selectionfunctions Documentation

Release v0.1.0

Douglas Boubert

Aug 14, 2022

Contents

`selectionfunctions` provides a unified interface for the selection functions of several major astronomical surveys. This module is entirely derivative of the excellent `dustmaps` package by Gregory M. Green. The `selectionfunctions` package is a product of the Completeness of the Gaia-verse (CoG) collaboration.

To get started, take a look at *Installation* and *Examples*. To see a list of all available selection functions, take a look at *Available Selection Functions*. For a complete reference to the API, see *selectionfunctions modules*.

If you make use of `selectionfunctions` in your research, please cite Green (2018):

```
@ARTICLE{2018JOSS....3..695M,
       author = {{Green}, {Gregory M.}},
        title = "{dustmaps: A Python interface for maps of interstellar dust}",
      journal = {The Journal of Open Source Software},
         year = "2018",
        month = "Jun",
       volume = {3},
       number = {26},
        pages = {695},
          doi = {10.21105/joss.00695},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2018JOSS....3..695M},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER 1

Contents

1.1 Installation

There are two ways to install `selectionfunctions`. If you are familiar with the installation of the `dustmaps` module, then this will be familiar.

1.1.1 1. Using `pip`

From the commandline, run

```
pip install selectionfunctions
```

You may have to use `sudo`.

Next, we'll configure the package and download the selection functions we'll want to use. Start up a python interpreter and type:

```
from selectionfunctions.config import config
config['data_dir'] = '/path/to/store/maps/in'

import selectionfunctions.cog_ii
selectionfunctions.cog_ii.fetch()
```

All the selection functions should now be in the path you gave to `config['data_dir']`. Note that these selection functions can be very large - some are several Gigabytes! Only download those you think you'll need.

1.1.2 2. Using `setup.py`

An alternative way to download `selectionfunctions`, if you don't want to use `pip`, is to download or clone the respository from https://https://github.com/gaiaverse/selectionfunctions.

In this case, you will have to manually make sure that the dependencies are satisfied:

- numpy
- scipy
- astropy
- h5py
- healpy
- requests
- six
- progressbar2

These packages can typically be installed using the Python package manager, `pip`.

Once these dependencies are installed, run the following command from the root directory of the `selectionfunctions` package:

```
python setup.py install --large-data-dir=/path/to/store/maps/in
```

Then, fetch the selection functions you'd like to use. Depending on which selection functions you choose to download, this step can take up several Gigabytes of disk space. Be careful to only download those you think you'll need:

```
python setup.py fetch --map-name=cog_ii
```

That's it!

1.2 Examples

1.2.1 Getting Started

Here, we'll look up a selection function at a number of different locations on the sky and a number of different magnitudes. The principal object in *selectionfunctions* is the *Source* object, which has both a *SkyCoord* attribute giving the position and a *Photometry* attribute giving the photometric measurements. We specify coordinates on the sky using astropy.coordinates.SkyCoord objects. This allows us a great deal of flexibility in how we specify sky coordinates. We can use different coordinate frames (e.g., Galactic, equatorial, ecliptic), different units (e.g., degrees, radians, hour angles), and either scalar or vector input.

For our first example, let's load the *Boubert & Everall (2020, submitted)* – or "cog_ii" – selection function for Gaia DR2, and then query the selection function of a G=21 source at one location on the sky:

```
from selectionfunctions.source import Source
from selectionfunctions import cog_ii

coords = Source('12h30m25.3s', '15d15m58.1s', frame='icrs', photometry={'gaia_g':21.0}
↪)
dr2_sf = cog_ii.dr2_sf(version='modelAB',crowding=True)
prob_selection = dr2_sf(coords)

print('Probability of selection = {:.3f}%'.format(prob_selection*100.0))

>>> Probability of selection = 69.877%
```

A couple of things to note here:

1. Above, we used the ICRS coordinate system, by specifying `frame='icrs'`.

2. We specified the apparent Gaia G magnitude of the star through a Python dictionary. Photometric transformations have not yet been implemented.
3. We used the keywords `version='modelAB'` and `crowding=True` when constructing the selection function. By default, `crowding=False`.

In future, you will be able to query other selection funtions from the `selectionfunctions` package with only minor modification to the above code.

1.2.2 Querying Selection Function at an Array of Coordinates

We can also query an array of coordinates, as follows:

```
import numpy as np
from selectionfunctions.source import Source
from selectionfunctions import cog_ii

l = np.array([0., 90., 180.])
b = np.array([15., 0., -15.])
g = np.array([20.8,21.0,21.2])

coords = Source(l, b, unit='deg', frame='galactic', photometry={'gaia_g':g})

dr2_sf = cog_ii.dr2_sf()
dr2_sf(coords)
>>> array([0.99997069, 0.96233884, 0.58957493])
```

The input need not be a flat array. It can have any shape – the shape of the output will match the shape of the input:

```
import numpy as np
from selectionfunctions.source import Source
from selectionfunctions import cog_ii

l = np.linspace(0., 180., 12)
b = np.zeros(12)
g = 21.0*np.ones(12)
l.shape = (3, 4)
b.shape = (3, 4)
g.shape = (3, 4)

coords = Source(l, b, unit='deg', frame='galactic', photometry={'gaia_g':g})

dr2_sf = cog_ii.dr2_sf()

prob_selection = dr2_sf(coords)

print(prob_selection)
>>> [[0.74045863 0.69877491 0.74045863 0.94768624]
     [0.98794938 0.93834743 0.95561436 0.96803869]
     [0.99962099 0.97286789 0.91445208 0.59940653]]

print(prob_selection.shape)
>>> (3, 4)
```

1.2.3 Plotting a Selection Function

We'll finish by plotting the Gaia DR2 selection function. First, we'll import the necessary modules:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

import astropy.units as units

from selectionfunctions.source import Source
from selectionfunctions import cog_ii
```

Next, we'll set up a grid of coordinates to plot:

```
l = np.linspace(-180.0, 180.0, 1000)
b = np.linspace(-90.0,90.0, 500)
l, b = np.meshgrid(l, b)
g = 21.0*np.ones(l.shape)
coords = Source(l*units.deg, b*units.deg, frame='galactic', photometry={'gaia_g':g})
```

Then, we'll load up and query the Gaia DR2 selection function:

```
dr2_sf = cog_ii.dr2_sf(version='modelAB',crowding=True)
prob_selection = dr2_sf(coords)
```

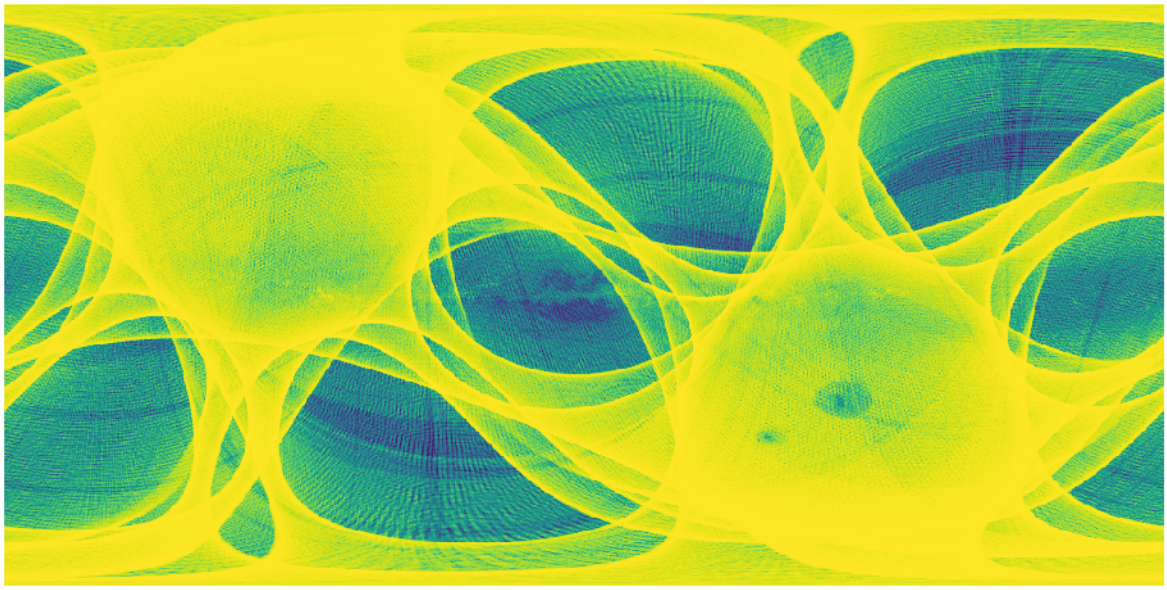
Finally, we create the figure using `matplotlib`:

```
fig = plt.figure(figsize=(12,4), dpi=150)

plt.imshow(
    prob_selection[::,::-1],
    vmin=0.,
    vmax=1.,
    origin='lower',
    interpolation='nearest',
    cmap='viridis',
    aspect='equal',
    extent=[-180,180,-90,90]
)

plt.axis('off')
plt.savefig('map.png', bbox_inches='tight', dpi=150)
```

Here's the result:



1.3 Available Selection Functions

1.3.1 Gaia selection functions

Gaia DR2 (cog_ii.dr2_sf)

The selection function of the Gaia source catalogue is principally determined by the scanning law, which gives the number of times that Gaia observes every location on the sky. Sources must be detected at least five times to have made it into Gaia DR2, but sources are not detected every time Gaia observes them. This selection function models the probability that a source is detected as a function of G magnitude and, optionally, the local source density.

