dmsky Documentation

Release 0.2.5+39.gbafb2ca.dirty

Alex Drlica-Wagner, Eric Charles, Matthew Wood

Jul 15, 2023

Contents

CHAPTER 1

Documentation Contents

1.1 Installation

1.1.1 Installing with pip

These instructions cover installation with the `pip` package management tool. This will install dmsky and its dependencies into your python distribution.

Before starting the installation process, you will need to determine whether you have setuptools and pip installed in your local python environment.

```
$ curl https://bootstrap.pypa.io/get-pip.py | python -
```

Check if pip is correctly installed:

```
$ which pip
```

Once again, if this isn't the pip in your python environment something went wrong. Now install dmsky by running:

```
$ pip install dmsky
```

To run the ipython notebook examples you will also need to install jupyter notebook:

```
$ pip install jupyter
```

Finally, check that dmsky imports:

```
$ python
Python 2.7.8 (default, Aug 20 2015, 11:36:15)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import dmsky
>>> dmsky.__file__
```

The instructions describe how to install development versions of Dmsky. Before installing a development version we recommend first installing a tagged release following the *Installing with pip* instructions above.

The development version of Dmsky can be installed by running `pip install` with the URL of the git repository:

```
$ pip install git+https://github.com/fermiPy/dmsky.git
```

This will install the most recent commit on the master branch. Note that care should be taken when using development versions as features/APIs under active development may change in subsequent versions without notice.

1.1.2 Upgrading

By default installing dmsky with `pip` or `conda` will get the latest tagged released available on the PyPi package respository. You can check your currently installed version of dmsky with `pip show`:

```
$ pip show dmsky
```

To upgrade your dmsky installation to the latest version run the pip installation command with `--upgrade --no-deps` (remember to also include the `--user` option if you're running at SLAC):

```
$ pip install dmsky --upgrade --no-deps
Collecting dmsky
Installing collected packages: dmsky
  Found existing installation: dmsky 0.2.0
    Uninstalling dmsky-0.2.0:
      Successfully uninstalled dmsky-0.2.0
Successfully installed dmsky-0.2.1
```

1.1.3 Developer Installation

These instructions describe how to install dmsky from its git source code repository using the `setup.py` script. Installing from source can be useful if you want to make your own modifications to the dmsky source code. Note that non-developers are recommended to install a tagged release of dmsky following the *Installing with pip* or instructions above.

First clone the dmsky git repository and cd to the root directory of the repository:

```
$ git clone https://github.com/fermiPy/dmsky.git
$ cd dmsky
```

To install the latest commit in the master branch run `setup.py install` from the root directory:

```
# Install the latest commit
$ git checkout master
$ python setup.py install --user
```

A useful option if you are doing active code development is to install your working copy of the package. This will create an installation in your python distribution that is linked to the copy of the code in your local repository. This allows you to run with any local modifications without having to reinstall the package each time you make a change. To install your working copy of dmsky run with the `develop` argument:

```
# Install a link to your source code installation
$ python setup.py develop --user
```

You can later remove the link to your working copy by running the same command with the `--uninstall` flag:

```
# Install a link to your source code installation
$ python setup.py develop --user --uninstall
```

Specific release tags can be installed by running `git checkout` before running the installation command:

```
# Checkout a specific release tag
$ git checkout X.X.X
$ python setup.py install --user
```

To see the list of available release tags run `git tag`.

1.1.4 Issues

If you get an error about importing matplotlib (specifically something about the macosx backend) you might change your default backend to get it working. The customizing matplotlib page details the instructions to modify your default matplotlibrc file (you can pick GTK or WX as an alternative). Specifically the `TkAgg` and `macosx` backends currently do not work on OSX if you upgrade matplotlib to the version required by dmsky. To get around this issue you can switch to the `Agg` backend at runtime before importing dmsky:

```
>>> import matplotlib
>>> matplotlib.use('Agg')
```

However note that this backend does not support interactive plotting.

If you are running OSX El Capitan or newer you may see errors like the following:

```
dyld: Library not loaded
```

In this case you will need to disable the System Integrity Protections (SIP). See here for instructions on disabling SIP on your machine.

1.2 Quickstart Guide

For a few dmsky tutorials see the *IPython Notebook Tutorials*.

1.2.1 Creating a Target File

A set of pre-defhined targets is included in the package.

1.2.2 Creating a Roster File

A set of pre-defhined targets is included in the package.

1.3 IPython Notebook Tutorials

Additional tutorials with more detailed examples are available as IPython notebooks in the examples directory.

These notebooks can run interactively by running `jupyter notebook` in the examples directory:

```
$ cd dmsky/examples
$ jupyter notebook roster_example.ipynb
```

Note that this will require you to have both ipython and jupyter installed in your python environment. These can be installed in a conda- or pip-based installation as follows:

```
# Install with conda
$ conda install ipython jupyter

# Install with pip
$ pip install ipython jupyter
```

1.4 dmsky package

1.4.1 Module contents

Dark matter skymaps.

