns.

Return type

dict

Note: Season delineated by the bounds attached to Instrument object.

---

pysatSeasons.occur_prob.**by_orbit3D**(*const*, *bin1*, *label1*, *bin2*, *label2*, *bin3*, *label3*, *data_label*, *gate*, *returnBins=None*, *return_bins=False*)

3D Occurrence Probability of *data_label* orbit-by-orbit over a season.

Deprecated since version 0.2.0: *returnBins* will be removed in v0.3.0, it is replaced by *return_bins* because it has standards compliant case.

If *data_label* is greater than *gate* at least once per orbit, then a 100% occurrence probability results. Season delineated by the bounds attached to Instrument object. Prob = (# of times with at least one hit) / (# of times in bin)

Parameters

- **const** (*pysat.Instrument or pysat.Constellation*) – Instrument/Constellation to use for calculating occurrence probability.
- **bin1** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **bin2** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.

- **bin3** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **label1** (*str*) – Identifies data product for binX.
- **label2** (*str*) – Identifies data product for binX.
- **label3** (*str*) – Identifies data product for binX.
- **data_label** (*list of str*) – Identifies data product(s) to calculate occurrence probability e.g. inst[data_label].
- **gate** (*list of values*) – Values that *data_label* must achieve to be counted as an occurrence.
- **returnBins** (*bool or NoneType*) – If True, also return arrays with values of bin edges, useful for pcolor. Deprecated in favor of *return_bins*. (default=None)
- **return_bins** (*bool*) – If True, also return arrays with values of bin edges, useful for pcolor. (default=False)

Returns

occur_prob – A dict of dicts indexed by data_label. Each entry is dict with entries 'prob' for the probability and 'count' for the number of orbits with any data; 'bin_x', 'bin_y', and 'bin_z' are also returned if requested. Note that arrays are organized for direct plotting, z, y, x.

Return type

dict

Note: Season delineated by the bounds attached to Instrument object.

---

pysatSeasons.occur_prob.**daily2D**(*const*, *bin1*, *label1*, *bin2*, *label2*, *data_label*, *gate*, *returnBins=None*, *return_bins=False*)

2D Daily Occurrence Probability of *data_label* > *gate* over a season.

Deprecated since version 0.2.0: *returnBins* will be removed in v0.3.0, it is replaced by *return_bins* because it has standards compliant case.

If *data_label* is greater than *gate* at least once per day, then a 100% occurrence probability results. Season delineated by the bounds attached to Instrument object. Probability = (# of times with at least one hit) / (# of times in bin)

Parameters

- **const** (*pysat.Instrument or pysat.Constellation*) – Instrument/Constellation to use for calculating occurrence probability.
- **bin1** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **bin2** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **label1** (*str*) – Identifies data product for binX.
- **label2** (*str*) – Identifies data product for binX.
- **data_label** (*list of str*) – Identifies data product(s) to calculate occurrence probability e.g. inst[data_label].
- **gate** (*list of values*) – Values that *data_label* must achieve to be counted as an occurrence.

- **returnBins** (*bool or NoneType*) – If True, also return arrays with values of bin edges, useful for pcolor. Deprecated in favor of *return_bins*. (default=None)
- **return_bins** (*bool*) – If True, also return arrays with values of bin edges, useful for pcolor. (default=False)

Returns

occur_prob – A dict of dicts indexed by *data_label*. Each entry is dict with entries 'prob' for the probability and 'count' for the number of days with any data; 'bin_x' and 'bin_y' are also returned if requested. Note that arrays are organized for direct plotting, y values along rows, x along columns.

Return type

dict

Note: Season delineated by the bounds attached to Instrument object.

---

pysatSeasons.occur_prob.**daily3D**(*const*, *bin1*, *label1*, *bin2*, *label2*, *bin3*, *label3*, *data_label*, *gate*, *returnBins=None*, *return_bins=False*)

3D Daily Occurrence Probability of *data_label* > *gate* over a season.