There are four variants of this selection function implemented, which can be used by changing the *version* and *crowding* parameters. We recommend setting `version='modelAB'` and `crowding=True` for most applications, but these are not set by default.

- **Reference**: *Boubert & Everall (2020, submitted)*

1.4 selectionfunctions modules

1.4.1 cog_ii (Boubert & Everall, 2020, submitted)

class selectionfunctions.cog_ii.**dr2_sf**(*map_fname=None*, *version='modelAB'*, *crowding=False*, *bounds=True*)

Bases: *selectionfunctions.map.SelectionFunction*

Queries the Gaia DR2 selection function (Boubert & Everall, 2019).

__init__(*map_fname=None*, *version='modelAB'*, *crowding=False*, *bounds=True*)

Parameters

- **map_fname** (Optional[str]) – Filename of the BoubertEverall2019 selection function. Defaults to None, meaning that the default location is used.

- **version** (Optional[str]) – The selection function version to download. Valid versions are 'modelT' and 'modelAB' Defaults to 'modelT'.
- **crowding** (Optional[bool]) – Whether or not the selection function includes crowding. Defaults to 'False'.
- **bounds** (Optional[bool]) – Whether or not the selection function is bounded to 0.0 < G < 25.0. Defaults to 'True'.

query(*sources*)

Returns the selection function at the requested coordinates.

Parameters **coords** (astropy.coordinates.SkyCoord) – The coordinates to query.

Returns Selection function at the specified coordinates, as a fraction.

class selectionfunctions.cog_ii.**dr3_sf**(*map_fname_dr3=None*, *map_fname_dr2=None*, *version='modelAB'*, *crowding=False*, *bounds=True*)

Bases: *selectionfunctions.cog_ii.dr2_sf*

__init__(*map_fname_dr3=None*, *map_fname_dr2=None*, *version='modelAB'*, *crowding=False*, *bounds=True*)

Args: map_fname (Optional[str]): Filename of the BoubertEverall2019 selection function. Defaults to None, meaning that the default location is used.

version (Optional[str]): The selection function version to download. Valid versions are 'modelT' and 'modelAB' Defaults to 'modelT'.

crowding (Optional[bool]): Whether or not the selection function includes crowding. Defaults to 'False'.

bounds (Optional[bool]): Whether or not the selection function is bounded to 0.0 < G < 25.0. Defaults to 'True'.

selectionfunctions.cog_ii.**fetch**()

Downloads the specified version of the Bayestar dust map.

Parameters **version** (Optional[str]) – The map version to download. Valid versions are 'bayestar2019' (Green, Schlafly, Finkbeiner et al. 2019), 'bayestar2017' (Green, Schlafly, Finkbeiner et al. 2018) and 'bayestar2015' (Green, Schlafly, Finkbeiner et al. 2015). Defaults to 'bayestar2019'.

Raises

- **ValueError** – The requested version of the map does not exist.
- ***DownloadError*** – Either no matching file was found under the given DOI, or the MD5 sum of the file was not as expected.
- **requests.exceptions.HTTPError** – The given DOI does not exist, or there was a problem connecting to the Dataverse.

1.4.2 fetch_utils

exception selectionfunctions.fetch_utils.**DownloadError**

Bases: *selectionfunctions.fetch_utils.Error*

An exception that occurs while trying to download a file.

exception selectionfunctions.fetch_utils.**Error**
Bases: Exception

__weakref__
list of weak references to the object (if defined)

class selectionfunctions.fetch_utils.**FileTransferProgressBar**(*content_length*)
Bases: progressbar.bar.ProgressBar

__init__(*content_length*)
Initializes a progress bar with sane defaults

selectionfunctions.fetch_utils.**check_md5sum**(*fname*, *md5sum*, *chunk_size=1024*)
Checks that a file exists, and has the correct MD5 checksum.

Parameters

- **fname** (*str*) – The filename of the file.
- **md5sum** (*str*) – The expected MD5 sum.
- **chunk_size** (*Optional[int]*) – Process in chunks of this size (in Bytes). Defaults to 1024.

selectionfunctions.fetch_utils.**dataverse_download_doi**(*doi*, *local_fname=None*, *file_requirements={}*, *original=False*, *clobber=False*)
Downloads a file from the Dataverse, using a DOI and set of metadata parameters to locate the file.