1.4.2 Priors on J-factor

class dmsky.priors.**PriorFunctor**(*funcname*, *scale=1.0*)

Bases: object

A functor class that wraps simple functions we use to make priors on parameters.

funcname
A string identifying the function.

log_value(*x*)
Return the log of the function value

Parameters **x** (*numpy.ndarray*) – Input values

Returns **y** – Output values, same shape as x

Return type *numpy.ndarray*

marginalization_bins()
Binning to use to do the marginalization integrals

Default is to marginalize over two decades, centered on mean, using 1000 bins

mean()
Mean value of the function.

normalization()
Normalization, i.e. the integral of the function over the normalization_range.

profile_bins()
The binning to use to do the profile fitting

Default is to profile over +-5 sigma, Centered on mean, using 100 bins

scale
The scale factor applied to input values

sigma()
The 'width' of the function. What this means depend on the function being used.

class dmsky.priors.**FunctionPrior**(*funcname*, *mu*, *sigma*, *fn*, *lnfn=None*, *scale=1.0*)
Bases: *dmsky.priors.PriorFunctor*

Implementation of a prior that simply wraps an existing function

log_value(*x*)
"Return the log of the function value

Parameters **x** (*numpy.ndarray*) – Input values

Returns **y** – Output values, same shape as x

Return type *numpy.ndarray*

mean()
Return the mean value of the function.

normalization()
The normalization i.e., the intergral of the function over the normalization_range

sigma()
Return the 'width' of the function. What this means depend on the function being used.

class dmsky.priors.**GaussPrior**(*mu*, *sigma*, *scale=1.0*)
Bases: *dmsky.priors.FunctionPrior*

Implemenation of a Prior that wraps a Gaussian

class dmsky.priors.**LGaussPrior**(*mu*, *sigma*, *scale=1.0*)
Bases: *dmsky.priors.FunctionPrior*

Implemenation of a Prior that wraps a log Gaussian

class dmsky.priors.**LGaussLogPrior**(*mu*, *sigma*, *scale=1.0*)
Bases: *dmsky.priors.FunctionPrior*

Implemenation of a Prior that wraps the inverse of the log of a Gaussian (i.e., x and y axes are swapped) The prior is implemented in log-space.

class dmsky.priors.**LGaussLikePrior**(*mu*, *sigma*, *scale=1.0*)
Bases: *dmsky.priors.FunctionPrior*

Implemenation of a Prior that wraps the inverse of the log of a Gaussian (i.e., x and y axes are swapped)

class dmsky.priors.**LognormPrior**(*mu*, *sigma*, *scale=1.0*)
Bases: *dmsky.priors.PriorFunctor*

A wrapper around the lognormal function.

A note on the highly confusing scipy.stats.lognorm function... The three inputs to this function are:

s : This is the variance of the underlying gaussian distribution

scale = 1.0 : This is the mean of the linear-space lognormal distribution. The mean of the underlying normal distribution occurs at ln(scale)

loc = 0 : This linearly shifts the distribution in x (DO NOT USE)

The convention is different for numpy.random.lognormal

mean : This is the mean of the underlying normal distribution (so mean = log(scale))

sigma : This is the standard deviation of the underlying normal distribution (so sigma = s)

For random sampling: numpy.random.lognormal(mean, sigma, size)

mean : This is the mean of the underlying normal distribution (so mean = exp(scale))

sigma : This is the standard deviation of the underlying normal distribution (so sigma = s)

scipy.stats.lognorm.rvs(s, scale, loc, size)

s : This is the standard deviation of the underlying normal distribution

scale : This is the mean of the generated random sample scale = exp(mean)

Remember, pdf in log space is plot(log(x), stats.lognorm(sigma,scale=exp(mean)).pdf(x)*x)

Parameters

- **mu** (*float*) – Mean value of the function
- **sigma** (*float*) – Variance of the underlying gaussian distribution

mean()
Mean value of the function.

normalization()
Normalization, i.e. the integral of the function over the normalization_range.

sigma()
The 'width' of the function. What this means depend on the function being used.

class dmsky.priors.**FileFuncPrior**(*filename*, *scale=1.0*)
Bases: *dmsky.priors.PriorFunctor*

A wrapper around the interpolated function.

Parameters **filename** (*string*) – File with the function parameters

mean()
Mean value of the function.

sigma()
The 'width' of the function. What this means depend on the function being used.

1.4.3 Dark Matter Denisty Porfiles

class dmsky.density.**DensityProfile**(***kwargs*)
Bases: pymodeler.model.Model

Am abstract base class for DM density profiles

At a minimum sub-classes need to implement the self._rho(r) method to compute the density as a function of radius from the center of the halo

deriv_params
Return the list of paramters we can take derivatives w.r.t.

mass(*r=None*)
Compute the mass of the object out to a particular radius.

Parameters **r** (*numpy.array* or float) – The radii

Returns **values** – Return values, same shape as the input radii

Return type *numpy.array*

rho(*r*)
Return the density for given radii.

Parameters **r** (*numpy.array* or float) – The radii

Returns **values** – Return values, same shape as the input radii

Return type *numpy.array*

rho_deriv(*r*, *paramNames*)

Return the derivatives of the density as a function of radius, w.r.t. a list of parameters

Parameters

- **r** (*numpy.array* or float) – The radii
- **paramNames** (*list*) – The names of the parameters to differentiation w.r.t.

