Introduction to PySPEDAS plug-in development

Eric Grimes
egrimes@igpp.ucla.edu
UCLA Department of Earth, Planetary, and Space Sciences

Last updated: Jan 17, 2023

Introduction

- We’re doing development on GitHub:

  https://github.com/spedas/pyspedas

- Plug-in developers can have their own branch, e.g.,

  https://github.com/spedas/pyspedas/tree/elfin/pyspedas

- When you’re ready to merge the updates from your branch, submit a pull request to master. You can also create a fork of PySPEDAS in your own account and submit a pull request.

Introduction

- We’re distributing PySPEDAS using the Python Package Index (PyPI); you can get the latest version using pip:

pip install pyspedas

- You can update PySPEDAS in your environment using the --upgrade flag:

pip install --upgrade pyspedas

- You can install the bleeding edge from your development branch using:

pip install --upgrade https://github.com/spedas/pyspedas/archive/branch.zip

Introduction

- Advice on creating isolated virtual environments can be found in our README:

https://github.com/spedas/pyspedas

- Google has a free tool that allows you to run PySPEDAS using Jupyter notebooks in the cloud called “Colab”, available at:

https://colab.research.google.com/

- I suggest using this to get to know PySPEDAS, test your plug-in code and create examples, reproduce issues others are running into, etc.

Introduction

- We use the MIT license (available in the repo), so the code you contribute will be under this license.

- We have user documentation available at:

https://pyspedas.readthedocs.io/

- These docs are built automatically using the files in the ‘docs’ directory, and the docstrings in your code:

https://github.com/spedas/pyspedas/tree/master/docs

(when code is contributed to the master branch)

Introduction

We only have a few big rules:

- Please do not modify code that belongs to someone else without talking to them first. Changes are always welcome, but it’s important to make sure the owner doesn’t have their own changes locally that might conflict with your changes. They may also want to review your changes before merging with the master branch.

- PySPEDAS is cross-platform, so please be sure to properly handle file paths in a way that works on unix-like machines as well as Windows

Introduction

And some suggestions:

- Please use the logging module instead of print() for sending output to the console (some users prefer to turn off all console messages other than errors, e.g., in processing scripts)

- We suggest following PEP-8, at least where it makes sense (i.e., where it improves the readability of the code)

- If your code depends on a package that requires a compiler to install, you'll have to provide instructions to install the package in your documentation (because we won't be able to install it automatically), and use a try/except to catch the possible ImportError

- Minimize adding additional external dependencies where possible

Introduction

- We try to support all versions of Python that numpy supports (which I think goes back 5 years), so limit usage of features that require newer versions of Python. I think we'll be bumping to requiring Python 3.8+ soon

- Dockerfiles for quickly running PySPEDAS in different versions of Python can be found here:

https://github.com/supervised/pyspedas-docker

Mission plug-in design

- A simple plug-in for loading mission data in PySPEDAS looks like this:
  - README.md: simple documentation that gets rendered on GitHub
  - config.py: contains a CONFIG dictionary with remote data directory and your other configuration options
  - load.py: core load routine; mostly instrument independent code for accessing the data and loading the CDF files
  - __init__.py: instrument load routines (or wrappers for instrument load routines if you decide to put them into their own files/directories)
  - tests/: directory containing the test suite

tests	Large update to the test suite	3 months ago
README.md	Update README.md	3 years ago
__init__.py	Adding datasets() functions to the missions that load data from SPDF;...	2 months ago
config.py	Removing .sci. from SPDF paths	2 years ago
load.py	Removing unused imports	last year

README.md

Colorado Student Space Weather Experiment (CSSWE)

The routines in this module can be used to load data from the Colorado Student Space Weather Experiment (CSSWE) mission.