Parameters

- **doi** (*str*) – Digital Object Identifier (DOI) containing the file.
- **local_fname** (*Optional[str]*) – Local filename to download the file to. If *None*, then use the filename provided by the Dataverse. Defaults to *None*.
- **file_requirements** (*Optional[dict]*) – Select the file containing the given metadata entries. If multiple files meet these requirements, only the first in downloaded. Defaults to *{}*, corresponding to no requirements.
- **original** (*Optional[bool]*) – Should the original version of the file be downloaded? Only applicable for tabular data. Defaults to *False*.

Raises

- ***DownloadError*** – Either no matching file was found under the given DOI, or the MD5 sum of the file was not as expected.
- **requests.exceptions.HTTPError** – The given DOI does not exist, or there was a problem connecting to the Dataverse.

selectionfunctions.fetch_utils.**dataverse_search_doi**(*doi*)
Fetches metadata pertaining to a Digital Object Identifier (DOI) in the Harvard Dataverse.

Parameters **doi** (*str*) – The Digital Object Identifier (DOI) of the entry in the Dataverse.

Raises **requests.exceptions.HTTPError** – The given DOI does not exist, or there was a problem connecting to the Dataverse.

selectionfunctions.fetch_utils.**download**(*url*, *fname=None*)
Downloads a file.

Parameters

- **url** (*str*) – The URL to download.

- **fname** (*Optional[str]*) – The filename to store the downloaded file in. If *None*, take the filename from the URL. Defaults to *None*.

Returns The filename the URL was downloaded to.

Raises **requests.exceptions.HTTPError** – There was a problem connecting to the URL.

selectionfunctions.fetch_utils.**download_and_verify**(*url*, *md5sum*, *fname=None*, *chunk_size=1024*, *clobber=False*, *verbose=True*)

Downloads a file and verifies the MD5 sum.

Parameters

- **url** (*str*) – The URL to download.
- **md5sum** (*str*) – The expected MD5 sum.
- **fname** (*Optional[str]*) – The filename to store the downloaded file in. If *None*, infer the filename from the URL. Defaults to *None*.
- **chunk_size** (*Optional[int]*) – Process in chunks of this size (in Bytes). Defaults to 1024.
- **clobber** (*Optional[bool]*) – If *True*, any existing, identical file will be overwritten. If *False*, the MD5 sum of any existing file with the destination filename will be checked. If the MD5 sum does not match, the existing file will be overwritten. Defaults to *False*.
- **verbose** (*Optional[bool]*) – If *True* (the default), then a progress bar will be shownd during downloads.

Returns The filename the URL was downloaded to.

Raises

- ***DownloadError*** – The MD5 sum of the downloaded file does not match *md5sum*.
- **requests.exceptions.HTTPError** – There was a problem connecting to the URL.

selectionfunctions.fetch_utils.**get_md5sum**(*fname*, *chunk_size=1024*)

Returns the MD5 checksum of a file.

Parameters

- **fname** (*str*) – Filename
- **chunk_size** (*Optional[int]*) – Size (in Bytes) of the chunks that should be read in at once. Increasing chunk size reduces the number of reads required, but increases the memory usage. Defaults to 1024.

Returns The MD5 checksum of the file, which is a string.

selectionfunctions.fetch_utils.**h5_file_exists**(*fname*, *size_guess=None*, *rtol=0.1*, *atol=1.0*, *dsets={}*)

Returns `True` if an HDF5 file exists, has the expected file size, and contains (at least) the given datasets, with the correct shapes.

Parameters

- **fname** (*str*) – Filename to check.
- **size_guess** (*Optional[int]*) – Expected size (in Bytes) of the file. If `None` (the default), then filesize is not checked.
- **rtol** (*Optional[float]*) – Relative tolerance for filesize.
- **atol** (*Optional[float]*) – Absolute tolerance (in Bytes) for filesize.

- **dsets** (*Optional[dict]*) – Dictionary specifying expected datasets. Each key is the name of a dataset, while each value is the expected shape of the dataset. Defaults to `{}`, meaning that no datasets are checked.

Returns `True` if the file matches by all given criteria.

1.4.3 map

class selectionfunctions.map.**SelectionFunction**

Bases: `object`

Base class for querying selectionfunctions. For each individual selection function, a different subclass should be written, implementing the *query()* function.

__call__(*coords*, ***kwargs*)

An alias for *SelectionFunction.query*.

query(*coords*, ***kwargs*)

Query the selection function at a set of coordinates.