Returns **matrix** – An n x m array, where: n is the number of radii m is the number of parameters

Return type *numpy.array*

rho_uncertainty(*r*)

Calculate the uncertainty of the density at given radii

Parameters **r** (*numpy.array* or float) – The radii

Returns **values** – Return values, same shape as the input radii

Return type *numpy.array*

set_mvir_c(*mvir*, *c*)

Fix the mass inside the virial radius.

Parameters

- **mvir** (*float*) – The virial radius
- **c** (*float*) – Scale factor

set_rho_r(*rho*, *r*)

Fix the density normalization at a given radius.

Parameters

- **rho** (*float*) – The normalization density
- **r** (*float*) – The corresponding radius

class dmsky.density.**UniformProfile**(***kwargs*)

Bases: *dmsky.density.DensityProfile*

Uniform spherical profile rho(r) = rhos for r <= rs rho(r) = 0 otherwise

class dmsky.density.**IsothermalProfile**(***kwargs*)

Bases: *dmsky.density.DensityProfile*

Non-Singular Isothermal Profile: Begeman et al. MNRAS 249, 523 (1991) http://adsabs.harvard.edu/full/1991MNRAS.249..523B rho(r) = rhos/(1+(r/rs))**2

class dmsky.density.**BurkertProfile**(***kwargs*)

Bases: *dmsky.density.DensityProfile*

Burkert ApJ 447, L25 (1995) [Eqn. 2] http://arxiv.org/abs/astro-ph/9504041 rho(r) = rho0 * r0**3 / ((r + r0)*(r**2+r0**2)) ==> rho(r) = rhos / ((1+r/rs)*(1+(r/rs)**2))

class dmsky.density.**NFWProfile**(***kwargs*)

Bases: *dmsky.density.DensityProfile*

Navarro, Frenk, and White, ApJ 462, 563 (1996) http://arxiv.org/abs/astro-ph/9508025 rho(r) = rhos / ((r/rs) * (1+r/rs)**2)

jvalue_fast(*r=None*)
Fast integrated J-factor computation

class dmsky.density.**GNFWProfile**(***kwargs*)
Bases: *dmsky.density.DensityProfile*

Generalized NFW Profile Strigari et al. ApJ 678, 614 (2008) [Eqn. 3] http://arxiv.org/abs/0709.1510 rho(r) = rhos / ((r/rs)**gamma * (1+r/rs)**(3-gamma))

deriv_params
Return the list of paramters we can take derivatives w.r.t.

class dmsky.density.**EinastoProfile**(***kwargs*)
Bases: *dmsky.density.DensityProfile*

Einasto profile Einasto Trudy Inst. Astrofiz. Alma-Ata 5, 87 (1965) (Russian) [Eqn. 4] http://adsabs.harvard.edu/abs/1965TrAlm...5...87E rho(r) = rhos*exp(-2*((r/rs)**alpha-1)/alpha) ==>

deriv_params
Return the list of paramters we can take derivatives w.r.t.

class dmsky.density.**ZhouProfile**(***kwargs*)
Bases: *dmsky.density.DensityProfile*

Generalized double power-law models Zhou MNRAS 278, 488 (1996) [Eqn. 1] http://arxiv.org/abs/astro-ph/9509122 rho(r) = C * (r/rs)**-gamma * (1 + (r/rs)**1/alpha))**-(beta-gamma)*alpha C = 4 * rhos

also see... Zhou MNRAS 287, 525 (1997) [Eqn. 2] http://arxiv.org/abs/astro-ph/9605029 Strigari et al., Nature 454 (2008) [Eqn. 8] http://arxiv.org/abs/0808.3772

deriv_params
Return the list of paramters we can take derivatives w.r.t.

1.4.4 Line of sight integration

class dmsky.jcalc.**LoSFn**(*dp*, *d*, *xi*, *alpha=3.0*)
Bases: object

Integrand function (luminosity density) for LoS integration. The parameter *alpha* introduces a change of variables:

x' = x^(1/alpha).

A value of alpha > 1 samples the integrand closer to x = 0 (distance of closest approach). The change of variables requires the substitution:

dx = alpha * (x')^(alpha-1) dx'

func(*r*)
Function to compute the integrand

class dmsky.jcalc.**LoSAnnihilate**(*dp*, *d*, *xi*, *alpha=3.0*)
Bases: *dmsky.jcalc.LoSFn*

Integrand function for LoS annihilation (J-factor).

func(*r*)
Function to compute the line-of-sight integrand

class dmsky.jcalc.**LoSAnnihilate_Deriv**(*dp*, *d*, *xi*, *paramNames*, *alpha=3.0*)
Bases: *dmsky.jcalc.LoSFn*

Integrand function for LoS annihilation (J-factor).

func(*r*)
Function to compute the line-of-sight integrand

class dmsky.jcalc.**LoSDecay**(*dp*, *d*, *xi*, *alpha=3.0*)
Bases: *dmsky.jcalc.LoSFn*

