dustmaps Documentation

Release v0.1a9

Gregory M. Green

Apr 01, 2017

Contents

A unified interface for 2D and 3D maps of interstellar dust reddening and extinction.

CHAPTER 1

Contents

Installation

There are two ways to install `dustmaps`.

1. Using `pip`

From the commandline, run

```
pip install dustmaps
```

You may have to use `sudo`.

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

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

import dustmaps.sfd
dustmaps.sfd.fetch()

import dustmaps.planck
dustmaps.planck.fetch()

import dustmaps.bayestar
dustmaps.bayestar.fetch()

import dustmaps.iphas
dustmaps.iphas.fetch()

import dustmaps.marshall
dustmaps.marshall.fetch()
```

```
import dustmaps.chen2014
dustmaps.chen2014.fetch()
```

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

2. Using `setup.py`

An alternative way to download `dustmaps`, if you don't want to use `pip`, is to download or clone the respository from https://github.com/gregreen/dustmaps. Then, from the root directory of the package, run

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

Then, fetch the maps you'd like to use:

```
python setup.py fetch --map-name=sfd
python setup.py fetch --map-name=planck
python setup.py fetch --map-name=bayestar
python setup.py fetch --map-name=iphas
python setup.py fetch --map-name=marshall
python setup.py fetch --map-name=chen2014
```

Since these maps are very large - up to several Gigabytes - be careful to only download those you think you'll need. That's it!

Examples

Getting Started

Here, we'll look up the reddening at a number of different locations on the sky. 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 Schlegel, Finkbeiner & Davis (1998) – or "SFD" – dust reddening map, and then query the reddening at one location on the sky:

```
from __future__ import print_function
from astropy.coordinates import SkyCoord
from dustmaps.sfd import SFDQuery

coords = SkyCoord('12h30m25.3s', '15d15m58.1s', frame='icrs')
sfd = SFDQuery()
ebv = sfd(coords)

print('E(B-V) = {:.3f} mag'.format(ebv))

>>> E(B-V) = 0.030 mag
```

A couple of things to note here:

1. In this example, we used `from __future__ import print_function` in order to ensure compatibility with both Python 2 and 3.

2. Above, we used the ICRS coordinate system, by specifying `frame='icrs'`.
3. `SFDQuery` returns reddening in a unit that is similar to magnitudes of **E(B-V)**. However, care should be taken: a unit of SFD reddening is not quite equivalent to a magnitude of **E(B-V)**. The way to correctly convert SFD units to extinction in various broadband filters is to use the conversions in Table 6 of Schlafly & Finkbeiner (2011).

We can query the other maps in the `dustmaps` package with only minor modification to the above code. For example, here's how we would query the Planck Collaboration (2013) dust map:

```
from __future__ import print_function
from astropy.coordinates import SkyCoord
from dustmaps.planck import PlanckQuery

coords = SkyCoord('12h30m25.3s', '15d15m58.1s', frame='icrs')
planck = PlanckQuery()
ebv = planck(coords)

print('E(B-V) = {:.3f} mag'.format(ebv))

>>> E(B-V) = 0.035 mag
```

Querying Reddening at an Array of Coordinates

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

```
from __future__ import print_function
import numpy as np
from astropy.coordinates import SkyCoord
from dustmaps.planck import PlanckQuery
from dustmaps.sfd import SFDQuery

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

coords = SkyCoord(l, b, unit='deg', frame='galactic')

planck = PlanckQuery()
planck(coords)
>>> array([ 0.50170666,  1.62469053,  0.29259142])

sfd = SFDQuery()
sfd(coords)
>>> array([ 0.55669367,  2.60569382,  0.37351534], dtype=float32)
```

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:

```
from __future__ import print_function
import numpy as np
from astropy.coordinates import SkyCoord
from dustmaps.planck import PlanckQuery

l = np.linspace(0., 180., 12)
b = np.zeros(12, dtype='f8')
l.shape = (3, 4)
b.shape = (3, 4)
```

