
Fermipy Documentation

Release 0.13.5+dirty

Matthew Wood

March 10, 2017

1	Introduction	1
1.1	Getting Help	1
1.2	Documentation Contents	1
1.2.1	Installation	1
1.2.2	Quickstart Guide	7
1.2.3	Configuration	12
1.2.4	Output File	23
1.2.5	ROI Optimization and Fitting	26
1.2.6	Customizing the Model	28
1.2.7	Advanced Analysis Methods	31
1.2.8	fermipy package	53
1.2.9	fermipy.jobs subpackage	106
1.2.10	fermipy.diffuse subpackage	124
1.2.11	Changelog	145
2	Indices and tables	149
	Python Module Index	151

Introduction

This is the Fermipy documentation page. Fermipy is a python package that facilitates analysis of data from the Large Area Telescope (LAT) with the [Fermi Science Tools](#). For more information about the Fermi mission and the LAT instrument please refer to the [Fermi Science Support Center](#).

The Fermipy package is built on the pyLikelihood interface of the Fermi Science Tools and provides a set of high-level tools for performing common analysis tasks:

- Data and model preparation with the gt-tools (gtselect, gtmktime, etc.).
- Extracting a spectral energy distribution (SED) of a source.
- Generating TS and residual maps for a region of interest.
- Finding new source candidates.
- Localizing a source or fitting its spatial extension.

Fermipy uses a configuration-file driven workflow in which the analysis parameters (data selection, IRFs, and ROI model) are defined in a YAML configuration file. Analysis is executed through a python script that calls the methods of *GTAnalysis* to perform different analysis operations.

For instructions on installing Fermipy see the [Installation](#) page. For a short introduction to using Fermipy see the [Quickstart Guide](#).

Getting Help

If you have questions about using Fermipy please open a [GitHub Issue](#) or email the [Fermipy developers](#).

Documentation Contents

Installation

Note: Fermipy is only compatible with Science Tools v10r0p5 or later. If you are using an earlier version, you will need to download and install the latest version from the [FSSC](#). Note that it is recommended to use the *non-ROOT* binary distributions of the Science Tools.

These instructions assume that you already have a local installation of the Fermi Science Tools (STs). For more information about installing and setting up the STs see [Installing the Fermi Science Tools](#). If you are running at

SLAC you can follow the [Running at SLAC](#) instructions. For Unix/Linux users we currently recommend following the [Installing with Anaconda Python](#) instructions. For OSX users we recommend following the [Installing with pip](#) instructions. The [Installing with Docker](#) instructions can be used to install the STs on both OSX and Linux machines that are new enough to support Docker.

Installing the Fermi Science Tools

The Fermi STs are a prerequisite for fermipy. To install the STs we recommend using one of the non-ROOT binary distributions available from the [FSSC](#). The following example illustrates how to install the binary distribution on a Linux machine running Ubuntu Trusty:

```
$ curl -OL http://fermi.gsfc.nasa.gov/ssc/data/analysis/software/tar/ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.19-10-without-rootA.tar.gz
$ tar xzf ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.19-10-without-rootA.tar.gz
$ export FERMI_DIR=ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.19-10-without-rootA
$ source $FERMI_DIR/fermi-init.sh
```

More information about installing the STs as well as the complete list of the available binary distributions is available on the [FSSC software page](#).

Installing with pip

These instructions cover installation with the `pip` package management tool. This method will install fermipy and its dependencies into the python distribution that comes with the Fermi Science Tools. First verify that you're running the python from the Science Tools

```
$ which python
```

If this doesn't point to the python in your Science Tools install (i.e. it returns `/usr/bin/python` or `/usr/local/bin/python`) then the Science Tools are not properly setup.

Before starting the installation process, you will need to determine whether you have `setuptools` and `pip` installed in your local python environment. You may need to install these packages if you are running with the binary version of the Fermi Science Tools distributed by the FSSC. The following command will install both packages in your local 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 the Science Tools, something went wrong. Now install fermipy by running

```
$ pip install fermipy
```

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

```
$ pip install jupyter
```

Finally, check that fermipy 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.
>>> from fermipy.gtanalysis import GTAnalysis
>>> help(GTAnalysis)
```

Installing with Anaconda Python

Note: The following instructions have only been verified to work with binary Linux distributions of the Fermi STs. If you are using OSX or you have installed the STs from source you should follow the [Installing with pip](#) thread above.

These instructions cover how to use fermipy with a new or existing anaconda python installation. These instructions assume that you have already downloaded and installed the Fermi STs from the FSSC and you have set the FERMI_DIR environment variable to point to the location of this installation.

If you already have an existing anaconda python installation then fermipy can be installed from the conda-forge channel as follows:

```
$ conda config --append channels conda-forge
$ conda install fermipy
```

If you do not have an anaconda installation, the `condainstall.sh` script can be used to create a minimal anaconda installation from scratch. First download and source the `condainstall.sh` script from the fermipy repository:

```
$ curl -OL https://raw.githubusercontent.com/fermiPy/fermipy/master/condainstall.sh
$ source condainstall.sh
```

If you do not already have anaconda python installed on your system this script will create a new installation under `$HOME/miniconda`. If you already have anaconda installed and the `conda` command is in your path the script will use your existing installation. After running `condainstall.sh` fermipy can be installed with conda:

```
$ conda install fermipy
```

Alternatively fermipy can be installed from source following the instructions in [Building from Source](#).

Once fermipy is installed you can initialize the ST/fermipy environment by running `condasetup.sh`:

```
$ curl -OL https://raw.githubusercontent.com/fermiPy/fermipy/master/condasetup.sh
$ source condasetup.sh
```

If you installed fermipy in a specific conda environment you should switch to this environment before running the script:

```
$ source activate fermi-env
$ source condasetup.sh
```

Installing with Docker

Note: This method for installing the STs is currently experimental and has not been fully tested on all operating systems. If you encounter issues please try either the pip- or anaconda-based installation instructions.

Docker is a virtualization tool that can be used to deploy software in portable containers that can be run on any operating system that supports Docker. Before following these instruction you should first install docker on your machine following the [installation instructions](#) for your operating system. Docker is currently supported on the following operating systems:

- macOS 10.10.3 Yosemite or later
- Ubuntu Precise 12.04 or later
- Debian 8.0 or later

- RHEL7 or later
- Windows 10 or later

Note that Docker is not supported by RHEL6 or its variants (CentOS6, Scientific Linux 6).

These instructions describe how to create a docker-based ST installation that comes preinstalled with anaconda python and fermipy. The installation is fully contained in a docker image that is roughly 2GB in size. To see a list of the available images go to the [fermipy Docker Hub page](#). Images are tagged with the release version of the STs that was used to build the image (e.g. 11-05-00). The *latest* tag points to the image corresponding to the most recent ST release.

To install the *latest* image first download the image file:

```
$ docker pull fermipy/fermipy
$ docker tag fermipy/fermipy fermipy
```

This will create an image called *fermipy*. Now change to the directory where you plan to do your analysis and run the following command to launch a docker container instance:

```
$ docker run -it --rm -p 8888:8888 -v $PWD:/workdir -w /workdir fermipy
```

This will start an ipython notebook server that will be attached to port 8888. Once the server is running you can start a notebook session by navigating to the URL <http://localhost:8888/>. The `-v $PWD:/workdir` argument mounts the current directory to the working area of the container. Additional directories may be mounted by adding more volume arguments `-v` with host and container paths separated by a colon.

The same docker image may be used to launch python, ipython, or a bash shell by passing the command as an argument to `docker run`:

```
$ docker run -it --rm -v $PWD:/workdir -w /workdir fermipy ipython
$ docker run -it --rm -v $PWD:/workdir -w /workdir fermipy python
$ docker run -it --rm -v $PWD:/workdir -w /workdir fermipy /bin/bash
```

By default interactive graphics will not be enabled. The following code can be inserted at the top of your analysis script to fall-back to a non-interactive backend:

```
from fermipy.utils import init_matplotlib_backend
init_matplotlib_backend()
```

The following commands can be used to enable X11 forwarding for interactive graphics on an OSX machine. This requires you to have installed XQuartz 2.7.10 or later. First enable remote connections by default and start the X server:

```
$ defaults write org.macosforge.xquartz.X11 nolisten_tcp -boolean false
$ open -a XQuartz
```

Now check that the X server is running and listening on port 6000:

```
$ lsof -i :6000
```

If you don't see X11 listening on port 6000 then try restarting XQuartz.

Once you have XQuartz configured you can enable forwarding by setting `DISPLAY` environment variable to the IP address of the host machine:

```
$ export HOST_IP=`ifconfig en0 | grep "inet " | cut -d " " -f2`
$ xhost +local:
$ docker run -it --rm -e DISPLAY=$HOST_IP:0 -v $PWD:/workdir -w /workdir fermipy ipython
```

Running at SLAC

This section provides specific installation instructions for running in the SLAC computing environment. First download and source the `slacsetup.sh` script:

```
$ wget https://raw.githubusercontent.com/fermiPy/fermipy/master/slacsetup.sh -O slacsetup.sh
$ source slacsetup.sh
```

To initialize the ST environment run the `slacsetup` function:

```
$ slacsetup
```

This will setup your `GLAST_EXT` path and source the setup script for one of the pre-built ST installations (the current default is 11-05-00). To manually override the ST version you can provide the release tag as an argument to `slacsetup`:

```
$ slacsetup XX-XX-XX
```

Because users don't have write access to the ST python installation all pip commands that install or uninstall packages must be executed with the `--user` flag. After initializing the STs environment, install `fermipy` with pip:

```
$ pip install fermipy --user
```

This will install `fermipy` in `$HOME/.local`. You can verify that the installation has succeeded by importing `GTAnalysis`:

```
$ python
Python 2.7.8 |Anaconda 2.1.0 (64-bit)| (default, Aug 21 2014, 18:22:21)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>> from fermipy.gtanalysis import GTAnalysis
```

Upgrading

By default installing `fermipy` with `pip` or `conda` will get the latest tagged released available on the [PyPi](#) package repository. You can check your currently installed version of `fermipy` with `pip show`:

```
$ pip show fermipy
```

or `conda info`:

```
$ conda info fermipy
```

To upgrade your `fermipy` 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 fermipy --upgrade --no-deps
Collecting fermipy
Installing collected packages: fermipy
  Found existing installation: fermipy 0.6.6
    Uninstalling fermipy-0.6.6:
      Successfully uninstalled fermipy-0.6.6
Successfully installed fermipy-0.6.7
```

If you installed `fermipy` with `conda` the equivalent command is:

```
$ conda update fermipy
```

Building from Source

These instructions describe how to install fermipy 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 fermipy source code or test features in an untagged commit. Note that non-expert users are recommended to install a tagged release of fermipy following the [Installing with pip](#) or [Installing with Anaconda Python](#) instructions above.

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

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

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 fermipy 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
```

You also have the option of installing a previous release tag. To see the list of release tags run `git tag`. To install a specific release tag, run `git checkout` with the tag name followed by `setup.py install`:

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

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 fermipy. To get around this issue you can switch to the Agg backend at runtime before importing fermipy:

```
>>> 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.

