HelioPy Documentation

Release 0.6.0

David Stansby

Aug 26, 2018

Contents

1 Getting started **3**
- 1.1 Examples . . . 3
- 1.2 HelioPy guide . . . 14

2 Requesting new features **21**

3 Module documentation **23**
- 3.1 Data import (`heliopy.data`) . . . 23
- 3.2 Coordinates (`heliopy.coordinates`) . . . 50
- 3.3 SPICE (`heliopy.spice`) . . . 53
- 3.4 Reading cdf files . . . 55

4 Development **57**
- 4.1 Development guide . . . 57

Python Module Index **59**

HelioPy is a free and open source set of tools for heliopsheric and planetary physics. For more information see the module documentation below.

CHAPTER 1

Getting started

1.1 Examples

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

1.1.1 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* for more information).

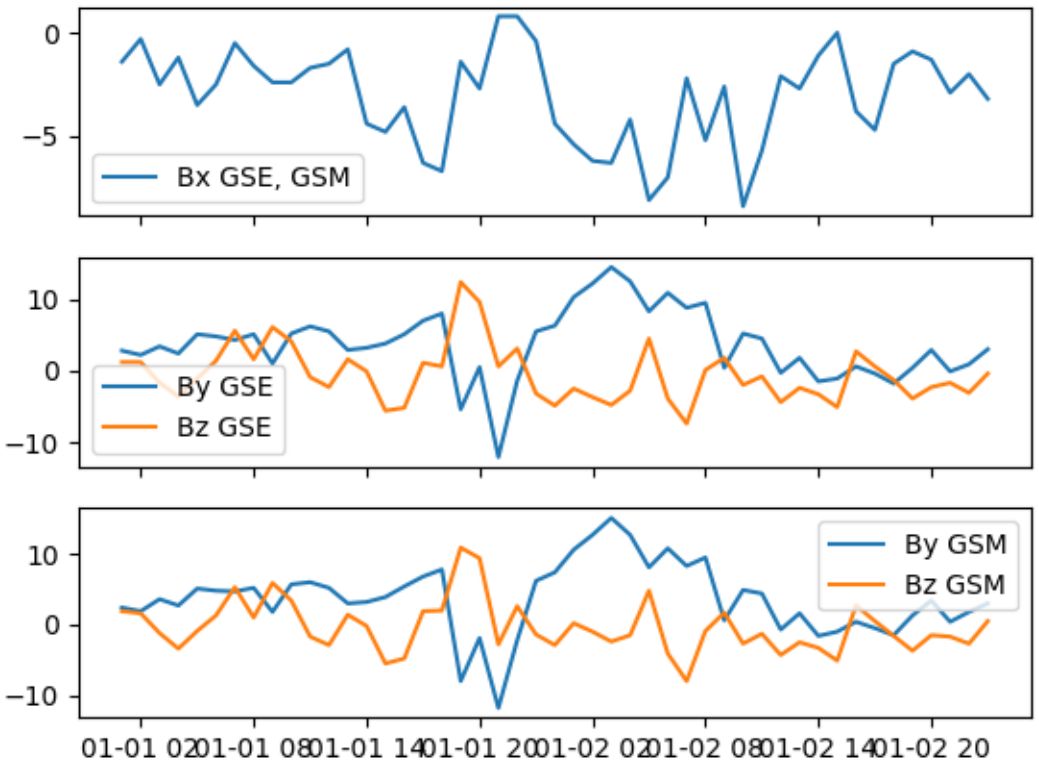
To check how much data is stored in both it's original format and hdf format see *Local data inventory*.

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

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

1.1.2 Plotting OMNI Data

Importing and plotting data from OMNI Web Interface. OMNI provides interspersed data from various spacecrafts. There are 55 variables provided in the OMNI Data Import.



Out:

```
Creating new directory /home/docs/heliopy/data/omni/low
Downloading https://cdaweb.gsfc.nasa.gov/pub/data/omni//low_res_omni/omni2_1970.dat
```

```
import heliopy.data.omni as omni
import matplotlib.pyplot as plt
from datetime import datetime

starttime = datetime(1970, 1, 1, 0, 0, 0)
endtime = datetime(1970, 1, 3, 0, 0, 0)

omni_data = omni.low(starttime, endtime)

fig, axs = plt.subplots(3, 1, sharex=True)
axs[0].plot(omni_data.data['Bx GSE, GSM'])
axs[1].plot(omni_data.data['By GSE'])
axs[1].plot(omni_data.data['Bz GSE'])
axs[2].plot(omni_data.data['By GSM'])
axs[2].plot(omni_data.data['Bz GSM'])
```

(continues on next page)

(continued from previous page)

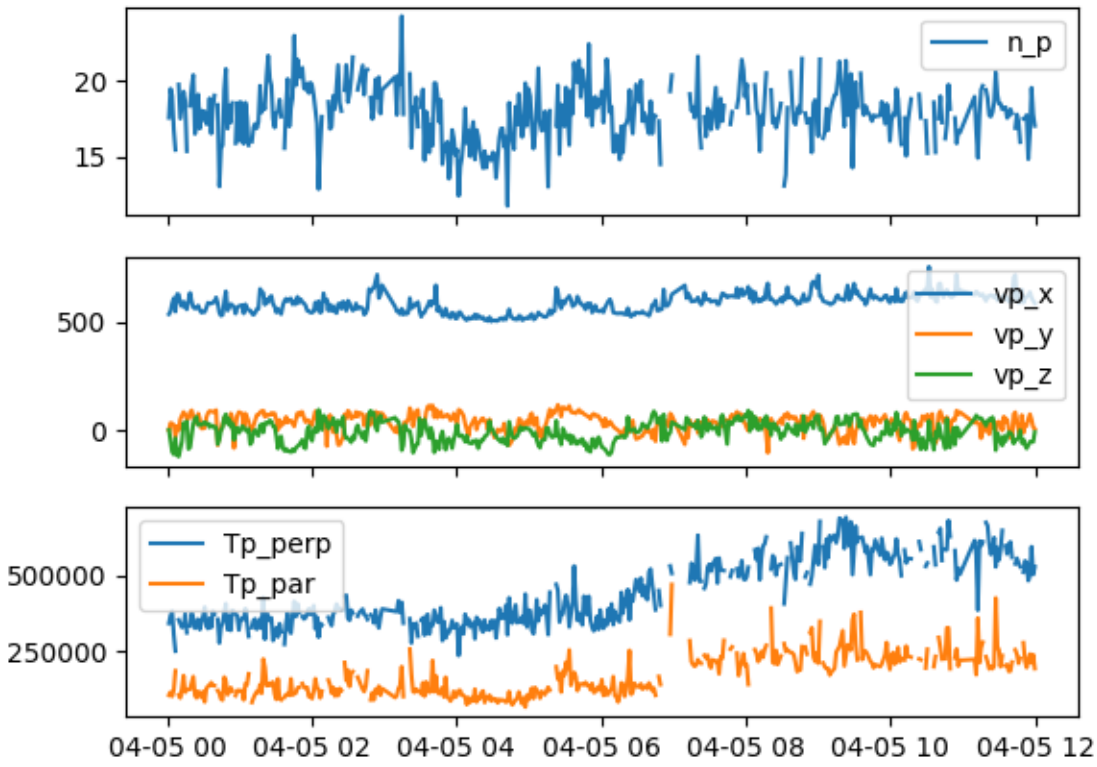
```
for ax in axs:
    ax.legend()
plt.show()
```