```
coords = SkyCoord(l, b, unit='deg', frame='galactic')

planck = PlanckQuery()

ebv = planck(coords)

print(ebv)
>>> [[ 315.52438354   28.11778831   23.53047562   20.72829247]
     [   2.20861101   15.68559361    1.46233201    1.70338535]
     [   0.94013882    1.11140835    0.38023439    0.81017196]]

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

Querying 3D Reddening Maps

When querying a 3D dust map, there are two slight complications:

1. There is an extra axis – distance – to care about.
2. Many 3D dust maps are probabilistic, so we need to specify whether we want the median reddening, mean reddening, a random sample of the reddening, etc.

Let's see how this works out with the "Bayestar" dust map of Green, Schlafly & Finkbeiner (2015).

How Distances are Handled

If we don't provide distances in our input, `dustmaps` will assume we want dust reddening along the entire line of sight.

```
from __future__ import print_function
from astropy.coordinates import SkyCoord
from dustmaps.bayestar import BayestarQuery

coords = SkyCoord(180., 0., unit='deg', frame='galactic')
bayestar = BayestarQuery(max_samples=2)

ebv = bayestar(coords, mode='random_sample')

print(ebv)
>>> [ 0.00476     0.00616     0.0073      0.00773     0.00796     0.07453
      0.07473     0.0748      0.07807     0.07831     0.18957999  0.2013
      0.20448001  0.20734     0.21008     0.73733997  0.75415999  0.93702
      0.93956     1.09001005  1.09141004  1.11407995  1.11925006  1.12212002
      1.12284994  1.12289     1.12296999  1.12305999  1.12308002  1.12309003
      1.12311995]
```

Here, the Bayestar map has given us a single random sample of the cumulative dust reddening *along the entire line of sight* – that is, to a set of distances. To see what those distances are, we can call:

```
bayestar.distances
>>> <Quantity [  0.06309573,  0.07943282,  0.1       ,  0.12589255,
                 0.15848933,  0.19952621,  0.25118864,  0.31622776,
                 0.3981072 ,  0.50118726,  0.63095725,  0.79432821,
                 1.        ,  1.2589252 ,  1.58489335,  1.99526215,
                 2.51188707,  3.1622777 ,  3.98107076,  5.01187277,
```

```
                6.3095727 ,  7.94328403, 10.        , 12.58925152,
               15.84893322, 19.95262146, 25.11886978, 31.62277603,
               39.81070709, 50.11872864, 63.09572601] kpc>
```

The return type is an astropy.unit.Quantity instance, which keeps track of units.

If we provide Bayestar with distances, then it will do the distance interpolation for us, returning the cumulative dust reddening out to specific distances:

```
import astropy.units as units

coords = SkyCoord(180.*units.deg, 0.*units.deg,
                  distance=500.*units.pc, frame='galactic')
ebv = bayestar(coords, mode='median')

print(ebv)
>>> 0.10705789
```

Because we have explicitly told Bayestar what distance to evaluate the map at, it returns only a single value.

How Probability is Handled

The Bayestar 3D dust map is probabilistic, meaning that it stores random samples of how dust reddening could increase along each sightline. Sometimes we might be interested in the median reddening to a given point in space, or we might want to have all the samples of reddening out to that point. We specify how we want to deal with the probabilistic nature of the map by providing the keyword argument `mode` to `dustmaps.bayestar.BayestarQuery.__call__`.

For example, if we want all the reddening samples, we invoke:

```
l = np.array([30.,  60., 90.]) * units.deg
b = np.array([10., -10., 15.]) * units.deg
d = np.array([1.5,  0.3, 4.0]) * units.kpc

coords = SkyCoord(l, b, distance=d, frame='galactic')

ebv = bayestar(coords, mode='samples')

print(ebv.shape) # (# of coordinates, # of samples)
>>> (3, 2)

print(ebv)
>>> [[ 0.24641787  0.27142054]    # Two samples at the first coordinate
     [ 0.01696703  0.0149225 ]    # Two samples at the second coordinate
     [ 0.08348     0.11068   ]]   # Two samples at the third coordinate
```

If we instead ask for the mean reddening, the shape of the output is different:

```
ebv = bayestar(coords, mode='mean')

print(ebv.shape) # (# of coordinates)
>>> (3,)

print(ebv)
>>> [ 0.25891921  0.09121627  0.09708   ]
```

The only axis is for the different coordinates, because we have reduced the samples axis by taking the mean.