In some cases the `setup.py` script will fail to properly install the fermipy package dependencies. If installation fails you can try running a forced upgrade of these packages with `pip install --upgrade`:

```
$ pip install --upgrade --user numpy matplotlib scipy astropy pyyaml healpy wcsaxes ipython jupyter
```

Quickstart Guide

This page walks through the steps to setup and perform a basic spectral analysis of a source. For additional fermipy tutorials see the *IPython Notebook Tutorials*. To more easily follow along with this example a directory containing pre-generated input files (FT1, source maps, etc.) is available from the following link:

```
$ curl -OL https://raw.githubusercontent.com/fermiPy/fermipy-extras/master/data/mkn421.tar.gz
$ tar xzf mkn421.tar.gz
$ cd mkn421
```

Creating a Configuration File

The first step is to compose a configuration file that defines the data selection and analysis parameters. Complete documentation on the configuration file and available options is given in the *Configuration* page. fermiPy uses the **YAML format** for its configuration files. The configuration file has a hierarchical organization that groups related parameters into separate dictionaries. In this example we will compose a configuration file for a SOURCE-class analysis of Markarian 421 with FRONT+BACK event types (`evtype=3`):

```
data:
  evfile : ft1.lst
  scfile : ft2.fits
  ltcube : ltcube.fits

binning:
  roiwidth : 10.0
  binsz : 0.1
  binsperdec : 8

selection :
  emin : 100
  emax : 316227.76
  zmax : 90
  evclass : 128
  evtype : 3
  tmin : 239557414
  tmax : 428903014
  filter : null
  target : 'mkn421'

gtlike:
  edisp : True
  irfs : 'P8R2_SOURCE_V6'
  edisp_disable : ['isodiff', 'galdiff']

model:
  src_roiwidth : 15.0
  galdiff : '$FERMI_DIFFUSE_DIR/gll_iem_v06.fits'
```

```
isodiff : 'iso_P8R2_SOURCE_V6_v06.txt'
catalogs : ['3FGL']
```

The *data* section defines the input data set and spacecraft file for the analysis. Here *evfile* points to a list of FIT files that encompass the chosen ROI, energy range, and time selection. The parameters in the *binning* section define the dimensions of the ROI and the spatial and energy bin size. The *selection* section defines parameters related to the data selection (energy range, zmax cut, and event class/type). The *target* parameter in this section defines the ROI center to have the same coordinates as the given source. The *model* section defines parameters related to the ROI model definition (diffuse templates, point sources).

Fermipy gives the user the option to combine multiple data selections into a joint likelihood with the *components* section. The components section contains a list of dictionaries with the same hierarchy as the root analysis configuration. Each element of the list defines the analysis parameters for an independent sub-selection of the data. Any parameters not defined within the component dictionary default to the value defined in the root configuration. The following example shows the *components* section that could be appended to the previous configuration to define a joint analysis with four PSF event types:

```
components:
- { selection : { evtype : 4 } } # PSF0
- { selection : { evtype : 8 } } # PSF1
- { selection : { evtype : 16 } } # PSF2
- { selection : { evtype : 32 } } # PSF3
```

Any configuration parameter can be changed with this mechanism. The following example is a configuration in which a different zmax selection and isotropic template is used for each of the four PSF event types:

```
components:
- model: {isodiff: isotropic_source_psf0_4years_P8V3.txt}
  selection: {evtype: 4, zmax: 70}
- model: {isodiff: isotropic_source_psf1_4years_P8V3.txt}
  selection: {evtype: 8, zmax: 75}
- model: {isodiff: isotropic_source_psf2_4years_P8V3.txt}
  selection: {evtype: 16, zmax: 85}
- model: {isodiff: isotropic_source_psf3_4years_P8V3.txt}
  selection: {evtype: 32, zmax: 90}
```

Creating an Analysis Script

Once the configuration file has been composed, the analysis is executed by creating an instance of *GTAnalysis* with the configuration file as its argument and calling its analysis methods. *GTAnalysis* serves as a wrapper over the underlying *pyLikelihood* classes and provides methods to fix/free parameters, add/remove sources from the model, and perform a fit to the ROI. For a complete documentation of the available methods you can refer to the *fermipy package* page.

In the following python examples we show how to initialize and run a basic analysis of a source. First we instantiate a *GTAnalysis* object with the path to the configuration file and run *setup()*.

```
from fermipy.gtanalysis import GTAnalysis

gta = GTAnalysis('config.yaml', logging={'verbosity' : 3})
gta.setup()
```

The *setup()* method performs the data preparation and response calculations needed for the analysis (selecting the data, creating counts and exposure maps, etc.). Depending on the data selection and binning of the analysis this will often be the slowest step in the analysis sequence. The output of *setup()* is cached in the analysis working directory so subsequent calls to *setup()* will run much faster.

Before running any other analysis methods it is recommended to first run `optimize()`:

```
gta.optimize()
```

This will loop over all model components in the ROI and fit their normalization and spectral shape parameters. This method also computes the TS of all sources which can be useful for identifying weak sources that could be fixed or removed from the model. We can check the results of the optimization step by calling `print_roi()`:

```
gta.print_roi()
```

By default all models parameters are initially fixed. The `free_source()` and `free_sources()` methods can be used to free or fix parameters of the model. In the following example we free the normalization of catalog sources within 3 deg of the ROI center and free the galactic and isotropic components by name.

```
# Free Normalization of all Sources within 3 deg of ROI center
gta.free_sources(distance=3.0,pars='norm')

# Free all parameters of isotropic and galactic diffuse components
gta.free_source('galdiff')
gta.free_source('isodiff')
```

The `minmax_ts` and `minmax_npred` arguments to `free_sources()` can be used to free or fixed sources on the basis of their current TS or Npred values:

```
# Free sources with TS > 10
gta.free_sources(minmax_ts=[10,None],pars='norm')

# Fix sources with TS < 10
gta.free_sources(minmax_ts=[None,10],free=False,pars='norm')

# Fix sources with 10 < Npred < 100
gta.free_sources(minmax_npred=[10,100],free=False,pars='norm')
```

When passing a source name argument both case and whitespace are ignored. When using a FITS catalog file a source can also be referred to by any of its associations. When using the 3FGL catalog, the following calls are equivalent ways of freeing the parameters of Mkn 421:

```
# These calls are equivalent
gta.free_source('mkn421')
gta.free_source('Mkn 421')
gta.free_source('3FGL J1104.4+3812')
gta.free_source('3fglj1104.4+3812')
```

After freeing parameters of the model we can execute a fit by calling `fit()`. This will maximize the likelihood with respect to the model parameters that are currently free.

```
gta.fit()
```

After the fitting is complete we can write the current state of the model with `write_roi()`:

```
gta.write_roi('fit_model')
```

This will write several output files including an XML model file and an ROI dictionary file. The names of all output files will be prepended with the `prefix` argument to `write_roi()`.

Once we have optimized our model for the ROI we can use the `residmap()` and `tsmap()` methods to assess the fit quality and look for new sources.

```
# Dictionary defining the spatial/spectral parameters of the test source
model = {'SpatialModel' : 'PointSource', 'Index' : 2.0,
```

```
'SpectrumType' : 'PowerLaw'}

# Both methods return a dictionary with the maps
m0 = gta.residmap('fit_model', model=model, make_plots=True)
m1 = gta.tsmmap('fit_model', model=model, make_plots=True)
```

More documentation on these methods is available in the [TS Map](#) and [Residual Map](#) pages.

By default, calls to `fit()` will execute a global spectral fit over the entire energy range of the analysis. To extract a bin-by-bin flux spectrum (i.e. a SED) you can call `sed()` method with the name of the source:

```
gta.sed('mkn421', make_plots=True)
```

More information about `sed()` method can be found in the [SED Analysis](#) page.

Extracting Analysis Results

Results of the analysis can be extracted from the dictionary file written by `write_roi()`. This method writes information about the current state of the analysis to a python dictionary. More documentation on the contents of the output file are available in the [Output File](#) page.

By default the output dictionary is written to a file in the [numpy format](#) and can be loaded from a python session after your analysis is complete. The following demonstrates how to load the analysis dictionary that was written to `fit_model.npy` in the Mkn421 analysis example:

```
>>> # Load analysis dictionary from a npy file
>>> import np
>>> c = np.load('fit_model.npy').flat[0]
>>> print(c.keys())
['roi', 'config', 'sources', 'version']
```

The output dictionary contains the following top-level elements:

Table 1.1: File Dictionary

Key	Description	
roi	dict	A dictionary containing information about the ROI as a whole.
sources	dict	A dictionary containing information for individual sources in the model (diffuse and point-like). Each element of this dictionary maps to a single source in the ROI model.
config	dict	The configuration dictionary of the GTAnalysis instance.
version	str	The version of the fermiPy package that was used to run the analysis. This is automatically generated from the git release tag.

Each source dictionary collects the properties of the given source (TS, NPred, best-fit parameters, etc.) computed up to that point in the analysis.

```
>>> print(c['sources'].keys())
['3FGL J1032.7+3735',
 '3FGL J1033.2+4116',
 ...
 '3FGL J1145.8+4425',
 'galdiff',
 'isodiff']
>>> print(c['sources']['3FGL J1104.4+3812']['ts'])
87455.9709683
```

```
>>> print c['sources']['3FGL J1104.4+3812']['npred']
31583.7166495
```

Information about individual sources in the ROI is also saved to a catalog FITS file with the same string prefix as the dictionary file. This file can be loaded with the `astropy.io.fits` or `astropy.table.Table` interface:

```
>>> # Load the source catalog file
>>> from astropy.table import Table
>>> tab = Table.read('fit_model.fits')
>>> print(tab[['name', 'class', 'ts', 'npred', 'flux']])
```

name	class	ts	npred	flux [2] 1 / (cm ² s)
3FGL J1104.4+3812	BLL	87455.9709683	31583.7166495	2.20746290445e-07 .. 1.67062058528e-09
3FGL J1109.6+3734	bll	42.34511826	93.7971922425	5.90635786943e-10 .. 3.6620894143e-10
...				
3FGL J1136.4+3405	fsrq	4.78089819776	261.427034151	1.86805869704e-08 .. 8.62638727067e-09
3FGL J1145.8+4425	fsrq	3.78006883967	237.525501441	7.25611442299e-08 .. 3.77056557247e-08

The FITS file contains columns for all scalar and vector elements of the source dictionary. Spectral fit parameters are contained in the `param_names`, `param_values`, and `param_errors` columns:

```
>>> print(tab[['param_names', 'param_values', 'param_errors']][0])
<Row 0 of table
  values=(['Prefactor', 'Index', 'Scale', '', '', ''],
          [2.1301351784512767e-11, -1.7716399431228638, 1187.1300048828125, nan, nan, nan],
          [1.6126233510314277e-13, nan, nan, nan, nan, nan])
  dtype=[('param_names', 'S32', (6,)),
         ('param_values', '>f8', (6,)),
         ('param_errors', '>f8', (6,))]>
```

Reloading from a Previous State

One can reload an analysis instance that was saved with `write_roi()` by calling either the `create()` or `load_roi()` methods. The `create()` method can be used to construct an entirely new instance of `GTAnalysis` from a previously saved results file:

```
from fermipy.gtanalysis import GTAnalysis
gta = GTAnalysis.create('fit_model.npy')

# Continue running analysis starting from the previously saved
# state
gta.fit()
```

where the argument is the path to an output file produced with `write_roi()`. This function will instantiate a new analysis object, run the `setup()` method, and load the state of the model parameters at the time that `write_roi()` was called.