Total running time of the script: (0 minutes 2.061 seconds)

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

1.1.3 Importing data

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



Out:

```
Creating new directory /home/docs/heliopy/data/helios/E1_experiment/New_proton_
↪corefit_data_2017/ascii/helios2/1976
Downloading ftp://apollo.ssl.berkeley.edu/pub/helios-data/E1_experiment/New_proton_
↪corefit_data_2017/ascii/helios2/1976/h2_1976_096_corefit.csv
```

(continues on next page)

(continued from previous page)

```
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'

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

print(corefit.data.keys())

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

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

Total running time of the script: (0 minutes 1.502 seconds)

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

1.1.4 TimeSeries Plotting Example

An example to show Sunpy's TimeSeries in HelioPy.

For more information on TimeSeries, see http://docs.sunpy.org/en/stable/guide/data_types/timeseries.html

For more information on AstroPy Units, see http://docs.astropy.org/en/stable/units/

Import modules

```
import heliopy.data.ulysses as ulysses
import matplotlib.pyplot as plt
from datetime import datetime
```

Set up support for plotting data with units

```
from astropy.visualization import quantity_support
quantity_support()
```

Load data. In this example we use Ulysses data.

```
starttime = datetime(1993, 1, 1, 0, 0, 0)
endtime = datetime(1993, 2, 1, 0, 0, 0)
timeseries_data = ulysses.swics_abundances(starttime, endtime)
```

Out:

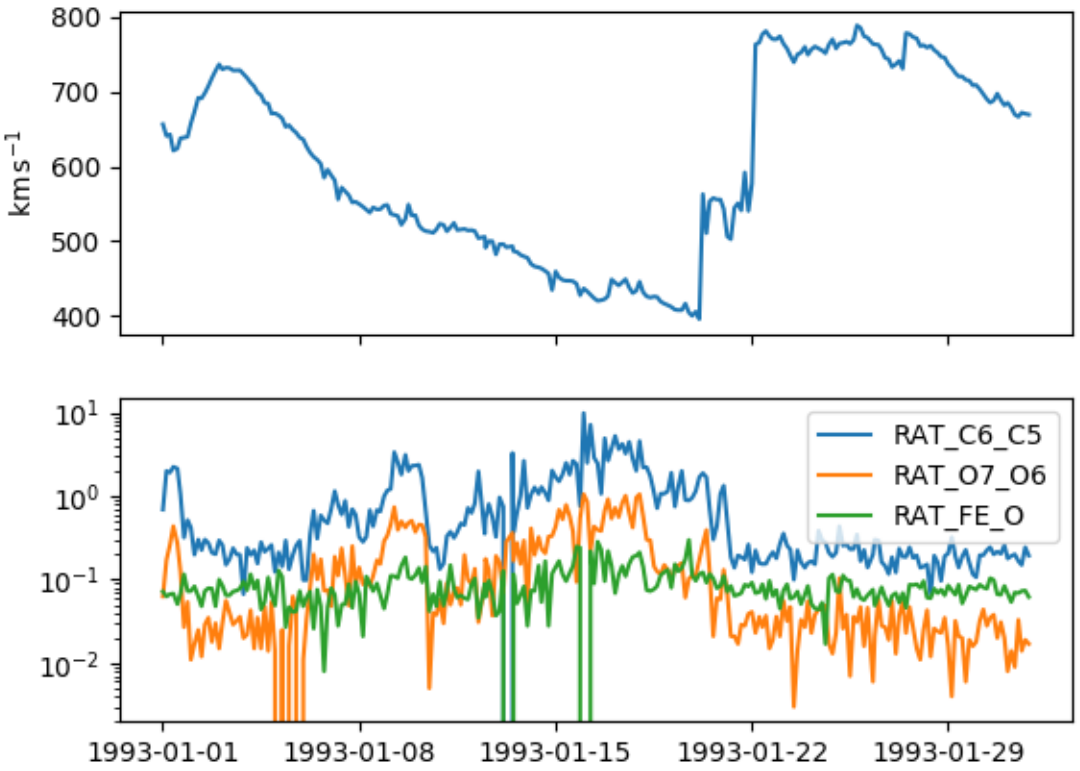
```
Creating new directory /home/docs/heliopy/data/ulysses/swics
Downloading http://ufa.esac.esa.int/ufa-sl-server/data-action?PROTOCOL=HTTP&PRODUCT_
↪TYPE=ALL&FILE_NAME=uswichst93.dat&FILE_PATH=/ufa/HiRes/data/swics&/uswichst93.dat
```

timeseries_data is a TimeSeries data type. The .index attribute gets the time index of the data the .quantity() method can be used to extract data with units attached

```
print(timeseries_data.data.keys())
fig, axs = plt.subplots(2, 1, sharex=True)
axs[0].plot(timeseries_data.index, timeseries_data.quantity('VEL_ALPHA'))

ion_ratios = ['RAT_C6_C5', 'RAT_O7_O6', 'RAT_FE_O']
for r in ion_ratios:
    axs[1].plot(timeseries_data.index, timeseries_data.quantity(r), label=r)

axs[1].set_yscale('log')
axs[1].legend()
plt.show()
```



Out:

```
Index(['VEL_ALPHA', 'RAT_C6_C5', 'RAT_O7_O6', 'RAT_FE_O', 'CHARGE_FE',
       'N_CYC'],
      dtype='object')
```

Total running time of the script: (0 minutes 1.815 seconds)

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

1.1.5 Importing sunspot data

Importing and plotting sunspot number data.

```
import heliopy.data.sunspot as sunspot
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()

# Plotting Daily
data_daily = sunspot.daily()
print(data_daily.keys())
```

(continues on next page)

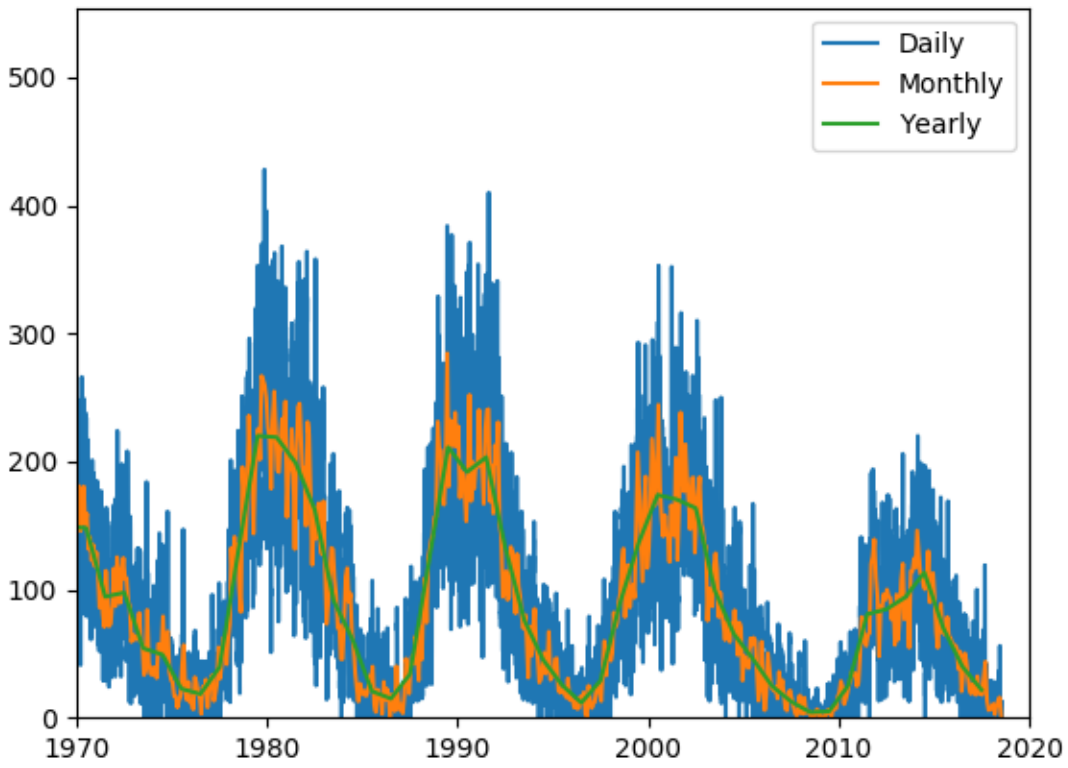
(continued from previous page)