Integrand function for LoS decay (D-factor).

func(*r*)
Function to compute the line-of-sight integrand

class dmsky.jcalc.**LoSDecay_Deriv**(*dp*, *d*, *xi*, *paramNames*, *alpha=1.0*)
Bases: *dmsky.jcalc.LoSFn*

Integrand function for LoS decay (D-factor).

func(*r*)
Function to compute the line-of-sight integrand

class dmsky.jcalc.**LoSIntegral**(*density*, *dhalo*, *alpha=3.0*, *ann=True*, *derivPar=None*)
Bases: object

Slowest (and most accurate?) LoS integral. Uses scipy.integrate.quad with a change of variables to better sample the LoS close to the halo center.

angularIntegral(*angle=None*)
Compute the solid-angle integrated j-value within a given radius

Parameters **angle** (*numpy.ndarray* or None) – Maximum integration angle (in degrees) If None, use the 'rmax' and 'dhalo' parameters.

Returns **values** – Return values, same shape as the input xp

Return type *numpy.array*

name
Return the name of this profile

rmax
Return the maximum integration radius

class dmsky.jcalc.**LoSIntegralFast**(*density*, *dhalo*, *alpha=3.0*, *ann=True*, *nsteps=400*, *derivPar=None*)
Bases: *dmsky.jcalc.LoSIntegral*

Vectorized version of LoSIntegral that performs midpoint integration with a fixed number of steps.

rmax
Return the maximum integration radius

class dmsky.jcalc.**LoSIntegralInterp**(*density*, *dhalo*, *alpha=3.0*, *ann=True*, *nsteps=400*, *derivPar=None*)
Bases: *dmsky.jcalc.LoSIntegralFast*

Interpolate fast integral a for even faster look-up.

create_func(*dhalo*)
Create the spline function

Parameters **dhalo** (*numpy.ndarray*) – Array of halo distances

Returns **func** – A function that return J-factor as a function of psi and dhalo

Return type function

create_profile(*dhalo*, *nsteps=None*)
Create a spatial J-factor profile

Parameters

- **dhalo** (*numpy.ndarray*) – Array of halo distances
- **nsteps** (*int*) – Number of steps for vectorization

Returns

- **dhalo, psi** (*numpy.meshgrid*) – Array of halo distances and angular offsets
- **jval** (*numpy.array*) – Corresponding J-factors

class dmsky.jcalc.**LoSIntegralFile**(*dp*, *dist*, *filename*, *ann=True*)
Bases: *dmsky.jcalc.LoSIntegralInterp*

Interpolate over a pre-generated file.

NOT IMPLEMENTED YET

create_profile(*dhalo*, *nsteps=300*)
Build the profile values

class dmsky.jcalc.**ROIIntegrator**(*jspline*, *lat_cut*, *lon_cut*, *source_list=None*)
Bases: object

Class to integrate a J-factor over a region of interest

compute()
Integrate the ROI

eval(*rgc*, *decay=False*)
Evaluate the J-factor

print_profile(*decay=False*)
Print the profile

1.4.5 Dark matter targets

class dmsky.targets.**Target**(***kwargs*)
Bases: pymodeler.model.Model

A DM search target

create_dmap(*npix=150*, *subsample=4*, *coordsys='CEL'*, *projection='AIT'*)
Create a D-factor map

Parameters

- **npix** (*int*) – Number of pixels along one axis of output map
- **subsample** (*int*) – Number of subsamples to take per pixel
- **coordsys** (*str*) – Coordniate system: 'GAL' or 'CEL'
- **projection** (*str*) – Map projection

Returns

- **image** (*numpy.ndarray*) – Image data
- **pix** (*numpy.ndarray*) – Pixel coordinatates
- **wcs** (*WCS.wcs*) – WCS object for map

create_jmap(*npix=150*, *subsample=4*, *coordsys='CEL'*, *projection='AIT'*)
Create a J-factor map

Parameters

- **npix** (*int*) – Number of pixels along one axis of output map
- **subsample** (*int*) – Number of subsamples to take per pixel
- **coordsys** (*str*) – Coordniate system: 'GAL' or 'CEL'
- **projection** (*str*) – Map projection

Returns

- **image** (*numpy.ndarray*) – Image data
- **pix** (*numpy.ndarray*) – Pixel coordinatates
- **wcs** (*WCS.wcs*) – WCS object for map

dsigma(*ra*, *dec*)
Return the uncertainty of J in any direction

Parameters

- **ra** (*numpy.ndarray*) – Right-accension (in degrees)
- **dec** (*numpy.ndarray*) – Declination (in degrees)

Returns **values** – Return values, same shape as the input ra, dec

Return type *numpy.array*

dvalue(*ra*, *dec*)
Return the D-factor in any direction

Parameters

- **ra** (*numpy.ndarray*) – Right-accension (in degrees)
- **dec** (*numpy.ndarray*) – Declination (in degrees)

Returns **values** – Return values, same shape as the input ra, dec

Return type *numpy.array*

jsigma(*ra*, *dec*)
Return the uncertainty of J in any direction

Parameters

- **ra** (*numpy.ndarray*) – Right-accension (in degrees)
- **dec** (*numpy.ndarray*) – Declination (in degrees)

