pysatModels Documentation

Release 0.1.0-alpha

Angeline G Burrell, Russell Stoneback, Jeffrey Klenzing

May 27, 2021

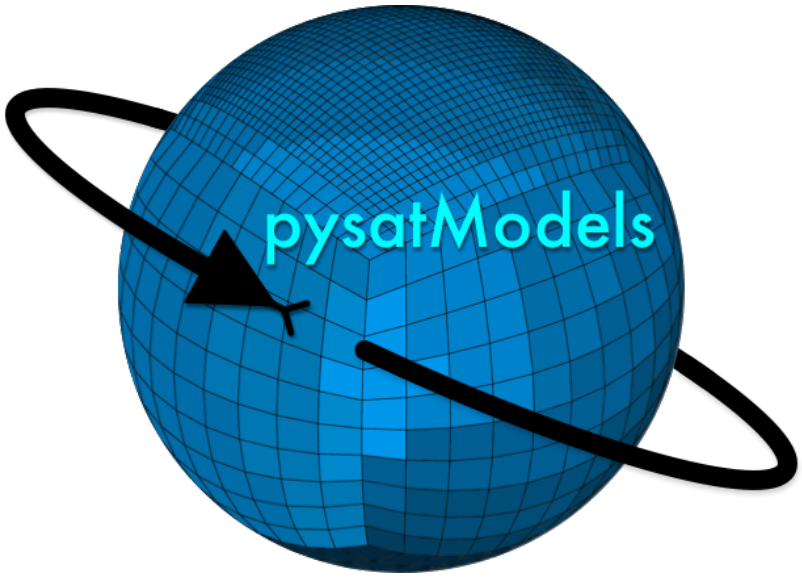
Contents

This documentation describes the pysatModels module, which contains routines to load model data as pysat.Instrument objects and utilities to perform typical model-oriented analysis, such as model validation.

CHAPTER 1

Overview

Working together with modelled and observational data sets can be challenging. Modelled data sets can be very large, with each model populating data along a different grid. This package provides a variety of tools to aid model-data analysis. This includes tools for model valiation, modelling instrument output, and incorporating modelled data sets into general data analysis routines written to use data stored in pysat Instrument objects or xarray Datasets.

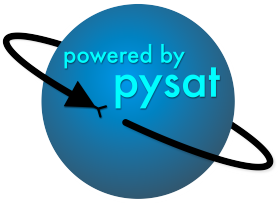


CHAPTER 2

Installation

The following instructions will allow you to install pysatModels.

2.1 Prerequisites



pysatModels 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
numpy	pysat
scipy	pyForecastTools
pandas	
xarray	

2.2 Installation Options

1. Clone the git repository

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

2. Install pysatModels: 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 3

Citation Guidelines

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

3.1 pysatModels

- Burrell, A. G., R. Stoneback, and J. H. Klenzing. (2020). pysat/pysatModels: Alpha Release (Version 0.1). https://github.com/pysat/pysatModels

```
@Misc{pysatModels,
      author = {Burrell, A. G. and Stoneback, R. and Klenzing, J. H.},
      title  = {pysat/pysatModels: Alpha Release},
      year   = {2020},
      date   = {2020-04-01},
      url    = {https://github.com/pysat/pysatModels},
      }
```

CHAPTER 4

Supported Models

4.1 SAMI2

Supports the SAMI2 (Sami2 is Another Model of the Ionosphere) model through the sami2py interface. Sami2py is a python module that runs the SAMI2 model, as well as archives, loads and plots the resulting modeled values. SAMI2 is a model developed by the Naval Research Laboratory to simulate the motions of plasma in a 2D ionospheric environment along a dipole magnetic field [Huba et al, 2000]. Information about this model can be found at the sami2py github page, along with a list of the SAMI2 principle papers.

4.2 TIE-GCM

Supports the UCAR (University Corporation for Atmospheric Research) model, Thermosphere-Ionosphere-Electrodynamics General Circulation Model (TIE-GCM). Information about this model can be found at the UCAR TIE-GCM website, along with a list of the TIE-GCM principle papers.

CHAPTER 5

Utilities

5.1 Match

5.2 Extract

5.3 Compare

CHAPTER 6

Examples

Here are some examples that demonstrate how to use various pysatModels tools

6.1 Loading Model Data

6.1.1 Load Model Data into a pysat Instrument

pysatModels uses pysat to load modelled data sets. As specified in the pysat tutorial, data may be loaded using the following commands. TIE-GCM is used as an example, and so to execute this code snippet the user will need to obtain a TIE-GCM data file from UCAR.

```
import pysat
import pysatModels as ps_mod

filename = 'tiegcm_filename.nc'
tiegcm = pysat.Instrument(platform='ucar', name='tiegcm')
tiegcm.load(fname=filename)
```

6.1.2 Load Model Data into an xarray Dataset

There are situations (such as when developing a new model) when it may be inconvenient to create a pysat Instrument object for a modelled data set. Many of the pysatModels utilities allow xarray Datasets as input. For these routines or to retrieve an xarray Dataset for other purposes, you can use the `load_model_xarray` routine in utils.match.