The `load_roi()` method can be used to reload a previous state of the analysis to an existing instance of `GTAnalysis`.

```
from fermipy.gtanalysis import GTAnalysis

gta = GTAnalysis('config.yaml')
gta.setup()

gta.write_roi('prefit_model')
```

```
# Fit a source
gta.free_source('mkn421')
gta.fit()

# Restore the analysis to its prior state before the fit of mkn421
# was executed
gta.load_roi('prefit_model')
```

Using `load_roi()` is generally faster than `create()` when an analysis instance already exists.

IPython Notebook Tutorials

Additional tutorials with more detailed examples are available as IPython notebooks in the [notebooks](#) directory of the [fermipy-extra](#) repository. These notebooks can be browsed as [static web pages](#) or run interactively by downloading the [fermipy-extra](#) repository and running `jupyter notebook` in the `notebooks` directory:

```
$ git clone https://github.com/fermiPy/fermipy-extra.git
$ cd fermipy-extra/notebooks
$ jupyter notebook index.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
```

Configuration

This page describes the configuration management scheme used within the Fermipy package and the documents the configuration parameters that can be set in the configuration file.

Class Configuration

Classes in the Fermipy package follow a common convention for configuring the runtime behavior of a class instance. Internally every class instance has a dictionary that defines its configuration state. Elements of the configuration dictionary can be scalars (str, int, float) or dictionaries defining nested groups of parameters.

The class configuration dictionary is initialized at the time of object creation by passing a dictionary or a path to a YAML configuration file to the class constructor. Keyword arguments can be passed to the constructor to override configuration parameters in the input dictionary. For instance in the following example the `config` dictionary defines values for the parameters `emin` and `emax`. By passing a dictionary for the `selection` keyword argument, the value of `emax` in the keyword argument (10000) overrides the value of this parameter in the input dictionary.

```
config = {
    'selection' : { 'emin' : 100,
                   'emax' : 1000 }
}

gta = GTAnalysis(config, selection={'emax' : 10000})
```

The first argument can also be the path to a YAML configuration file rather than a dictionary:

```
gta = GTAnalysis('config.yaml',selection={'emax' : 10000})
```

Configuration File

Fermipy uses YAML files to read and write its configuration in a persistent format. The configuration file has a hierarchical organization that groups parameters into dictionaries that are keyed to a section name (*data*, *binning*, etc.).

Listing 1.1: Sample Configuration

```
data:
  evfile : ft1.lst
  scfile : ft2.fits
  ltfile : ltcube.fits

binning:
  roiwidth : 10.0
  binsz : 0.1
  binsperdec : 8

selection :
  emin : 100
  emax : 316227.76
  zmax : 90
  evclass : 128
  evtype : 3
  tmin : 239557414
  tmax : 428903014
  filter : null
  target : 'mkn421'

gtlike:
  edisp : True
  irfs : 'P8R2_SOURCE_V6'
  edisp_disable : ['isodiff', 'galdiff']

model:
  src_roiwidth : 15.0
  galdiff : '$FERMI_DIFFUSE_DIR/gll_iem_v06.fits'
  isodiff : 'iso_P8R2_SOURCE_V6_v06.txt'
  catalogs : ['3FGL']
```

The configuration file mirrors the layout of the configuration dictionary. Most of the configuration parameters are optional and if not set explicitly in the configuration file will be set to a default value. The parameters that can be set in each section are described below.

binning

Options in the *binning* section control the spatial and spectral binning of the data.

Listing 1.2: Sample *binning* Configuration

```
binning:

  # Binning
  roiwidth : 10.0
```

```

npix      : null
binsz     : 0.1 # spatial bin size in deg
binsperdec : 8  # nb energy bins per decade
projtype  : WCS

```

Listing 1.3: *binning* Options

Option	De- fault	Description
<code>binsperdec</code>	8	Number of energy bins per decade.
<code>binsz</code>	0.1	Spatial bin size in degrees.
<code>coordsys</code>	CEL	Coordinate system of the spatial projection (CEL or GAL).
<code>enumbins</code>	None	Number of energy bins. If none this will be inferred from energy range and <code>binsperdec</code> parameter.
<code>hpx_ebin</code>	True	Include energy binning
<code>hpx_order</code>	10	Order of the map (int between 0 and 12, included)
<code>hpx_ordering_scheme</code>	RING	HEALPix Ordering Scheme
<code>npix</code>	None	Number of pixels. If none then this will be set from <code>roiwidth</code> and <code>binsz</code> .
<code>proj</code>	AIT	Spatial projection for WCS mode.
<code>projtype</code>	WCS	Projection mode (WCS or HPX).
<code>roiwidth</code>	10.0	Width of the ROI in degrees. The number of pixels in each spatial dimension will be set from <code>roiwidth/binsz</code> (rounded up).

components

The *components* section can be used to define analysis configurations for a sequence of independent subselections of the data. Each subselection will have its own binned likelihood instance that will be combined in a global likelihood function for the whole ROI (implemented with the `SummedLikelihood` class in `pyLikelihood`). This section is optional and when this section is empty (the default) `fermiPy` will construct a single likelihood with the parameters of the root analysis configuration.

The component section is defined as a list of dictionaries where each element sets analysis parameters for a different subcomponent of the analysis. Dictionary elements have the same hierarchy of parameters as the root analysis configuration. Parameters not defined in a given element will default to the values set in the root analysis configuration.

The following example illustrates how to define a Front/Back analysis with the a list of dictionaries. Files associated to each component will be given a suffix according to their order in the list (e.g. `file_00.fits`, `file_01.fits`, etc.).

```

# Component section for Front/Back analysis
- { selection : { evtype : 1 } } # Front
- { selection : { evtype : 2 } } # Back

```

data

The *data* section defines the input data files for the analysis (FT1, FT2, and livetime cube). `evfile` and `scfile` can either be individual files or group of files. The optional `ltcube` option can be used to choose a pre-generated livetime cube. If `ltcube` is null a livetime cube will be generated at runtime with `gtltcube`.

Listing 1.4: Sample *data* Configuration

```

data :
  evfile : ft1.lst

```

```
scfile : ft2.fits
ltcube : null
```

Listing 1.5: *data* Options

Option	Default	Description
cacheft1	True	Cache FT1 files when performing binned analysis. If false then only the counts cube is retained.
evfile	None	Path to FT1 file or list of FT1 files.
ltcube	None	Path to livetime cube. If none a livetime cube will be generated with <code>gtmktime</code> .
scfile	None	Path to FT2 (spacecraft) file.

extension

The options in *extension* control the default behavior of the *extension* method. For more information about using this method see the [Extension Fitting](#) page.

Listing 1.6: *extension* Options

Option	Default	Description
fit_position	False	Perform a simultaneous fit to the source position and extension.
free_background	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
free_radius	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
make_plots	False	Generate diagnostic plots.
psf_scale_factor	None	Tuple of two vectors (logE,f) defining an energy-dependent PSF scaling function that will be applied when building spatial models for the source of interest. The tuple (logE,f) defines the fractional corrections f at the sequence of energies logE = log10(E/MeV) where f=0 corresponds to no correction. The correction function f(E) is evaluated by linearly interpolating the fractional correction factors f in log(E). The corrected PSF is given by $P'(x;E) = P(x/(1+f(E));E)$ where x is the angular separation.
save_model	False	Save model counts cubes for the best-fit model of extension.
spatial_model	Radial-Gaussian	Spatial model that will be used to test the source extension. The spatial scale parameter of the model will be set such that the 68% containment radius of the model is equal to the width parameter.
sqrt_ts_threshold	None	Threshold on sqrt(TS_ext) that will be applied when update is True. If None then no threshold is applied.
update	False	Update this source with the best-fit model for spatial extension if TS_ext > tsext_threshold.
width	None	Sequence of values in degrees for the likelihood scan over spatial extension (68% containment radius). If this argument is None then the scan points will be determined from width_min/width_max/width_nstep.
width_max	1.0	Maximum value in degrees for the likelihood scan over spatial extent.
width_min	0.01	Minimum value in degrees for the likelihood scan over spatial extent.
width_nstep	21	Number of scan points between width_min and width_max. Scan points will be spaced evenly on a logarithmic scale between width_min and width_max.
write_fits	True	Write the output to a FITS file.
write_numpy	True	Write the output dictionary to a numpy file.

fileio

The *fileio* section collects options related to file bookkeeping. The `outdir` option sets the root directory of the analysis instance where all output files will be written. If `outdir` is null then the output directory will be automatically set to the directory in which the configuration file is located. Enabling the `usescratch` option will stage all output data files to a temporary scratch directory created under `scratchdir`.

Listing 1.7: Sample *fileio* Configuration

```
fileio:
  outdir : null
  logfile : null
  usescratch : False
  scratchdir : '/scratch'
```

Listing 1.8: *fileio* Options

Option	Default	Description
logfile	None	Path to log file. If None then log will be written to fermipy.log.
outdir	None	Path of the output directory. If none this will default to the directory containing the configuration file.
outdir_regex	'^\\.[a-z]\$\\.[a-z]\$\\.fit\$\\.xml\$\\.npz\$\\.png\$\\.pdf\$\\.yaml\$'	Stage files to the output directory that match at least one of the regular expressions in this list. This option only takes effect when <code>usescratch</code> is True.
savefits	True	Save intermediate FITS files.
scratchdir	/scratch	Path to the scratch directory. If <code>usescratch</code> is True then a temporary working directory will be created under this directory.
usescratch	False	Run analysis in a temporary working directory under <code>scratchdir</code> .
workdir	None	Path to the working directory.
workdir_regex	'^\\.[a-z]\$\\.[a-z]\$\\.fit\$\\.xml\$\\.npz\$'	Stage files to the working directory that match at least one of the regular expressions in this list. This option only takes effect when <code>usescratch</code> is True.

gtlike

Options in the *gtlike* section control the setup of the likelihood analysis include the IRF name (`irfs`).

Listing 1.9: *gtlike* Options

Option	De- fault	Description
bexpmap	None	
bexpmap_roi	None	
convolve	True	
edisp	True	Enable the correction for energy dispersion.
edisp_disable	None	Provide a list of sources for which the edisp correction should be disabled.
expscale	None	Exposure correction that is applied to all sources in the analysis component. This correction is superseded by <code>src_expscale</code> if it is defined for a source.
irfs	None	Set the IRF string.
llscan_npts	20	Number of evaluation points to use when performing a likelihood scan.
minbinsz	0.05	Set the minimum bin size used for resampling diffuse maps.
resample	True	
rfactor	2	
src_expscale	None	Dictionary of exposure corrections for individual sources keyed to source name. The exposure for a given source will be scaled by this value. A value of 1.0 corresponds to the nominal exposure.
srcmap	None	Set the source maps file. When defined this file will be used instead of the local source maps file.
use_external_srcmap	False	Use an external precomputed source map file.
wmap	None	Likelihood weights map.

lightcurve

The options in *lightcurve* control the default behavior of the *lightcurve* method. For more information about using this method see the [Light Curves](#) page.