Returns **values** – Return values, same shape as the input ra, dec

Return type *numpy.array*

jvalue(*ra*, *dec*)
Return the J-factor in any direction

Parameters

- **ra** (*numpy.ndarray*) – Right-accension (in degrees)
- **dec** (*numpy.ndarray*) – Declination (in degrees)

Returns **values** – Return values, same shape as the input ra, dec

Return type *numpy.array*

write_d_rad_file(*d_rad_file=None*, *npts=50*, *minpsi=0.0001*)
Write a text file with the D-fractor radial profile

Parameters

- **d_rad_file** (*str*) – Filename to write to
- **npts** (*int*) – Number of angles to write
- **minpsi** (*float*) – Value for smallest angle to write

write_dmap(*filename*, *npix=150*, *clobber=False*, *map_kwargs=None*, *file_kwargs=None*)
Write the D-factor to a template map.

write_dmap_hpx(*filename*)
Write the D-factor to a template map.

write_dmap_wcs(*filename*, *npix=150*, *clobber=False*, *map_kwargs=None*, *file_kwargs=None*)
Write the D-factor to a template map.

write_j_rad_file(*j_rad_file=None*, *npts=50*, *minpsi=0.0001*)
Write a text file with the J-fractor radial profile

Parameters

- **j_rad_file** (*str*) – Filename to write to
- **npts** (*int*) – Number of angles to write
- **minpsi** (*float*) – Value for smallest angle to write

write_jmap(*filename*, *npix=150*, *clobber=False*, *map_kwargs=None*, *file_kwargs=None*)
Write the J-factor to a template map.

write_jmap_hpx(*filename*)
Write the J-factor to a template map.

write_jmap_wcs(*filename*, *npix=150*, *clobber=False*, *map_kwargs=None*, *file_kwargs=None*)
Write the J-factor to a template map.

class dmsky.targets.**Galactic**(***kwargs*)
Bases: *dmsky.targets.Target*

Class to add specifics for Galactic DM targets

class dmsky.targets.**Dwarf**(***kwargs*)
Bases: *dmsky.targets.Target*

Class to add specifics for Dwarf Galaxy DM targets

j_photo(*a=18.17*, *b=0.23*)
Photometric J-factor from Eq 14 of Pace & Strigari (2019) [1802.06811v2]

J = 10^{a} * (Lv / 1e4 Lsun)^{b} * (D/100 kpc)^-2 * (rhalf/100 pc)^-0.5

For Pace & Strigari (2019): a = 18.17, b = 0.23 For Pace & Strigari 1802.06811v1: a = 17.93, b = 0.32

Parameters

- **a** (*normalization exponent*) –
- **b** (*luminosity scaling*) –

Returns **J**

Return type photometric J-factor within 0.5 deg

class dmsky.targets.**Galaxy**(***kwargs*)
Bases: *dmsky.targets.Target*

Class to add specifics for Galaxy DM targets

class dmsky.targets.**Cluster**(***kwargs*)
Bases: *dmsky.targets.Target*

Class to add specifics for Galaxy Cluster DM targets

class dmsky.targets.**Isotropic**(***kwargs*)
Bases: *dmsky.targets.Target*

Class to add specifics for Isotropic DM targets

1.4.6 Plotting fucntions

Module for plotting stuff in dmsky.

dmsky.plotting.**plot_density**(*name*, *targets*, *xlims=None*, *nstep=100*)
Make a plot of the density as a function of distance from the target center.

Parameters

- **name** (*str*) – Name for the plot
- **targets** (*list*) – List of targets to include in the plot
- **xlims** (*tuple*) – Range for the x-axis of the plot
- **nteps** (*int*) – Number of points to include in the plot

Returns

- **fig** (*matplotlib.Figure*)
- **ax** (*matplotlib.Axes*)
- **leg** (*matplotlib.Legend*)

dmsky.plotting.**plot_j_integ**(*name*, *targets*, *xlims=None*, *nstep=100*, *ylims=None*)
Make a plot of the integrated J-factor as a function of the angle from the target center.

Parameters

- **name** (*str*) – Name for the plot
- **targets** (*list*) – List of targets to include in the plot
- **xlims** (*tuple*) – Range for the x-axis of the plot
- **nteps** (*int*) – Number of points to include in the plot
- **ylims** (*tuple*) – Range for the y-axis of the plot

Returns

- **fig** (*matplotlib.Figure*)
- **ax** (*matplotlib.Axes*)
- **leg** (*matplotlib.Legend*)

dmsky.plotting.**plot_j_profile**(*name*, *targets*, *xlims=None*, *nstep=100*, *ylims=None*)
Make a plot of the J-factor as a function of the angle from the target center.

Parameters

- **name** (*str*) – Name for the plot
- **targets** (*list*) – List of targets to include in the plot
- **xlims** (*tuple*) – Range for the x-axis of the plot
- **nteps** (*int*) – Number of points to include in the plot
- **ylims** (*tuple*) – Range for the y-axis of the plot

Returns

- **fig** (*matplotlib.Figure*)
- **ax** (*matplotlib.Axes*)
- **leg** (*matplotlib.Legend*)

1.4.7 Skymap class

1.4.8 Roster class

class dmsky.roster.**Roster**(**targets*, ***kwargs*)
Bases: collections.OrderedDict

A self-consistent set of search targets, typically with exactly one version for any given target.