Deprecated since version 0.2.0: *returnBins* will be removed in v0.3.0, it is replaced by *return_bins* because it has standards compliant case.

If *data_label* is greater than *gate* at least once per day, then a 100% occurrence probability results. Season delineated by the bounds attached to Instrument object. Probability = (# of times with at least one hit) / (# of times in bin)

Parameters

- **const** (*pysat.Instrument or pysat.Constellation*) – Instrument/Constellation to use for calculating occurrence probability.
- **bin1** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **bin2** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **bin3** (*array-like*) – List holding [min, max, number of bins] or array-like containing bin edges.
- **label1** (*str*) – Identifies data product for binX.
- **label2** (*str*) – Identifies data product for binX.
- **label3** (*str*) – Identifies data product for binX.
- **data_label** (*list of str*) – Identifies data product(s) to calculate occurrence probability e.g. inst[data_label].
- **gate** (*list of values*) – Values that *data_label* must achieve to be counted as an occurrence.
- **returnBins** (*bool or NoneType*) – If True, also return arrays with values of bin edges, useful for pcolor. Deprecated in favor of *return_bins*. (default=None)
- **return_bins** (*bool*) – If True, also return arrays with values of bin edges, useful for pcolor. (default=False)

Returns

occur_prob – A dict of dicts indexed by *data_label*. Each entry is dict with entries 'prob' for the probability and 'count' for the number of days with any data; 'bin_x', 'bin_y', and 'bin_z' are also returned if requested. Note that arrays are organized for direct plotting, z, y, x.

Return type

dict

Note: Season delineated by the bounds attached to Instrument object.

4.3 Scatter Plot

Support scatterplot production over seasons of interest.

`pysatSeasons.plot.`**`scatterplot`**(*const*, *labelx*, *labely*, *data_label*, *datalim*, *xlim=None*, *ylim=None*)

Return 2D and 3D scatterplot of *data_label* over *label** for a season.

Parameters

- **`const`** (*`pysat.Instrument or pysat.Constellation`*) – Instrument/Constellation to scatterplot.
- **`labelx`** (*`str`*) – Data product for x-axis.
- **`labely`** (*`str`*) – Data product for y-axis.
- **`data_label`** (*`str or array-like of str`*) – Data product(s) to be scatter plotted.
- **`datalim`** (*`numpy.array`*) – Plot limits for data_label.
- **`xlim`** (*`numpy.array or None.`*) – Array for limits along x or y axes. If None, limits are determined automatically. (default=None)
- **`ylim`** (*`numpy.array or None.`*) – Array for limits along x or y axes. If None, limits are determined automatically. (default=None)

Returns

figs – Scatter plots of *data_label* as a function of *labelx* and *labely* over the season delineated by *inst.bounds*.

Return type

list

CHAPTER

FIVE

PYSATSEASONS EXAMPLES

pysat tends to reduce certain science data investigations to the construction of a routine(s) that makes that investigation unique, a call to a seasonal analysis routine, and some plotting commands. Several demonstrations are offered in this section.

5.1 Seasonal Occurrence by Orbit

How often does a particular thing occur on a orbit-by-orbit basis? As an example, let us calculate the occurrence of a positive perturbation in the meridional component of the geomagnetic field as measured by the Vector Electric Field Instrument (VEFI) onboard the Communication/Navigation Outage Forecasting System (C/NOFS) satellite. The full code can be found at: https://github.com/pysat/pysatSeasons/blob/main/demo/ssnl_occurrence_by_orbit.py

```
# Demonstrates iterating over an instrument data set by orbit and
# determining the occurrence probability of an event occurring.

import datetime as dt
import os
import matplotlib.pyplot as plt
import numpy as np

import pysat
import pysatNASA
import pysatSeasons

# Ensure all pysatNASA data plugins are registered with pysat. Only needs
# to be performed once per installation/upgrade.
pysat.utils.registry.register_by_module(pysatNASA.instruments)

# Set the directory where the plots will be saved. Setting nothing will put
# the plots in the current directory
results_dir = ''

# Select C/NOFS VEFI DC magnetometer data, use longitude to determine where
# there are changes in the orbit (local time info not in file)
orbit_info = {'index': 'longitude', 'kind': 'longitude'}
vefi = pysat.Instrument(platform='cnofs', name='vefi', tag='dc_b',
                        clean_level=None, orbit_info=orbit_info)


# Define function to remove flagged values
```