In general, the shape of the output from the Bayestar map is:

```
(coordinate, distance, sample)
```

where any of the axes can be missing (e.g., if only one coordinate was specified, if distances were provided, or if the median reddening was requested).

Plotting the Dust Maps

We'll finish by plotting a comparison of the SFD, Planck Collaboration and Bayestar Dust maps. First, we'll import the necessary modules:

```
from __future__ import print_function

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

import astropy.units as units
from astropy.coordinates import SkyCoord

from dustmaps.sfd import SFDQuery
from dustmaps.planck import PlanckQuery
from dustmaps.bayestar import BayestarQuery
```

Next, we'll set up a grid of coordinates to plot, centered on the Aquila South cloud:

```
l0, b0 = (37., -16.)
l = np.arange(l0 - 5., l0 + 5., 0.05)
b = np.arange(b0 - 5., b0 + 5., 0.05)
l, b = np.meshgrid(l, b)
coords = SkyCoord(l*units.deg, b*units.deg,
                  distance=1.*units.kpc, frame='galactic')
```

Then, we'll load up and query three different dust maps:

```
sfd = SFDQuery()
Av_sfd = 2.742 * sfd(coords)

planck = PlanckQuery()
Av_planck = 3.1 * planck(coords)

bayestar = BayestarQuery(max_samples=1)
Av_bayestar = 2.742 * bayestar(coords)
```

We've assumed $R_V = 3.1$, and used the coefficient from Table 6 of Schlafly & Finkbeiner (2011) to convert SFD and Bayestar reddenings to magnitudes of A_V.

Finally, we create the figure using `matplotlib`:

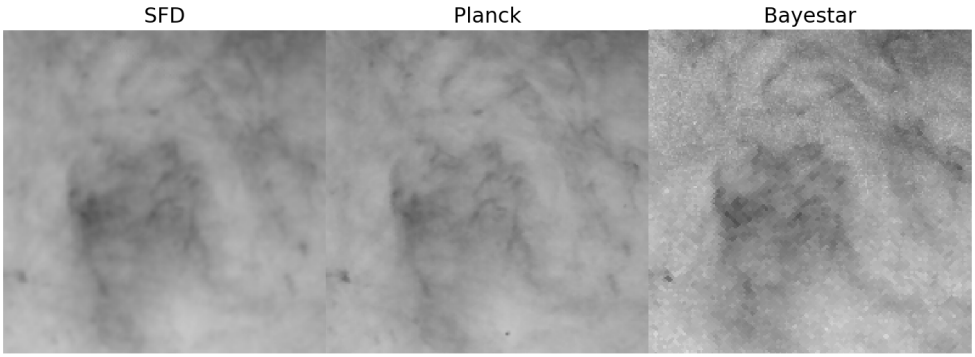
```
fig = plt.figure(figsize=(12,4), dpi=150)

for k,(Av,title) in enumerate([(Av_sfd, 'SFD'),
                               (Av_planck, 'Planck'),
                               (Av_bayestar, 'Bayestar')]):
    ax = fig.add_subplot(1,3,k+1)
    ax.imshow(
        np.sqrt(Av)[::,::-1],
        vmin=0.,
```

```
        vmax=2.,
        origin='lower',
        interpolation='nearest',
        cmap='binary',
        aspect='equal'
    )
    ax.axis('off')
    ax.set_title(title)

fig.subplots_adjust(wspace=0., hspace=0.)
plt.savefig('comparison.png', dpi=150)
```

Here's the result:



Available Dust Maps

Two-Dimensional Dust Maps

SFD

A two-dimensional map of dust reddening across the entire sky. The "SFD" dust map is based on far-infrared emission of dust. The authors model the temperature and optical depth of the dust, and then calibrate a relationship between the dust's far-infrared optical depth and optical reddening. This calibration was updated by Schlafly & Finkbeiner (2011).