Listing 1.10: *lightcurve* Options

Option	De- fault	Description
binsz	86400.0	Set the lightcurve bin size in seconds.
free_background	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
free_params	None	Set the parameters of the source of interest that will be re-fit in each time bin. If this list is empty then all parameters will be freed.
free_radius	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
free_sources	None	List of sources to be freed. These sources will be added to the list of sources satisfying the <code>free_radius</code> selection.
make_plots	False	Generate diagnostic plots.
nbins	None	Set the number of lightcurve bins. The total time range will be evenly split into this number of time bins.
time_bins	None	Set the lightcurve bin edge sequence in MET. This option takes precedence over <code>binsz</code> and <code>nbins</code> .
use_local_fit	True	Use local fit.
write_fits	True	Write the output to a FITS file.
write_npy	True	Write the output dictionary to a numpy file.

model

The *model* section collects options that control the inclusion of point-source and diffuse components in the model. `galdiff` and `isodiff` set the templates for the Galactic IEM and isotropic diffuse respectively. `catalogs` defines a list of catalogs that will be merged to form a master analysis catalog from which sources will be drawn. Valid entries in this list can be FITS files or XML model files. `sources` can be used to insert additional point-source or extended components beyond those defined in the master catalog. `src_radius` and `src_roiwidth` set the maximum distance from the ROI center at which sources in the master catalog will be included in the ROI model.

Listing 1.11: Sample *model* Configuration

```
model :  
  
    # Diffuse components  
    galdiff : '$FERMI_DIR/refdata/fermi/galdiffuse/gll_iem_v06.fits'  
    isodiff : '$FERMI_DIR/refdata/fermi/galdiffuse/iso_P8R2_SOURCE_V6_v06.txt'  
  
    # List of catalogs to be used in the model.  
    catalogs :  
        - '3FGL'  
        - 'extra_sources.xml'  
  
    sources :  
        - { 'name' : 'SourceA', 'ra' : 60.0, 'dec' : 30.0, 'SpectrumType' : PowerLaw }  
        - { 'name' : 'SourceB', 'ra' : 58.0, 'dec' : 35.0, 'SpectrumType' : PowerLaw }  
  
    # Include catalog sources within this distance from the ROI center  
    src_radius : null  
  
    # Include catalog sources within a box of width roisrc.  
    src_roiwidth : 15.0
```

Listing 1.12: *model* Options

Option	Default	Description
assoc_xmatch	['3FGL', 'None']	Choose a set of association columns on which to cross-match catalogs.
catalogs	None	
diffuse	None	
diffuse_dir	None	
extdir	None	Set a directory that will be searched for extended source FITS templates. Template files in this directory will take precedence over catalog source templates with the same name.
extract_diffuse	False	Extract a copy of all mapcube components centered on the ROI.
galdiff	None	Set the path to one or more galactic IEM mapcubes. A separate component will be generated for each item in this list.
isodiff	None	Set the path to one or more isotropic templates. A separate component will be generated for each item in this list.
limbdiff	None	
merge_sources	True	Merge properties of sources that appear in multiple source catalogs. If merge_sources=false then subsequent sources with the same name will be ignored.
sources	None	
src_radius	None	Radius of circular region in degrees centered on the ROI that selects sources for inclusion in the model. If this parameter is none then no selection is applied. This selection is ORed with the src_roiwidth selection.
src_radius_r	None	Half-width of src_roiwidth selection. This parameter can be used in lieu of src_roiwidth.
src_roiwidth	None	Width of square region in degrees centered on the ROI that selects sources for inclusion in the model. If this parameter is none then no selection is applied. This selection will be ORed with the src_radius selection.

optimizerListing 1.13: *optimizer* Options

Option	Default	Description
init_lambda	0.0001	Initial value of damping parameter for step size calculation when using the NEWTON fitter. A value of zero disables damping.
max_iter	100	Maximum number of iterations for the Newtons method fitter.
min_fit_quality	2	Set the minimum fit quality.
optimizer	MINUIT	Set the optimization algorithm to use when maximizing the likelihood function.
retries	3	Set the number of times to retry the fit when the fit quality is less than min_fit_quality.
tol	0.001	Set the optimizer tolerance.
verbosity	0	

plotting

Listing 1.14: *plotting* Options

Option	Default	Description
catalogs	None	
cmap	magma	Set the colormap for 2D plots.
cmap_resid	RdBu_r	Set the colormap for 2D residual plots.
figsize	[8.0, 6.0]	Set the default figure size.
format	png	
graticule_radii	None	Define a list of radii at which circular graticules will be drawn.
label_ts_threshold	0.0	TS threshold for labeling sources in sky maps. If None then no sources will be labeled.
loge_bounds	None	

residmap

The options in *residmap* control the default behavior of the *residmap* method. For more information about using this method see the [Residual Map](#) page.

Listing 1.15: *residmap* Options

Option	De-fault	Description
exclude	None	List of sources that will be removed from the model when computing the residual map.
loge_bounds	None	Restrict the analysis to an energy range (emin,emax) in log10(E/MeV) that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either emin/emax are None then only an upper/lower bound on the energy range wil be applied.
make_plots	False	Generate diagnostic plots.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
write_fits	True	Write the output to a FITS file.
write_numpy	True	Write the output dictionary to a numpy file.

roiapt

The options in *roiapt* control the default behavior of the *optimize* method. For more information about using this method see the [ROI Optimization and Fitting](#) page.

Listing 1.16: *roiapt* Options

Option	De-fault	Description
max_free_sources	5	Maximum number of sources that will be fit simultaneously in the first optimization step.
npred_frac	0.95	
npred_threshold	1.0	
shape_ts_threshold	25.0	Threshold on source TS used for determining the sources that will be fit in the third optimization step.
skip	None	List of str source names to skip while optimizing.

sed

The options in *sed* control the default behavior of the *sed* method. For more information about using this method see the *SED Analysis* page.

Listing 1.17: *sed* Options

Option	De-fault	Description
<code>bin_index</code>	2.0	Spectral index that will be use when fitting the energy distribution within an energy bin.
<code>cov_scale</code>	3.0	Scale factor that sets the strength of the prior on nuisance parameters that are free. Setting this to None disables the prior.
<code>free_background</code>	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
<code>free_radius</code>	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
<code>make_plots</code>	False	Generate diagnostic plots.
<code>ul_confidence</code>	0.95	Confidence level for flux upper limit.
<code>use_local_index</code>	False	Use a power-law approximation to the shape of the global spectrum in each bin. If this is false then a constant index set to <code>bin_index</code> will be used.
<code>write_fits</code>	True	Write the output to a FITS file.
<code>write_numpy</code>	True	Write the output dictionary to a numpy file.

selection

The *selection* section collects parameters related to the data selection and target definition. The majority of the parameters in this section are arguments to *gtselect* and *gtmktime*. The ROI center can be set with the *target* parameter by providing the name of a source defined in one of the input catalogs (defined in the *model* section). Alternatively the ROI center can be defined by giving explicit sky coordinates with *ra* and *dec* or *glon* and *glat*.

```
selection:

# gtselect parameters
emin      : 100
emax      : 100000
zmax      : 90
evclass   : 128
evtype    : 3
tmin      : 239557414
tmax      : 428903014

# gtmktime parameters
filter    : 'DATA_QUAL>0 && LAT_CONFIG==1'
roicut    : 'no'

# Set the ROI center to the coordinates of this source
target    : 'mkn421'
```

Listing 1.18: *selection* Options

Option	De- fault	Description
convtype	None	Conversion type selection.
dec	None	
emax	None	Maximum Energy (MeV)
emin	None	Minimum Energy (MeV)
evclass	None	Event class selection.
evtype	None	Event type selection.
filter	None	Filter string for gtmktime selection.
glat	None	
glon	None	
logemax	None	Maximum Energy (log10(MeV))
logemin	None	Minimum Energy (log10(MeV))
phasemax	None	Maximum pulsar phase
phasemin	None	Minimum pulsar phase
ra	None	
radius	None	Radius of data selection. If none this will be automatically set from the ROI size.
roicut	no	
target	None	Choose an object on which to center the ROI. This option takes precedence over ra/dec or glon/glat.
tmax	None	Maximum time (MET).
tmin	None	Minimum time (MET).
zmax	None	Maximum zenith angle.

sourcefind

The options in *sourcefind* control the default behavior of the *find_sources* method. For more information about using this method see the *Source Finding* page.

Listing 1.19: *sourcefind* Options

Option	De- fault	Description
free_params	None	
max_iter	5	Maximum number of source finding iterations. The source finder will continue adding sources until no additional peaks are found or the number of iterations exceeds this number.
min_separation	1.0	Minimum separation in degrees between sources detected in each iteration. The source finder will look for the maximum peak in the TS map within a circular region of this radius.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
multithread	False	Split the calculation across all available cores.
sources_per_iter	4	Maximum number of sources that will be added in each iteration. If the number of detected peaks in a given iteration is larger than this number, only the N peaks with the largest TS will be used as seeds for the current iteration.
sqrt_ts_threshold	5.0	Source threshold in sqrt(TS). Only peaks with sqrt(TS) exceeding this threshold will be used as seeds for new sources.
tmap_fitter	tmap	Set the method for generating the TS map. Valid options are tmap or tscube.

tmap

The options in *tmap* control the default behavior of the *tmap* method. For more information about using this method see the [TS Map](#) page.

Listing 1.20: *tmap* Options

Option	De- fault	Description
<code>exclude</code>	None	List of sources that will be removed from the model when computing the TS map.
<code>loge_bounds</code>	None	Restrict the analysis to an energy range (emin,emax) in log10(E/MeV) that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either emin/emax are None then only an upper/lower bound on the energy range will be applied.
<code>make_plots</code>	False	Generate diagnostic plots.
<code>max_kernel_radius</code>	30.0	Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy.
<code>model</code>	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
<code>multithread</code>	False	Split the calculation across all available cores.
<code>write_fits</code>	True	Write the output to a FITS file.
<code>write_npy</code>	True	Write the output dictionary to a numpy file.

tscube

The options in *tscube* control the default behavior of the *tscube* method. For more information about using this method see the [TS Cube](#) page.

Listing 1.21: *tscube* Options

Option	De- fault	Description
<code>cov_scale</code>	-1.0	Scale factor to apply to broadband fitting cov. matrix in bin-by-bin fits (< 0 -> fixed)
<code>cov_scale_bb</code>	-1.0	Scale factor to apply to global fitting cov. matrix in broadband fits. (< 0 -> no prior)
<code>do_sed</code>	True	Compute the energy bin-by-bin fits
<code>init_lambda</code>	0	Initial value of damping parameter for newton step size calculation. A value of zero disables damping.
<code>max_iter</code>	30	Maximum number of iterations for the Newtons method fitter.
<code>model</code>	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
<code>nnorm</code>	10	Number of points in the likelihood v. normalization scan
<code>norm_sigma</code>	5.0	Number of sigma to use for the scan range
<code>remake_test_image</code>	False	If true, recomputes the test source image (otherwise just shifts it)
<code>st_scan_level</code>	0	Level to which to do ST-based fitting (for testing)
<code>tol</code>	0.001	Criteria for fit convergence (estimated vertical distance to min < tol)
<code>tol_type</code>	0	Absolute (0) or relative (1) criteria for convergence.

Output File

The current state of the ROI can be written at any point by calling *write_roi*.

```
>>> gta.write_roi('output.npy')
```

The output file will contain all information about the state of the ROI as calculated up to that point in the analysis including model parameters and measured source characteristics (flux, TS, NPred). An XML model file will also be saved for each analysis component.

