Python Satellite Data Analysis Toolkit

Russell Stoneback
Angeline Burrell
Jeff Klenzing
Tool for System Science

• If we are going to integrate the full array of space science measurements then we need a common ground for all instruments and an implementation of the process of space science analysis

• Package with support for common problems
  • Downloading
  • Organizing Files
  • Loading
  • Cleaning
  • Modifying/Processing
  • Exploit routines from other packages
  • Instrument specific analysis
  • Instrument independent analysis

• Support for a variety of unique datasets and processing chains

• Web Tour - **Python 2/3 Compatible!!**
Short History

• pysat was forged in the fires of the C/NOFS mission

• Began life in IDL as a GUI

  • Lack of behind the scenes capabilities prompted a refactor of the back end

  • Attended scipy 2012 in Austin, TX - Refactored into Python immediately after

• Goals expanded with capabilities

• Feature set and structure reflects my scientific requirements for space science
Since Last Update

• IVM processing for the upcoming ICON and COSMIC-2 Missions runs on top of pysat
  • pysat runs at UC Berkeley, UCAR, UTD, and …
  • Officially speaking lots of money depends upon pysat
• Added significant unit testing coverage: 80%
• Added additional instruments
Installation

- pip install pysat
- Support for system science
- or, python setup.py install
- pip install pysatCDF
- Support for NASA’s CDF library, includes everything you need including NASA’s CDF library
- Going to pursue inclusion into Enthought Repository
What is pysat again?

Process of Space Science Data Analysis
Implemented like a music recording signal chain
Currently 1 channel - no automatic mixing
C/NOFS IVM

In [31]: import pysat
   ...: pysat.utils.set_data_dir('/Users/rstoneba/demo')
   ...: ivm = pysat.Instrument('cnofs', 'ivm', clean_level='clean')
   ...: ivm.download(pysat.datetime(2010, 1, 1), pysat.datetime(2010, 1, 2))
   ...: ivm.load(2010, 1)
   ...: np.log10(ivm[0:1000, 'ionDensity']).plot(title='IVM Ion Density')
   ...: plt.ylabel('Log Density (N/cc)')

Downloading data to: /Users/rstoneba/demo/cnofs/ivm/
Downloading file for 01/01/10
Downloading file for 01/02/10
Updating pysat file list
pysat is searching for cnofs ivm files.
Found 2 of them.
Updating instrument object bounds.
Returning cnofs ivm data for 01/01/10
Out[31]: <matplotlib.text.Text at 0x123eb2790>

Data is preliminary
C/NOFS IVM

Data is preliminary
C/NOFS IVM by Orbit

In [40]: ivm = pysat.Instrument('cnofs', 'ivm',
                           clean_level='dirty',
                           orbit_info={'index': 'mlt'})

ivm.load(2010, 2)
ivm.orbits.next()
np.log10(ivm['ionDensity']).plot(title='IVM Ion Density')
plt.ylabel('Log Density (N/cc)')

Returning cnofs ivm data for 01/02/10
Returning cnofs ivm data for 01/01/10
Returning cnofs ivm data for 01/02/10
Loaded Orbit: 0
Out[40]: <matplotlib.text.Text at 0x1250fab90>

In [48]: ivm = pysat.Instrument('cnofs', 'ivm',
                           clean_level='dirty',
                           orbit_info={'index': 'mlt'})

ivm.load(2010, 2)
ivm.orbits.next()
ivm.data.plot(x='mlt', y='ionDensity',
              title='IVM Ion Density',
              logy=True,
              xticks=[0, 6, 12, 18, 24])
plt.ylabel('Ion Density (N/cc)')

Returning cnofs ivm data for 01/02/10
Returning cnofs ivm data for 01/01/10
Returning cnofs ivm data for 01/02/10
Loaded Orbit: 0
Out[48]: <matplotlib.text.Text at 0x1284af410>
In [17]: ivm = pysat.Instrument('rocsat', 'ivm',
   ...:         clean_level='none',
   ...:         orbit_info={'index': 'lhr'})
   ...:
   ...: ivm.download(pysat.datetime(2002,1,1), pysat.datetime(2002,1,2))
   ...: ivm.load(2002,2)
   ...: ivm.orbits.next()
   ...: ivm.data.plot(x='lhr', y='logN',
   ...:         title='IVM Ion Density',
   ...:         xticks=[0,6,12,18,24])
   ...: plt.ylabel('Ion Density (N/cc)')

pysat is searching for rocsat ivm files.
Unable to find any files. If you have the necessary files please check pysat settings and file locations.
Downloading data to: /Users/rstoneba/demo/rocsat/ivm/
Downloading file for 01/01/02
Downloading file for 01/02/02
Updating pysat file list
pysat is searching for rocsat ivm files.
Found 2 of them.
Updating instrument object bounds.
Returning rocsat ivm data for 01/02/02
Returning rocsat ivm data for 01/01/02
Returning rocsat ivm data for 01/02/02
Loaded Orbit: 0
Out[17]: <matplotlib.text.Text at 0x124132e50>
In [17]: ivm = pysat.Instrument('rocsat', 'ivm',
   clean_level='none',
   orbit_info={'index': 'lhr'})
   ivm.download(pysat.datetime(2002,1,1), pysat.datetime(2002,1,2))
   ivm.load(2002,2)
   ivm.orbits.next()
   ivm.data.plot(x='lhr', y='logN',
   title='IVM Ion Density',
   xticks=[0,6,12])
   plt.ylabel('Ion Density (N/cc)')