In order to convert SFD values of E(B-V) to extinction, one should use the conversions provided in Table 6 of Schlafly & Finkbeiner (2011).

Reference: Schlegel, Finkbeiner & Davis (1998)

Recalibration: Schlafly & Finkbeiner (2011)

Planck

Two-dimensional maps of dust reddening across the entire sky. The Planck Collaboration (2013) fits a modified blackbody dust emission model to the Planck and IRAS far-infrared maps, and provides three different conversions to dust reddening.

The three maps provided by Planck Collaboration (2013) are based on:

1. τ_{353}: dust optical depth at 353 GHz.

2. : thermal dust radiance.
3. A recommende extragalactic reddening estimate, based on thermal dust radiance, but with point sources removed.

Reference: Planck Collaboration (2013)

Website: Planck Explanatory Supplement

Burstein & Heiles

Primarily of historical interest, the Burstein & Heiles (1982) dust reddening maps are derived from HI column density and galaxy counts.

Reference: Burstein & Heiles (1982)

Three-Dimensional Dust Maps

Bayestar

A three-dimensional map of Milky Way dust reddening, covering the three quarters of the sky north of a declination of -30°. The map is probabilistic. containing samples of the reddening along each line of sight. The "Bayestar" dust map is inferred from stellar photometry of 800 million stars observed by Pan-STARRS 1, and 2MASS photometry for a quarter of the stars.

Bayestar values of E(B-V) are in the same units as those of SFD. Therefore, in order to convert Bayestar E(B-V) to extinction in different bands, one should use the conversions provided in Table 6 of Schlafly & Finkbeiner (2011).

Reference: Green, Schlafly, Finkbeiner et al. (2015)

Website: argonaut.skymaps.info

Chen et al. (2014)

A three-dimensional map of dust extinction in the Galactic anticenter. The map covers about 6000 deg^2, from 140° < < 240° and -60° < b < 40°, and is based on stellar photometry from the Xuyi Schmidt Telescope Photometric Survey of the Galactic Anticentre (XSTPS-GAC), 2MASS and *WISE*. The map has an angular resolution of 3 to 9 arcminutes, and reports *r*-band extinction, along with Gaussian error estimates.

Reference: Chen et al. (2014)

Website: http://lamost973.pku.edu.cn

IPHAS

A three-dimensional map of Milky Way dust extinction, covering a 10°-thick strip of the Galactic plane, between 30° < < 120°. The map is probabilistic, containing samples of the cumulative extinction along each line of sight. The map is based on IPHAS imaging of stars. The map returns A_0, the monochromatic extinction.

Reference: Sale et al. (2014)

Website: www.iphas.org/extinction

Marshall et al. (2006)

A three-dimensional map of Milky Way dust extinction, covering a 10°-thick strip of the Galactic plane, between -100° < < 100°. The map is contains 2MASS K_s-band extinctions with a Gaussian uncertainty estimates. The map is based on a comparison of 2MASS colors of stars with expectations from the Besançon model of the Galaxy.

Reference: Marshall et al. (2006)

Website: http://cds.u-strasbg.fr/

dustmap modules

bayestar (Green et al. 2015)

class `dustmaps.bayestar.`**`BayestarQuery`**(*map_fname=None*, *max_samples=None*)
Bases: *`dustmaps.map_base.DustMap`*

Queries the Bayestar 3D dust maps, including Green, Schlafly & Finkbeiner (2015). The maps cover the Pan-STARRS 1 footprint, Dec > -30 deg, amounting to three-quarters of the sky.