(continues on next page)

(continued from previous page)

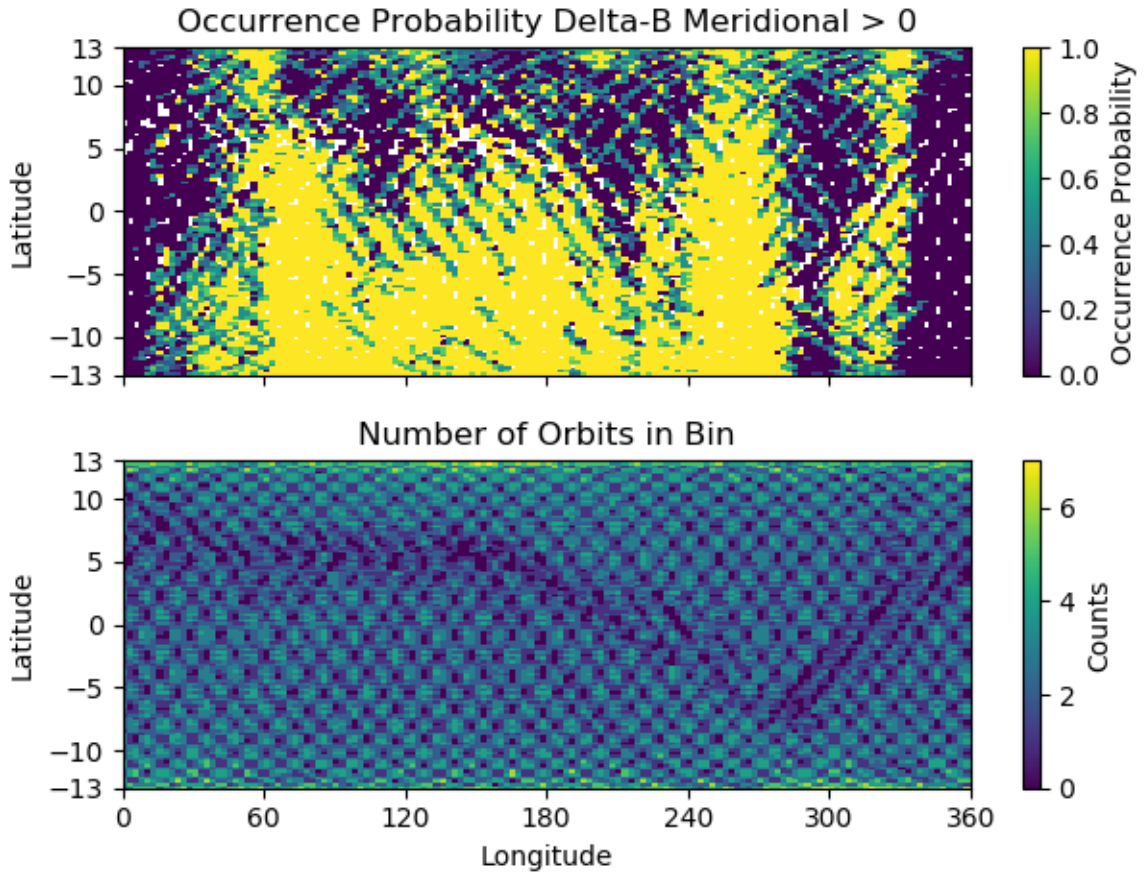
```
def filter_vefi(inst):
    idx, = np.where(inst['B_flag'] == 0)
    inst.data = inst[idx]
    return


# Attach filtering function to `vefi` object.
vefi.custom_attach(filter_vefi)

# Set limits on dates analysis will cover, inclusive
start = dt.datetime(2010, 5, 9)
stop = dt.datetime(2010, 5, 15)

# Check if data already on system, if not, download.
if len(vefi.files[start:stop]) < (stop - start).days:
    vefi.download(start, stop)

# Specify the analysis time limits using `bounds`, otherwise all VEFI DC
# data will be processed.
vefi.bounds = (start, stop)

# Perform occurrence probability calculation.
# Any data added by custom functions is available within analysis below.
ans = pysatSeasons.occur_prob.by_orbit2D(vefi, [0, 360, 144], 'longitude',
                                         [-13, 13, 104], 'latitude',
                                         ['dB_mer'], [0.], returnBins=True)

# A dict indexed by data_label is returned.
ans = ans['dB_mer']

# Plot occurrence probability
f, axarr = plt.subplots(2, 1, sharex=True, sharey=True)

# Mask for locations not observed.
masked = np.ma.array(ans['prob'], mask=np.isnan(ans['prob']))

# Plot occurrence probability
im = axarr[0].pcolor(ans['bin_x'], ans['bin_y'], masked)
axarr[0].set_title('Occurrence Probability Delta-B Meridional > 0')
axarr[0].set_ylabel('Latitude')
axarr[0].set_yticks((-13, -10, -5, 0, 5, 10, 13))
axarr[0].set_ylim((ans['bin_y'][0], ans['bin_y'][-1]))
plt.colorbar(im, ax=axarr[0], label='Occurrence Probability')

# Plot number of orbits per bin.
im = axarr[1].pcolor(ans['bin_x'], ans['bin_y'], ans['count'])
axarr[1].set_title('Number of Orbits in Bin')
axarr[1].set_xlabel('Longitude')
axarr[1].set_xticks((0, 60, 120, 180, 240, 300, 360))
axarr[1].set_xlim((ans['bin_x'][0], ans['bin_x'][-1]))
axarr[1].set_ylabel('Latitude')
plt.colorbar(im, ax=axarr[1], label='Counts')
```