Instruments

- Relativistic Electron and Proton Telescope integrated little experiment (REPTile)

Examples

Get started by importing pyspedas and tplot; these are required to load and plot the data:

```
import pyspedas
from pytplot import tplot
```

Relativistic Electron and Proton Telescope integrated little experiment (REPTile)

```
reptile_vars = pyspedas.csswe.reptile(trange=['2013-11-5', '2013-11-6'])

tplot(['E1flux', 'E2flux', 'E3flux', 'P1flux', 'P2flux', 'P3flux'])
```

supervised Removing .sci. from SPDF paths Latest commit 93c2a45 on Aug 9, 2020 History

1 contributor

11 lines (8 sloc) | 409 Bytes Raw Blame

```
1  import os
2
3  CONFIG = {'local_data_dir': 'cssswe_data/',
4            'remote_data_dir': 'https://spdf.gsfc.nasa.gov/pub/data/cssswe/'}
5
6  # override local data directory with environment variables
7  if os.environ.get('SPEDAS_DATA_DIR'):
8      CONFIG['local_data_dir'] = os.sep.join([os.environ['SPEDAS_DATA_DIR'], 'cssswe'])
9
10 if os.environ.get('CSSWE_DATA_DIR'):
11     CONFIG['local_data_dir'] = os.environ['CSSWE_DATA_DIR']
```

54 lines (41 sloc) | 1.63 KB Raw Blame

```python
from pyspedas.utilities.dailynames import dailynames
from pyspedas.utilities.download import download
from pyspedas.analysis.time_clip import time_clip as tclip
from pytplot import cdf_to_tplot

from .config import CONFIG

def load(trange=['2013-11-5', '2013-11-6'],
         instrument='reptile',
         datatype='flux',
         level='l2',
         suffix='',
         get_support_data=False,
         varformat=None,
         varnames=[],
         downloadonly=False,
         notplot=False,
         no_update=False,
         time_clip=False):
    """
    This function loads data from the CSSWE mission; this function is not meant
    to be called directly; instead, see the wrapper:
        pyspedas.csswe.reptile

    """

```

```
        no_update=False,
        time_clip=False):
    """
    This function loads data from the CSSWE mission; this function is not meant
    to be called directly; instead, see the wrapper:
        pyspedas.csswe.reptile

    """

    if instrument == 'reptile':
        pathformat = level+'/'+instrument+'/'+datatype+'/%Y/csswe_'+instrument+'_6sec-'+datatype+'-'+level+'_%Y%m%d_v??.cdf'

    # find the full remote path names using the trange
    remote_names = dailynames(file_format=pathformat, trange=trange)

    out_files = []

    files = download(remote_file=remote_names, remote_path=CONFIG['remote_data_dir'], local_path=CONFIG['local_data_dir'],
no_download=no_update)
    if files is not None:
        for file in files:
            out_files.append(file)

    out_files = sorted(out_files)

    if downloadonly:
        return out_files

    tvars = cdf_to_tplot(out_files, suffix=suffix, get_support_data=get_support_data, varformat=varformat, varnames=varnames,
notplot=notplot)

    if notplot:
        return tvars

    if time_clip:
        for new_var in tvars:
            tclip(new_var, trange[0], trange[1], suffix='')

    return tvars
```

```python
from .load import load
from pyspedas.utilities.datasets import find_datasets


def reptile(trange=['2013-11-5', '2013-11-6'],
        datatype='flux',
        level='l2',
        suffix='',
        get_support_data=False,
        varformat=None,
        varnames=[],
        downloadonly=False,
        notplot=False,
        no_update=False,
        time_clip=False):
    """
    This function loads data from the Relativistic Electron and Proton Telescope integrated little experiment (REPTile)

    Parameters
    ----------
        trange : list of str
            time range of interest [starttime, endtime] with the format
            'YYYY-MM-DD','YYYY-MM-DD'] or to specify more or less than a day
            ['YYYY-MM-DD/hh:mm:ss','YYYY-MM-DD/hh:mm:ss']

        datatype: str
            Data type; Valid options:
                'counts' for L1 data
                'flux' for L2 data

        level: str
            Data level; options: 'l1', 'l2' (default: l2)

        suffix: str
            The tplot variable names will be given this suffix.  By default,
            no suffix is added.
```

```
            "VAR_TYPE" attribute of "data".

        varformat: str
            The file variable formats to load into tplot.  Wildcard character
            "*" is accepted.  By default, all variables are loaded in.

        varnames: list of str
            List of variable names to load (if not specified,
            all data variables are loaded)

        downloadonly: bool
            Set this flag to download the CDF files, but not load them into
            tplot variables

        notplot: bool
            Return the data in hash tables instead of creating tplot variables

        no_update: bool
            If set, only load data from your local cache

        time_clip: bool
            Time clip the variables to exactly the range specified in the trange keyword

    Returns
    ----------
        List of tplot variables created.

    """
    return load(instrument='reptile', trange=trange, level=level, datatype=datatype, suffix=suffix,
get_support_data=get_support_data, varformat=varformat, varnames=varnames, downloadonly=downloadonly, notplot=notplot,
time_clip=time_clip, no_update=no_update)


def datasets(instrument=None, label=True):
    out = find_datasets(mission='Smallsats/Cubesats', instrument='csswe', label=label)
    return out
```

Documentation

- The docs at https://pyspedas.readthedocs.io/ are built using the docstrings in your files

- For this to work properly, use **Numpy** style docstrings (see any of our current code for examples)

- To change the default in PyCharm use Preferences -> Tools -> Python Integrated Tools and change the docstring style to Numpy

Unit tests

- We use the unittest framework to ensure that the code runs.

- The full test suite runs every time code is merged with the master branch; the config file is at .github/workflows/pythonpackage.yml. By default, the tests run on Ubuntu, but you can change to macOS or Windows by modifying pythonpackage.yml.

- The test status, including full logs, can be found by clicking ‘Actions’ in the Github repo, or at:

https://github.com/spedas/pyspedas/actions

- You’ll want to look for the long-running ‘build’ workflows to see the logs

Merge pull request #294 from spedas/eric #1343

Re-run all jobs | Latest #2 | ···

Summary

Jobs

build (ubuntu-latest, 3.9)

Run details

Usage

Workflow file

build (ubuntu-latest, 3.9)

succeeded yesterday in 2h 14m 59s

Search logs

Set up job	2s
Run actions/checkout@v2	2s
Set up Python 3.9	0s
Install dependencies (Linux)	1m 56s
Install dependencies (Windows)	0s
Install dependencies (macOS)	0s
Lint with flake8	12s
Test with unittest	2h 12m 45s

```
1   ▶ Run # coverage run -a -m pyspedas.akebono.tests.tests
89  .........
90  ----------------------------------------------------------------------
91  Ran 9 tests in 18.919s
92
93  OK
94  ...
95  ----------------------------------------------------------------------
96  Ran 3 tests in 3.505s
97
98  OK
99  ....
100 ----------------------------------------------------------------------
101 Ran 4 tests in 3.646s
```

Code coverage of the unit tests

- We’re using coveralls to measure test coverage:

https://coveralls.io/github/spedas/pyspedas

- This tool will show you which lines in your code have been executed by the test suite, and which haven’t

- Note: files that the test suite doesn’t see won’t be included, so be sure to have tests that execute code in every file

COVERALLS

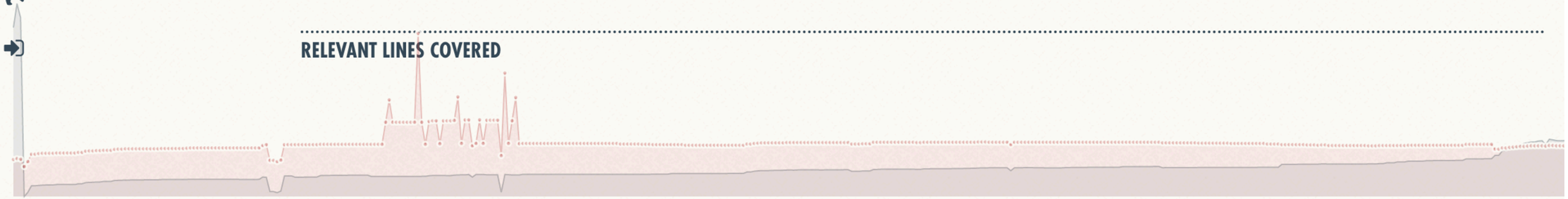
LAST BUILD ON BRANCH MASTER

BRANCH: MASTER

COMMITTED 11 JAN 2023 - 3:20 — COVERAGE: 71.114%. REMAINED THE SAME

BUILD #	BUILD TYPE	COMMITTED BY	COMMIT MESSAGE	RUN DETAILS
3886764794	push github-actions	GitHub	Merge pull request #294 from spedas/eric	**16130** of **22682** relevant lines covered **(71.11%)** **0.71** hits per line

RELEVANT LINES COVERED



22682 RELEVANT LINES 16130 COVERED LINES 0.71 HITS PER LINE

SOURCE FILES ON MASTER

TREE | LIST 492 | CHANGED 0 | SOURCE CHANGED 0 | COVERAGE CHANGED 0

- ▼ 71.11 pyspedas/
  - ▼ 99.19 ace/
    - ▶ 100.0 tests/
      - 96.67 __init__.py
      - 100.0 config.py
      - 100.0 load.py
    - ▶ akebono/

Validation tests

- We have tools for checking that IDL matches Python available at:

  https://github.com/spedas/pyspedas-validation

- These tools run in IDL and use the mgunit unit test framework.

- See the following example:

  https://github.com/spedas/pyspedas-validation/blob/main/src/example/validation_example_ut__define.pro

Creating a new mission plug-in

- I have a tool that uses AI to generate all of the initial files for a mission:

https://github.com/supervised/pyspedas-add-project

This requires an API key for Codex

```
;
; NOTES:
;       To run:
;         IDL> mgunit, 'validation_example_ut'
;
;       For the MMS validation tests, see:
;         projects/mms/common/tests/mms_python_validation_ut__define.pro
;
; $LastChangedBy: egrimes $
; $LastChangedDate: 2020-10-08 10:37:39 -0700 (Thu, 08 Oct 2020) $
; $LastChangedRevision: 29225 $
; $URL: svn+ssh://thmsvn@ambrosia.ssl.berkeley.edu/repos/spdsoft/trunk/general/spedas_tools/python_validation/py_validation_examp
;-

; the individual unit tests are implemented as methods that start with "test_"
function validation_example_ut::test_example
  ; first run IDL code to produce some tplot variables
  store_data, 'tplot_variable', data={x: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], y: [0, 0, 1, 0, 0, 2, 2, 1, 0, 1]}

  ; next, create an array containing a script to run in Python that should produce the same variables as above
  pyscript = ["from pytplot import store_data", $
              "store_data('tplot_variable', data={'x': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'y': [0, 0, 1, 0, 0, 2, 2, 1, 0, 1]})"]

  ; next, create an array containing the variables you would like to check
  vars = ['tplot_variable']

  ; the unit tests must return 1 if they pass, 0 if they fail
  ; spd_run_py_validation returns 1 if the variables in the array 'vars' match
  ; in both IDL and Python, and 0 if differences are found
  ; note: for performance reasons, only N data points are checked, where N is specified
  ; by the points_to_check keyword (default: 10)
  ; the maximum difference is specified by the tolerance keyword (default: 1e-6)
  return, spd_run_py_validation(pyscript, vars)
end

; the setup procedure runs before each test runs
pro validation_example_ut::setup
  del_data, '*'
end
```