`__init__`(*map_fname=None*, *max_samples=None*)

Parameters

- **`map_fname`** (*`Optional[str]`*) – Filename of the Bayestar map. Defaults to *None*, meaning that the default location is used.
- **`max_samples`** (*`Optional[int]`*) – Maximum number of samples of the map to load. Use a lower number in order to decrease memory usage. Defaults to *None*, meaning that all samples will be loaded.

`distances`
Returns the distance bins that the map uses. The return type is `astropy.units.Quantity`, which stores unit-full quantities.

`query`(*coords*, ***kwargs*)
Returns E(B-V) at the requested coordinates. There are several different query modes, which handle the probabilistic nature of the map differently.

Parameters

- **`coords`** (`astropy.coordinates.SkyCoord`) – The coordinates to query.
- **`mode`** (*`Optional[str]`*) – Five different query modes are available: 'random_sample', 'random_sample_per_pix' 'samples', 'median' and 'mean'. The `mode` determines how the output will reflect the probabilistic nature of the Bayestar dust maps.

Returns

Reddening at the specified coordinates, in mags of E(B-V). The shape of the output depends on the `mode`, and on whether `coords` contains distances.

If `coords` does not specify distance(s), then the shape of the output begins with *coords.shape*. If `coords` does specify distance(s), then the shape of the output begins with `coords.shape + ([number of distance bins],)`.

If `mode` is 'random_sample', then at each coordinate/distance, a random sample of reddening is given.

If `mode` is 'random_sample_per_pix', then the sample chosen for each angular pixel of the map will be consistent. For example, if two query coordinates lie in the same map pixel, then the same random sample will be chosen from the map for both query coordinates.

If `mode` is 'median', then at each coordinate/distance, the median reddening is returned.

If `mode` is 'mean', then at each coordinate/distance, the mean reddening is returned.

Finally, if `mode` is 'samples', then all at each coordinate/distance, all samples are returned.

dustmaps.bayestar.**fetch**()
Downloads the Bayestar dust map of Green, Schlafly, Finkbeiner et al. (2015).

dustmaps.bayestar.**lb2pix**(*nside*, *l*, *b*, *nest=True*)
Converts Galactic (l, b) to HEALPix pixel index.

Parameters

- **nside** (*int*) – The HEALPix *nside* parameter.
- **l** (*float, or array of floats*) – Galactic longitude, in degrees.
- **b** (*float, or array of floats*) – Galactic latitude, in degrees.
- **nest** (*Optional[bool]*) – If *True* (the default), nested pixel ordering will be used. If *False*, ring ordering will be used.

Returns The HEALPix pixel index or indices. Has the same shape as the input *l* and *b*.

bh (Burstein & Heiles 1982)

class dustmaps.bh.**BHQuery**(*bh_dir=None*)
Bases: *dustmaps.map_base.DustMap*

Queries the Burstein & Heiles (1982) reddening map.

__init__(*bh_dir=None*)

Parameters **bh_dir** (*Optional[str]*) – The directory containing the Burstein & Heiles dust map. Defaults to *None*, meaning that the default directory is used.

query(*coords*, ***kwargs*)
Returns E(B-V) at the specified location(s) on the sky.

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

Returns A float array of reddening, in units of E(B-V), at the given coordinates. The shape of the output is the same as the shape of the coordinates stored by *coords*.

dustmaps.bh.**ascii2h5**(*bh_dir=None*)
Convert the Burstein & Heiles (1982) dust map from ASCII to HDF5.

chen2014 (Chen et al. 2014)

class dustmaps.chen2014.**Chen2014Query**(*map_fname=None*)
Bases: *dustmaps.unstructured_map.UnstructuredDustMap*

The 3D dust map of Chen et al. (2014), based on stellar photometry from the Xuyi Schmidt Telescope Photometric Survey of the Galactic Anticentre. The map covers 140 deg < l < 240 deg, -60 deg < b < 40 deg.