The output file can be read with `load`:

```
>>> o = np.load('output.npy').flat[0]
>>> print(o.keys())
['roi', 'config', 'sources', 'version']
```

The output file is organized in four top-level of dictionaries:

Listing 1.22: File Dictionary

Key	Type	Description
roi	dict	A dictionary containing information about the ROI as a whole.
sources	dict	A dictionary containing information for individual sources in the model (diffuse and point-like). Each element of this dictionary maps to a single source in the ROI model.
config	dict	The configuration dictionary of the <i>GTAnalysis</i> instance.
version	str	The version of the fermiPy package that was used to run the analysis. This is automatically generated from the git release tag.

ROI Dictionary

Source Dictionary

The `sources` dictionary contains one element per source keyed to the source name. The following table lists the elements of the source dictionary and their descriptions.

Table 1.2: Source Dictionary

Key	Type	Description
name	str	Name of the source.
Source_Name	str	Name of the source.
SpatialModel	str	Spatial model.
SpatialWidth	float	Spatial size parameter.
SpatialType	str	Spatial type string. This corresponds to the type attribute of the spatialModel component.
SourceType	str	Source type string (PointSource or DiffuseSource).
SpectrumType	str	Spectrum type string. This corresponds to the type attribute of the spectrum component.
Spatial_Filename	str	Path to spatial template associated to this source.
Spectrum_Filename	str	Path to file associated to the spectral model of this source.
correlation	dict	Dictionary of correlation coefficients.
model_counts	ndarray	Vector of predicted counts for this source in each analysis energy bin.
sed	dict	Output of SED analysis. See <i>SED Analysis</i> for more information.
ra	float	Right ascension of the source in deg.
dec	float	Declination of the source in deg.
glon	float	Galactic Longitude of the source in deg.
glat	float	Galactic Latitude of the source in deg.
offset_ra	float	Angular offset from ROI center along RA.
offset_dec	float	Angular offset from ROI center along DEC.
offset_glon	float	Angular offset from ROI center along GLON.
offset_glat	float	Angular offset from ROI center along GLAT.
offset_roi_edge	float	Distance from the edge of the ROI in deg. Negative (positive) values indicate locations inside (outside) the ROI.

Table 1.2 – continued from previous page

Key	Type	Description
offset	float	Angular offset from ROI center.
pos_sigma	float	1-sigma uncertainty (deg) on the source position.
pos_sigma_semimajor	float	1-sigma uncertainty (deg) on the source position along major axis.
pos_sigma_semiminor	float	1-sigma uncertainty (deg) on the source position along minor axis.
pos_angle	float	Position angle (deg) of the positional uncertainty ellipse.
pos_r68	float	68% uncertainty (deg) on the source position.
pos_r95	float	95% uncertainty (deg) on the source position.
pos_r99	float	99% uncertainty (deg) on the source position.
param_names	ndarray	Names of spectral parameters.
param_values	ndarray	Spectral parameter values.
param_errors	ndarray	Spectral parameters errors.
ts	float	Source test statistic.
loglike	float	Log-likelihood of the model evaluated at the best-fit normalization of the source.
loglike_scan	ndarray	Log-likelihood values for scan of source normalization.
dloglike_scan	ndarray	Delta Log-likelihood values for scan of source normalization.
eflux_scan	ndarray	Energy flux values for scan of source normalization.
flux_scan	ndarray	Flux values for scan of source normalization.
norm_scan	ndarray	Normalization parameters values for scan of source normalization.
npred	float	Number of predicted counts from this source integrated over the analysis energy range.
pivot_energy	float	Decorrelation energy in MeV.
flux	float	Photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated over analysis energy range
flux100	float	Photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000	float	Photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000	float	Photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 10 GeV to 316 GeV.
flux_err	float	Photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1}$) integrated over analysis energy range
flux100_err	float	Photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000_err	float	Photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000_err	float	Photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 10 GeV to 316 GeV.
flux_ul95	float	95% CL upper limit on the photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated over analysis energy range
flux100_ul95	float	95% CL upper limit on the photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000_ul95	float	95% CL upper limit on the photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000_ul95	float	95% CL upper limit on the photon flux ($\text{cm}^{-2} \text{s}^{-1}$) integrated from 10 GeV to 316 GeV.
eflux	float	Energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated over analysis energy range
eflux100	float	Energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000	float	Energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000	float	Energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 10 GeV to 316 GeV.
eflux_err	float	Energy flux uncertainty ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated over analysis energy range
eflux100_err	float	Energy flux uncertainty ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000_err	float	Energy flux uncertainty ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000_err	float	Energy flux uncertainty ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 10 GeV to 316 GeV.
eflux_ul95	float	95% CL upper limit on the energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated over analysis energy range
eflux100_ul95	float	95% CL upper limit on the energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000_ul95	float	95% CL upper limit on the energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000_ul95	float	95% CL upper limit on the energy flux ($\text{MeV cm}^{-2} \text{s}^{-1}$) integrated from 10 GeV to 316 GeV.
dnde	float	Differential photon flux ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at the pivot energy.
dnde100	float	Differential photon flux ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at 100 MeV.
dnde1000	float	Differential photon flux ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at 1 GeV.
dnde10000	float	Differential photon flux ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at 10 GeV.
dnde_err	float	Differential photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at the pivot energy.

Table 1.2 – continued from previous page

Key	Type	Description
dnde100_err	float	Differential photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at 100 MeV.
dnde1000_err	float	Differential photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at 1 GeV.
dnde10000_err	float	Differential photon flux uncertainty ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$) evaluated at 10 GeV.
dnde_index	float	Logarithmic slope of the differential photon spectrum evaluated at the pivot energy.
dnde100_index	float	Logarithmic slope of the differential photon spectrum evaluated at 100 MeV.
dnde1000_index	float	Logarithmic slope of the differential photon spectrum evaluated evaluated at 1 GeV.
dnde10000_index	float	Logarithmic slope of the differential photon spectrum evaluated at 10 GeV.

ROI Optimization and Fitting

Source fitting with fermipy is generally performed with the *optimize* and *fit* methods.

Fitting

fit is a wrapper on the pyLikelihood fit method and performs a likelihood fit of all free parameters of the model. This method can be used to manually optimize of the model by calling it after freeing one or more source parameters. The following example demonstrates the commands that would be used to fit the normalizations of all sources within 3 deg of the ROI center:

```
>>> gta.free_sources(distance=2.0,pars='norm')
>>> gta.print_params(True)
```

idx	parname	value	error	min	max	scale	free
3FGL J1104.4+3812							
18	Prefactor	1.77	0	1e-05	100	1e-11	*
3FGL J1109.6+3734							
24	Prefactor	0.33	0	1e-05	100	1e-14	*
galdiff							
52	Prefactor	1	0	0.1	10	1	*
isodiff							
55	Normalization	1	0	0.001	1e+03	1	*

```
>>> o = gta.fit()
2016-04-19 14:07:55 INFO GTAnalysis.fit(): Starting fit.
2016-04-19 14:08:56 INFO GTAnalysis.fit(): Fit returned successfully.
2016-04-19 14:08:56 INFO GTAnalysis.fit(): Fit Quality: 3 LogLike: -77279.869 DeltaLogLike:
>>> gta.print_params(True)
```

idx	parname	value	error	min	max	scale	free
3FGL J1104.4+3812							
18	Prefactor	2.13	0.0161	1e-05	100	1e-11	*
3FGL J1109.6+3734							
24	Prefactor	0.342	0.0904	1e-05	100	1e-14	*
galdiff							
52	Prefactor	0.897	0.0231	0.1	10	1	*
isodiff							
55	Normalization	1.15	0.016	0.001	1e+03	1	*

By default *fit* will repeat the fit until a fit quality of 3 is obtained. After the fit returns all sources with free parameters will have their properties (flux, TS, NPred, etc.) updated in the `ROIModel` instance. The return value of the method is a dictionary containing the following diagnostic information about the fit:

Listing 1.23: *fit* Output Dictionary

Key	Type	Description
<code>fit_quality</code>	<code>int</code>	Fit quality parameter for MINUIT and NEWMINUIT optimizers (3 - Full accurate covariance matrix, 2 - Full matrix, but forced positive-definite (i.e. not accurate), 1 - Diagonal approximation only, not accurate, 0 - Error matrix not calculated at all)
<code>errors</code>	<code>ndarray</code>	Vector of parameter errors (unscaled).
<code>loglike</code>	<code>float</code>	Post-fit log-likelihood value.
<code>correlation</code>	<code>ndarray</code>	Correlation matrix between free parameters of the fit.
<code>config</code>	<code>dict</code>	Copy of input configuration to this method.
<code>values</code>	<code>ndarray</code>	Vector of best-fit parameter values (unscaled).
<code>dloglike</code>	<code>float</code>	Improvement in log-likelihood value.
<code>fit_status</code>	<code>int</code>	Optimizer return code (0 = ok).
<code>covariance</code>	<code>ndarray</code>	Covariance matrix between free parameters of the fit.
<code>edm</code>	<code>float</code>	Estimated distance to maximum of log-likelihood function.

The *fit* also accepts keyword arguments which can be used to configure its behavior at runtime:

```
>>> o = gta.fit(min_fit_quality=2, optimizer='NEWMINUIT', reoptimize=True)
```

Reference/API

`GTAnalysis.fit` (*update=True*, ***kwargs*)

Run the likelihood optimization. This will execute a fit of all parameters that are currently free in the model and update the characteristics of the corresponding model components (TS, npred, etc.). The fit will be repeated `N` times (set with the `retries` parameter) until a fit quality greater than or equal to `min_fit_quality` and a fit status code of 0 is obtained. If the fit does not succeed after `N` retries then all parameter values will be reverted to their state prior to the execution of the fit.

Parameters

- **update** (*bool*) – Update the model dictionary for all sources with free parameters.
- **tol** (*float*) – Set the optimizer tolerance.
- **verbosity** (*int*) – Set the optimizer output level.
- **optimizer** (*str*) – Set the likelihood optimizer (e.g. MINUIT or NEWMINUIT).
- **retries** (*int*) – Set the number of times to rerun the fit when the fit quality is < 3.
- **min_fit_quality** (*int*) – Set the minimum fit quality. If the fit quality is smaller than this value then all model parameters will be restored to their values prior to the fit.
- **reoptimize** (*bool*) – Refit background sources when updating source properties (TS and likelihood profiles).

Returns *fit* – Dictionary containing diagnostic information from the fit (fit quality, parameter covariances, etc.).

Return type `dict`

ROI Optimization

The *optimize* method performs an automatic optimization of the ROI by fitting all sources with an iterative strategy.

```
>>> o = gta.optimize()
```

It is generally good practice to run this method once at the start of your analysis to ensure that all parameters are close to their global likelihood maxima.

Listing 1.24: *optimization* Output Dictionary

Key	Type	Description
loglike1	float	Post-optimization log-likelihood value.
loglike0	float	Pre-optimization log-likelihood value.
config	dict	Copy of input configuration to this method.
dloglike	float	Improvement in log-likelihood value.

Reference/API

GTAnalysis.**optimize** (**kwargs)

Iteratively optimize the ROI model. The optimization is performed in three sequential steps:

- Free the normalization of the N largest components (as determined from NPred) that contain a fraction `npred_frac` of the total predicted counts in the model and perform a simultaneous fit of the normalization parameters of these components.
- Individually fit the normalizations of all sources that were not included in the first step in order of their `npred` values. Skip any sources that have `NPred < npred_threshold`.
- Individually fit the shape and normalization parameters of all sources with `TS > shape_ts_threshold` where TS is determined from the first two steps of the ROI optimization.