1.4.9 Top-level bookkeeping

Factory for generating instances of classes

dmsky.factory.**factory**(*cls*, *module=None*, ***kwargs*)
Factory for creating objects. Arguments are passed directly to the constructor of the chosen class.

Parameters

- **cls** (*str*) – Class name of the object to create
- **module** (*str*) – python module defining the class in question

Returns **object** – Newly created object

Return type *object*

class dmsky.library.**ObjectLibrary**(*path=None*)
Bases: object

Library that keeps track of object we are building

classmethod load_library(*paths*)
Build the library by walking through paths

class dmsky.targets.**TargetLibrary**(*path=None*)
Bases: *dmsky.library.ObjectLibrary*

A top-level object, keeping track of all the *Target* objects that we have created

create_target(*name*, *version=None*, *default='default'*, ***kwargs*)
Create a *Target*

Parameters

- **name** (*str*) – A name for the *Target*

- **version** (*str*) – Key that specifies which set of parameters to used for this target
- **default** (*str*) – Name of default parameter dictionary to use

Returns **target** – The newly created Target

Return type *Target*

get_target_dict(*name*, *version=None*, *default='default'*, ***kwargs*)

Step through the various levels of dependencies to get the full dictionary for a target.

target: version -> ... -> target: default -> default: type

Parameters

- **name** (*str*) – target name
- **version** (*str*) – version for target parameters
- **default** (*str*) – name of default parameters to use
- **kwargs** (*dict*) – keyword arguments passed to target dict

Returns **dict**

Return type dictionary of target parameters

class dmsky.roster.**RosterLibrary**(*path=None*)

Bases: *dmsky.library.ObjectLibrary*

A top-level object, keeping track of all the *Roster* objects that we have created

create_roster(*name*, **targets*, ***kwargs*)

Create a roster

Parameters

- **name** (*str*) – A name for the Roster
- **targets** (*list*) – Objects that will go on the Roster

Returns **roster** – The newly created Roster

Return type *Roster*

get_roster_list(*name*, **targets*)

Get the list of objects on a roster, creating them in needed.

Parameters

- **name** (*str*) – A name for the list
- **targets** (*list*) – Objects that will go on the list

Returns **items** – List of items needed to create a *Roster*

Return type list

1.5 dmsky.utils subpackage

1.5.1 dmsky.utils.coords module

Utilities for working with coordinates.

dmsky.utils.coords.**angsep**(*lon1*, *lat1*, *lon2*, *lat2*)
Angular separation (deg) between two sky coordinates. Faster than creating astropy coordinate objects.

Parameters

- **lon1** (*numpy.array*) –
- **lat1** (*numpy.array*) –
- **lon2** (*numpy.array*) –
- **lat2** (*numpy.array*) – Coordinates (in degrees)

Returns **dist** – Angular seperations (in degrees)

Return type *numpy.array*

Notes

The angular separation is calculated using the Vincenty formula [1], which is slighly more complex and computationally expensive than some alternatives, but is stable at at all distances, including the poles and antipodes.

[1] http://en.wikipedia.org/wiki/Great-circle_distance

dmsky.utils.coords.**cel2gal**(*ra*, *dec*)
Convert celestial coordinates (J2000) to Galactic coordinates. (Much faster than astropy for small arrays.)

Parameters

- **ra** (*numpy.array*) –
- **dec** (*numpy.array*) – Celestical Coordinates (in degrees)

Returns

- **glon** (*numpy.array*)
- **glat** (*numpy.array*) – Galactic Coordinates (in degrees)

dmsky.utils.coords.**gal2cel**(*glon*, *glat*)
Convert Galactic coordinates to celestial coordinates (J2000). (Much faster than astropy for small arrays.)

Parameters

- **glon** (*numpy.array*) –
- **glat** (*numpy.array*) – Galactic Coordinates (in degrees)

Returns

- **ra** (*numpy.array*)
- **dec** (*numpy.array*) – Celestical Coordinates (in degrees)

1.5.2 dmsky.utils.speed module

Utilities for speed testing

dmsky.utils.speed.**speedtest**(*func*, **args*, ***kwargs*)
Test the speed of a function.

1.5.3 dmsky.utils.healpix module

1.5.4 dmsky.utils.stat_funcs module

Utilities for statistical operations

dmsky.utils.stat_funcs.**gauss**(*x*, *mu*, *sigma=1.0*)
Gaussian

dmsky.utils.stat_funcs.**lgauss**(*x*, *mu*, *sigma=1.0*, *logpdf=False*)
Log10 normal distribution...