```
x_daily = data_daily['DecD']
y_daily = data_daily['Daily']
ax.plot(x_daily, y_daily, label='Daily')

# Plotting Monthly
data_monthly = sunspot.monthly()
print(data_monthly.keys())
x_monthly = data_monthly['DecD']
y_monthly = data_monthly['Monthly']
ax.plot(x_monthly, y_monthly, label='Monthly')

# Plotting Yearly
data_yearly = sunspot.yearly()
print(data_yearly.keys())
x_yearly = data_yearly['Y']
y_yearly = data_yearly['Y_Mean']
ax.plot(x_yearly, y_yearly, label='Yearly')

# Set some sensible plotting limits
ax.set_xlim(left=1970, right=2020)
ax.set_ylim(bottom=0)
ax.legend()
plt.show()
```



Out:

```
Index(['Y', 'M', 'D', 'DecD', 'Daily', 'Std Dev', 'No Obs', 'Def/Prov Ind'], dtype=
↪'object')
Index(['Y', 'M', 'DecD', 'Monthly', 'Std Dev ', 'No Obs', 'Def/Prov Ind'], dtype=
↪'object')
Index(['Y', 'Y_Mean', 'Std Dev', 'No Obs', 'Def/Prov Ind'], dtype='object')
```

Total running time of the script: (0 minutes 1.698 seconds)

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

1.1.6 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 |
  ---------------------------------------
'''
```

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

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

1.1.7 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('solo_2020')
spice.furnish(orbiter_kernel)
orbiter = spice.Trajectory('Solar Orbiter')
```

Out:

```
Downloading https://issues.cosmos.esa.int/solarorbiterwiki/download/attachments/
↪7274724/solo_ANC_soc-orbit_20200207-20300902_V01.bsp
Downloading https://naif.jpl.nasa.gov/pub/naif/generic_kernels/lsk/naif0012.tls
Downloading https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/de430.bsp
```

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)
```

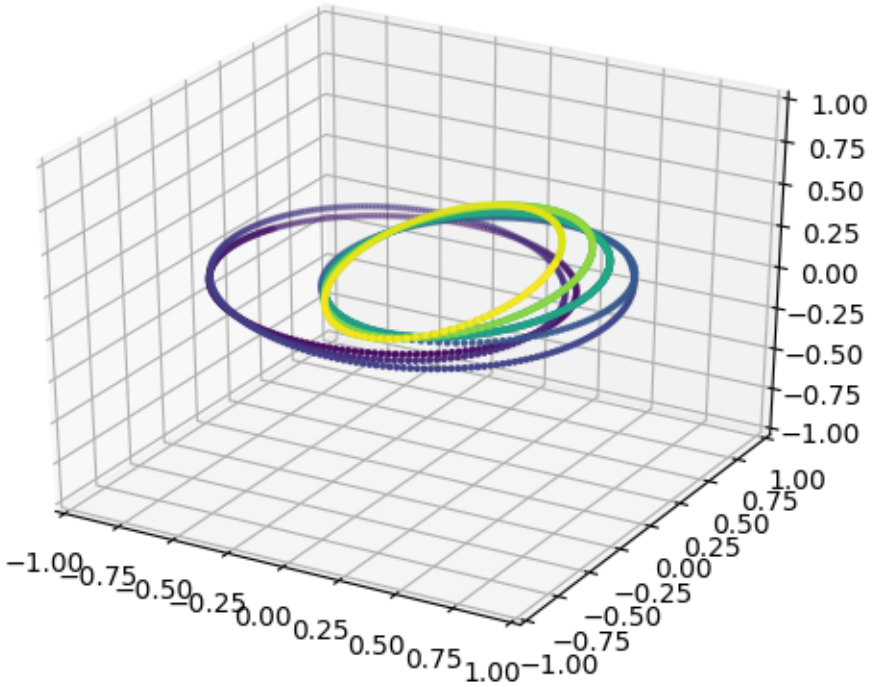
Generate positions

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



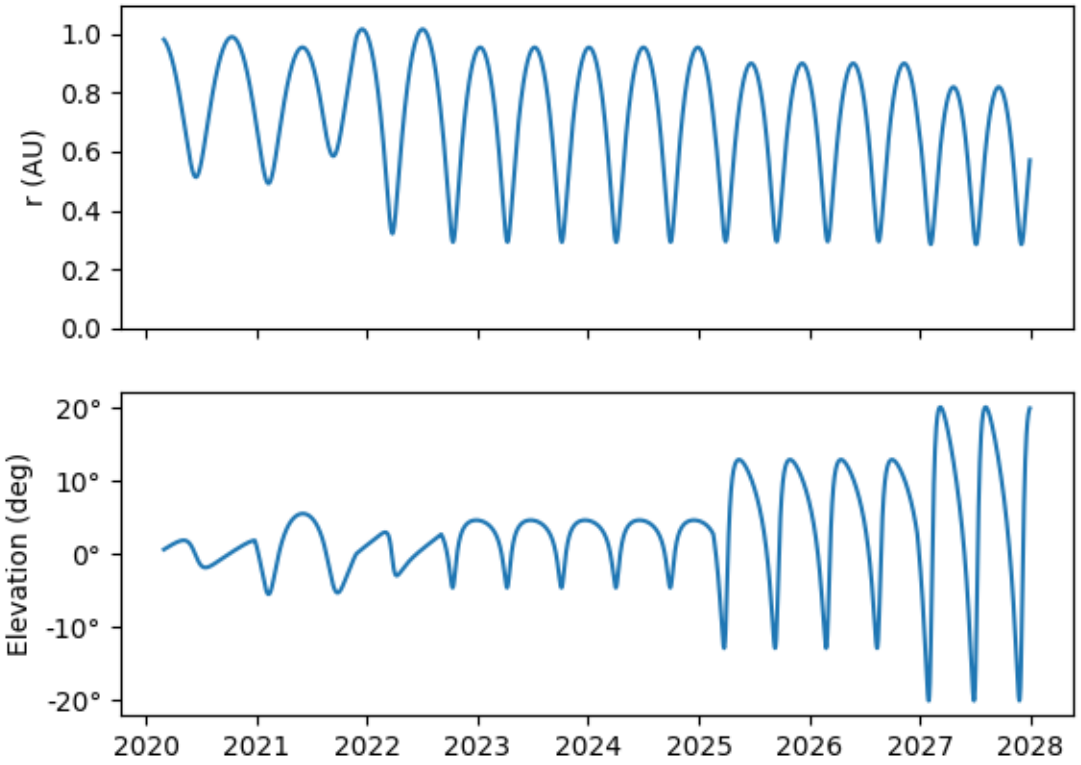
Plot radial distance and elevation as a function of time

```
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 5.914 seconds)

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

1.1.8 TimeSeries Basics

An example to show some basic functions of TimeSeries.

For more information about TimeSeries, http://docs.sunpy.org/en/stable/guide/data_types/timeseries.html For more information about AstroPy Units, http://docs.astropy.org/en/stable/units/