To ensure that the model is fully optimized this method can be run multiple times.

Parameters

- **npred_frac** (*float*) – Threshold on the fractional number of counts in the N largest components in the ROI. This parameter determines the set of sources that are fit in the first optimization step.
- **npred_threshold** (*float*) – Threshold on the minimum number of counts of individual sources. This parameter determines the sources that are fit in the second optimization step.
- **shape_ts_threshold** (*float*) – Threshold on source TS used for determining the sources that will be fit in the third optimization step.
- **max_free_sources** (*int*) – Maximum number of sources that will be fit simultaneously in the first optimization step.
- **skip** (*list*) – List of str source names to skip while optimizing.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Customizing the Model

The ROIModel class is responsible for managing the source and diffuse components in the ROI. Configuration of the model is controlled with the *model* block of YAML configuration file.

Configuring Diffuse Components

The simplest configuration uses a single file for the galactic and isotropic diffuse components. By default the galactic diffuse and isotropic components will be named *galdiff* and *isodiff* respectively. An alias for each component will

also be created with the name of the mapcube or file spectrum. For instance the galactic diffuse can be referred to as *galdiff* or *gll_iem_v06* in the following example.

```
model:
  src_roiwidth : 10.0
  galdiff      : '$FERMI_DIFFUSE_DIR/gll_iem_v06.fits'
  isodiff      : '$FERMI_DIFFUSE_DIR/isotropic_source_4years_P8V3.txt'
  catalogs     : ['gll_psc_v14.fit']
```

To define two or more galactic diffuse components you can optionally define the *galdiff* and *isodiff* parameters as lists. A separate component will be generated for each element in the list with the name *galdiffXX* or *isodiffXX* where *XX* is an integer position in the list.

```
model:
  galdiff      :
    - '$FERMI_DIFFUSE_DIR/diffuse_component0.fits'
    - '$FERMI_DIFFUSE_DIR/diffuse_component1.fits'
```

To explicitly set the name of a component you can define any element as a dictionary containing *name* and *file* fields:

```
model:
  galdiff      :
    - { 'name' : 'component0' : 'file' : '$FERMI_DIFFUSE_DIR/diffuse_component0.fits' }
    - { 'name' : 'component1' : 'file' : '$FERMI_DIFFUSE_DIR/diffuse_component1.fits' }
```

Configuring Source Components

The list of sources for inclusion in the ROI model is set by defining a list of catalogs with the *catalogs* parameter. Catalog files can be in either XML or FITS format. Sources from the catalogs in this list that satisfy either the *src_roiwidth* or *src_radius* selections are added to the ROI model. If a source is defined in multiple catalogs the source definition from the last file in the catalogs list takes precedence.

```
model:
  src_radius: 5.0
  src_roiwidth: 10.0
  catalogs :
    - 'gll_psc_v16.fit'
    - 'extra_sources.xml'
```

Individual sources can also be defined within the configuration file with the *sources* parameter. This parameter contains a list of dictionaries that defines the spatial and spectral parameters of each source. The keys of the source dictionary map to the spectral and spatial source properties as they would be defined in the XML model file.

```
model:
  sources :
    - { name: 'SourceA', glon : 120.0, glat : -3.0,
        SpectrumType : 'PowerLaw', Index : 2.0, Scale : 1000, Prefactor : !!float 1e-11,
        SpatialModel: 'PointSource' }
    - { name: 'SourceB', glon : 122.0, glat : -3.0,
        SpectrumType : 'LogParabola', norm : !!float 1E-11, Scale : 1000, beta : 0.0,
        SpatialModel: 'PointSource' }
```

For parameters defined as scalars, the scale and value properties will be assigned automatically from the input value. To set these manually a parameter can also be initialized with a dictionary that explicitly sets the value and scale properties:

```
model:
  sources :
    - { name: 'SourceA', glon : 120.0, glat : -3.0,
        SpectrumType : 'PowerLaw', Index : 2.0, Scale : 1000,
        Prefactor : { value : 1.0, scale : !!float 1e-11, free : '0' },
        SpatialModel: 'PointSource' }
```

Spatial Models

Fermipy supports four spatial models which are defined with the `SpatialModel` property:

- `PointSource` : A point source (`SkyDirFunction`).
- `RadialGaussian` : A symmetric 2D Gaussian with width parameter ‘Sigma’.
- `RadialDisk` : A symmetric 2D Disk with radius ‘Radius’.
- `SpatialMap` : An arbitrary 2D shape with morphology defined by a FITS template.

The spatial extension of `RadialDisk` and `RadialGaussian` can be controlled with the `SpatialWidth` parameter which sets the 68% containment radius in degrees. Note for ST releases prior to 11-01-01, `RadialDisk` and `RadialGaussian` sources will be represented with the `SpatialMap` type.

```
model:
  sources :
    - { name: 'PointSource', glon : 120.0, glat : 0.0,
        SpectrumType : 'PowerLaw', Index : 2.0, Scale : 1000, Prefactor : !!float 1e-11,
        SpatialModel: 'PointSource' }
    - { name: 'DiskSource', glon : 120.0, glat : 0.0,
        SpectrumType : 'PowerLaw', Index : 2.0, Scale : 1000, Prefactor : !!float 1e-11,
        SpatialModel: 'RadialDisk', SpatialWidth: 1.0 }
    - { name: 'GaussSource', glon : 120.0, glat : 0.0,
        SpectrumType : 'PowerLaw', Index : 2.0, Scale : 1000, Prefactor : !!float 1e-11,
        SpatialModel: 'RadialGaussian', SpatialWidth: 1.0 }
    - { name: 'MapSource', glon : 120.0, glat : 0.0,
        SpectrumType : 'PowerLaw', Index : 2.0, Scale : 1000, Prefactor : !!float 1e-11,
        SpatialModel: 'SpatialTemplate', file : 'template.fits' }
```

Editing the Model at Runtime

The model can be manually editing at runtime with the `add_source()` and `delete_source()` methods. Sources can be added either before or after calling `setup()` as shown in the following example.

```
from fermipy.gtanalysis import GTAnalysis

gta = GTAnalysis('config.yaml', logging={'verbosity' : 3})

# Remove isodiff from the model
gta.delete_source('isodiff')

# Add SourceA to the model
gta.add_source('SourceA', { 'glon' : 120.0, 'glat' : -3.0,
                             'SpectrumType' : 'PowerLaw', 'Index' : 2.0,
                             'Scale' : 1000, 'Prefactor' : 1e-11,
                             'SpatialModel' : 'PointSource' })

gta.setup()
```

```
# Add SourceB to the model
gta.add_source('SourceB',{ 'glon' : 121.0, 'glat' : -2.0,
                           'SpectrumType' : 'PowerLaw', 'Index' : 2.0,
                           'Scale' : 1000, 'Prefactor' : 1e-11,
                           'SpatialModel' : 'PointSource' })
```

Sources added before calling `setup()` will be appended to the XML model definition. Sources added after calling `setup()` will be created dynamically through the `pyLikelihood` object creation mechanism.

Advanced Analysis Methods

This page documents some of the more advanced methods and features available in Fermipy:

- *TS Map*: Generate a test statistic (TS) map for a new source centered at each spatial bin in the ROI.
- *TS Cube*: Generate a TS map using the `gttscube` ST application. In addition to generating a TS map this method can also extract a test source likelihood profile as a function of energy and position over the whole ROI.
- *Residual Map*: Generate a residual map by evaluating the difference between smoothed data and model maps (residual) at each spatial bin in the ROI.
- *Source Finding*: Find new sources using an iterative source-finding algorithm. Adds new sources to the ROI by looking for peaks in the TS map.
- *SED Analysis*: Compute the spectral energy distribution of a source by fitting its amplitude in a sequence of energy bins.
- *Light Curves*: Compute the lightcurve of a source by fitting its amplitude in a sequence of time bins.
- *Extension Fitting*: Fit the angular extension of a source.
- *Source Localization*: Find the best-fit position of a source.
- *Phased Analysis*: Instructions for performing a phased-selected analysis.
- *Sensitivity Tools*: Scripts and classes for estimating sensitivity.

SED Analysis

The `sed()` method computes a spectral energy distribution (SED) by fitting for the flux normalization of a source in a sequence of energy bins. The normalization in each bin is fit independently using a power-law spectrum with a fixed index. The value of this index can be set with the `bin_index` parameter or allowed to vary over the energy range according to the local slope of the global spectral model (with the `use_local_index` parameter).

The `free_background` and `cov_scale` parameters can be used to control how nuisance parameters are dealt with in the fit. By default this method will fix the parameters of background components ROI when fitting the source normalization in each energy bin (`free_background = False`). Setting `free_background` to `True` will profile the normalizations of all background components that were free when the method was executed. In order to minimize overfitting, background normalization parameters are constrained with priors taken from the global fit. The strength of the priors is controlled with the `cov_scale` parameter. A larger (smaller) value of `cov_scale` applies a weaker (stronger) constraint on the background amplitude. Setting `cov_scale` to `None` can be used to perform the fit without priors.

Examples

The `sed()` method is executed by passing the name of a source in the ROI as a single argument. Additional keyword argument can also be provided to override the default configuration of the method:

```
# Run analysis with default energy binning
sed = gta.sed('sourceA')

# Override the energy binning and the assumed power-law index
# within the bin
sed = gta.sed('sourceA', loge_bins=[2.0,2.5,3.0,3.5,4.0,4.5,5.0], bin_index=2.3)

# Profile background normalization parameters with prior scale of 5.0
sed = gta.sed('sourceA', free_background=True, cov_scale=5.0)
```

By default the method will use the energy bins of the underlying analysis. The `loge_bins` keyword argument can be used to override the default binning with the restriction that the SED energy bins most align with the analysis bins.

The return value of `sed()` is a dictionary with the results of the analysis. The output dictionary is also saved to the `sed` dictionary of the *Source* instance which is written to the output file generated by `write_roi()`.

The following example shows how the output dictionary can be captured from either from the method return value or later accessed from the *ROIModel* instance:

```
# Get the sed results from the return argument
sed = gta.sed('sourceA')

# Get the sed results from the source object
sed = gta.roi['sourceA']

# Print the SED flux values
print(sed['flux'])
```

The contents of the FITS file and output dictionary are documented in *SED FITS File* and *SED Dictionary*.

SED FITS File

The following table describes the contents of the FITS file written by `sed()`. The SED HDU uses that data format specification for SEDs documented [here](#).