Parameters **coords** (`astropy.coordinates.SkyCoord`) – The coordinates at which to query the selection function.

Raises **NotImplementedError** – This function must be defined by derived classes.

query_equ(*ra*, *dec*, *d=None*, *frame='icrs'*, ***kwargs*)

Query using Equatorial coordinates. By default, the ICRS frame is used, although other frames implemented by `astropy.coordinates` may also be specified.

Parameters

- **ra** (`float`, scalar or array-like) – Galactic longitude, in degrees, or as an `astropy.unit.Quantity`.
- **dec** (*float*, scalar or array-like) – Galactic latitude, in degrees, or as an `astropy.unit.Quantity`.
- **d** (Optional[`float`, scalar or array-like]) – Distance from the Solar System, in kpc, or as an `astropy.unit.Quantity`. Defaults to `None`, meaning no distance is specified.
- **frame** (Optional[`icrs`]) – The coordinate system. Can be `'icrs'` (the default), `'fk5'`, `'fk4'` or `'fk4noeterms'`.
- ****kwargs** – Any additional keyword arguments accepted by derived classes.

Returns The results of the query, which must be implemented by derived classes.

query_gal(*l*, *b*, *d=None*, ***kwargs*)

Query using Galactic coordinates.

Parameters

- **l** (`float`, scalar or array-like) – Galactic longitude, in degrees, or as an `astropy.unit.Quantity`.
- **b** (`float`, scalar or array-like) – Galactic latitude, in degrees, or as an `astropy.unit.Quantity`.
- **d** (Optional[`float`, scalar or array-like]) – Distance from the Solar System, in kpc, or as an `astropy.unit.Quantity`. Defaults to `None`, meaning no distance is specified.
- ****kwargs** – Any additional keyword arguments accepted by derived classes.

Returns The results of the query, which must be implemented by derived classes.

class selectionfunctions.map.**WebSelectionFunction**(*api_url=None*, *map_name=''*)
Bases: object

Base class for querying selection functions through a web API. For each individual selection functions, a different subclass should be written, specifying the base URL.

__call__(*coords*, ***kwargs*)
An alias for WebDustMap.query().

query(*coords*, ***kwargs*)
A web API version of *SelectionFunction.query*. See the documentation for the corresponding local query object.

Parameters **coords** (astropy.coordinates.SkyCoord) – The coordinates at which to query the selection function.

query_equ(*ra*, *dec*, *d=None*, *frame='icrs'*, ***kwargs*)
A web API version of *SelectionFunction.query_equ()*. See the documentation for the corresponding local query object. Queries using Equatorial coordinates. By default, the ICRS frame is used, although other frames implemented by astropy.coordinates may also be specified.

Parameters

- **ra** (float, scalar or array-like) – Galactic longitude, in degrees, or as an astropy.unit.Quantity.
- **dec** (float, scalar or array-like) – Galactic latitude, in degrees, or as an astropy.unit.Quantity.
- **d** (Optional[float, scalar or array-like]) – Distance from the Solar System, in kpc, or as an astropy.unit.Quantity. Defaults to None, meaning no distance is specified.
- **frame** (*Optional[icrs]*) – The coordinate system. Can be 'icrs' (the default), 'fk5', 'fk4' or 'fk4noeterms'.
- ****kwargs** – Any additional keyword arguments accepted by derived classes.

Returns The results of the query.

query_gal(*l*, *b*, *d=None*, ***kwargs*)
A web API version of *SelectionFunction.query_gal()*. See the documentation for the corresponding local query object. Queries using Galactic coordinates.

Parameters

- **l** (float, scalar or array-like) – Galactic longitude, in degrees, or as an astropy.unit.Quantity.
- **b** (float, scalar or array-like) – Galactic latitude, in degrees, or as an astropy.unit.Quantity.
- **d** (Optional[float, scalar or array-like]) – Distance from the Solar System, in kpc, or as an astropy.unit.Quantity. Defaults to None, meaning no distance is specified.
- ****kwargs** – Any additional keyword arguments accepted by derived classes.

Returns The results of the query.

selectionfunctions.map.**coord2healpix**(*coords*, *frame*, *nside*, *nest=True*)
Calculate HEALPix indices from an astropy SkyCoord. Assume the HEALPix system is defined on the coordinate frame frame.

Parameters

- **coords** (astropy.coordinates.SkyCoord) – The input coordinates.

- **frame** (str) – The frame in which the HEALPix system is defined.
- **nside** (int) – The HEALPix nside parameter to use. Must be a power of 2.
- **nest** (Optional[bool]) – True (the default) if nested HEALPix ordering is desired. False for ring ordering.