```
import numpy as np
import datetime
import pandas as pd
import sunpy.timeseries as ts
from collections import OrderedDict
import astropy.units as u

# The index of the SunPy Timeseries is always datetime
base = datetime.datetime.today()
times = [base - datetime.timedelta(minutes=x) for x in range(24*60, 0, -1)]
intensity = np.sin(np.arange(0, 12 * np.pi, ((12 * np.pi) / (24*60))))
```

(continues on next page)

(continued from previous page)

```
# This example shows how a TimeSeries object is made from a Pandas DataFrame
data = pd.DataFrame(intensity, index=times, columns=['intensity'])

# TimeSeries can have a metadata attached to it.
meta = OrderedDict({'key':'value'})

# AstroPy Units are attached to the TimeSeries by passing it alongside the data.
# The units are stored in an OrderedDict object.
# Each key is the unit, and the value is the astropy representation of the same.
units = OrderedDict([('intensity', u.W/u.m**2)])
ts_custom = ts.TimeSeries(data, meta, units)

# Using sunpy.timeseries.TimeSeries.data will return a Pandas DataFrame of the␣
↪TimeSeries object.
print(ts_custom.data)

# To view the units, sunpy.timeserise.TimeSeries.units can be used.
print(ts_custom.units)

# The values can be extracted along with their units as well.
#sunpy.timeseries.TimeSeries.quantity(column_name)[index]
print(ts_custom.quantity('intensity')[1])
```

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

1.2 HelioPy guide

1.2.1 Installing

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.

The minimum supported version of python is python 3.6.

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

```
conda install -c conda-forge heliopy
```

or pip:

```
pip install heliopy
```

Module requirements

Each module has a set of dependencies that are required for that module to be used. To automatically install dependencies for a specific module, use *pip install heliopy[modname]*, e.g. for the 'coordinates' module:

```
pip install heliopy[coordinates]
```

Alternatively, to install all of the optional dependencies use:

```
pip install heliopy[all]
```

In addition, there are the following optional requirements that add extra functionality to HelioPy.

HDF file reader/writer

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

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.

1.2.2 Configuring

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.

1.2.3 Citing

If you use HelioPy for research presented in a publication, please cite HelioPy using the reference provided at the bottom right of this page. Please use the following ready made sentence:

> "This research made use of HelioPy, a community-developed Python package for space physics [HelioPy reference]"

1.2.4 What's new

- *Version 0.6.0*
- *Version 0.5.3*
- *Version 0.5.2*
- *Version 0.5.1*
- *Version 0.5.0*
- *Version 0.4*
- *Version 0.3*
- *Version 0.2*
- *Version 0.1.3*

Version 0.6.0

HelioPy now only supports Python versions 3.6 and higher.

New features

- HelioPy has been integrated with SunPy TimeSeries and AstroPy Units. All of the HelioPy modules now return physical units with data.
- Added a new `data.util.cdf_units()` function that can extract the UNIT attribute from CDF files.
- Low resolution OMNI data import has been added in `data.omni.low()` function.
- Magnetic Field data from DSCOVR Spacecraft can now be imported using the `data.dscovr.mag_h0()` function.

Backwards incompatible changes

- Methods in `heliopy.data` no longer returns a Pandas DataFrame, but now return a SunPy timeseries object. To get the underlying data, you can still do:

```
dataframe = timeseries.data
```

  For an example of how to use the new object, see *TimeSeries Plotting Example*.
- Data import has had a major overhaul, so that every column in CDF files now gets automatically imported and retains its name without being changed by HelioPy. This means column names in several data products are now different, to reflect their original name in the CDF files instead of a custom name that was previously assigned by HelioPy.
- `data.helios.merged()`, `data.helios.mag_4hz()`, `data.helios.corefit()` and `data.helios.mag_ness()` no longer take a *verbose* keyword argument. #467

Fixed bugs

- *data.imp.merged()* no longer imports redundant columns.

Version 0.5.3

New features

- Lots of small documentation updates.
- *.data.helios.distparams* now has an extra `'data_rate'` column, which determines whether a given distribution function was transmitted in high or low data mode. #529

Version 0.5.2

New features

- The new HelioPy logo has been added to the documentation. #448, #447

Fixed bugs

- The new data version number of `data.mms.fpi_dis_moms()` has been updated.

Version 0.5.1

New features

- HelioPy can now be installed using conda.

Backwards incompatible changes

- The list of kernels available for automatic download in *data.spice* has been updated, and some names changed. #408

Fixed bugs

- *spice.Trajectory.generate_positions()* can now generate positions at a resolution of one second instead of one day. #405
- A duplicate "z gsm" column header in the data returned by `data.imp.mag15s()` has been corrected. #396

Version 0.5.0

New features

- *heliopy.data.sunspot()* added an additional functionality to import sunspot data in three different timeframes - daily, monthly and yearly.

- The inventory of spice kernels in *heliopy.data.spice* now includes "Helios 1 Reconstructed", "Helios 1 Predicted", "Juno Reconstructed", "Juno Predicted" and "Helios 2" kernels.
- *heliopy.spice.furnish()* now accepts a list of filenames as well as individual filenames.
- A lot of new functions for downloading ACE data have been added to *heliopy.data.ace*.

Backwards incompatible changes

- *heliopy.data.spice.get_kernel()* now returns a list of filenames instead of a single filename string.
- Most of the functions that were in *heliopy.data.helper* have been moved to *heliopy.data.util*. The ones the remain in *heliopy.data.helper* are useful for users, and the ones in *heliopy.data.util* are used internally as utility functions for data import.

Removed features

- `heliopy.data.helios.trajectory()` has been removed. To get Helios trajectory data use the *heliopy.spice* and *heliopy.data.spice* modules.

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.

1.2.5 HelioPy community Code of Conduct

The community of participants in open source projects, including HelioPy, is made up of members from around the globe with a diverse set of skills, personalities, and experiences. It is through these differences that our community experiences success and continued growth. We expect everyone in our community to follow these guidelines when interacting with others both inside and outside of our community. Our goal is to keep ours a positive, inclusive, successful, and growing community.

As members of the community:

- We pledge to treat all people with respect and provide a harassment- and bullying-free environment, regardless of sex, sexual orientation and/or gender identity, disability, physical appearance, body size, race, nationality, ethnicity, and religion. In particular, sexual language and imagery, sexist, racist, or otherwise exclusionary jokes are not appropriate.
- We pledge to respect the work of others by recognising acknowledgment/citation requests of original authors. As authors, we pledge to be explicit about how we want our own work to be cited or acknowledged.
- We pledge to welcome those interested in joining the community, and realise that including people with a variety of opinions and backgrounds will only serve to enrich our community. In particular, discussions relating to pros/cons of various technologies, programming languages, and so on are welcome, but these should be done with respect, taking proactive measure to ensure that all participants are heard and feel confident that they can freely express their opinions.
- We pledge to welcome questions and answer them respectfully, paying particular attention to those new to the community. We pledge to provide respectful criticisms and feedback in forums, especially in discussion threads resulting from code contributions.
- We pledge to be conscientious of the perceptions of the wider community and to respond to criticism respectfully. We will strive to mode behaviours that encourage productive debate and disagreement, both within our community and where we are criticised. We will treat those outside our community with the same respect as people within our community.
- We pledge to help the entire community follow the code of conduct, and to not remain silent when we see violations of the code of conduct.

This code of conduct applies to all community situations online and offline, including mailing lists, forums, social media, conferences, meetings, associated social events, and one-to-one interactions.

This code of conduct has been adapted from the astropy code of conduct, which in turn was partly adapted from the PSF code of conduct.

CHAPTER 2

Requesting new features

If you would like a new feature to be built into HelioPy, you can either open a bug report in the github issue tracker, or send an email to heliopy@googlegroups.com.

CHAPTER 3

Module documentation

3.1 Data import (`heliopy.data`)

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

3.1.1 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.
swi_h2(starttime, endtime)	Import hourly SWICS data.
swi_h3(starttime, endtime)	Import hourly SWICS composition data.
swi_h6(starttime, endtime)	Import 12 minute SWICS proton data.

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 TimeSeries

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 TimeSeries

swi_h2

heliopy.data.ace.**swi_h2**(*starttime*, *endtime*)

Import hourly SWICS data.

See https://cdaweb.sci.gsfc.nasa.gov/misc/NotesA.html#AC_H2_SWI for more information.

Parameters

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

Returns **data**

Return type TimeSeries

swi_h3

heliopy.data.ace.**swi_h3**(*starttime*, *endtime*)

Import hourly SWICS composition data.

See https://cdaweb.sci.gsfc.nasa.gov/misc/NotesA.html#AC_H3_SWI for more information.

Parameters

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

Returns **data**

Return type TimeSeries

swi_h6

heliopy.data.ace.**swi_h6**(*starttime*, *endtime*)
Import 12 minute SWICS proton data.

See https://cdaweb.sci.gsfc.nasa.gov/misc/NotesA.html#AC_H6_SWI for more information.

Parameters

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

Returns data

Return type TimeSeries

3.1.2 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 TimeSeries

3.1.3 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[, try_download])	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 http://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 `TimeSeries`

mag_hires

heliopy.data.cassini.**mag_hires**(*starttime*, *endtime*, *try_download=True*)

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

3.1.4 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[, try_download])	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'*, *try_download=True*)

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*, *try_download=True*)

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*, *try_download=True*)
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*, *try_download=True*)
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

3.1.5 DSCOVR

heliopy.data.dscovr Module

Methods for importing data from the DSCOVR.

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

Functions

mag_h0(starttime, endtime)	Imports magnetic field data from DSCOVR Spacecraft.

mag_h0

heliopy.data.dscovr.**mag_h0**(*starttime*, *endtime*)

Imports magnetic field data from DSCOVR Spacecraft. :param starttime: Interval start time. :type starttime: datetime :param endtime: Interval end time. :type endtime: datetime

Returns **data**

Return type TimeSeries

3.1.6 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/

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[, try_download])	Read in merged data set.

corefit

heliopy.data.helios.**corefit**(*probe*, *starttime*, *endtime*, *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
- **try_download** (*bool, optional*) – If `False` don't try to download data if it is missing locally.

Returns **data** – Data set

Return type `TimeSeries`

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*, *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
- **try_download** (*bool, optional*) – If `False` don't try to download data if it is missing locally.

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

Return type `TimeSeries`

mag_ness

heliopy.data.helios.**mag_ness**(*probe*, *starttime*, *endtime*, *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
- **try_download** (*bool, optional*) – If `False` don't try to download data if it is missing locally.

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

Return type DataFrame

merged

heliopy.data.helios.**merged**(*probe*, *starttime*, *endtime*, *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
- **try_download** (*bool, optional*) – If `False` don't try to download data if it is missing locally.

Returns **data** – Merged data set

Return type `TimeSeries`

Notes

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

3.1.7 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[, try_download])	Import merged plasma data.
mitplasma_h0(probe, starttime, endtime[, ...])	Import mit h0 plasma data.

mag15s

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

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*, *try_download=True*)

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*, *try_download=True*)

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 `TimeSeries`

mitplasma_h0

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

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

3.1.8 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[, try_download])	Import magnetic field in RTN coordinates from Messenger.

mag_rtn

heliopy.data.messenger.**mag_rtn**(*starttime*, *endtime*, *try_download=True*)

Import magnetic field in RTN coordinates from Messenger.