__init__(*map_fname=None*)

Parameters **map_fname** (*Optional[str]*) – Filename at which the map is stored. Defaults to *None*, meaning that the default filename is used.

distances
Returns the distance bins that the map uses. The return type is `astropy.units.Quantity`, which stores unit-full quantities.

query(*coords*, ***kwargs*)
Returns r-band extinction, A_r, at the given coordinates. Can also return uncertainties.

Parameters

- **coords** (*astropy.coordinates.SkyCoord*) – The coordinates to query.
- **return_sigma** (*Optional[bool]*) – If True, returns the uncertainty in extinction as well. Defaults to False.

Returns

Extinction in the r-band at the specified coordinates, in mags. The shape of the output depends on whether `coords` contains distances.

If *coords* does not specify distance(s), then the shape of the output begins with *coords.shape*. If *coords* does specify distance(s), then the shape of the output begins with *coords.shape + ([number of distance bins],)*.

dustmaps.chen2014.**ascii2h5**(*dat_fname*, *h5_fname*)
Converts from the original ASCII format of the Chen+ (2014) 3D dust map to the HDF5 format.

Parameters

- **dat_fname** (*str*) – Filename of the original ASCII .dat file.
- **h5_fname** (*str*) – Output filename to write the resulting HDF5 file to.

dustmaps.chen2014.**fetch**(*clobber=False*)
Downloads the Chen et al. (2014) dust map.

Parameters **clobber** (*Optional[bool]*) – If `True`, any existing file will be overwritten, even if it appears to match. If `False` (the default), `fetch()` will attempt to determine if the dataset already exists. This determination is not 100% robust against data corruption.

iphas (Sale et al. 2014)

class dustmaps.iphas.**IPHASQuery**(*map_fname=None*)
Bases: *dustmaps.unstructured_map.UnstructuredDustMap*

The 3D dust map of Sale et al. (2014), based on IPHAS imaging in the Galactic plane. The map covers 30 deg < l < 115 deg, -5 deg < b < 5 deg.

__init__(*map_fname=None*)

Parameters **map_fname** (*Optional[str]*) – Filename at which the map is stored. Defaults to *None*, meaning that the default filename is used.

distances
Returns the distance bins that the map uses. The return type is `astropy.units.Quantity`, which stores unit-full quantities.

query(*coords*, ***kwargs*)
Returns A0 at the given coordinates. There are several different query modes, which handle the probabilistic nature of the map differently.

Parameters

- **coords** (astropy.coordinates.SkyCoord) – The coordinates to query.
- **mode** (*Optional[str]*) – Five different query modes are available: 'random_sample', 'random_sample_per_pix' 'samples', 'median' and 'mean'. The mode determines how the output will reflect the probabilistic nature of the IPHAS dust map.

Returns

Monochromatic extinction, A0, at the specified coordinates, in mags. The shape of the output depends on the mode, and on whether coords contains distances.

If coords does not specify distance(s), then the shape of the output begins with *coords.shape*. If *coords* does specify distance(s), then the shape of the output begins with coords.shape + ([number of distance bins],).

If mode is 'random_sample', then at each coordinate/distance, a random sample of reddening is given.

If mode is 'random_sample_per_pix', then the sample chosen for each angular pixel of the map will be consistent. For example, if two query coordinates lie in the same map pixel, then the same random sample will be chosen from the map for both query coordinates.

If mode is 'median', then at each coordinate/distance, the median reddening is returned.

If mode is 'mean', then at each coordinate/distance, the mean reddening is returned.

Finally, if mode is 'samples', then all at each coordinate/distance, all samples are returned.

dustmaps.iphas.**ascii2h5**(*dirname*, *output_fname*)

Converts from a directory of tarballed ASCII ".samp" files to a single HDF5 file. Essentially, converts from the original release format to a single HDF5 file.

dustmaps.iphas.**fetch**(*clobber=False*)

Downloads the IPHAS 3D dust map of Sale et al. (2014).