Returns An array of pixel indices (integers), with the same shape as the input SkyCoord coordinates (coords.shape).

Raises **sfexceptions.CoordFrameError** – If the specified frame is not supported.

selectionfunctions.map.**ensure_coord_type**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where coords is an astropy.coordinates.SkyCoord object.

The decorator raises a TypeError if the coords that gets passed to Class.method is not an astropy.coordinates.SkyCoord instance.

Parameters **f** (*class method*) – A function with the signature (self, coords, **kwargs), where coords is a SkyCoord object containing an array.

Returns A function that raises a TypeError if coords is not an astropy.coordinates.SkyCoord object, but which otherwise behaves the same as the decorated function.

selectionfunctions.map.**ensure_flat_coords**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where coords is an astropy.coordinates.SkyCoord object.

The decorator ensures that the coords that gets passed to Class.method is a flat array. It also reshapes the output of Class.method to have the same shape (possibly scalar) as the input coords. If the output of Class.method is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature (self, coords, **kwargs), where coords is a SkyCoord object containing an array.

Returns A function that takes SkyCoord input with any shape (including scalar).

selectionfunctions.map.**ensure_flat_galactic**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where coords is an astropy.coordinates.SkyCoord object.

The decorator ensures that the coords that gets passed to Class.method is a flat array of Galactic coordinates. It also reshapes the output of Class.method to have the same shape (possibly scalar) as the input coords. If the output of Class.method is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature (self, coords, **kwargs), where coords is a SkyCoord object containing an array.

Returns A function that takes SkyCoord input with any shape (including scalar).

selectionfunctions.map.**ensure_flat_icrs**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where `coords` is an `astropy.coordinates.SkyCoord` object.

The decorator ensures that the `coords` that gets passed to `Class.method` is a flat array of Equatorial coordinates. It also reshapes the output of `Class.method` to have the same shape (possibly scalar) as the input `coords`. If the output of `Class.method` is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature `(self, coords, **kwargs)`, where `coords` is a `SkyCoord` object containing an array.

Returns A function that takes `SkyCoord` input with any shape (including scalar).

1.4.4 source

`selectionfunctions.source.`**`ensure_distance`**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where `coords` is an `astropy.coordinates.SkyCoord` object.

The decorator ensures that the `coords` that gets passed to `Class.method` is a flat array of Equatorial coordinates. It also reshapes the output of `Class.method` to have the same shape (possibly scalar) as the input `coords`. If the output of `Class.method` is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature `(self, coords, **kwargs)`, where `coords` is a `SkyCoord` object containing an array.

Returns A function that takes `SkyCoord` input with any shape (including scalar).

`selectionfunctions.source.`**`ensure_gaia_g`**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where `coords` is an `astropy.coordinates.SkyCoord` object.

The decorator ensures that the `coords` that gets passed to `Class.method` is a flat array of Equatorial coordinates. It also reshapes the output of `Class.method` to have the same shape (possibly scalar) as the input `coords`. If the output of `Class.method` is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature `(self, coords, **kwargs)`, where `coords` is a `SkyCoord` object containing an array.

Returns A function that takes `SkyCoord` input with any shape (including scalar).

`selectionfunctions.source.`**`ensure_gaia_g_gaia_rp`**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where `coords` is an `astropy.coordinates.SkyCoord` object.

The decorator ensures that the `coords` that gets passed to `Class.method` is a flat array of Equatorial coordinates. It also reshapes the output of `Class.method` to have the same shape (possibly scalar) as the input

coords. If the output of Class.method is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature (self, coords, **kwargs), where coords is a SkyCoord object containing an array.

Returns A function that takes SkyCoord input with any shape (including scalar).

selectionfunctions.source.**ensure_tmass_hjk**(*f*)

A decorator for class methods of the form

```
Class.method(self, coords, **kwargs)
```

where coords is an astropy.coordinates.SkyCoord object.

The decorator ensures that the coords that gets passed to Class.method is a flat array of Equatorial coordinates. It also reshapes the output of Class.method to have the same shape (possibly scalar) as the input coords. If the output of Class.method is a tuple or list (instead of an array), each element in the output is reshaped instead.

Parameters **f** (*class method*) – A function with the signature (self, coords, **kwargs), where coords is a SkyCoord object containing an array.

Returns A function that takes SkyCoord input with any shape (including scalar).

1.4.5 healpix_map

class selectionfunctions.healpix_map.**HEALPixFITSQuery**(*fname*, *coord_frame*, *hdu=0*, *field=None*, *dtype='f8'*)

Bases: *selectionfunctions.healpix_map.HEALPixQuery*

A HEALPix map class that is initialized from a FITS file.