(continues on next page)

(continued from previous page)

```
f.tight_layout()
plt.savefig(os.path.join(results_dir, 'ssnl_occurrence_by_orbit_demo'))
plt.close()
```

Result



The top plot shows the occurrence probability of a positive magnetic field perturbation as a function of geographic longitude and latitude. The bottom plot shows the number of times the satellite was in each bin with data (on per orbit basis). Individual orbit tracks may be seen. The hatched pattern is formed from the satellite traveling North to South and vice-versa. At the latitudinal extremes of the orbit the latitudinal velocity goes through zero providing a greater coverage density. The satellite doesn't return to the same locations on each pass so there is a reduction in counts between orbit tracks. All local times are covered by this plot, over-representing the coverage of a single satellite.

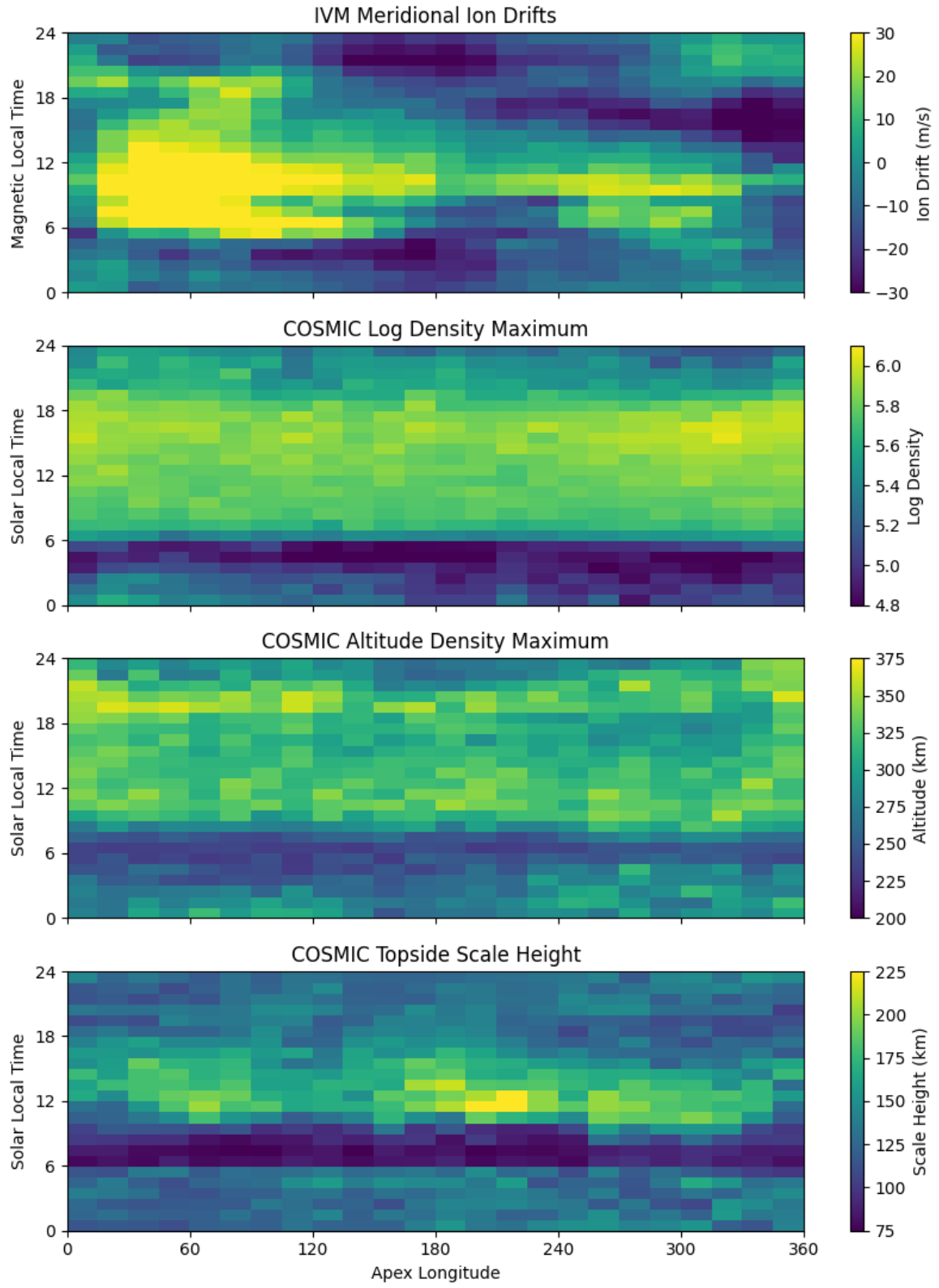
The horizontal blue band that varies in latitude as a function of longitude is the location of the magnetic equator. Torque rod firings that help C/NOFS maintain proper attitude are performed at the magnetic equator. Data during these firings is excluded by the custom function attached to the `vefi` instrument object.

5.2 Seasonal Averaging of Ion Drifts and Density Profiles

In-situ measurements of the ionosphere by the Ion Velocity Meter onboard C/NOFS provides information on plasma density, composition, ion temperature, and ion drifts. This provides a great deal of information on the ionosphere though this information is limited to the immediate vicinity of the satellite. COSMIC GPS measurements, with some processing, provide information on the vertical electron density distribution in the ionosphere. The vertical motion of ions measured by IVM should be reflected in the vertical plasma densities measured by COSMIC. To look at this relationship over all longitudes and local times, for magnetic latitudes near the geomagnetic equator, the code excerpts below provides a framework for the user. The full code can be found at https://github.com/pysat/pysatSeasons/blob/main/demo/cosmic_and_ivm_demo.py