Parameters **clobber** (*Optional[bool]*) – If True, any existing file will be overwritten, even if it appears to match. If False (the default), fetch() will attempt to determine if the dataset already exists. This determination is not 100% robust against data corruption.

marshall (Marshall et al. 2006)

class dustmaps.marshall.**MarshallQuery**(*map_fname=None*)

Bases: *dustmaps.map_base.DustMap*

Galactic-plane 3D dust map of Marshall et al. (2006), based on 2MASS photometry.

__init__(*map_fname=None*)

Parameters **map_fname** (*Optional[str]*) – Filename at which the map is stored. Defaults to *None*, meaning that the default filename is used.

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

Returns 2MASS Ks-band extinction at the given coordinates.

Parameters

- **coords** (*astropy.coordinates.SkyCoord*) – The coordinates to query. Must contain distances.
- **return_sigma** (*Optional[bool]*) – If True, returns the uncertainty in extinction as well. Defaults to False.

Returns Extinction at the specified coordinates, in mags of 2MASS Ks-band extinction. If `return_sigma` is `True`, then the uncertainty in reddening is also returned, so that the output is `(A, sigma_A)`, where both `A` and `sigma_A` have the same shape as the input coordinates.

`dustmaps.marshall.`**`dat2hdf5`**(*table_dir*)
Convert the Marshall et al. (2006) map from *.dat.gz to *.hdf5.

`dustmaps.marshall.`**`fetch`**(*clobber=False*)
Downloads the Marshall et al. (2006) dust map, which is based on 2MASS stellar photometry.

Parameters **`clobber`** (*`Optional[bool]`*) – If `True`, any existing file will be overwritten, even if it appears to match. If `False` (the default), `fetch()` will attempt to determine if the dataset already exists. This determination is not 100% robust against data corruption.

planck (Planck Collaboration 2013)

class `dustmaps.planck.`**`PlanckQuery`**(*map_fname=None*, *component='extragalactic'*)
Bases: *`dustmaps.healpix_map.HEALPixFITSQuery`*

Queries the Planck Collaboration (2013) dust map.

`__init__`(*map_fname=None*, *component='extragalactic'*)

Parameters

- **`map_fname`** (*`Optional[str]`*) – Filename of the Planck map. Defaults to `None`, meaning that the default location is used.
- **`component`** (*`Optional[str]`*) – Which measure of reddening to use. There are seven valid components. Three denote reddening measures: 'extragalactic', 'tau' and 'radiance'. Four refer to dust properties: 'temperature', 'beta', 'err_temp' and 'err_beta'. Defaults to 'extragalactic'.

`query`(*coords*, ***kwargs*)
Returns E(B-V) (or a different Planck dust inference, depending on how the class was intialized) at the specified location(s) on the sky.

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

Returns A float array of the selected Planck component, at the given coordinates. The shape of the output is the same as the shape of the coordinates stored by `coords`. If extragalactic E(B-V), tau_353 or radiance was chosen, then the output has units of magnitudes of E(B-V). If the selected Planck component is temperature (or temperature error), then an `astropy.Quantity` is returned, with units of Kelvin. If beta (or beta error) was chosen, then the output is unitless.

`dustmaps.planck.`**`fetch`**()
Downloads the Planck Collaboration (2013) dust map, placing it in the default `dustmaps` data directory.

sfd (Schlegel, Finkbeiner & Davis 1998)

class `dustmaps.sfd.`**`SFDQuery`**(*map_dir=None*)
Bases: *`dustmaps.map_base.DustMap`*

Queries the Schlegel, Finbeiner & Davis (1998) dust reddening map.

`__init__`(*map_dir=None*)

Parameters **map_dir** (*Optional[str]*) – The directory containing the SFD map. Defaults to *None*, which means that *dustmaps* will look in its default data directory.

query(*coords*, ***kwargs*)
Returns E(B-V) at the specified location(s) on the sky. See Table 6 of Schlafly & Finkbeiner (2011) for instructions on how to convert this quantity to extinction in various passbands.