Parameters

- **x** (*numpy.array* or list) – Parameter of interest for scanning the pdf
- **mu** (*float*) – Peak of the lognormal distribution (mean of the underlying normal distribution is log10(mu)
- **sigma** (*float*) – Standard deviation of the underlying normal distribution
- **logpdf** (*bool*) – Define the PDF in log space

Returns **vals** – Output values, same shape as x

Return type *numpy.array*

dmsky.utils.stat_funcs.**ln_log10norm**(*x*, *mu*, *sigma=1.0*)
Natural log of base 10 lognormal

dmsky.utils.stat_funcs.**ln_norm**(*x*, *mu*, *sigma=1.0*)
Natural log of scipy norm function truncated at zero

dmsky.utils.stat_funcs.**lngauss**(*x*, *mu*, *sigma=1.0*)
Natural log of a Gaussian

dmsky.utils.stat_funcs.**lnlgauss**(*x*, *mu*, *sigma=1.0*, *logpdf=False*)
Log-likelihood of the natural log of a Gaussian

dmsky.utils.stat_funcs.**log10norm**(*x*, *mu*, *sigma=1.0*)
Scale scipy lognorm from natural log to base 10 x : input parameter mu : mean of the underlying log10 gaussian sigma : variance of underlying log10 gaussian

dmsky.utils.stat_funcs.**lognorm**(*x*, *mu*, *sigma=1.0*)
Log-normal function from scipy

dmsky.utils.stat_funcs.**norm**(*x*, *mu*, *sigma=1.0*)
Scipy norm function

1.5.5 dmsky.utils.tools module

Random python tools

dmsky.utils.tools.**get_items**(*items*, *library*)
Grab list items recursing through existing rosters. Careful of recursion traps.

Some prefix comprehension: '++' = Always add '+' = Add only if unique '–' = Remove last occurence '-' = Remove all occurences

dmsky.utils.tools.**getnest**(*d*, **keys*)
Function to return nested keys.

dmsky.utils.tools.**item_prefix**(*item*)
Get the item prefix ('+','++','-','–","").

dmsky.utils.tools.**item_version**(*item*)
Split the item and version based on sep ':'

dmsky.utils.tools.**merge_dict**(*d0*, *d1*, *add_keys=True*, *append=False*)
Merge two target dicts into a new dict.

Parameters

- **d0** (*Base dictionary*) –
- **d1** (*Update dictionary*) –

Returns **d**

Return type A new dictionary merging d and u

dmsky.utils.tools.**update_dict**(*d0*, *d1*, *add_keys=True*, *append=False*)
Recursively update the contents of dictionary d0 with the contents of python dictionary d1.

Parameters

- **d0** (*Base dictionary*) –
- **d1** (*Update dictionary*) –

dmsky.utils.tools.**yaml_dump**(*x*, *filename*)
Dump object to a yaml file (use libyaml when available) x : output to dump to the file filename : output file (can be file-type or path string)

dmsky.utils.tools.**yaml_load**(*filename*)
Load a yaml file (use libyaml when available)

1.5.6 dmsky.utils.units module

Simple module to deal with unit conversion

class dmsky.utils.units.**Units**
Bases: object

Simple class to keep track of unit conversions

cm = 1

cm3_s = 1.0

static convert_from(*value*, *key*)
Convert to cgs units from a different type of units

Parameters

- **value** (*scalar or array-like, the input value(s)*) –
- **key** (*str, a key corresponding to one of the globals defined above*) –

Returns

Return type the input values, converted to cgs units

static convert_to(*value*, *key*)
Convert from cgs units to a different type of units

Parameters

- **value** (*scalar or array-like, the input value(s)*) –
- **key** (*str, a key corresponding to one of the globals defined above*) –

Returns

Return type the input values, converted to requested units

deg2 = 0.00030461741978670857

g = 1.0

g_cm3 = 1.0

static get_value(*key*)

Get a conversion value based on a key

This is here to make it easy to automate unit conversion

Parameters key (*str, a key corresponding to one of the globals defined above*) –

Returns

Return type the conversion constant

gev = 1.78266e-24

gev2_cm5 = 3.1778766755999995e-48

gev_cm2 = 1.78266e-24

gev_cm3 = 1.78266e-24

hr = 3600.0

k = 'gev_cm3'

km = 100000.0

kpc = 3.08568e+21

m = 100.0

m2 = 10000.0

map_from_astropy = {'GeV / cm2': 'gev_cm2', 'GeV / cm3': 'gev_cm3', 'GeV2 / cm5': '

map_to_astropy = {'cm3_s': 'cm3 / s', 'g_cm3': 'g / cm3', 'gev2_cm5': 'GeV2 / cm5',

msun = 1.98892e+33

msun2_kpc5 = 1.4141002588357058e-41

msun2_pc5 = 1.4141002588357058e-26

msun_kpc3 = 6.769625720905608e-32

msun_pc3 = 6.769625720905608e-23

pc = 3.08568e+18

v = 'GeV / cm3'

1.5.7 dmsky.utils.wcs module

Interface with wcs.