Note the same averaging routine is used for both COSMIC and IVM, and that both 1D and 2D data are handled correctly. The demo code requires pysatCDAAC > 0.0.2.

```
# Instantiate IVM Object
ivm = pysat.Instrument(platform='cnofs', name='ivm', tag='',
                       clean_level='clean')

# Restrict measurements to those near geomagnetic equator.
ivm.custom_attach(restrict_abs_values, args=['mlat', 25.])

# Perform seasonal average
ivm.bounds = (startDate, stopDate)
ivmResults = pysatSeasons.avg.median2D(ivm, [0, 360, 24], 'alon',
                                       [0, 24, 24], 'mlt',
                                       ['ionVelmeridional'])

# Create COSMIC instrument object. Engage supported keyword `altitude_bin`
# to bin all altitude profiles into 3 km increments.
cosmic = pysat.Instrument(platform='cosmic', name='gps', tag='ionprf',
                          clean_level='clean', altitude_bin=3)

# Apply custom functions to all data that is loaded through cosmic
cosmic.custom_attach(add_magnetic_coordinates)

# Select locations near the magnetic equator
cosmic.custom_attach(filter_values, args=['edmax_qd_lat', (-10., 10.)])

# Take the log of NmF2 and add to the dataframe
cosmic.custom_attach(add_log_density)

# Calculates the height above hmF2 to reach Ne < NmF2/e
cosmic.custom_attach(add_scale_height)

# Perform a bin average of multiple COSMIC data products, from startDate
# through stopDate. A mixture of 1D and 2D data is averaged.
cosmic.bounds = (startDate, stopDate)
cosmicResults = pysatSeasons.avg.median2D(cosmic, [0, 360, 24], 'edmax_qd_lon',
                                          [0, 24, 24], 'edmaxlct',
                                          ['ELEC_dens', 'edmaxalt',
                                           'lognm', 'thf2'])

# The work is done, plot the results!
```



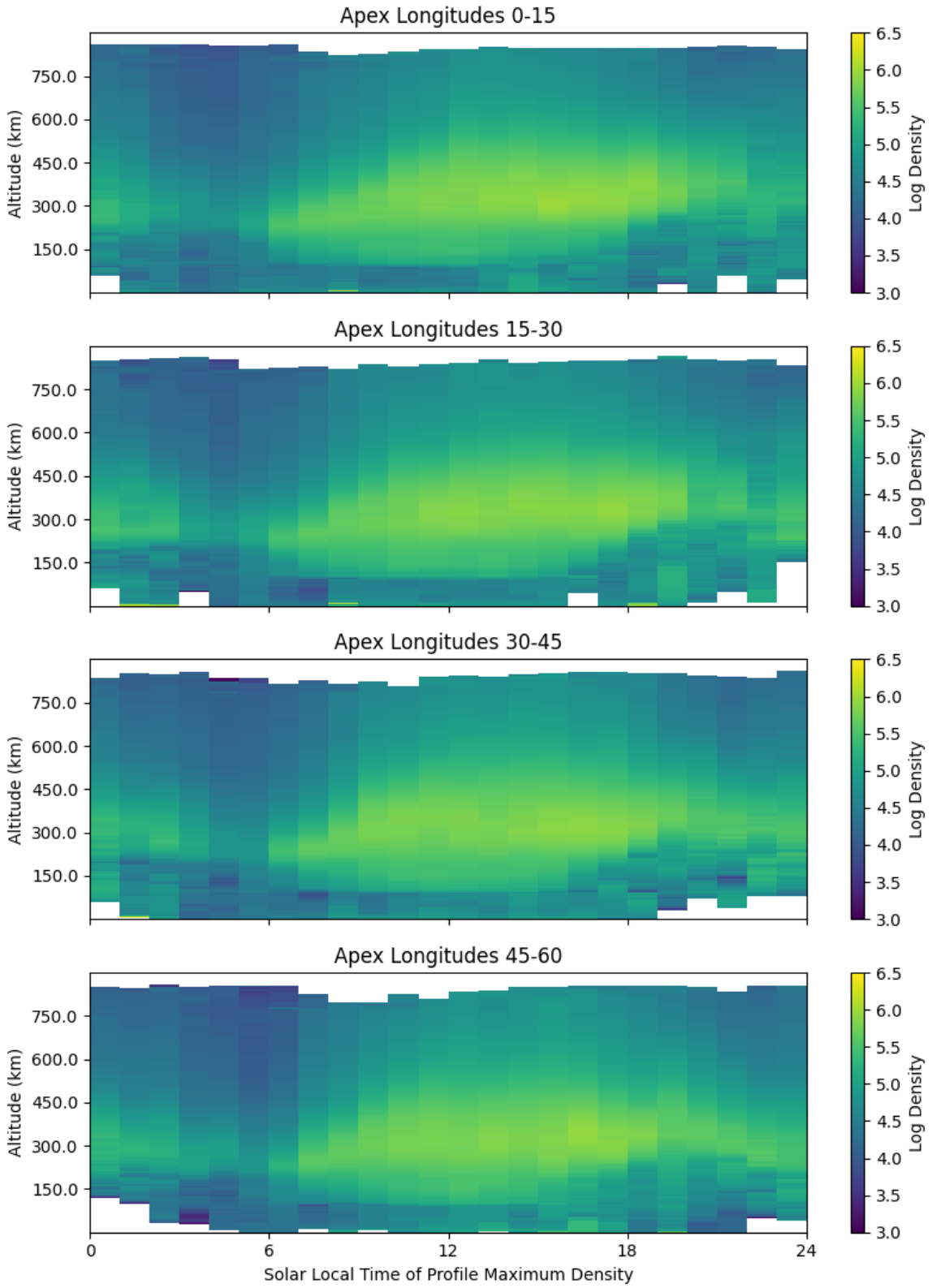
The top image is the median ion drift from the IVM, while the remaining plots are derived from the COSMIC density profiles. COSMIC data does not come with the location of the profiles in magnetic coordinates, so this information is added using the nano-kernel.

```
cosmic.custom_attach(add_magnetic_coordinates)
```