__init__(*fname*, *coord_frame*, *hdu=0*, *field=None*, *dtype='f8'*)

Parameters

- **fname** (*str, HDUList, TableHDU or BinTableHDU*) – The filename, HDUList or HDU from which the map should be loaded.
- **coord_frame** (*str*) – The coordinate system in which the HEALPix map is defined. Must be a coordinate frame which astropy understands.
- **hdu** (*Optional[int or str]*) – Specifies which HDU to load the map from. Defaults to 0.
- **field** (*Optional[int or str]*) – Specifies which field (column) to load the map from. Defaults to None, meaning that hdu.data[:] is used.
- **dtype** (*Optional[str or type]*) – The data will be coerced to this datatype. Can be any type specification that numpy understands. Defaults to 'f8', for IEEE754 double precision.

class selectionfunctions.healpix_map.**HEALPixQuery**(*pix_val*, *nest*, *coord_frame*)

Bases: *selectionfunctions.map.SelectionFunction*

A class for querying HEALPix maps.

__init__(*pix_val*, *nest*, *coord_frame*)

Parameters

- **pix_val** (*array*) – Value of the map in every pixel. The length of the array must be of the form *12 * nside**2*, where *nside* is a power of two.
- **nest** (*bool*) – *True* if the map uses nested ordering. *False* if ring ordering is used.
- **coord_frame** (*str*) – The coordinate system that the HEALPix map is in. Should be one of the frames supported by *astropy.coordinates*.

query(*coords*)

Parameters **coords** (*astropy.coordinates.SkyCoord*) – The coordinates to query.

Returns A float array of the value of the map at the given coordinates. The shape of the output is the same as the shape of the coordinates stored by *coords*.

1.4.6 unstructured_map

1.4.7 config

exception selectionfunctions.config.**ConfigError**

Bases: Exception

class selectionfunctions.config.**Configuration**(*fname*)

Bases: object

A class that stores the package configuration.

get(*key*, *default=None*)

Gets a configuration option, returning a default value if the specified key isn't set.

remove(*key*)

Deletes a key from the configuration.

reset()

Resets the configuration, and overwrites the existing configuration file.

save(*force=False*)

Saves the configuration to a JSON, in the standard config location.

Parameters **force** (Optional[bool]) – Continue writing, even if the original config file was not loaded properly. This is dangerous, because it could cause the previous configuration options to be lost. Defaults to False.

Raises ***ConfigError*** – if the configuration file was not successfully loaded on initialization of the class, and force is False.

selectionfunctions.config.**config = <selectionfunctions.config.Configuration object>**

The package configuration. This is the object that the user should interact with in order to change settings. For example, to set the directory where large files (e.g., selections functions) will be stored:

```
from selectionfunctions.config import config
config['data_dir'] = '/path/to/data/directory'
```

1.4.8 std_paths

selectionfunctions.std_paths.**data_dir**()

Returns the directory used to store large data files (e.g., dust maps).

selectionfunctions.std_paths.**fix_path**(*path*)
Returns an absolute path, with '~' expanded to the user's home directory.

selectionfunctions.std_paths.**output_dir**()
Returns a directory that can be used to store temporary output.

1.4.9 json_serializers

class selectionfunctions.json_serializers.**MultiJSONDecoder**(**args*, ***kwargs*)
Bases: json.decoder.JSONDecoder

A JSON decoder that can handle:

- numpy.ndarray
- numpy.dtype
- astropy.units.Quantity
- astropy.coordinates.SkyCoord

__init__(**args*, ***kwargs*)
object_hook, if specified, will be called with the result of every JSON object decoded and its return value will be used in place of the given dict. This can be used to provide custom deserializations (e.g. to support JSON-RPC class hinting).

object_pairs_hook, if specified will be called with the result of every JSON object decoded with an ordered list of pairs. The return value of object_pairs_hook will be used instead of the dict. This feature can be used to implement custom decoders. If object_hook is also defined, the object_pairs_hook takes priority.

parse_float, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

parse_int, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

parse_constant, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

If strict is false (true is the default), then control characters will be allowed inside strings. Control characters in this context are those with character codes in the 0-31 range, including '\t' (tab), '\n', '\r' and '\0'.

selectionfunctions.json_serializers.**deserialize_dtype**(*d*)
Deserializes a JSONified numpy.dtype.

Parameters **d** (dict) – A dictionary representation of a dtype object.