Parameters

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

Returns **data**

Return type DataFrame

3.1.9 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

3.1.10 OMNI

heliopy.data.omni Module

Methods for importing data from the OMNI.

All data is publically available at https://cdaweb.gsfc.nasa.gov/pub/data/omni.

Functions

low(starttime, endtime[, try_download])	Import data from OMNI Web Interface.

low

heliopy.data.omni.**low**(*starttime*, *endtime*, *try_download=True*)
Import data from OMNI Web Interface.

Parameters

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

Returns data

Return type TimeSeries

3.1.11 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[, try_download])	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[, try_download])	Import SWOOPS ion data.

fgm_hires

heliopy.data.ulysses.**fgm_hires**(*starttime*, *endtime*, *try_download=True*)

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*, *try_download=True*)

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 TimeSeries

swics_heavy_ions

heliopy.data.ulysses.**swics_heavy_ions**(*starttime*, *endtime*, *try_download=True*)

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 TimeSeries

swoops_ions

heliopy.data.ulysses.**swoops_ions**(*starttime*, *endtime*, *try_download=True*)

Import SWOOPS ion data.

Parameters

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

Returns **data** – Requested data

Return type TimeSeries

3.1.12 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. :param starttime: Interval start time. :type starttime: datetime :param endtime: Interval end time. :type endtime: datetime

Returns **data**

Return type TimeSeries

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) :param starttime: Interval start time. :type starttime: datetime :param endtime: Interval end time. :type endtime: datetime

Returns data

Return type TimeSeries

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. :param starttime: Interval start time. :type starttime: datetime :param endtime: Interval end time. :type endtime: datetime

Returns data

Return type TimeSeries

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 :param starttime: Interval start time. :type starttime: datetime :param endtime: Interval end time. :type endtime: datetime

Returns data

Return type TimeSeries

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 TimeSeries

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 :param starttime: Interval start time. :type starttime: datetime :param endtime: Interval end time. :type endtime: datetime

Returns data

Return type TimeSeries

Each mission does not have a complete set of data import methods, but the goal of HelioPy is to be as complete as possible. If you want to import a data set that is not yet supported please open an issue on the bug tracker at https://github.com/heliopython/heliopy/issues

There are also modules for downloading SPICE kernels and sunspot number data:

3.1.13 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.

Table 13: Generic kernels

Name	Identifier	Kernel URL(s)
Leap Second Kernel	lsk	[1]
Planet trajectories	planet_trajectories	[1]
Planet orientations	planet_orientations	[1]

Table 14: Spacecraft kernels

Name	Identifier	Kernel URL(s)
Clementine-Goddard Space Flight Center	clem_gsfc	[1]
Clementine-Jet Propulsion Laboratory	clem_jpl	[1]
Clementine-Naval Research Laboratory	clem_nrl	[1]
Deep Impact-Adjusted	deepimpact_adjusted	[1]
Deep Impact-Complete	deepimpact_complete	[1] [2] [3] [4]
Deep Impact-Original	deepimpact_original	[1]
Deep Space	deepspace	[1]
EPOXI-DIXI	epoxi_dixi	[1]
EPOXI-ePOCh	epoxi_epoch	[1]
Hayabusa-Complete	hayabusa_complete	[1] [2] [3] [4]
Helios 1 Predicted	helios1_pred	[1]
Helios 1 Reconstructed	helios1_rec	[1]
Helios 2	helios2	[1]
Juno Predicted	juno_pred	[1]
Juno Reconstructed	juno_rec	[1]
Solar Orbiter 2020	solo_2020	[1]

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

Get the local location of a kernel.

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

Parameters **name** (*str*) – Kernel name. See *Generic kernels* and *Spacecraft kernels* for lists of available names. The name should be a string from the "Identifier" column of one of the tables.

Returns List of the locations of kernels that have been downloaded.

Return type list

3.1.14 Sunspot

heliopy.data.sunspot Module

Sunspot

Methods for automatically downloading sunspot number data.

For more info about the sunspot number data, visit http://www.sidc.be/silso/datafiles.

Functions

daily()	Import daily sunspot number.
monthly()	Import monthly sunspot number.
yearly()	Import yearly sunspot number.

daily

heliopy.data.sunspot.**daily**()
Import daily sunspot number.

For more information, see http://www.sidc.be/silso/infosndtot.

monthly

heliopy.data.sunspot.**monthly**()
Import monthly sunspot number.

For more information, see http://www.sidc.be/silso/infosnmtot.

yearly

heliopy.data.sunspot.**yearly**()
Import yearly sunspot number.

For more information, see http://www.sidc.be/silso/infosnytot.

a module with helper functions that apply generally to all data is available:

3.1.15 Helper methods

heliopy.data.helper Module

Helper methods for data import.

Functions

cdf_dict(unit_string)	Method to obtain the unit denoted by the strings inside the CDF files in the UNIT attribute.
cdfpeek(cdf_loc)	List all the variables present in a CDF file, along with their size.
listdata([probes])	Print amount of data stored locally in the heliopy data directory.

cdf_dict

heliopy.data.helper.**cdf_dict**(*unit_string*)

Method to obtain the unit denoted by the strings inside the CDF files in the UNIT attribute.

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.

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

and utility functions that much of the data import uses are also available in the util module:

3.1.16 Utility methods

heliopy.data.util Module

Utility functions for data downloading.

Note: these methods are liable to change at any time.

Functions

cdf2df(cdf, index_key[, dtimeindex, ...])	Converts a cdf file to a pandas dataframe.
cdf_units(cdf_[, manual_units, length])	Takes the CDF File and the required keys, and finds the units of the selected keys.
doy2ymd(y, doy)	Converts day of year and year to year, month, day
dtime2doy(dt)	Returns day of year of a datetime object.
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.
process(dirs, fnames, extension, ...[, ...])	The main utility method for systematically loading, downloading, and saving data.
timefilter(data, starttime, endtime)	Puts data in a single dataframe, and filters it between times.
units_attach(data, units)	Takes the units defined by the user and attaches them to the TimeSeries.

cdf2df

heliopy.data.util.**cdf2df**(*cdf*, *index_key*, *dtimeindex=True*, *badvalues=None*, *ignore=None*)

Converts a cdf file to a pandas dataframe.

Note that this only works for 1 dimensional data, other data such as distribution functions or pitch angles will not work properly.

Parameters

- **cdf** (*cdf*) – Opened CDF file.
- **index_key** (*string*) – The CDF key to use as the index in the output DataFrame.
- **dtimeindex** (*bool, optional*) – If `True`, the DataFrame index is parsed as a datetime. Default is `True`.
- **badvalues** (*dict, list, optional*) – A dictionary that maps the new DataFrame column keys to a list of bad values to replace with nans. Alternatively a list of numbers which are replaced with nans in all columns.
- **ignore** (*list, optional*) – In case a CDF file has columns that are unused / not required, then the column names can be passed as a list into the function.

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

Return type `pandas.DataFrame`

cdf_units

heliopy.data.util.**cdf_units**(*cdf_*, *manual_units=None*, *length=None*)

Takes the CDF File and the required keys, and finds the units of the selected keys.

Parameters

- **cdf** (*cdf*) – Opened cdf file
- **manual_units** (*OrderedDict, optional*) – Manually defined units to be attached to the data that will be returned.

Returns **out** – Returns an OrderedDict with units of the selected keys.

Return type collections.OrderedDict

doy2ymd

heliopy.data.util.**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.util.**dtime2doy**(*dt*)

Returns day of year of a datetime object.

Parameters **dt** (*datetime*) –

Returns **doy** – Day of year

Return type int

load

heliopy.data.util.**load**(*filename*, *local_dir*, *remote_url*, *try_download=True*, *remote_error=False*)

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

- **try_download** (*bool*) – If a file isn't available locally, try to downloaded it. Default is *True*.
- **remote_error** (*bool*) – If `True`, raise an error if the requested file isn't present locally or remotely. If `False`, the method returns `None` if the file can't be found.

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.

If the file can't be found locally or remotely, and *remote_errror* is `False`, `None` is returned.

Return type CDF, open file, None

pitchdist_cdf2df

heliopy.data.util.**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`

process

heliopy.data.util.**process**(*dirs*, *fnames*, *extension*, *local_base_dir*, *remote_base_url*, *download_func*, *processing_func*, *starttime*, *endtime*, *try_download=True*, *units=None*, *processing_kwargs={}*, *download_info=[]*, *remote_fnames=None*)

The main utility method for systematically loading, downloading, and saving data.

Parameters

- **dirs** (*list*) – A list of directories relative to *local_base_dir*.
- **fnames** (*list or str or regex*) – A list of filenames **without** their extension. These are the filenames that will be downloaded from the remote source. Must be the same length as *dirs*. Each filename is saved in it's respective entry in *dirs*. Can also be a regular expression that is used to match the filename (e.g. for version numbers)
- **extension** (*str*) – File extension of the raw files. **Must include leading dot**.
- **local_base_dir** (*str*) – Local base directory. `fname[i]` will be stored in `local_base_dir / dirs[i] / fname[i] + extension`.
- **remote_base_url** (*str*) – Remote base URL. `fname[i]` will be downloaded from `Remote / dirs[i] / fname[i] + extension`.
- **download_func** – Function that takes
  - The remote base url
  - The local base directory
  - The relative directory (relative to the base url)
  - The local filename to download to
  - The remote filename
  - A file extension

  and downloads the remote file. The signature must be:

```
def download_func(remote_base_url, local_base_dir,
                  directory, fname, remote_fname, extension)
```

  The function can also return the filename of the file it downloaded, if this is different to the filename it is given. *download_func* should **not** raise any errors, and just silently do nothing if a given file is not available.
- **processing_func** – Function that takes an open CDF file or open plain text file, and returns a pandas DataFrame. The signature must be:

```
def processing_func(file, **processing_kwargs)
```

- **starttime** (*datetime*) – Start of requested interval.
- **endtime** (*datetime*) – End of requested interval.
- **try_download** (*bool, optional*) – If `True`, try to download data. If `False` don't. Default is `True`.
- **units** (*OrderedDict, optional*) – Manually defined units to be attached to the data that will be returned.

Must map column headers (strings) to `Quantity` objects. If units are present, then a TimeSeries object is returned, else a Pandas DataFrame.

- **processing_kwargs** (*dict, optional*) – Extra keyword arguments to be passed to the processing funciton.
- **download_info** (*list, optional*) – A list with the same length as *fnames*, which contains extra info that is handed to *download_func* for each file individually.
- **remote_fnames** (*list of str, optional*) – If the remote filenames are different from the desired downloaded filenames, this should be a list of length `len(fnames)` with the files to be downloaded. The ordering must be the same as *fnames*.

Returns Requested data.

Return type `DataFrame` or `TimeSeries`

timefilter

heliopy.data.util.**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`

units_attach

heliopy.data.util.**units_attach**(*data*, *units*)

Takes the units defined by the user and attaches them to the TimeSeries.

Parameters

- **data** (`pandas.DataFrame`) – Input data. Takes the DataFrame which needs to have units attached.
- **units** (`collections.OrderedDict`) – The units manually defined by the user.

Returns **out** – DataFrame converted into TimeSeries with units attached.

Return type `TimeSeries`

Classes

RemoteFileNotPresentError

RemoteFileNotPresentError

exception heliopy.data.util.**RemoteFileNotPresentError**

3.1.17 SunPy and AstroPy Integration

HelioPy is built to be used alongside SunPy and astropy. Before v0.6, HelioPy returned a pandas DataFrame object. After adding physical units to the data, HelioPy now returns a SunPy TimeSeries object. The TimeSeries object is capable of storing the data in a DataFrame, and also stores the units that are associated with each data column.

An example on how to use TimeSeries data and astropy units is also available in *TimeSeries Plotting Example*.

3.2 Coordinates (`heliopy.coordinates`)

3.2.1 heliopy.coordinates Package

Creating coordinate objects

Coordinate objects can be created using the coordinate frame classes in *`heliopy.coordinates.frames`*, for example, to create a coordinate in a GSE frame:

```
>>> from astropy.constants import au
>>> import heliopy.coordinates.frames as frames
>>> hee_coord = frames.HeliocentricEarthEcliptic(1 * au, 0 * au, 0 * au)
>>> hee_coord
<HeliocentricEarthEcliptic Coordinate (obstime=None): (x, y, z) in m
    (1.49597871e+11, 0., 0.)>
```

Transforming between coordinate systems

To transform between coordinate frames, the `transform_to()` method can be called on a coordinate object:

```
>>> from datetime import datetime
>>> from astropy.constants import au
>>> import heliopy.coordinates.frames as frames
>>>
>>> hee_coord = frames.HeliocentricEarthEcliptic(1 * au, 0 * au, 0 * au,
...     obstime=datetime(1992, 12, 21))
>>> gse_coord = hee_coord.transform_to(frames.GeocentricSolarEcliptic)
>>> gse_coord
<GeocentricSolarEcliptic Coordinate (obstime=None): (x, y, z) in m
    (-2.42947355e+09, 0., 0.)>
```

3.2.2 heliopy.coordinates.frames Module

This submodule contains various space physics coordinate frames for use with the `astropy.coordinates` module.

Warning: The functions in this submodule should never be called directly, transforming between coordinate frames should be done using `transform_to()` on coordinate frame objects. See above for an example.

Functions

hee_to_gse(hee_coord, gse_frame)	Convert from HEE to GSE coordinates.

hee_to_gse

heliopy.coordinates.frames.**hee_to_gse**(*hee_coord*, *gse_frame*)
Convert from HEE to GSE coordinates.

Classes

GeocentricSolarEcliptic(*args[, copy, ...])	A coordinate frame in the Geocentric Solar Ecliptic (GSE) system.
HeliocentricEarthEcliptic(*args[, copy, ...])	A coordinate frame in the Heliocentric Earth Ecliptic (HEE) system.

GeocentricSolarEcliptic

class heliopy.coordinates.frames.**GeocentricSolarEcliptic**(**args*, *copy=True*, *representation_type=None*, *differential_type=None*, ***kwargs*)

Bases: astropy.coordinates.baseframe.BaseCoordinateFrame

A coordinate frame in the Geocentric Solar Ecliptic (GSE) system.

Possible call signatures:

```
gse = GeocentricSolarEcliptic(x, y, z)
gse = GeocentricSolarEcliptic(x, y, z, obstime)
```

The x-y plane is the Earth mean ecliptic, the x-axis points from the Earth to the Sun, and the z-axis points North out of the ecliptic plane.

Parameters

- **x** (*Quantity*) – x-coordinate(s)
- **y** (*Quantity*) – y-coordinate(s)
- **z** (*Quantity*) – z-coordinate(s)
- **obstime** (*datetime, optional*) – Observation time. Required for some transformations between different coordinate systems.

Attributes Summary

default_differential
default_representation
frame_attributes
frame_specific_representation_info
name
obstime

Attributes Documentation

default_differential

default_representation

frame_attributes = {'obstime': <astropy.coordinates.attributes.TimeAttribute object a

frame_specific_representation_info

name = 'GSE'

obstime = None

HeliocentricEarthEcliptic

class heliopy.coordinates.frames.**HeliocentricEarthEcliptic**(**args*, *copy=True*, *representation_type=None*, *differential_type=None*, ***kwargs*)

Bases: astropy.coordinates.baseframe.BaseCoordinateFrame

A coordinate frame in the Heliocentric Earth Ecliptic (HEE) system.

Possible call signatures:

```
hee = HeliocentricEarthEcliptic(x, y, z)
hee = HeliocentricEarthEcliptic(x, y, z, obstime=obstime)
```

The x-y plane is the Earth mean ecliptic, the x-axis points from the Sun to the Earth, and the z-axis points North out of the ecliptic plane.

Parameters

- **x** (*Quantity*) – x-coordinate(s)
- **y** (*Quantity*) – y-coordinate(s)
- **z** (*Quantity*) – z-coordinate(s)
- **obstime** (*datetime, optional*) – Observation time. Required for some transformations between different coordinate systems.

Attributes Summary

default_differential
default_representation
frame_attributes
frame_specific_representation_info
name
obstime

Attributes Documentation

default_differential

```
default_representation
frame_attributes = {'obstime':  <astropy.coordinates.attributes.TimeAttribute object a
frame_specific_representation_info
name = 'HEE'
obstime = None
```

3.3 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.

3.3.1 heliopy.spice Module

Functions

furnish(fname)	Furnish SPICE with a kernel.

furnish

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

Furnish SPICE with a kernel.

Parameters **fname** (*str or list*) – Filename of a spice kernel to load, or list of filenames 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.

3.4 Reading cdf files

cdflib is a module for reading CDF files into python. cdflib is developed by MAVENSDC. This module allows requires no extra dependencies to work, therefore, there is no requirement for CDF installation

All the documentation for cdflib can be found at https://github.com/MAVENSDC/cdflib

cdflib is written by Bryan Harter and Michael Liu.

CHAPTER 4

Development

4.1 Development guide

4.1.1 Testing

Running local tests

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

```
pytest
```

To run all tests apart from the ones that require data to be downloaded run:

```
pytest -m "not data"
```

Doctests

To test code snippets in the documentaiton, change to the `doc` directory and run:

```
make doctest
```

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

4.1.2 Creating a release

1. Fetch the latest copy of the upstream repository

```
git checkout master
git fetch upstream
git merge --ff-only 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 and a python wheel

```
python setup.py sdist
python setup.py bdist_wheel
```

5. Upload created wheels to pypi

```
twine upload dist/*
```

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

6. Update conda package at https://github.com/conda-forge/heliopy-feedstock

4.1.3 Minor and major releases

Major releases

Major releases have version numbers 0.x.0, and contain major new features and any API breaking code. They should be released whenever new features are ready. Development is done on the *master* branch.