call runs a routine that adds the needed information using the community package `apexpy`. Similarly, using custom functions, locations away from the magnetic equator are filtered out and a couple new quantities are added.

There is a strong correspondence between the distribution of downward drifts between noon and midnight and a reduction in the height of the peak ionospheric density around local sunset. There isn't the same strong correspondence with the other parameters but ion density profiles are also affected by production and loss processes, not measured by IVM.

The median averaging routine also produced a series a median altitude profiles as a function of longitude and local time. A selection are shown below.



There is a gradient in the altitude distribution over longitude near sunset. Between 0-15 longitude an upward slope is seen in bottom-side density levels with local time though higher altitudes have a flatter gradient. This is consistent with the upward ion drifts reported by IVM. Between 45-60 the bottom-side ionosphere is flat with local time, while

densities at higher altitudes drop steadily. Ion drifts in this sector become downward at night. Downward drifts lower plasma into exponentially higher neutral densities, rapidly neutralizing plasma and producing an effective flat bottom. Thus, the COSMIC profile in this sector is also consistent with the IVM drifts.

Between 15-30 degrees longitude, ion drifts are upward, but less than the 0-15 sector. Similarly, the density profile in the same sector has a weaker upward gradient with local time than the 0-15 sector. Between 30-45 longitude, drifts are mixed, then transition into weaker downward drifts than between 45-60 longitude. The corresponding profiles have a flatter bottom-side gradient than sectors with upward drift (0-30), and a flatter top-side gradient than when drifts are more downward (45-60), consistent with the ion drifts.

CHAPTER

SIX

GUIDE FOR DEVELOPERS

6.1 Contributor Covenant Code of Conduct

6.1.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

6.1.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

6.1.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

6.1.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

6.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at pysat.developers@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

6.1.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

6.2 Contributing

Bug reports, feature suggestions and other contributions are greatly appreciated! PysatSeasons is a community-driven project and welcomes both feedback and contributions.