In this example, the time is irrelevent because a full filename is provided:

```
import datetime as dt
import pysat
import pysatModels as ps_mod

ftime = dt.datetime(2010, 1, 1)
```

(continues on next page)

(continued from previous page)

```
filename = 'tiegcm_filename.nc'
tiegcm = pysat.Instrument(platform='ucar', name='tiegcm')

tg_dataset = ps_mod.utils.match.load_model_xarray(ftime, tiegcm, filename)
```

In this example, the filename includes temporal information, which is provided within the loading function by the input time:

```
import datetime as dt
import pysat
import pysatModels as ps_mod

ftime = dt.datetime(2010, 1, 1)
filename = 'tiegcm_%Y%j.nc'
tiegcm = pysat.Instrument(platform='ucar', name='tiegcm')

tg_dataset = ps_mod.utils.match.load_model_xarray(ftime, tiegcm, filename)
```

In this example, the routine takes advantage of the pysat file organization system, and will return a `NoneType` object if no files are found for the specified time:

```
import datetime as dt
import pysat
import pysatModels as ps_mod

ftime = dt.datetime(2010, 1, 1)
tiegcm = pysat.Instrument(platform='ucar', name='tiegcm')

tg_dataset = ps_mod.utils.match.load_model_xarray(ftime, tiegcm)
```

6.2 Pair Modelled and Observed Data

One common analytical need is to obtain a combined data set of modelled and observed observations at the same times and locations. This can be done using the `collect_inst_model_pairs` routine in utils.match. This routine takes a date range as input, and then extracts modelled observations at the specified instrument location. However, it does not interpolate in time. Details about the interpolation of modelled data onto the instrument location can be found in the utils.extract routine: `extract_modelled_observations`.

In the example below, we load a DINEOFs file created for testing purposes and pair it with C/NOFS IVM data. This is a global example for a single time, since in this instance DINEOFs was used to create a day-specific emperical moodel. Comparisons with output from a global circulation model would look different, as one would be more likely to desire the the closest observations to the model time rather than all observations within the model time.

This example uses the external modules:

- pysat
- pysatNASA

```
import datetime as dt
from os import path
import pysat
import pysatNASA
import pysatModels as ps_mod
```

(continues on next page)

(continued from previous page)

```
stime = dt.datetime(2009, 1, 1)
tinc = dt.timedelta(days=1)  # Required input is not used in this example
input_kwargs = dict()

# Initialize the model input information, including the keyword arguements
# needed to load the model Instrument into an xarray Dataset using
# the match.load_model_xarray routine
dineofs = pysat.Instrument(inst_module=ps_mod.models.pydineof_dineof,
                           tag='test')
if len(dineofs.files.files) == 0:
    dineofs.download(stime, stime)
filename = dineofs.files.files[stime]
input_kwargs["model_load_kwargs"] = {'model_inst': dineofs,
                                    'filename': filename}

# Ended HERE
# Initialize the DMSP IVM input, and ensure the best model interpolation
# by extracting the clean model data after matching.
dmsp_f18 = pysat.Instrument(inst_module=pysatMadrigal.instruments.dmsp_ivm, sat_id=
↪'f18', clean_level='none')
dmsp_f18.download(stime, stime) # Skip this if you already have the data
dmsp_f18.custom.attach(sami2_meridian, kind='modify', at_pos='end', args=['glon', lon_
↪min, lon_max])
input_kwargs["inst_clean_rout"] = pysatMadrigal.instruments.dmsp_ivm.clean
input_kwargs["inst_download_kwargs"] = {"skip_download": True}

# Many keyword arguements are required, as they provide information on
# the names of coordinates needed to extract data
input_kwargs["inst_lon_name"] = "glon"
input_kwargs["mod_lon_name"] = "glon"
input_kwargs["inst_name"] = ["gdlat", "gdalt"] # Not considering longitude
input_kwargs["mod_name"] = ["glat", "zalt"]    # in the data matching
input_kwargs["mod_units"] = ["deg", "km"]
input_kwargs["mod_datetime_name"] = "time"
input_kwargs["mod_time_name"] = "time"

# Now we are ready to run using the defaults!
matched_inst = ps_mod.utils.match.collect_inst_model_pairs(stime, stime+tinc, tinc,␣
↪dmsp_f18, **input_kwargs)
```

This returns a pysat Instrument object with the DMSP IVM data and SAMI2 data at the same times, latitudes, and altitudes along the SAMI2 meridian. The DMSP IVM data has the same names as the normal Instrument, and the SAMI2 data has the same name as the SAMI2 data, but with `model_` as a prefix to prevent confusion. You can change this prefix using the `model_label` keyword argument, allowing multiple models to be matched to the same observational data set.

```
# Using the results from the prior example
print([cc for cc in matched_inst.data.variables.keys() if cc.find("model_") == 0])
```

This produces the output line: `['model_deni', 'model_vsi', 'model_ti', 'model_te', 'model_slt']`

You can also match model and data results by location alone. This is done by setting the `time_method` keyword arguement to `'max'`.

CHAPTER 7

Guide for Developers

7.1 Contributor Covenant Code of Conduct

7.1.1 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.

7.1.2 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

7.1.3 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.

7.1.4 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.

7.1.5 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.

7.1.6 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

7.2 Contributing

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

7.3 Short version

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

7.4 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

7.5 Feature requests and feedback

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

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 :)

7.6 Development

To set up `pysatModels` for local development:

- Fork pysat on GitHub.
- Clone your fork locally:

```
git clone git@github.com:your_name_here/pysatModels.git
```

- 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 `pysatModels/tests`. For example, the averaging routines in avg.py are tested in `pysatModels/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_`.

- **When you're done making changes, run all the checks from the** `pysatModels/tests` directory to ensure that nothing is broken on your local system. You may need to install pytest and pytest-flake8 first.

```
python -m pytest -vs --flake8
```

- **Update or add documentation (in `docs`), if relevant. If you have added** a new routine, you will need to add an example in the `docs/examples` folder.
- **Commit your changes and push your branch to GitHub. Our commit statements** follow the basic rules in the Numpy/SciPy workflow:

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

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

7.6.1 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).

If you don't have all the necessary Python versions available locally or 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.

CHAPTER 8

Change Log

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

8.1 [0.1.0] - 2019-xx-xx

- Initial release

CHAPTER 9

Indices and tables

- genindex
- modindex
- search