Minor releases

Minor releases have version numbers 0.x.y, where y is greater than zero, and contain bug fixes and small improvements (including documentation improvements). Development should be done on the *master* branch, and then changes backported to the *v0.x.x* branch. Minor releases can be released rapidly, when any changes are made.

Python Module Index

h

heliopy.coordinates, 50
heliopy.coordinates.frames, 50
heliopy.data.ace, 23
heliopy.data.artemis, 25
heliopy.data.cassini, 25
heliopy.data.cluster, 27
heliopy.data.dscovr, 28
heliopy.data.helios, 29
heliopy.data.helper, 43
heliopy.data.imp, 34
heliopy.data.messenger, 36
heliopy.data.mms, 36
heliopy.data.omni, 37
heliopy.data.spice, 42
heliopy.data.sunspot, 43
heliopy.data.ulysses, 38
heliopy.data.util, 45
heliopy.data.wind, 40
heliopy.spice, 53

Index

C

cdf2df() (in module heliopy.data.util), 45
cdf_dict() (in module heliopy.data.helper), 44
cdf_units() (in module heliopy.data.util), 46
cdfpeek() (in module heliopy.data.helper), 44
change_units() (heliopy.spice.Trajectory method), 55
cis_codif_h1_moms() (in module heliopy.data.cluster), 27
cis_hia_onboard_moms() (in module heliopy.data.cluster), 27
corefit() (in module heliopy.data.helios), 29

D

daily() (in module heliopy.data.sunspot), 43
default_differential (heliopy.coordinates.frames.GeocentricSolarEcliptic attribute), 52
default_differential (heliopy.coordinates.frames.HeliocentricEarthEcliptic attribute), 52
default_representation (heliopy.coordinates.frames.GeocentricSolarEcliptic attribute), 52
default_representation (heliopy.coordinates.frames.HeliocentricEarthEcliptic attribute), 52
distparams() (in module heliopy.data.helios), 30
distparams_single() (in module heliopy.data.helios), 30
doy2ymd() (in module heliopy.data.util), 46
dtime2doy() (in module heliopy.data.util), 46

E

electron_dist_single() (in module heliopy.data.helios), 30
electron_dists() (in module heliopy.data.helios), 31

F

fgm() (in module heliopy.data.artemis), 25
fgm() (in module heliopy.data.cluster), 28
fgm_hires() (in module heliopy.data.ulysses), 38
fgm_survey() (in module heliopy.data.mms), 37
fpi_dis_moms() (in module heliopy.data.mms), 37
frame_attributes (heliopy.coordinates.frames.GeocentricSolarEcliptic attribute), 52
frame_attributes (heliopy.coordinates.frames.HeliocentricEarthEcliptic attribute), 53
frame_specific_representation_info (heliopy.coordinates.frames.GeocentricSolarEcliptic attribute), 52
frame_specific_representation_info (heliopy.coordinates.frames.HeliocentricEarthEcliptic attribute), 53
furnish() (in module heliopy.spice), 53

G

generate_positions() (heliopy.spice.Trajectory method), 55
generated (heliopy.spice.Trajectory attribute), 54
GeocentricSolarEcliptic (class in heliopy.coordinates.frames), 51
get_kernel() (in module heliopy.data.spice), 42

H

hee_to_gse() (in module heliopy.coordinates.frames), 51
HeliocentricEarthEcliptic (class in heliopy.coordinates.frames), 52
heliopy.coordinates (module), 50
heliopy.coordinates.frames (module), 50
heliopy.data.ace (module), 23
heliopy.data.artemis (module), 25
heliopy.data.cassini (module), 25
heliopy.data.cluster (module), 27
heliopy.data.dscovr (module), 28
heliopy.data.helios (module), 29
heliopy.data.helper (module), 43
heliopy.data.imp (module), 34
heliopy.data.messenger (module), 36
heliopy.data.mms (module), 36
heliopy.data.omni (module), 37

heliopy.data.spice (module), 42
heliopy.data.sunspot (module), 43
heliopy.data.ulysses (module), 38
heliopy.data.util (module), 45
heliopy.data.wind (module), 40
heliopy.spice (module), 53

I

integrated_dists() (in module heliopy.data.helios), 31
integrated_dists_single() (in module heliopy.data.helios), 32
ion_dist_single() (in module heliopy.data.helios), 32
ion_dists() (in module heliopy.data.helios), 33

L

listdata() (in module heliopy.data.helper), 44
load() (in module heliopy.data.util), 46
low() (in module heliopy.data.omni), 37

M

mag15s() (in module heliopy.data.imp), 34
mag320ms() (in module heliopy.data.imp), 35
mag_1min() (in module heliopy.data.cassini), 26
mag_4hz() (in module heliopy.data.helios), 33
mag_h0() (in module heliopy.data.dscovr), 29
mag_hires() (in module heliopy.data.cassini), 26
mag_ness() (in module heliopy.data.helios), 33
mag_rtn() (in module heliopy.data.messenger), 36
merged() (in module heliopy.data.helios), 34
merged() (in module heliopy.data.imp), 35
mfi_h0() (in module heliopy.data.ace), 23
mfi_h0() (in module heliopy.data.wind), 40
mfi_h2() (in module heliopy.data.wind), 41
mitplasma_h0() (in module heliopy.data.imp), 35
monthly() (in module heliopy.data.sunspot), 43

N

name (heliopy.coordinates.frames.GeocentricSolarEcliptic attribute), 52
name (heliopy.coordinates.frames.HeliocentricEarthEcliptic attribute), 53

O

observing_body (heliopy.spice.Trajectory attribute), 54
obstime (heliopy.coordinates.frames.GeocentricSolarEcliptic attribute), 52
obstime (heliopy.coordinates.frames.HeliocentricEarthEcliptic attribute), 53

P

peace_moments() (in module heliopy.data.cluster), 28
pitchdist_cdf2df() (in module heliopy.data.util), 47
process() (in module heliopy.data.util), 48

R

r (heliopy.spice.Trajectory attribute), 54
RemoteFileNotPresentError, 49

S

swe_h0() (in module heliopy.data.ace), 24
swe_h1() (in module heliopy.data.wind), 41
swe_h3() (in module heliopy.data.wind), 41
swi_h2() (in module heliopy.data.ace), 24
swi_h3() (in module heliopy.data.ace), 24
swi_h6() (in module heliopy.data.ace), 25
swics_abundances() (in module heliopy.data.ulysses), 38
swics_heavy_ions() (in module heliopy.data.ulysses), 39
swoops_ions() (in module heliopy.data.ulysses), 40

T

target (heliopy.spice.Trajectory attribute), 54
threedp_pm() (in module heliopy.data.wind), 41
threedp_sfpd() (in module heliopy.data.wind), 42
timefilter() (in module heliopy.data.util), 49
times (heliopy.spice.Trajectory attribute), 54
Trajectory (class in heliopy.spice), 53

U

units_attach() (in module heliopy.data.util), 49

X

x (heliopy.spice.Trajectory attribute), 54

Y

y (heliopy.spice.Trajectory attribute), 54
yearly() (in module heliopy.data.sunspot), 43

Z

z (heliopy.spice.Trajectory attribute), 54