Parameters

- **coords** (*astropy.coordinates.SkyCoord*) – The coordinates to query.
- **order** (*Optional[int]*) – Interpolation order to use. Defaults to *1*, for linear interpolation.

Returns A float array containing the SFD E(B-V) at every input coordinate. The shape of the output will be the same as the shape of the coordinates stored by *coords*.

dustmaps.sfd.**fetch**()
Downloads the Schlegel, Finkbeiner & Davis (1998) dust map, placing it in the data directory for *dustmap*.

fetch_utils

exception dustmaps.fetch_utils.**DownloadError**
Bases: dustmaps.fetch_utils.Error

An exception that occurs while trying to download a file.

dustmaps.fetch_utils.**dataverse_download_doi**(*doi*, *local_fname=None*, *file_requirements={}*, *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.

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.

dustmaps.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.

dustmaps.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.

`dustmaps.fetch_utils.`**download_and_verify**(*url*, *md5sum*, *fname=None*, *chunk_size=1024*, *clobber=False*)

Download a file and verify 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*.

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.

`dustmaps.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.

`dustmaps.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.

map_base

class `dustmaps.map_base.`**DustMap**
Bases: `object`

Base class for querying dust maps. For each individual dust map, a different subclass should be written, implementing the *query* function.

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

query(*coords*, ***kwargs*)
Query the map at a set of coordinates.

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

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.
- **dec** (*float, scalar or array-like*) – Galactic latitude, in degrees.
- **d** (*Optinal[float, scalar or array-like]*) – Distance from the Solar System. 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.
- **b** (*float, scalar or array-like*) – Galactic latitude, in degrees.
- **d** (*Optinal[float, scalar or array-like]*) – Distance from the Solar System. 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.

`dustmaps.map_base.`**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 `dustexceptions.CoordFrameError` – If the specified frame is not supported.

`dustmaps.map_base.`**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 `coord` is not an `astropy.coordinates.SkyCoord` object, but which otherwise behaves the same as the decorated function.

`dustmaps.map_base.`**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).

healpix_map

class `dustmaps.healpix_map.`**HEALPixFITSQuery**(*fname*, *coord_frame*, *hdu=0*, *field=0*, *dtype='f8'*)

Bases: *dustmaps.healpix_map.HEALPixQuery*

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

__init__(*fname*, *coord_frame*, *hdu=0*, *field=0*, *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 *0*.
- **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 dustmaps.healpix_map.**HEALPixQuery**(*pix_val*, *nest*, *coord_frame*)
Bases: *dustmaps.map_base.DustMap*

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*.

unstructured_map

class dustmaps.unstructured_map.**UnstructuredDustMap**(*pix_coords*, *max_pix_scale*, *metric_p=2*, *frame=None*)
Bases: *dustmaps.map_base.DustMap*

A class for querying dust maps with unstructured pixels. Sky coordinates are assigned to the nearest pixel.

__init__(*pix_coords*, *max_pix_scale*, *metric_p=2*, *frame=None*)

Parameters

- **pix_coords** (array-like `astropy.coordinates.SkyCoord`) – The sky coordinates of the pixels.
- **max_pix_scale** (scalar `astropy.units.Quantity`) – Maximum angular extent of a pixel. If no pixel is within this distance of a query point, NaN will be returned for that query point.
- **metric_p** (*Optional[float]*) – The metric to use. Defaults to 2, which is the Euclidean metric. A value of 1 corresponds to the Manhattan metric, while a value approaching infinity yields the maximum component metric.
- **frame** (*Optional[str]*) – The coordinate frame to use internally. Must be a frame understood by `astropy.coordinates.SkyCoord`. Defaults to `None`, meaning that the frame will be inferred from `pix_coords`.

config

class dustmaps.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*.

dustmaps.config.**config = <dustmaps.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., dust maps) will be stored:

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

std_paths

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

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

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

License

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

The MIT License (MIT)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

U