pysat is searching for rocsat ivm files.
Unable to find any files. If you have
Download data to: /Users/rstoneba/
Downloading file for 01/01/02
Downloading file for 01/02/02
Updating pysat file list
pysat is searching for rocsat ivm files.
Found 2 of them.
Updating instrument object bounds.
Returning rocsat ivm data for 01/02/02
Returning rocsat ivm data for 01/01/02
Returning rocsat ivm data for 01/02/02
Loaded Orbit:0
Out[17]: <matplotlib.text.Text at 0x12>
C/NOFS VEFI

Sun May 9 01:42:58 2010 - Sun May 9 03:27:12 2010

Magnetic Field (North)

Magnetic Field (Up)

Magnetic Field (West)

Delta Magnetic Field - Meridional

Delta Magnetic Field - Parallel

Delta Magnetic Field - Zonal

Occurrence Probability Delta-B Meridional > 0

Number of Orbits in Bin

Full Code in Demo Area of Repo

# select 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)

# perform occurrence probability calculation

# any data added by custom functions is available within routine below

ans = pysat.ssnl.occur_prob.by_orbit2D(vefi, [0, 360, 144], 'longitude',
                                       [-13, 13, 104], 'latitude',
                                       [1, 0], returnBins=True)
COSMIC and IVM Demo

- def, etc.
- download has password
- plot altitude sweep
- Andrea's poster
COSMIC and IVM Demo

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

# restrict measurements to those near geomagnetic equator
ivm.custom.add(restrictMLAT, 'modify', maxMLAT=25.)

# perform seasonal average
ivm.bounds = (startDate, stopDate)
ivmResults = pysat.ssnl.avgs.median2D(ivm, [0, 360, 24], 'alon',
                                       [0, 24, 24], 'mlt', ['ionVelmeridional'])
```
COSMIC and IVM Demo

```
# instantiate IVM Object
ivm = pysat.Instrument(platform='cnofs',
                        name='ivm', tag='',
                        clean_level='clean')
# restrict measurements to those near geomagnetic equator
ivm.custom.add(restrictMLAT, 'modify', maxMLAT=25.)
# perform seasonal average
ivm.bounds = (startDate, stopDate)
ivmResults = pysat.ssnl.avg.median2D(ivm, [0,360,24], 'alon',
                                      [0,24,24], 'mlt', ['ionVelmeridional'])
```

# create COSMIC instrument object
cosmic = pysat.Instrument(platform='cosmic2013',
                           name='gps', tag='ionprof',
                           clean_level='clean',
                           altitude_bin=3)
# apply custom functions to all data that is loaded through cosmic
cosmic.custom.add(addApexLong, 'add')
# select locations near the magnetic equator
cosmic.custom.add(filterMLAT, 'modify', mlatRange=(0.,10.))
# take the log of NmF2 and add to the dataframe
cosmic.custom.add(addlogNm, 'add')
# calculates the height above hmF2 to reach Ne < NmF2/e
cosmic.custom.add(addTopsideScaleHeight, 'add')

# do an average of multiple COSMIC data products
# from startDate through stopDate
# a mixture of 1D and 2D data is averaged
cosmic.bounds = (startDate, stopDate)
cosmicResults = pysat.ssnl.avg.median2D(cosmic, [0,360,24], 'apex_long',
                                         [0,24,24], 'edmaxlct', ['profiles', 'edmaxalt', 'lognm', 'thf2'])
are added to remaining data. In addition, some vectors are 'measured' along a Line of sight, similar to the Figure 2: (Top) Single slices of a series of 2D vector fields with a percentage of data missing. Random errors to the areas measured by F13 over the input data set.

reconstruction of polar ion convection. The distribution of reconstructed points is limited here Figure 3: (Left) Single orbital pass from DMSP F13 in early January, 2001. (Right) DINEOF process is established through comparison to the original simulation.

the reconstruction. Errors and LOS directions are random. The performance of the measurement data, most of which are LOS measurements. On the bottom right is DINEOFs is shown in the right column. The bottom row covers (left) input vector gaps presented to DINEOFs. This slice is one of many. The reconstruction from test runs are shown below. The left column top shows a slice of scalar data with corresponding amplitudes in time that best reproduce the long term data set. Unit

DINEOFs are a data based method for isolating and exploring variations within a process. These combined measurements are analyzed using Data Interpolating Orthogonal Functions (DINEOFs), a method that produces a set of basis functions from DavitPy. These combined measurements are analyzed using Data Interpolating...
Future

• Transition to a hybrid Pandas/Xarray data model
  • Expected to be backwards compatible

• Update metadata to be practical but careful about handling case

• Polish

• Submitted proposal about extending this type of functionality to constellations - collections of instruments that can acted upon as one - multichannel audio mixer