Adapted from fermipy.skymap

dmsky.utils.wcs.**create_image_hdu**(*data*, *wcs*, *name=None*)
Create a *astropy.io.fits.ImageHDU* object

dmsky.utils.wcs.**create_image_wcs**(*skydir*, *cdelt*, *npix*, *coordsys='CEL'*, *projection='AIT'*)
Create a blank image and associated WCS object

dmsky.utils.wcs.**create_wcs**(*skydir*, *coordsys='CEL'*, *projection='AIT'*, *cdelt=1.0*, *crpix=1.0*, *naxis=2*, *energies=None*)
Create a WCS object. :param skydir: Sky coordinate of the WCS reference point. :type skydir: *~astropy.coordinates.SkyCoord* :param coordsys: :type coordsys: str :param projection: :type projection: str :param cdelt: :type cdelt: float :param crpix: In the first case the same value is used for x and y axes :type crpix: float or (float,float) :param naxis: Number of dimensions of the projection. :type naxis: {2, 3} :param energies: Array of energies that defines the third dimension if naxis=3. :type energies: array-like

dmsky.utils.wcs.**get_pixel_skydirs**(*npix*, *wcs*)
Get a list of sky coordinates for the centers of every pixel.

dmsky.utils.wcs.**write_image_hdu**(*filename*, *data*, *wcs*, *name=None*, *clobber=False*)
Write an image to a file

1.5.8 Module contents

1.6 dmsky.file_io subpackage

1.6.1 dmsky.file_io.table module

Module that handles file IO using *astropy.table.Table* class.

dmsky.file_io.table.**columns_from_derived**(*name*, *prop*)
Build a set of *Column* objects for a *Derived* object

In this case we only build a single column

Parameters

- **name** (*str*) – Name of the Column we will build
- **prop** (*dmsky.Property*) – Property object we are building the column for

Returns **cols** – A list of *astropy.table.Column*

Return type list

dmsky.file_io.table.**columns_from_parameter**(*name*, *prop*)
Build a set of *astropy.table.Column* objects for a *dmsky.Parameter* object

In this case we build a several columns

Parameters

- **name** (*str*) – Base of the names of the Column we will build
- **prop** (*dmsky.Property*) – Property object we are building the column for

Returns **cols** – A list of *astropy.table.Column*

Return type list

dmsky.file_io.table.**columns_from_property**(*name*, *prop*)
Build some of *astropy.table.Column* objects for a *dmsky.Property* object

In this case we only build a single column

Parameters

- **name** (*str*) – Name of the Column we will build
- **prop** (*dmsky.Property*) – Property object we are building the column for

Returns **cols** – A list of *astropy.table.Column*

Return type list

dmsky.file_io.table.**fill_table_from_targetlist**(*tab*, *targetList*)
Fill a table for a set of *dmsky.Target* objects

Parameters

- **tab** (*astropy.table.Table*) – The table we are filling
- **targetList** (*list*) – List of 'dmsky.Target' object used to fill the table

dmsky.file_io.table.**get_column_kwargs**(*name*, *prop*)
Get keyword arguments needed to build a column for a *dmsky.Property* object

Parameters

- **name** (*str*) – Name of the Column we will build
- **prop** (*dmsky.Property*) – Property object we are building the column for

Returns **opt_keys** – A dictionary we can use to constuct a *astropy.table.Column*

Return type dict

dmsky.file_io.table.**get_row_values**(*model*, *keys*)
Get the values needed to fill a row in the table

Parameters

- **model** (*dmsky.Model*) – Model that we are getting the data from
- **keys** (*list*) – Names of the properties we are reading

Returns **vals** – A dictionary we can use to fill an *astropy.table.Column* row

Return type dict

dmsky.file_io.table.**make_columns_for_model**(*names*, *model*)
Make a set of *Column* objects needed to describe a *Model* object.

Parameters

- **names** (*list*) – List of the names of the properties to convert
- **model** (*dmsky.Model*) – Model that we are getting the data from

Returns **clist** – A list of *astropy.table.Column*

Return type list

dmsky.file_io.table.**make_columns_for_prop**(*name*, *prop*)
Generic function to make a set of *astropy.table.Column* objects for any *dmsky.Property* sub-class

Parameters

- **name** (*str*) – Name of the Column we will build
- **prop** (*dmsky.Property*) – Property object we are building the column for

Returns **cols** – A list of *astropy.table.Column*

Return type list

dmsky.file_io.table.**make_table_for_roster**(*colNames*, *roster*)
Make a table for a *Roster* object

Parameters

- **colNames** (*list*) – The names of the properties to include in the table
- **roster** (*dmsky.Roster*) – Roster used to fill the table

Returns **table** – A table with the data from that Roster

Return type *astropy.table.Table*

dmsky.file_io.table.**make_table_for_targetlist**(*colNames*, *targetList*)
Build a table for a set of *Target* objects

Parameters

- **colNames** (*list*) – The names of the properties to include in the table
- **targetList** (*list*) – List of 'dmsky.Target' object used to fill the table

Returns **table** – A table with the data from those targets

Return type *astropy.table.Table*

dmsky.file_io.table.**make_target_from_row**(*row*)
Make a *Target* object from a *Table* row

dmsky.file_io.table.**row_to_dict**(*row*)
Convert a *Table* row into a dict

1.6.2 Module contents

File IO for dmsky package

CHAPTER 2

Indices and tables

Python Module Index

d

Index

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z