Listing 1.25: *sed* Output Dictionary

HDU	Column Name	Description
SED	e_min	Lower edges of SED energy bins (MeV).
SED	e_ref	Upper edges of SED energy bins (MeV).
SED	e_max	Centers of SED energy bins (MeV).
SED	ref_dnde	Differential flux of the reference model evaluated at the lower bin edge ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$)
SED	ref_dnde	Differential flux of the reference model evaluated at the upper bin edge ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$)
SED	ref_flux	Flux of the reference model in each bin ($\text{cm}^{-2} \text{s}^{-1}$).
SED	ref_elflux	Energy flux of the reference model in each bin ($\text{MeV cm}^{-2} \text{s}^{-1}$).
SED	ref_dnde	Differential flux of the reference model evaluated at the bin center ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$)
SED	ref_npred	Number of predicted counts in the reference model in each bin.
SED	norm	Normalization in each bin in units of the reference model.
SED	norm_err	Symmetric error on the normalization in each bin in units of the reference model.
SED	norm_errn	Lower 1-sigma error on the normalization in each bin in units of the reference model.
SED	norm_errp	Upper 1-sigma error on the normalization in each bin in units of the reference model.
SED	norm_UL	Upper limit on the normalization in each bin in units of the reference model.
SED	loglike	Log-likelihood value of the model for the best-fit amplitude.
SED	norm_scan	Array of NxM normalization values for the profile likelihood scan in N energy bins and M scan points. A row-wise multiplication with any of <code>ref</code> columns can be used to convert this matrix to the respective unit.
SED	dloglike_scan	Array of NxM delta-loglikelihood values for the profile likelihood scan in N energy bins and M scan points.
MODEL_FLUX	Energy	Energies at which the spectral band is evaluated (MeV).
MODEL_FLUX	Flux	Central value of spectral band ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
MODEL_FLUX	Flux_lo	Lower 1-sigma bound of spectral band ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
MODEL_FLUX	Flux_hi	Upper 1-sigma bound of spectral band ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
MODEL_FLUX	Flux_err	Symmetric error of spectral band ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
MODEL_FLUX	Flux_ferr	Fractional width of spectral band.
PARAMS	name	Name of the parameter.
PARAMS	value	Value of the parameter.
PARAMS	error	1-sigma parameter error (nan indicates that the parameter was not included in the fit).
PARAMS	covariance	Covariance matrix among free parameters.
PARAMS	correlation	Correlation matrix among free parameters.

SED Dictionary

The following table describes the contents of the `sed()` output dictionary:

Table

Key	Type	Description
loge_min	<code>ndarray</code>	Lower edges of SED energy bins ($\log_{10}(E/\text{MeV})$).
loge_max	<code>ndarray</code>	Upper edges of SED energy bins ($\log_{10}(E/\text{MeV})$).
loge_ctr	<code>ndarray</code>	Centers of SED energy bins ($\log_{10}(E/\text{MeV})$).
loge_ref	<code>ndarray</code>	Reference energies of SED energy bins ($\log_{10}(E/\text{MeV})$).
e_min	<code>ndarray</code>	Lower edges of SED energy bins (MeV).
e_max	<code>ndarray</code>	Upper edges of SED energy bins (MeV).

Table 1.3 -

Key	Type	Description
e_ctr	ndarray	Centers of SED energy bins (MeV).
e_ref	ndarray	Reference energies of SED energy bins (MeV).
ref_flux	ndarray	Flux of the reference model in each bin ($\text{cm}^{-2} \text{s}^{-1}$).
ref_eflux	ndarray	Energy flux of the reference model in each bin ($\text{MeV cm}^{-2} \text{s}^{-1}$).
ref_dnde	ndarray	Differential flux of the reference model evaluated at the bin center ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
ref_dnde_e_min	ndarray	Differential flux of the reference model evaluated at the lower bin edge ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
ref_dnde_e_max	ndarray	Differential flux of the reference model evaluated at the upper bin edge ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
ref_e2dnde	ndarray	E^2 x the differential flux of the reference model evaluated at the bin center ($\text{MeV cm}^{-2} \text{s}^{-1}$).
ref_npred	ndarray	Number of predicted counts in the reference model in each bin.
norm	ndarray	Normalization in each bin in units of the reference model.
flux	ndarray	Flux in each bin ($\text{cm}^{-2} \text{s}^{-1}$).
eflux	ndarray	Energy flux in each bin ($\text{MeV cm}^{-2} \text{s}^{-1}$).
dnde	ndarray	Differential flux in each bin ($\text{cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$).
e2dnde	ndarray	E^2 x the differential flux in each bin ($\text{MeV cm}^{-2} \text{s}^{-1}$).
dnde_err	ndarray	1-sigma error on dnde evaluated from likelihood curvature.
dnde_err_lo	ndarray	Lower 1-sigma error on dnde evaluated from the profile likelihood (MINOS errors).
dnde_err_hi	ndarray	Upper 1-sigma error on dnde evaluated from the profile likelihood (MINOS errors).
dnde_ul95	ndarray	95% CL upper limit on dnde evaluated from the profile likelihood (MINOS errors).
dnde_ul	ndarray	Upper limit on dnde evaluated from the profile likelihood using a $\text{CL} = \text{ul_confidence}$.
e2dnde_err	ndarray	1-sigma error on e2dnde evaluated from likelihood curvature.
e2dnde_err_lo	ndarray	Lower 1-sigma error on e2dnde evaluated from the profile likelihood (MINOS errors).
e2dnde_err_hi	ndarray	Upper 1-sigma error on e2dnde evaluated from the profile likelihood (MINOS errors).
e2dnde_ul95	ndarray	95% CL upper limit on e2dnde evaluated from the profile likelihood (MINOS errors).
e2dnde_ul	ndarray	Upper limit on e2dnde evaluated from the profile likelihood using a $\text{CL} = \text{ul_confidence}$.
ts	ndarray	Test statistic.
loglike	ndarray	Log-likelihood of model for the best-fit amplitude.
npred	ndarray	Number of model counts.
fit_quality	ndarray	Fit quality parameter for MINUIT and NEWMINUIT optimizers (3 - Full accurate covariance).
fit_status	ndarray	Fit status parameter (0=ok).
index	ndarray	Spectral index of the power-law model used to fit this bin.
norm_scan	ndarray	Array of NxM normalization values for the profile likelihood scan in N energy bins and M scans.
dloglike_scan	ndarray	Array of NxM delta-loglikelihood values for the profile likelihood scan in N energy bins and M scans.
loglike_scan	ndarray	Array of NxM loglikelihood values for the profile likelihood scan in N energy bins and M scans.
param_covariance	ndarray	Covariance matrix for the best-fit spectral parameters of the source.
param_names	ndarray	Array of names for the parameters in the global spectral parameterization of this source.
param_values	ndarray	Array of parameter values.
param_errors	ndarray	Array of parameter errors.
model_flux	dict	Dictionary containing the differential flux uncertainty band of the best-fit global spectral parameters.
config	dict	Copy of input configuration to this method.

Configuration

The default configuration of the method is controlled with the `sed` section of the configuration file. The default configuration can be overridden by passing the option as a *kwargs* argument to the method.

Listing 1.26: *sed* Options

Option	De- fault	Description
<code>bin_index</code>	2.0	Spectral index that will be use when fitting the energy distribution within an energy bin.
<code>cov_scale</code>	3.0	Scale factor that sets the strength of the prior on nuisance parameters that are free. Setting this to None disables the prior.
<code>free_background</code>	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
<code>free_radius</code>	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
<code>make_plots</code>	False	Generate diagnostic plots.
<code>ul_confidence</code>	0.95	Confidence level for flux upper limit.
<code>use_local_index</code>	False	Use a power-law approximation to the shape of the global spectrum in each bin. If this is false then a constant index set to <code>bin_index</code> will be used.
<code>write_fits</code>	True	Write the output to a FITS file.
<code>write_npy</code>	True	Write the output dictionary to a numpy file.

Reference/API

`GTAnalysis.sed(name, **kwargs)`

Generate a spectral energy distribution (SED) for a source. This function will fit the normalization of the source in each energy bin. By default the SED will be generated with the analysis energy bins but a custom binning can be defined with the `log_e_bins` parameter.

Parameters

- **name** (*str*) – Source name.
- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **log_e_bins** (*ndarray*) – Sequence of energies in $\log_{10}(E/\text{MeV})$ defining the edges of the energy bins. If this argument is None then the analysis energy bins will be used. The energies in this sequence must align with the bin edges of the underlying analysis instance.
- **bin_index** (*float*) – Spectral index that will be use when fitting the energy distribution within an energy bin. (default : 2.0)
- **cov_scale** (*float*) – Scale factor that sets the strength of the prior on nuisance parameters that are free. Setting this to None disables the prior. (default : 3.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **ul_confidence** (*float*) – Confidence level for flux upper limit. (default : 0.95)
- **use_local_index** (*bool*) – Use a power-law approximation to the shape of the global spectrum in each bin. If this is false then a constant index set to `bin_index` will be used. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)

- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns *sed* – Dictionary containing output of the SED analysis.

Return type *dict*

Light Curves

lightcurve() can be used to fit characteristics of a source (flux, TS, etc.) in a sequence of time bins.

Examples

```
# Generate a lightcurve with two bins
lc = gta.lightcurve('sourceA', nbins=2)

# Generate a lightcurve with 1-week binning
lc = gta.lightcurve('sourceA', binsz=86400.*7.0)

# Generate a lightcurve freeing sources within 3 deg of the source
# of interest
lc = gta.lightcurve('sourceA', binsz=86400.*7.0, free_radius=3.0)
```

Reference/API

GTAnalysis.**lightcurve** (*name*, ***kwargs*)

Generate a lightcurve for the named source. The function will complete the basic analysis steps for each bin and perform a likelihood fit for each bin. Extracted values (along with errors) are Integral Flux, spectral model, Spectral index, TS value, pred. # of photons.

Parameters

- **name** (*str*) – source name
- **binsz** (*float*) – Set the lightcurve bin size in seconds. (default : 86400.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_params** (*list*) – Set the parameters of the source of interest that will be re-fit in each time bin. If this list is empty then all parameters will be freed. (default : None)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **free_sources** (*list*) – List of sources to be freed. These sources will be added to the list of sources satisfying the free_radius selection. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **nbins** (*int*) – Set the number of lightcurve bins. The total time range will be evenly split into this number of time bins. (default : None)
- **time_bins** (*list*) – Set the lightcurve bin edge sequence in MET. This option takes precedence over binsz and nbins. (default : None)

- `use_local_ltcube` (*bool*) – (default : True)
- `write_fits` (*bool*) – Write the output to a FITS file. (default : True)
- `write_numpy` (*bool*) – Write the output dictionary to a numpy file. (default : True)

Returns `LightCurve` – Dictionary containing output of the LC analysis

Return type `dict`

Extension Fitting

The `extension()` method executes a source extension analysis for a given source by computing a likelihood ratio test with respect to the no-extension (point-source) hypothesis and a best-fit model for extension. The best-fit extension is found by performing a likelihood profile scan over the source width (68% containment) and fitting for the extension that maximizes the model likelihood. Currently this method supports two models for extension: a 2D Gaussian (*RadialGaussian*) or a 2D disk (*RadialDisk*).