6.3 Short version

- Submit bug reports and feature requests at GitHub
- Make pull requests to the `develop` branch

6.4 Bug reports

When reporting a bug please include:

- Your operating system name and version
- Any details about your local setup that might be helpful in troubleshooting
- Detailed steps to reproduce the bug

6.5 Feature requests and feedback

The best way to send feedback is to file an issue at GitHub.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

6.6 Development

To set up `pysatSeasons` for local development:

#. Fork pysat on GitHub. #.

   Clone your fork locally:

   git clone git@github.com:your_name_here/pysatSeasons.git

1. Create a branch for local development:

   git checkout -b name-of-your-bugfix-or-feature

   Now you can make your changes locally. Tests for new instruments are performed automatically. Tests for custom functions should be added to the appropriately named file in `pysatSeasons/tests`. For example, the averaging routines in avg.py are tested in `pysatSeasons/tests/test_avg.py`. If no test file exists, then you should create one. This testing uses pytest, which will run tests on any python file in the test directory that starts with `test_`.

2. When you're done making changes, run all the checks to ensure that nothing is broken on your local system:

   pytest -vs

3. Update/add documentation (in `docs`), if relevant

4. Commit your changes and push your branch to GitHub:

   git add . git commit -m "Brief description of your changes" git push origin name-of-your-bugfix-or-feature

5. Submit a pull request through the GitHub website. Pull requests should be made to the `develop` branch.

6.6.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code, just make a pull request. Pull requests should be made to the `develop` branch.

For merging, you should:

1. Include an example for use
2. Add a note to `CHANGELOG.md` about the changes
3. Ensure that all checks passed (current checks include Scrutinizer, Travis-CI, and Coveralls)[1]

```
have trouble building all the testing environments, you can rely on
Travis to run the tests for each change you add in the pull request.
Because testing here will delay tests by other developers, please ensure
that the code passes all tests on your local system first.
```

[1] If you don't have all the necessary Python versions available locally or

CHAPTER

SEVEN

CHANGE LOG

All notable changes to this project will be documented in this file. This project adheres to Semantic Versioning.

7.1 [0.2.0] - 2022-08-12

- New Features
  - Added support for xarray data in the seasonal averaging functions in `pysatSeasons.avg`
  - Added support for xarray data in the occurrence probability functions in `pysatSeasons.occur_prob`
  - Added support for Constellations in `pysatSeasons.occur_prob`
  - Added support for Constellations in `pysatSeasons.plot`
  - Renamed `computational_form` to `to_xarray_dataset` and refocused.
- Deprecations
  - Deprecated `returnBins` keyword in favor of `return_bins` in `pysatSeasons.occur_prob`.
  - Deprecated `returnData` keyword in favor of `return_data` in `pysatSeasons.avg`.
- Documentation
  - Improved docstrings throughout.
  - Updated documentation examples.
  - Documentation now available on readthedocs.org.
- Bug Fix
- Maintenance
  - Removed deprecated `pandas.Panel` from functions.
  - Removed old `__future__` imports.
  - Removed use of `collections.deque` in `pysatSeasons.avg`.
  - Migrated to GitHub Workflows for CI testing.
  - Migrated from nose to pytest.
  - Adopted setup.cfg
  - Updated style standards
  - Added automated style and docstring testing

7.2 [0.1.3] - 2021-06-18

- Updates style to match pysat 3.0.0 release candidate
- Improves discussion of rationale for version caps on readme page
- Migrates CI tests to github actions

7.3 [0.1.2] - 2020-07-29

- Updates demo codes to import objects from datetime and pandas for pysat 3.0.0 compatibility
- Fixed a bug where test routines used float where numpy 1.18 expects an int
- Import objects from datetime and pandas for pysat 3.0.0 compatibility
- Use conda to manage Travis CI
- Rename default branch as `main`
- Update to pysat documentation standards
- Add flake8 testing for code

7.4 [0.1.1] - 2019-10-09

- Add demo code
- Added DOI badge to documentation page

7.5 [0.1.0] - 2019-10-07

- Initial release

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX