HelioPy Documentation

Release 0+unknown

David Stansby

Jun 21, 2018

Contents

HelioPy is a free and open source set of tools for analysing space physics data in python. The main purpose is to make it easy to download and read in various space physics data sets.

CHAPTER 1

Getting started

1.1 HelioPy guide

1.1.1 Installing HelioPy

HelioPy is built on the Python programming language. The easiest way to install Python with the various required scientific python modules is to use Anaconda. Installation instructions can be found here.

Once you have a Python distribution installed, HelioPy can be installed using:

```
pip install heliopy
```

Optional dependencies

HDF file reader/writer

Saving data to hdf files for quicker access requires PyTables. (see *Speeding up file import* for more information)

CDF Library

Reading in any data that is stored in .cdf files requires an installation of the CDF library: https://cdf.gsfc.nasa.gov/

SPICE Toolkit

Working with spice kernels using the `heliopy.spice` module requires the python package spiceypy.

Installing from source

The latest source code is available at https://github.com/heliopython/heliopy/. To install from source follow these steps:

1. Install git
2. `git clone https://github.com/heliopython/heliopy.git`
3. `cd heliopy`
4. `pip install .`

This will install HelioPy from source and it's required dependancies.

1.1.2 Configuring HelioPy

HelioPy comes with a sample 'heliopyrc' file. In order to customise the file make a copy of it at `~/.heliopy/heliopyrc` and edit that copy. The default contents of the file are:

```
; Example heliopy configuration file. To make the configuration active, either
; edit this copy or move a copy to ~/.heliopy/heliopyrc
; The config parser will look in ~/.heliopy first.

[DEFAULT]
; The working directory is the parent directory in which all downloaded
; data will be stored.
download_dir = ~/heliopy/data

; Choose whether to convert all downloaded data to a hdf store, enabling much
; faster file reading after the initial load, but requiring the additional
; h5py and py-tables dependencies
use_hdf = False

; Cluster user cookie
cluster_cookie = none
```

Alternatively the copy included with HelioPy can be directly edited. To get the location of the configuration file in a python session run

```
from heliopy.util import config
print(config.get_config_file())
```

This will print the location of the configuration file that HelioPy is reading in.

1.1.3 What's new

Version 0.4

New features

- `swics_abundances()` and `swics_heavy_ions()` methods added for loading SWICS data from the Ulysses mission.
- `cdfpeek()` method added for peeking inside CDF files.

Backwards incompatible changes

- `heliopy.spice.Trajectory.generate_positions()` now takes a list of dates/times at which to generate orbital positions, instead of a start time, stop time, and number of steps. The old behaviour can be recovered by manually generating an evenly spaced list of times.

Version 0.3

New features

HelioPy now contiains code for working with SPICE kernels. See the following modules for more information:

- `heliopy.data.spice` module for downloading spice kernels
- `heliopy.spice` module for automatically processing spice kernels

Removed features

- The `heliopy.plasma` module has been removed (see http://www.plasmapy.org/ for the recommended alternative)
- `heliopy.plot` code removed

Version 0.2

New features

- Convert examples gallery to automatically generate plots
- Added `HelioPy.data.helper.listdata()` method for easily viewing the amount of data HelioPy is storing locally.
- Added `heliopy.data.wind.threedp_sfpd()` method for importing WIND 3DP sfpd data.

Version 0.1.3

Fixed bugs

- Correctly report download percentage when downloading files.
- Fix issue where `heliopy.data.helios.corefit()` made duplicate .hdf files on days where no data is available.

1.2 Examples

Note: Click *here* to download the full example code

1.2.1 Speeding up file import

For some files, reading in the original files can be very slow in Python. heliopy has the option to automatically save files into the binary hdf file format. Two copies of the data will be stored locally (original and hdf), but will significantly speed up loading files.

To enable this option, make sure PyTables is installed using:

```
pip install pytables
```

And then edit your heliopyrc file to enable hdf file saving (see *Configuring HelioPy* for more information).

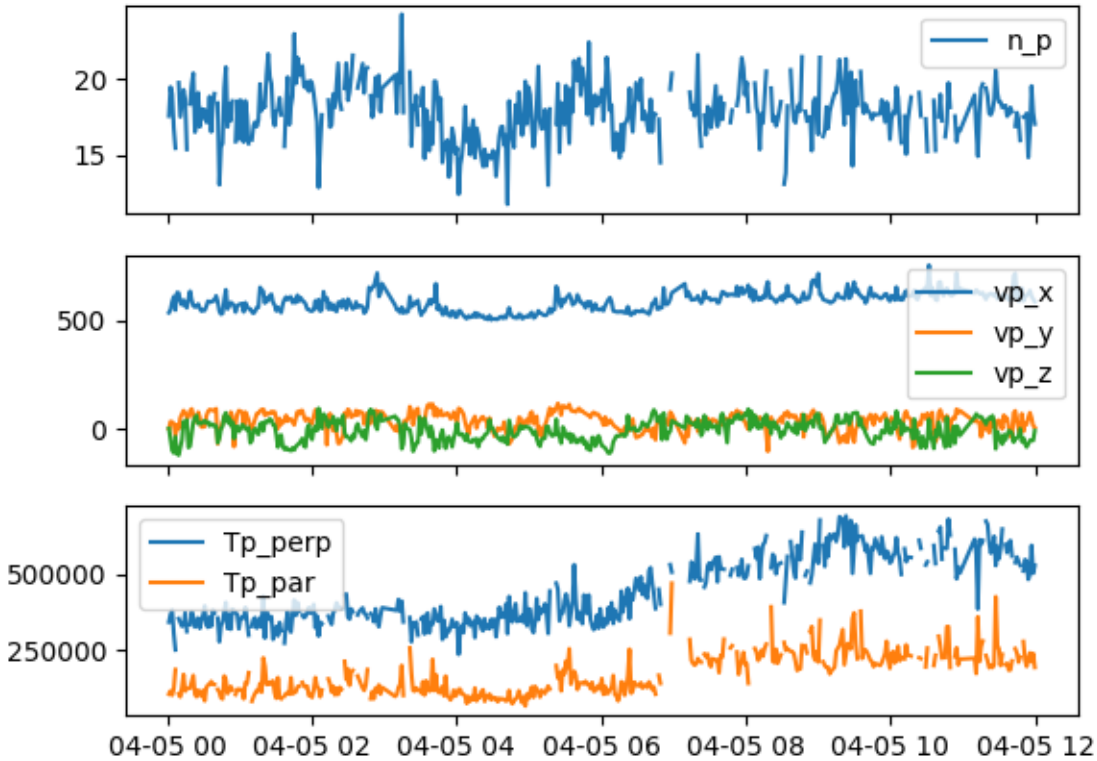
To check how much data is stored in both it's original format and hdf format see *Local data inventory*.

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

1.2.2 Importing data

A short example showing how to import and plot plasma data.



Out:

```
Index(['B instrument', 'Bx', 'By', 'Bz', 'sigma B', 'Ion instrument', 'Status',
       'Tp_par', 'Tp_perp', 'carrot', 'r_sun', 'clat', 'clong',
       'earth_he_angle', 'n_p', 'vp_x', 'vp_y', 'vp_z', 'vth_p_par',
       'vth_p_perp'],
      dtype='object')
```

```
from datetime import datetime, timedelta
import heliopy.data.helios as helios
import matplotlib.pyplot as plt

starttime = datetime(1976, 4, 5, 0, 0, 0)
endtime = starttime + timedelta(hours=12)
probe = '2'

data = helios.corefit(probe, starttime, endtime)

print(data.keys())
```

(continues on next page)

(continued from previous page)

```
fig, axs = plt.subplots(3, 1, sharex=True)
axs[0].plot(data['n_p'])
axs[1].plot(data['vp_x'])
axs[1].plot(data['vp_y'])
axs[1].plot(data['vp_z'])
axs[2].plot(data['Tp_perp'])
axs[2].plot(data['Tp_par'])

for ax in axs:
    ax.legend()
plt.show()
```

Total running time of the script: (0 minutes 6.084 seconds)

Note: Click *here* to download the full example code

1.2.3 A brief guide to pandas multi-index dataframes

The data sets imported by heliopy are returned in a `pandas.DataFrame`. Most data sets have a single variable for the `DataFrame` index. In this case DataFrame can be thought of as storing multiple variables, each of which depend on the same dependent variable. For example the x component of magnetic field depends only on time.

Particle distribution functions have more than one dependent variable however. Typically the distribution is measured as a function of time, energy, and two angles.

```
import pandas as pd
import numpy as np
```

As a simple example, lets make a multi-index DataFrame to imitate a particle distribution function. First, set the index values

```
# Energy values
E = np.arange(0, 4)
# Azimuthal angles
phi = np.arange(0, 360, 90)
# Elevation angles
theta = np.arange(-90, 90, 45)
# Fake distribution function
dist = np.random.rand(E.size * phi.size * theta.size)
```

The index values can then be created from `pandas.MultiIndex.from_product()`, and the complete DataFrame created from `pandas.DataFrame()`. The distribution function is sorted first by energy, then by theta, and then by phi.

```
index = pd.MultiIndex.from_product((E, theta, phi),
                                   names=('E', 'theta', 'phi'))
data = pd.DataFrame(data=dist, index=index, columns=['df'])
print(data.head())
```

Out:

```
df
E theta phi
```

(continues on next page)

(continued from previous page)

```
0 -90   0    0.639512
        90   0.429693
        180  0.247111
        270  0.380913
  -45   0    0.864078
```

To loop through the energy levels with the index label `'theta'` do the following. `theta` will give the value of the current index, and `theta_dist` will give the reduced DataFrame at each value of `theta`.

```
for theta, theta_data in data.groupby(level='theta'):
    print(theta)
```

Out:

```
-90
-45
0
45
```

Total running time of the script: (0 minutes 0.108 seconds)

Note: Click *here* to download the full example code

1.2.4 Local data inventory

`heliopy.data.helper` contains the method `heliopy.data.helper.listdata()` that can be useful for working out how much data is stored locally. It can be run using

```
from heliopy.data import helper as heliohelper
heliohelper.listdata()
```

This will print a table with each probe and the total raw data stored along with the total *.hdf* file data stored (*.hdf* files are binary files that are much faster for python to read than raw data).

Example output is:

```
'''
  Scanning files in /Users/dstansby/Data/
  ---------------------------------------
  |      Probe |        Raw |        HDF |
  |-------------------------------------|
  |        ace |    1.44 MB |  800.00  B |
  |    cluster |  200.39 MB |    0.00  B |
  |     helios |    2.37 GB |    1.41 GB |
  |        imp |   19.76 MB |   28.56 MB |
  |  messenger |   15.24 MB |   27.21 MB |
  |        mms |   70.11 MB |    0.00  B |
  |     themis |   64.31 MB |    0.00  B |
  |    ulysses |   54.78 MB |   47.98 MB |
  |       wind |  176.84 MB |   63.82 MB |
  |-------------------------------------|
  |-------------------------------------|
  |      Total |    2.96 GB |    1.57 GB |
```

(continues on next page)

(continued from previous page)

```
  ----------------------------------------
'''
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

1.2.5 SPICE orbit plotting

How to plot orbits from SPICE kernels.

In this example we download the Solar Orbiter SPICE kernel, and plot it's orbit from 2020 to 2028.

```
import heliopy.data.spice as spicedata
import heliopy.spice as spice
from datetime import datetime, timedelta
import astropy.units as u
import numpy as np
```

Load the solar orbiter spice kernel. HelioPy will automatically fetch the latest kernel

```
orbiter_kernel = spicedata.get_kernel('solar orbiter 2020')
spice.furnish(orbiter_kernel)
orbiter = spice.Trajectory('Solar Orbiter')
```

Generate a time for every day between starttime and endtime

```
starttime = datetime(2020, 3, 1)
endtime = datetime(2028, 1, 1)
times = []
while starttime < endtime:
    times.append(starttime)
    starttime += timedelta(days=1)
```

Generate positions

```
orbiter.generate_positions(times, 'Sun', 'ECLIPJ2000')
orbiter.change_units(u.au)
```

Plot the orbit. The orbit is plotted in 3D

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from astropy.visualization import quantity_support
quantity_support()

# Generate a set of timestamps to color the orbits by
times_float = [(t - orbiter.times[0]).total_seconds() for t in orbiter.times]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
kwargs = {'s': 3, 'c': times_float}
ax.scatter(orbiter.x, orbiter.y, orbiter.z, **kwargs)
ax.set_xlim(-1, 1)
```

(continues on next page)

(continued from previous page)

```
ax.set_ylim(-1, 1)
ax.set_zlim(-1, 1)
```

Plot radial distance and elevation as a function of time

```
elevation = np.rad2deg(np.arcsin(orbiter.z / orbiter.r))

fig, axs = plt.subplots(2, 1, sharex=True)
axs[0].plot(orbiter.times, orbiter.r)
axs[0].set_ylim(0, 1.1)
axs[0].set_ylabel('r (AU)')

axs[1].plot(orbiter.times, elevation)
axs[1].set_ylabel('Elevation (deg)')

plt.show()
```

Total running time of the script: (0 minutes 0.000 seconds)

CHAPTER 2

Module documentation

2.1 Data import (`heliopy.data`)

Methods for automatically importing data to python. Each spacecraft has its own sub-module:

2.1.1 ACE (`heliopy.data.ace`)

heliopy.data.ace Module

Methods for importing data from the ACE spacecraft.

All data is publically available at ftp://spdf.gsfc.nasa.gov/pub/data/ace/. The ACE spacecraft homepage can be found at http://www.srl.caltech.edu/ACE/.

Functions

mfi_h0(starttime, endtime)	Import 'mfi_h0' magnetic field data product from ACE.
swe_h0(starttime, endtime)	Import swe_h0 particle moment data product from ACE.

mfi_h0

`heliopy.data.ace.`**`mfi_h0`**(*starttime*, *endtime*)

Import 'mfi_h0' magnetic field data product from ACE. See ftp://spdf.gsfc.nasa.gov/pub/data/ace/mag/ for more information.

Parameters

- **`starttime`** (*datetime*) – Interval start time.
- **`endtime`** (*datetime*) – Interval end time.

Returns data

Return type DataFrame

swe_h0

`heliopy.data.ace.swe_h0`(*starttime*, *endtime*)

Import swe_h0 particle moment data product from ACE. See https://cdaweb.sci.gsfc.nasa.gov/misc/NotesA.html#AC_H0_SWE for more information.

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns data

Return type DataFrame

2.1.2 ARTEMIS

heliopy.data.artemis Module

Methods for importing data from the THEMIS/ARTEMIS spacecraft.

All data is publically available at http://themis.ssl.berkeley.edu/data/themis/.

Functions

fgm(probe, rate, coords, starttime, endtime)	Import fgm magnetic field data from THEMIS.

fgm

`heliopy.data.artemis.fgm`(*probe*, *rate*, *coords*, *starttime*, *endtime*)

Import fgm magnetic field data from THEMIS.

Parameters

- **probe** (*string*) – Alowed values are [a, b, c, d, e].
- **rate** (*string*) – Date rate to return. Allowed values are [e, h, l, s].
- **coords** (*string*) – Magnetic field co-ordinate system. Allowed values are [dsl, gse, gsm, ssl]. NOTE: Add link to co-ordinate system descriptions.
- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns data

Return type DataFrame

2.1.3 Cassini

heliopy.data.cassini Module

Methods for importing data from the Cassini spacecraft.

All data is publically available at http://pds-atmospheres.nmsu.edu/data_and_services/atmospheres_data/Cassini/Cassini.html

Functions

mag_1min(starttime, endtime, coords)	Import 1 minute magnetic field from Cassini.
mag_hires(starttime, endtime)	Import high resolution magnetic field from Cassini.

mag_1min

`heliopy.data.cassini.`**`mag_1min`**(*starttime*, *endtime*, *coords*)

Import 1 minute magnetic field from Cassini.

See https://pds-ppi.igpp.ucla.edu/search/view/?f=yes&id=pds://PPI/CO-E_SW_J_S-MAG-4-SUMM-1MINAVG-V1.0 for more information.

Cassini Orbiter Magnetometer Calibrated MAG data in 1 minute averages available covering the period 1999-08-16 (DOY 228) to 2016-12-31 (DOY 366). The data are provided in RTN coordinates throughout the mission, with Earth, Jupiter, and Saturn centered coordinates for the respective flybys of those planets.

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.
- **coords** (*strings*) – Requested coordinate system. Must be one of `['KRTP', 'KSM', 'KSO', 'RTN']`

Returns **data** – Requested data

Return type DataFrame

mag_hires

`heliopy.data.cassini.`**`mag_hires`**(*starttime*, *endtime*)

Import high resolution magnetic field from Cassini.

See http://pds-ppi.igpp.ucla.edu/search/view/?f=yes&id=pds://PPI/CO-E_SW_J_S-MAG-3-RDR-FULL-RES-V1.0 for more information.

Cassini Orbiter Magnetometer Calibrated MAG data at the highest time resolution available covering the period 1999-08-16 (DOY 228) to 2016-12-31 (DOY 366).

The data are in RTN coordinates prior Cassini's arrival at Saturn, and Kronographic (KRTP) coordinates at Saturn (beginning 2004-05-14, DOY 135).

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data** – Requested data

Return type DataFrame

2.1.4 Cluster

heliopy.data.cluster Module

Methods for importing data from the four Cluster spacecraft.

To download data you will need to be registered at the cluster science archive (http://www.cosmos.esa.int/web/csa/register-now), and have set either the environment variable CLUSTERCOOKIE to your cookie, or set your cookie in the *heliopyrc* file.

The data download method is described at https://csa.esac.esa.int/csa/aio/html/wget.shtml.

Functions

`cis_codif_h1_moms`(probe, starttime, endtime)	Load H+ moments from CIS instrument.
`cis_hia_onboard_moms`(probe, starttime, endtime)	Download onboard ion moments from the CIS instrument.
`fgm`(probe, starttime, endtime)	Download fluxgate magnetometer data.
`peace_moments`(probe, starttime, endtime)	Download electron moments from the PEACE instrument.

cis_codif_h1_moms

`heliopy.data.cluster.`**`cis_codif_h1_moms`**(*probe*, *starttime*, *endtime*, *sensitivity='high'*)

Load H+ moments from CIS instrument.

See https://caa.estec.esa.int/documents/UG/CAA_EST_UG_CIS_v35.pdf for more information on the CIS data.

Parameters

- **`probe`** (*`string`*) – Probe number. Must be '1', '2', '3', or '4'.
- **`starttime`** (*`datetime`*) – Interval start.
- **`endtime`** (*`datetime`*) – Interval end.
- **`sensitivity`**(*`string, 'high' or 'low', default: 'low'`*) – Load high or low sensitivity

Returns **data** – Requested data.

Return type DataFrame

cis_hia_onboard_moms

`heliopy.data.cluster.`**`cis_hia_onboard_moms`**(*probe*, *starttime*, *endtime*)

Download onboard ion moments from the CIS instrument.

See https://caa.estec.esa.int/documents/UG/CAA_EST_UG_CIS_v35.pdf for more information on the CIS data.

Parameters

- **probe** (*string*) – Probe number. Must be '1' or '3'
- **starttime** (*datetime*) – Interval start.
- **endtime** (*datetime*) – Interval end.

Returns **data** – Requested data.

Return type DataFrame

fgm

`heliopy.data.cluster.fgm`(*probe*, *starttime*, *endtime*)

Download fluxgate magnetometer data.

See https://caa.estec.esa.int/documents/UG/CAA_EST_UG_FGM_v60.pdf for more information on the FGM data.

Parameters

- **probe** (*string*) – Probe number. Must be '1', '2', '3', or '4'.
- **starttime** (*datetime*) – Interval start.
- **endtime** (*datetime*) – Interval end.

Returns **data** – Requested data.

Return type DataFrame

peace_moments

`heliopy.data.cluster.peace_moments`(*probe*, *starttime*, *endtime*)

Download electron moments from the PEACE instrument.

See https://caa.estec.esa.int/documents/UG/CAA_EST_UG_PEA_v25.pdf for more information on the PEACE data.

Parameters

- **probe** (*string*) – Probe number. Must be '1', '2', '3', or '4'.
- **starttime** (*datetime*) – Interval start.
- **endtime** (*datetime*) – Interval end.

Returns **data** – Requested data.

Return type DataFrame

2.1.5 Helios

heliopy.data.helios Module

Methods for importing Helios data.

In general the data are available form a number of sources (replace 'helios1' with 'helios2' in url to change probe):

- Distribution functions - Not publically available
- Merged plasma/mangetic field - ftp://cdaweb.gsfc.nasa.gov/pub/data/helios/helios1/merged/

- 6 second cadence magnetic field - ftp://cdaweb.gsfc.nasa.gov/pub/data/helios/helios1/mag/6sec_ness/
- Trajectory - ftp://cdaweb.gsfc.nasa.gov/pub/data/helios/helios1/traj/

If the data is publically available, it will be dowloaded automatically if it doesn't exist locally.

Functions

corefit(probe, starttime, endtime[, ...])	Read in merged data set
distparams(probe, starttime, endtime[, verbose])	Read in distribution parameters found in the header of distribution files.
distparams_single(probe, year, doy, hour, ...)	Read in parameters from a single distribution function measurement.
electron_dist_single(probe, year, doy, hour, ...)	Read in 2D electron distribution function.
electron_dists(probe, starttime, endtime[, ...])	Return 2D electron distributions between *starttime* and *endtime*
integrated_dists(probe, starttime, endtime)	Returns the integrated distributions from experiments i1a and i1b in Helios distribution function files.
integrated_dists_single(probe, year, doy, ...)	Returns the integrated distributions from experiments i1a and i1b in Helios distribution function files.
ion_dist_single(probe, year, doy, hour, ...)	Read in ion distribution function.
ion_dists(probe, starttime, endtime[, ...])	Return 3D ion distributions between *starttime* and *endtime*
mag_4hz(probe, starttime, endtime[, ...])	Read in 4Hz magnetic field data.
mag_ness(probe, starttime, endtime[, ...])	Read in 6 second magnetic field data.
merged(probe, starttime, endtime[, verbose, ...])	Read in merged data set.
trajectory(probe, starttime, endtime)	Read in trajectory data for Helios (and the Earth) - Position vectors are in AU - Velocities are in km/s

corefit

`heliopy.data.helios.`**`corefit`**(*probe*, *starttime*, *endtime*, *verbose=False*, *try_download=True*)

Read in merged data set

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Interval start time
- **endtime** (*datetime*) – Interval end time
- **verbose** (*bool, optional*) – If `True`, print information as data is loading. Default is `False`.

Returns **data** – Data set

Return type DataFrame

distparams

`heliopy.data.helios.`**`distparams`**(*probe*, *starttime*, *endtime*, *verbose=False*)

Read in distribution parameters found in the header of distribution files.

Parameters

- **probe** (*int*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval
- **verbose** (*bool, optional*) – If `True`, print information whilst loading. Default is `False`.

Returns **distinfo** – Infromation stored in the top of distribution function files

Return type Series

distparams_single

`heliopy.data.helios.`**`distparams_single`**(*probe*, *year*, *doy*, *hour*, *minute*, *second*)

Read in parameters from a single distribution function measurement.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **hour** (*int*) – Hour
- **minute** (*int*) – Minute
- **second** (*int*) – Second

Returns **distparams** – Distribution parameters from top of distribution function file.

Return type Series

electron_dist_single

`heliopy.data.helios.`**`electron_dist_single`**(*probe*, *year*, *doy*, *hour*, *minute*, *second*, *remove_advect=False*)

Read in 2D electron distribution function.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **hour** (*int*) – Hour.
- **minute** (*int*) – Minute
- **second** (*int*) – Second
- **remove_advect** (*bool, optional*) – If `False`, the distribution is returned in the spacecraft frame.

  If `True`, the distribution is returned in the solar wind frame, by subtracting the spacecraft velocity from the velcoity of each bin. Note this significantly slows down reading in the distribution.

Returns **dist** – 2D electron distribution function

Return type DataFrame

electron_dists

`heliopy.data.helios.`**`electron_dists`**(*probe*, *starttime*, *endtime*, *remove_advect=False*, *verbose=False*)

Return 2D electron distributions between *starttime* and *endtime*

Parameters

- **probe** (*int*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval
- **remove_advect** (*bool, optional*) – If *False*, the distribution is returned in the spacecraft frame.

  If *True*, the distribution is returned in the solar wind frame, by subtracting the spacecraft velocity from the velcoity of each bin. Note this significantly slows down reading in the distribution.
- **verbose** (*bool, optional*) – If `True`, print dates when loading files. Default is `False`.

Returns **dists** – Electron distribution functions

Return type DataFrame

integrated_dists

`heliopy.data.helios.`**`integrated_dists`**(*probe*, *starttime*, *endtime*, *verbose=False*)

Returns the integrated distributions from experiments i1a and i1b in Helios distribution function files.

The distributions are integrated over all angles and given as a function of proton velocity.

Parameters

- **probe** (*int*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval
- **verbose** (*bool, optional*) – If `True`, print information whilst loading. Default is `False`.

Returns **distinfo** – Infromation stored in the top of distribution function files.

Return type Series

integrated_dists_single

`heliopy.data.helios.`**`integrated_dists_single`**(*probe*, *year*, *doy*, *hour*, *minute*, *second*)

Returns the integrated distributions from experiments i1a and i1b in Helios distribution function files.

The distributions are integrated over all angles and given as a function of proton velocity.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **hour** (*int*) – Hour
- **minute** (*int*) – Minute.
- **second** (*int*) – Second

Returns

- **i1a** (*DataFrame*) – i1a integrated distribution function.
- **i1b** (*DataFrame*) – i1b integrated distribution function.

ion_dist_single

`heliopy.data.helios.ion_dist_single`(*probe*, *year*, *doy*, *hour*, *minute*, *second*, *remove_advect=False*)

Read in ion distribution function.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **hour** (*int*) – Hour
- **minute** (*int*) – Minute.
- **second** (*int*) – Second
- **remove_advect** (*bool, optional*) – If *False*, the distribution is returned in the spacecraft frame.

  If *True*, the distribution is returned in the solar wind frame, by subtracting the spacecraft velocity from the velcoity of each bin. Note this significantly slows down reading in the distribution.

Returns **dist** – 3D ion distribution function

Return type DataFrame

ion_dists

`heliopy.data.helios.ion_dists`(*probe*, *starttime*, *endtime*, *remove_advect=False*, *verbose=False*)

Return 3D ion distributions between *starttime* and *endtime*

Parameters

- **probe** (*int*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval

- **remove_advect** (*bool, optional*) – If *False*, the distribution is returned in the spacecraft frame.

  If *True*, the distribution is returned in the solar wind frame, by subtracting the spacecraft velocity from the velcoity of each bin. Note this significantly slows down reading in the distribution.

- **verbose** (*bool, optional*) – If `True`, print dates when loading files. Default is `False`.

Returns **distinfo** – Infromation stored in the top of distribution function files.

Return type Series

mag_4hz

`heliopy.data.helios.`**`mag_4hz`**(*probe*, *starttime*, *endtime*, *verbose=True*, *try_download=True*)

Read in 4Hz magnetic field data.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Interval start time
- **endtime** (*datetime*) – Interval end time
- **verbose** (*bool, optional*) – If `True`, print more information as data is loading. Default is `True`.
- **try_download** (*bool, optional*) – If `False` don't try to download data if it is missing locally. Default is `False`.

Returns **data** – 4Hz magnetic field data set

Return type DataFrame

mag_ness

`heliopy.data.helios.`**`mag_ness`**(*probe*, *starttime*, *endtime*, *verbose=True*, *try_download=True*)

Read in 6 second magnetic field data.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Interval start time
- **endtime** (*datetime*) – Interval end time
- **verbose** (*bool, optional*) – If `True`, print more information as data is loading. Default is `True`.

Returns **data** – 6 second magnetic field data set

Return type DataFrame

merged

`heliopy.data.helios.merged`(*probe*, *starttime*, *endtime*, *verbose=True*, *try_download=True*)

Read in merged data set.

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **starttime** (*datetime*) – Interval start time
- **endtime** (*datetime*) – Interval end time
- **verbose** (*bool, optional*) – If `True`, print information as data is loading. Default is `True`.

Returns **data** – Merged data set

Return type DataFrame

Notes

This is an old dataset, and it is recommended to use *corefit* instead.

trajectory

`heliopy.data.helios.trajectory`(*probe*, *starttime*, *endtime*)

Read in trajectory data for Helios (and the Earth) - Position vectors are in AU - Velocities are in km/s

Parameters

- **probe** (*int, string*) – Helios probe to import data from. Must be 1 or 2.
- **year** (*int*) – Year
- **doy** (*int*) – Day of year

Returns **data** – Helios trajectory data

Return type DataFrame

2.1.6 IMP

heliopy.data.imp Module

Methods for importing data from the IMP spacecraft.

All data is publically available at ftp://cdaweb.gsfc.nasa.gov/pub/data/imp/

Functions

mag15s(probe, starttime, endtime[, verbose])	Import 15s cadence magnetic field data.
mag320ms(probe, startTime, endTime)	Import 320ms cadence magnetic field data.
merged(probe, starttime, endtime[, verbose])	Import merged plasma data.
mitplasma_h0(probe, starttime, endtime)	Import mit h0 plasma data.

mag15s

`heliopy.data.imp.`**`mag15s`**(*probe*, *starttime*, *endtime*, *verbose=False*)

Import 15s cadence magnetic field data.

Parameters

- **probe** (*string*) – Probe number.
- **starttime** (*datetime*) – Start of interval.
- **endtime** (*datetime*) – End of interval.
- **verbose** (*bool, optional*) – If `True`, print information whilst loading. Default is `False`.

Returns **data** – Requested data.

Return type DataFrame

mag320ms

`heliopy.data.imp.`**`mag320ms`**(*probe*, *startTime*, *endTime*)

Import 320ms cadence magnetic field data.

Parameters

- **probe** (*string*) – Probe number.
- **starttime** (*datetime*) – Start of interval.
- **endtime** (*datetime*) – End of interval.

Returns **data** – Requested data.

Return type DataFrame

merged

`heliopy.data.imp.`**`merged`**(*probe*, *starttime*, *endtime*, *verbose=False*)

Import merged plasma data. See ftp://cdaweb.gsfc.nasa.gov/pub/data/imp/imp8/merged/00readme.txt for information on variables.

Parameters

- **probe** (*string*) – Probe number.
- **starttime** (*datetime*) – Start of interval.
- **endtime** (*datetime*) – End of interval.
- **verbose** (*bool, optional*) – If `True`, print information whilst loading. Default is `False`.

Returns **data** – Requested data.

Return type DataFrame

mitplasma_h0

`heliopy.data.imp.`**`mitplasma_h0`**(*probe*, *starttime*, *endtime*)

Import mit h0 plasma data.

Parameters

- **probe** (*string*) – Probe number.
- **starttime** (*datetime*) – Start of interval.
- **endtime** (*datetime*) – End of interval.

Returns **data** – Requested data.

Return type DataFrame

2.1.7 MESSENGER

heliopy.data.messenger Module

Methods for importing data from the Messenger spacecraft.

All data is publically available at ftp://spdf.gsfc.nasa.gov/pub/data/messenger

Functions

mag_rtn(starttime, endtime)	Import magnetic field in RTN coordinates from Messenger.

mag_rtn

`heliopy.data.messenger.`**`mag_rtn`**(*starttime*, *endtime*)

Import magnetic field in RTN coordinates from Messenger.

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

2.1.8 MMS

heliopy.data.mms Module

Methods for importing data from the four MMS spacecraft.

All data is publically available at https://lasp.colorado.edu/mms/sdc/public/data/, and the MMS science data centre is at https://lasp.colorado.edu/mms/sdc/public/.

Functions

fgm_survey(probe, starttime, endtime)	Import fgm survey mode magnetic field data.
fpi_dis_moms(probe, mode, starttime, endtime)	Import fpi ion distribution moment data.

fgm_survey

`heliopy.data.mms.`**`fgm_survey`**(*probe*, *starttime*, *endtime*)

Import fgm survey mode magnetic field data.

Parameters

- **probe** (*string*) – Probe number, must be 1, 2, 3, or 4
- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data** – Imported data.

Return type DataFrame

fpi_dis_moms

`heliopy.data.mms.`**`fpi_dis_moms`**(*probe*, *mode*, *starttime*, *endtime*)

Import fpi ion distribution moment data.

Parameters

- **probe** (*string*) – Probe number, must be 1, 2, 3, or 4
- **mode** (*string*) – Data mode, must be 'fast' or 'brst'
- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data** – Imported data.

Return type DataFrame

2.1.9 Ulysses

heliopy.data.ulysses Module

Methods for importing data from the Ulysses spacecraft.

All data is publically available at http://ufa.esac.esa.int/ufa/

Functions

fgm_hires(starttime, endtime)	Import high resolution fluxgate magnetometer data.
swics_abundances(starttime, endtime)	Import swics abundance data.
swics_heavy_ions(starttime, endtime)	Import swics heavy ion data.
swoops_ions(starttime, endtime)	Import SWOOPS ion data.

fgm_hires

`heliopy.data.ulysses.fgm_hires(starttime, endtime)`

Import high resolution fluxgate magnetometer data.

Parameters

- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval

Returns **data** – Requested data

Return type DataFrame

swics_abundances

`heliopy.data.ulysses.swics_abundances(starttime, endtime)`

Import swics abundance data.

The variables in this dataset are:

- VEL_ALPHA: alpha velocity
- RAT_C6_C5: ratio of carbon 6+ to 5+
- RAT_O7_O6: ratio of oxygen 7+ to 6+
- RAT_FE_O: abundance ratio of iron to oxygen
- CHARGE_FE: average charge state of iron
- N_CYC: number of instrument cycles in average

See http://ufa.esac.esa.int/ufa-sl-server/data-action?PROTOCOL=HTTP&PRODUCT_TYPE=ALL&FILE_NAME=readme.txt&FILE_PATH=/ufa/HiRes/data/swics for more information.

Parameters

- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval

Returns **data** – Requested data

Return type DataFrame

swics_heavy_ions

`heliopy.data.ulysses.swics_heavy_ions(starttime, endtime)`

Import swics heavy ion data.

The variables in this dataset are:

- DENS_ALPHA: alpha to oxygen 6+ density ratio
- VEL_ALPHA: alpha velocity
- TEMP_ALPHA: alpha temperature
- DENS_C6: carbon 6+ to oxygen 6+ density ratio
- VEL_C6: carbon 6+ velocity

- TEMP_C6: carbon 6+ temperature
- DENS_O6: oxygen 6+ density in cm^-3
- VEL_O6: oxygen 6+ velocity
- TEMP_O6: oxygen 6+ temperature
- DENS_NE8: neon 8+ to oxygen 6+ density ratio
- VEL_NE8: neon 8+ velocity
- TEMP_NE8: neon 8+ temperature
- DENS_MG10: magnesium 10+ to oxygen 6+ density ratio
- VEL_MG10: magnesium 10+ velocity
- TEMP_MG10: magnesium 10+ temperature
- DENS_SI9: silicon 9+ to oxygen 6+ density ratio
- VEL_SI9: silicon 9+ velocity
- TEMP_SI9: silicon 9+ temperature
- DENS_S10: sulphur 10+ to oxygen 6+ density ratio
- VEL_S10: sulphur 10+ velocity
- TEMP_S10: sulphur 10+ temperature
- DENS_FE11: iron 11+ to oxygen 6+ density ratio
- VEL_FE11: iron 11+ velocity
- TEMP_FE11: iron 11+ temperature

See http://ufa.esac.esa.int/ufa-sl-server/data-action?PROTOCOL=HTTP&PRODUCT_TYPE=ALL&FILE_NAME=readme.txt&FILE_PATH=/ufa/HiRes/data/swics for more information.

Parameters

- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval

Returns **data** – Requested data

Return type DataFrame

swoops_ions

`heliopy.data.ulysses.`**`swoops_ions`**(*starttime*, *endtime*)

Import SWOOPS ion data.

Parameters

- **starttime** (*datetime*) – Start of interval
- **endtime** (*datetime*) – End of interval

Returns **data** – Requested data

Return type DataFrame

2.1.10 WIND

heliopy.data.wind Module

Methods for importing data from the WIND spacecraft.

All data is publically available at ftp://spdf.gsfc.nasa.gov/pub/data/wind. See https://wind.nasa.gov/data_sources.php for more information on different data products.

Functions

mfi_h0(starttime, endtime)	Import 'mfi_h0' magnetic field data product from WIND.
mfi_h2(starttime, endtime)	Import 'mfi_h2' magnetic field data product from WIND.
swe_h1(starttime, endtime)	Import 'h1' (Bi-Maxwellian, Anisotropic Analysis of Protons and Alphas) solar wind ion data product from WIND.
swe_h3(starttime, endtime)	Import 'h3' solar wind electron data product from WIND.
threedp_pm(starttime, endtime)	Import 'pm' wind data.
threedp_sfpd(starttime, endtime)	Import 'sfpd' wind data.

mfi_h0

`heliopy.data.wind.`**`mfi_h0`**(*starttime*, *endtime*)

Import 'mfi_h0' magnetic field data product from WIND.

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

mfi_h2

`heliopy.data.wind.`**`mfi_h2`**(*starttime*, *endtime*)

Import 'mfi_h2' magnetic field data product from WIND.

The highest time resolution data (11 vectors/sec usually, and 22 vectors/sec when near Earth)

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

swe_h1

`heliopy.data.wind.`**`swe_h1`**(*starttime*, *endtime*)

Import 'h1' (Bi-Maxwellian, Anisotropic Analysis of Protons and Alphas) solar wind ion data product from WIND.

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

swe_h3

`heliopy.data.wind.`**`swe_h3`**(*starttime*, *endtime*)

Import 'h3' solar wind electron data product from WIND.

Electron pitch angle files providing electron fluxes at 30 directional bins relative to the instantaneous magnetic field direction at 13 different energy levels

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

threedp_pm

`heliopy.data.wind.`**`threedp_pm`**(*starttime*, *endtime*)

Import 'pm' wind data.

3 second time resolution solar wind proton and alpha particle moments from the PESA LOW sensor, computed on-board the spacecraft

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

threedp_sfpd

`heliopy.data.wind.`**`threedp_sfpd`**(*starttime*, *endtime*)

Import 'sfpd' wind data.

12 second energetic electron pitch-angle energy spectra from the foil SST

Parameters

- **starttime** (*datetime*) – Interval start time.
- **endtime** (*datetime*) – Interval end time.

Returns **data**

Return type DataFrame

There is also a module for downloading SPICE kernels:

2.1.11 SPICE

Methods for automatically downloading SPICE kernels for various objects.

This is essentially a library of SPICE kernels that are available online, so users don't have go go hunting for them. If you know a kernel is out of date, and HelioPy should be using a newer kernel please let us know at https://github.com/heliopython/heliopy/issues.

`heliopy.data.spice.get_kernel(name)`

Get the local location of a kernel.

If a kernel isn't available locally, it is download.

Parameters **name** (*str*) – Kernel name. See PUT LINK HERE for a list of available names.

Returns Local location of kernel.

Return type str

and helper methods that much of the data import uses are also available in the helper module:

2.1.12 Helper methods

heliopy.data.helper Module

Helper methods for importing data

Functions

cdf2df(cdf, index_key[, keys, dtimeindex, ...])	Converts a cdf file to a pandas dataframe.
cdfpeek(cdf_loc)	List all the variables present in a CDF file, along with their size.
doy2ymd(y, doy)	Converts day of year and year to year, month, day
dtime2doy(dt)	Returns day of year of a datetime object.
listdata([probes])	Print amount of data stored locally in the heliopy data directory.
load(filename, local_dir, remote_url[, ...])	Try to load a file from *local_dir*.
pitchdist_cdf2df(cdf, distkeys, energykey, ...)	Converts cdf file of a pitch angle distribution to a pandas dataframe.
timefilter(data, starttime, endtime)	Puts data in a single dataframe, and filters it between times.

cdf2df

`heliopy.data.helper.`**`cdf2df`**(*cdf*, *index_key*, *keys=None*, *dtimeindex=True*, *badvalues=None*)

Converts a cdf file to a pandas dataframe.

Parameters

- **cdf** (*cdf*) – Opened cdf file
- **index_key** (*string*) – The key to use as indexing in the output dataframe
- **keys** (*dict, optional*) – A dictionary that maps keys in the cdf file to the corresponding desired keys in the ouput dataframe. If a particular cdf key has multiple columns, the mapped keys must be in a list.
- **dtimeindex** (*bool, optional*) – If `True`, DataFrame index is parsed as a datetime. Default is `True`.
- **badvalues** (*dict, optional*) – A dictionary that maps the new DataFrame keys to a list of bad values to replace with nans.

Returns **df** – Data frame with read in data

Return type `pandas.DataFrame`

cdfpeek

`heliopy.data.helper.`**`cdfpeek`**(*cdf_loc*)

List all the variables present in a CDF file, along with their size.

Parameters **cdf_loc** (*string*) – Local location of the cdf file.

doy2ymd

`heliopy.data.helper.`**`doy2ymd`**(*y*, *doy*)

Converts day of year and year to year, month, day

Parameters

- **y** (*int*) – Year
- **doy** (*int*) – Day of year

Returns

- **year** (*int*) – Year
- **month** (*int*) – Month
- **day** (*int*) – Day of month

dtime2doy

`heliopy.data.helper.`**`dtime2doy`**(*dt*)

Returns day of year of a datetime object.

Parameters **dt** (*datetime*) –

Returns **doy** – Day of year

Return type int

listdata

`heliopy.data.helper.listdata(probes=None)`

Print amount of data stored locally in the heliopy data directory.

Prints a table to the terminal with a column for raw data and a column for converted hdf data files.

Example output

```
Scanning files in /Users/dstansby/Data/
----------------------------------------
|      Probe |        Raw |        HDF |
|--------------------------------------|
|        ace |    1.44 MB |  800.00  B |
|    cluster |  200.39 MB |    0.00  B |
|     helios |    2.37 GB |    1.41 GB |
|        imp |   19.76 MB |   28.56 MB |
|  messenger |   15.24 MB |   27.21 MB |
|        mms |   70.11 MB |    0.00  B |
|     themis |   64.31 MB |    0.00  B |
|    ulysses |   54.78 MB |   47.98 MB |
|       wind |  176.84 MB |   63.82 MB |
|--------------------------------------|
|--------------------------------------|
|      Total |    2.96 GB |    1.57 GB |
----------------------------------------
```

Parameters **probes** (*List of strings, optional*) – Probe names

load

`heliopy.data.helper.load(filename, local_dir, remote_url, guessversion=False, try_download=True)`

Try to load a file from *local_dir*.

If file doesn't exist locally, try to download from *remtote_url* instead.

Parameters

- **filename** (*string*) – Name of file
- **local_dir** (*string*) – Local location of file
- **remote_url** (*string*) – Remote location of file
- **guessversion** (*bool*) – If *True*, try to guess the version number in the filename. Only works for cdf files. Default is *False*.
- **try_download** (*bool*) – If a file isn't available locally, try to downloaded it. Default is *True*.

Returns

file – If *filename* ends in *.cdf* the CDF file will be opened and returned.

Otherwise it is assumed that the file is an ascii file, and *filename* will be opened using python's `open()` method.