Returns A dtype object.

selectionfunctions.json_serializers.**deserialize_ndarray**(*d*)
Deserializes a JSONified numpy.ndarray. Can handle arrays serialized using any of the methods in this module: "npy", "b64", "readable".

Parameters **d** (*dict*) – A dictionary representation of an ndarray object.

Returns An ndarray object.

selectionfunctions.json_serializers.**deserialize_ndarray_npy**(*d*)
Deserializes a JSONified numpy.ndarray that was created using numpy's save function.

Parameters **d** (dict) – A dictionary representation of an ndarray object, created using numpy.save.

Returns An ndarray object.

selectionfunctions.json_serializers.**deserialize_quantity**(*d*)
Deserializes a JSONified astropy.units.Quantity.

Parameters **d** (dict) – A dictionary representation of a Quantity object.

Returns A Quantity object.

selectionfunctions.json_serializers.**deserialize_skycoord**(*d*)
Deserializes a JSONified astropy.coordinates.SkyCoord.

Parameters **d** (dict) – A dictionary representation of a SkyCoord object.

Returns A SkyCoord object.

selectionfunctions.json_serializers.**deserialize_tuple**(*d*)
Deserializes a JSONified tuple.

Parameters **d** (dict) – A dictionary representation of the tuple.

Returns A tuple.

selectionfunctions.json_serializers.**get_encoder**(*ndarray_mode='b64'*)

Returns a JSON encoder that can handle:

- numpy.ndarray
- numpy.floating (converted to float)
- numpy.integer (converted to int)
- numpy.dtype
- astropy.units.Quantity
- astropy.coordinates.SkyCoord

Parameters **ndarray_mode** (Optional[str]) – Which method to use to serialize numpy.ndarray objects. Defaults to 'b64', which converts the array data to binary64 encoding (non-human-readable), and stores the datatype/shape in human-readable formats. Other options are 'readable', which produces fully human-readable output, and 'npy', which uses numpy's built-in save function and produces completely unreadable output. Of all the methods 'npy' is the most reliable, but also least human-readable. 'readable' produces the most human-readable output, but is the least reliable and loses precision.

Returns A subclass of json.JSONEncoder.

selectionfunctions.json_serializers.**hint_tuples**(*o*)
Annotates tuples before JSON serialization, so that they can be reconstructed during deserialization. Each tuple is converted into a dictionary of the form:

{'_type': 'tuple', 'items': (...)}

This function acts recursively on lists, so that tuples nested inside a list (or doubly nested, triply nested, etc.) will also be annotated.

selectionfunctions.json_serializers.**serialize_dtype**(*o*)
Serializes a numpy.dtype.

Parameters **o** (numpy.dtype) – dtype to be serialized.

Returns A dictionary that can be passed to `json.dumps`.

`selectionfunctions.json_serializers.`**`serialize_ndarray_b64`**`(`*`o`*`)`
Serializes a `numpy.ndarray` in a format where the datatype and shape are human-readable, but the array data itself is binary64 encoded.

Parameters **o** (`numpy.ndarray`) – `ndarray` to be serialized.

Returns A dictionary that can be passed to `json.dumps`.

`selectionfunctions.json_serializers.`**`serialize_ndarray_npy`**`(`*`o`*`)`
Serializes a `numpy.ndarray` using numpy's built-in `save` function. This produces totally unreadable (and very un-JSON-like) results (in "npy" format), but it's basically guaranteed to work in 100% of cases.

Parameters **o** (`numpy.ndarray`) – `ndarray` to be serialized.

Returns A dictionary that can be passed to `json.dumps`.

`selectionfunctions.json_serializers.`**`serialize_ndarray_readable`**`(`*`o`*`)`
Serializes a `numpy.ndarray` in a human-readable format.

Parameters **o** (`numpy.ndarray`) – `ndarray` to be serialized.

Returns A dictionary that can be passed to `json.dumps`.

`selectionfunctions.json_serializers.`**`serialize_quantity`**`(`*`o`*`)`
Serializes an `astropy.units.Quantity`, for JSONification.

Parameters **o** (`astropy.units.Quantity`) – `Quantity` to be serialized.

Returns A dictionary that can be passed to `json.dumps`.

`selectionfunctions.json_serializers.`**`serialize_skycoord`**`(`*`o`*`)`
Serializes an `astropy.coordinates.SkyCoord`, for JSONification.

Parameters **o** (`astropy.coordinates.SkyCoord`) – `SkyCoord` to be serialized.

Returns A dictionary that can be passed to `json.dumps`.

1.5 License

The `selectionfunctions` documentation is covered by the MIT License, as given below.

1.5.1 The MIT License (MIT)

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

C

D

E