Return type CDF or open file

pitchdist_cdf2df

`heliopy.data.helper.`**`pitchdist_cdf2df`**(*cdf*, *distkeys*, *energykey*, *timekey*, *anglelabels*)

Converts cdf file of a pitch angle distribution to a pandas dataframe.

MultiIndexing is used as a pitch angle distribution is essentially a 3D dataset *f(time, energy, angle)*. See http://pandas.pydata.org/pandas-docs/stable/advanced.html#multiindex-advanced-indexing for more information.

This has been constructed for importing wind swe pitch angle distributions, and might not generalise very well to other data sets.

Assumes that each energy in the cdf has its own 2D array (time, angle). In the below description of the function there are

- `n` time data points
- `m` energy data points
- `l` anglular data points

Parameters

- **`cdf`** (*`cdf`*) – Opened cdf file.
- **`distkeys`** (*`list`*) – A list of the cdf keys for a given energies. Each array accessed by distkeys is shape *(n, l)*, and there must be *m* distkeys.
- **`energykey`** (*`string`*) – The cdf key for the energy values. The array accessed by energykey must have shape *(m)* or *(a, m)* where *a* can be anything. If it has shape *(a, m)*, we assume energies measured don't change, and take the first row as the energies for all times.
- **`timekey`** (*`string`*) – The cdf key for the timestamps. The array access by timekey must have shape *(n)*
- **`anglelabels`** (*`list`*) – A list of the labels to give each anglular bin (eg. [0, 10, 20] in degrees). Must be of length *l*.

Returns **df** – Data frame with read in data.

Return type `pandas.DataFrame`

timefilter

`heliopy.data.helper.`**`timefilter`**(*data*, *starttime*, *endtime*)

Puts data in a single dataframe, and filters it between times.

Parameters

- **`data`** (`pandas.DataFrame` or list) – Input data. If a list, `pd.concat(data)` will be run to put it in a DataFrame.
- **`starttime`** (*`datetime`*) – Start of interval.
- **`endtime`** (*`datetime`*) – End of interval.

Returns **out** – Filtered data.

Return type `pandas.DataFrame`

2.2 SPICE (heliopy.spice)

A module for loading SPICE kernels.

SPICE is a NASA toolkit for calculating the position of bodies (including planets and spacecraft) within the solar system. This module builds on the spiceypy package to provide a high level interface to the SPICE toolkit for performing orbital calculations using spice kernels.

2.2.1 heliopy.spice Module

Functions

furnish(fname)	Furnish SPICE with a kernel.

furnish

heliopy.spice.**furnish**(*fname*)

Furnish SPICE with a kernel.

Parameters **fname** (*Filename of a spice kernel to load.*) –

See also:

heliopy.data.spice.get_kernel() For attempting to automatically download kernels based on spacecraft name.

Classes

Trajectory(target)	A generic class for the trajectory of a single body.

Trajectory

class heliopy.spice.**Trajectory**(*target*)

Bases: object

A generic class for the trajectory of a single body.

Objects are initially created using only the body. To perform the actual trajectory calculation run *generate_positions()*. The generated positions are then available via. the attributes *times*, *x*, *y*, and *z*.

Parameters **spacecraft** (*str*) – Name of the target. The name must be present in the loaded kernels.

Notes

When an instance of this class is created, a leapseconds kernel and a planets kernel are both automatically loaded.

See also:

furnish for loading in local spice kernels.

Attributes Summary

generated	`True` if positions have been generated, `False` otherwise.
observing_body	Observing body.
r	Magnitude of position vectors.
target	The body whose coordinates are being calculated.
times	The *list* of *datetime* at which positions were last sampled.
x	x coordinates of position.
y	y coordinates of position.
z	z coordinates of position.

Methods Summary

change_units(unit)	Convert the positions to different units.
generate_positions(times, observing_body, frame)	Generate positions from a spice kernel.

Attributes Documentation

generated
`True` if positions have been generated, `False` otherwise.

observing_body
Observing body. The position vectors are all specified relative to this body.

r
Magnitude of position vectors.

target
The body whose coordinates are being calculated.

times
The *list* of *datetime* at which positions were last sampled.

x
x coordinates of position.

y
y coordinates of position.

z
z coordinates of position.

Methods Documentation

change_units(*unit*)
Convert the positions to different units.

Parameters **unit** (*astropy.units.Quantity*) – Must be a unit of length (e.g. km, m, AU).

generate_positions (*times*, *observing_body*, *frame*)

Generate positions from a spice kernel.

Parameters

- **times** (iterable of *datetime*) – An iterable (e.g. a *list*) of *datetime* objects at which the positions are calculated.
- **observing_body** (*str*) – The observing body. Output position vectors are given relative to the position of this body. See https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/naif_ids.html for a list of bodies.
- **frame** (*str*) – The coordinate system to return the positions in. See https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/frames.html for a list of frames.

2.3 Reading cdf files (pycdf)

pycdf is a module for reading CDF files into python. It is forked directly without change from the spacepy project. However, this is a minimal-dependency version which should be much easier to install compared to the full spacepy distribution.

To use pycdf after installing HelioPy do

```
from pycdf import pycdf
```

All the documentation for pycdf can be found at https://pythonhosted.org/SpacePy/pycdf.html

spacepy (and therefore the pycdf code) is licensed under the Python Software Foundation license.

CHAPTER 3

Development

3.1 Development guide

3.1.1 Testing

Running local tests

To run tests locally, install pytest (https://docs.pytest.org/en/latest/). Then from the `heliopy` directory run:

```
pytest
```

Continuous integration tests

To continuously check the codebase is working properly, tests are automatically run every time a pull request is submitted or a pull request is merged.

Currently tests are run using these services:

- Linux: https://travis-ci.org/heliopython/heliopy
- Windows: https://ci.appveyor.com/
- Documentation: https://circleci.com/gh/heliopython/heliopy

3.1.2 Creating a release

1. Fetch the latest copy of the upstream repository

```
git fetch upstream
git merge upstream/master
```

2. Run `git clean -xfd` to clean out any old builds

3. Tag the current version using

```
git tag version-number
git push
git push --tags
```

4. Create a source distribution

```
python setup.py sdist
```

5. Create a python wheel

```
python setup.py bdist_wheel
```

6. Upload created wheels to pypi

```
twine upload dist/*
```

See https://packaging.python.org/tutorials/distributing-packages/#packaging-your-project for more information.

Python Module Index

h

Index

P

R

S

T

X

Y

Z