At runtime the default settings for the extension analysis can be overridden by passing one or more *kwargs* when executing `extension()`:

```
# Run extension fit of sourceA with default settings
>>> gta.extension('sourceA')

# Override default spatial model
>>> gta.extension('sourceA', spatial_model='RadialDisk')
```

By default the method will fix all background parameters before performing the extension fit. One can leave background parameters free by setting `free_background=True`:

```
# Free a nearby source that maybe be partially degenerate with the
# source of interest. The normalization of SourceB will be refit
# when testing the extension of sourceA
gta.free_norm('sourceB')
gta.extension('sourceA', free_background=True)

# Fix all background parameters when testing the extension
# of sourceA
gta.extension('sourceA', free_background=False)

# Free normalizations of sources within 2 degrees of sourceA
gta.extension('sourceA', free_radius=2.0)
```

The results of the extension analysis are written to a dictionary which is the return value of the extension method.

```
ext = gta.extension('sourceA', write_numpy=True, write_fits=True)
```

The contents of the output dictionary are given in the following table:

Listing 1.27: *extension* Output Dictionary

Key	Type	Description
width	<code>ndarray</code>	Vector of width values.
dloglike	<code>ndarray</code>	Sequence of delta-log-likelihood values for each point in the profile likelihood scan.
loglike	<code>ndarray</code>	Sequence of likelihood values for each point in the scan over the spatial extension.
loglike_ptsrc	<code>float</code>	Model log-Likelihood value of the best-fit point-source model.
loglike_ext	<code>float</code>	Model log-Likelihood value of the best-fit extended source model.
loglike_base	<code>float</code>	Model log-Likelihood value of the baseline model.
ext	<code>float</code>	Best-fit extension in degrees.
ext_err_hi	<code>float</code>	Upper (1 sigma) error on the best-fit extension in degrees.
ext_err_lo	<code>float</code>	Lower (1 sigma) error on the best-fit extension in degrees.
ext_err	<code>float</code>	Symmetric (1 sigma) error on the best-fit extension in degrees.
ext_ul95	<code>float</code>	95% CL upper limit on the spatial extension in degrees.
ts_ext	<code>float</code>	Test statistic for the extension hypothesis.
source_fit	<code>dict</code>	Dictionary with parameters of the best-fit extended source model.
config	<code>dict</code>	Copy of the input configuration to this method.

Configuration

The default configuration of the method is controlled with the *extension* section of the configuration file. The default configuration can be overridden by passing the option as a *kwargs* argument to the method.

Listing 1.28: *extension* Options

Option	Default	Description
<code>fit_position</code>	False	Perform a simultaneous fit to the source position and extension.
<code>free_background</code>	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
<code>free_radius</code>	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
<code>make_plots</code>	False	Generate diagnostic plots.
<code>psf_scale</code>	None	Tuple of two vectors (logE,f) defining an energy-dependent PSF scaling function that will be applied when building spatial models for the source of interest. The tuple (logE,f) defines the fractional corrections f at the sequence of energies logE = log10(E/MeV) where f=0 corresponds to no correction. The correction function f(E) is evaluated by linearly interpolating the fractional correction factors f in log(E). The corrected PSF is given by $P'(x;E) = P(x/(1+f(E));E)$ where x is the angular separation.
<code>save_model</code>	False	Save model counts cubes for the best-fit model of extension.
<code>spatial_model</code>	Radial-Gaussian	Spatial model that will be used to test the source extension. The spatial scale parameter of the model will be set such that the 68% containment radius of the model is equal to the width parameter.
<code>sqrt_ts_threshold</code>	None	Threshold on sqrt(TS_ext) that will be applied when update is True. If None then no threshold is applied.
<code>update</code>	False	Update this source with the best-fit model for spatial extension if TS_ext > tsext_threshold.
<code>width</code>	None	Sequence of values in degrees for the likelihood scan over spatial extension (68% containment radius). If this argument is None then the scan points will be determined from width_min/width_max/width_nstep.
<code>width_max</code>	1.0	Maximum value in degrees for the likelihood scan over spatial extent.
<code>width_min</code>	0.01	Minimum value in degrees for the likelihood scan over spatial extent.
<code>width_nstep</code>	21	Number of scan points between width_min and width_max. Scan points will be spaced evenly on a logarithmic scale between width_min and width_max.
<code>write_fits</code>	True	Write the output to a FITS file.
<code>write_numpy</code>	True	Write the output dictionary to a numpy file.

Reference/API

GTAnalysis.**extension** (*name*, ***kwargs*)

Test this source for spatial extension with the likelihood ratio method (TS_ext). This method will substitute an extended spatial model for the given source and perform a one-dimensional scan of the spatial extension parameter over the range specified with the width parameters. The 1-D profile likelihood is then used to compute the best-fit value, upper limit, and TS for extension. The background parameters that will be simultaneously profiled when performing the spatial scan can be controlled with the `free_background` and `free_radius` options. By default the position of the source will be fixed to its current position. A simultaneous fit to position and extension can be performed by setting `fit_position` to True.

Parameters

- **name** (*str*) – Source name.
- **fit_position** (*bool*) – Perform a simultaneous fit to the source position and extension. (default : False)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously

with the source of interest. (default : False)

- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **psf_scale_fn** (*tuple*) – Tuple of two vectors (logE,f) defining an energy-dependent PSF scaling function that will be applied when building spatial models for the source of interest. The tuple (logE,f) defines the fractional corrections f at the sequence of energies $\log E = \log_{10}(E/\text{MeV})$ where $f=0$ corresponds to no correction. The correction function $f(E)$ is evaluated by linearly interpolating the fractional correction factors f in log(E). The corrected PSF is given by $P'(x;E) = P(x/(1+f(E));E)$ where x is the angular separation. (default : None)
- **save_model_map** (*bool*) – Save model counts cubes for the best-fit model of extension. (default : False)
- **spatial_model** (*str*) – Spatial model that will be used to test the source extension. The spatial scale parameter of the model will be set such that the 68% containment radius of the model is equal to the width parameter. (default : RadialGaussian)
- **sqrt_ts_threshold** (*float*) – Threshold on sqrt(TS_ext) that will be applied when update is True. If None then no threshold is applied. (default : None)
- **update** (*bool*) – Update this source with the best-fit model for spatial extension if TS_ext > ts_ext_threshold. (default : False)
- **width** (*list*) – Sequence of values in degrees for the likelihood scan over spatial extension (68% containment radius). If this argument is None then the scan points will be determined from width_min/width_max/width_nstep. (default : None)
- **width_max** (*float*) – Maximum value in degrees for the likelihood scan over spatial extent. (default : 1.0)
- **width_min** (*float*) – Minimum value in degrees for the likelihood scan over spatial extent. (default : 0.01)
- **width_nstep** (*int*) – Number of scan points between width_min and width_max. Scan points will be spaced evenly on a logarithmic scale between width_min and width_max. (default : 21)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_numpy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns **extension** – Dictionary containing results of the extension analysis. The same dictionary is also saved to the dictionary of this source under ‘extension’.

Return type *dict*

TS Map

`tmap()` generates a test statistic (TS) map for an additional source component centered at each spatial bin in the ROI. The methodology is similar to that of the `gttsmap` ST application but with the following approximations:

- Evaluation of the likelihood is limited to pixels in the vicinity of the test source position.
- The background model is fixed when fitting the test source amplitude.

TS Cube is a related method that can also be used to generate TS maps as well as cubes (TS vs. position and energy).

For each spatial bin the method calculates the maximum likelihood test statistic given by

$$TS = 2 \sum_k \ln L(\mu, \theta | n_k) - \ln L(0, \theta | n_k)$$

where the summation index k runs over both spatial and energy bins, μ is the test source normalization parameter, and θ represents the parameters of the background model. The likelihood fitting implementation used by `tmap()` only fits the test source normalization (μ). Shape parameters of the test source and parameters of background components are fixed to their current values.

Examples

The spatial and spectral properties of the convolution kernel are defined with the `model` dictionary argument. The `model` dictionary format is the same as accepted by `add_source()`.

```
# Generate TS map for a power-law point source with Index=2.0
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
maps = gta.tmap('fit1', model=model)

# Generate TS map for a power-law point source with Index=2.0 and
# restricting the analysis to E > 3.16 GeV
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
maps = gta.tmap('fit1_emin35', model=model, erange=[3.5, None])

# Generate TS maps for a power-law point source with Index=1.5, 2.0, and 2.5
model={'SpatialModel' : 'PointSource'}
maps = []
for index in [1.5, 2.0, 2.5]:
    model['Index'] = index
    maps += [gta.tmap('fit1', model=model)]
```

The `multithread` option can be enabled to split the calculation across all available cores:

```
maps = gta.tmap('fit1', model=model, multithread=True)
```

Note that care should be taken when using this option in an environment where the number of cores per process is restricted such as a batch farm.

`tmap()` returns a `maps` dictionary containing *Map* representations of the TS and predicted counts (NPred) of the best-fit test source at each position.

```
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
maps = gta.tmap('fit1', model=model)
print('TS at Pixel (50,50): ', maps['ts'].counts[50,50])
```

The contents of the output dictionary are given in the following table.

Key	Type	Description
amplitude	<i>Map</i>	Best-fit test source amplitude expressed in terms of the spectral prefactor.
npred	<i>Map</i>	Best-fit test source amplitude expressed in terms of the total model counts (NPred).
ts	<i>Map</i>	Test source TS (twice the logLike difference between null and alternate hypothesis).
sqrt_ts	<i>Map</i>	Square-root of the test source TS.
file	str	Path to a FITS file containing the maps (TS, etc.) generated by this method.
src_dict	dict	Dictionary defining the properties of the test source.

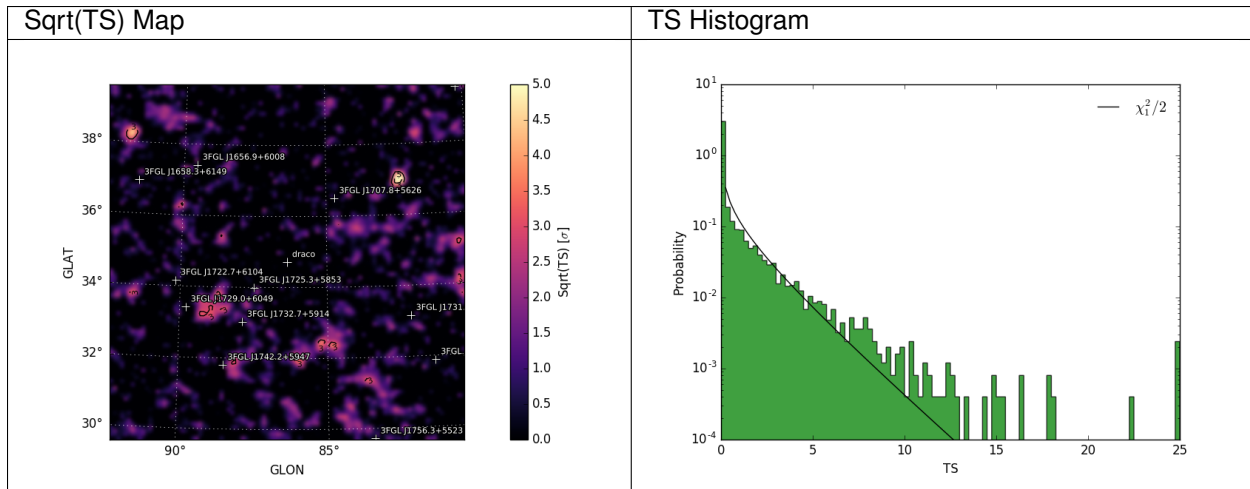
The `write_fits` and `write_npy` options can be used to write the output to a FITS or numpy file. All output files are prepended with the `prefix` argument.

Diagnostic plots can be generated by setting `make_plots=True` or by passing the output dictionary to `make_residmap_plots`:

```
maps = gta.tsmap('fit1', model=model, make_plots=True)
gta.plotter.make_tsmap_plots(maps, roi=gta.roi)
```

This will generate the following plots:

- `tsmap_sqrt_ts`: Map of $\sqrt{\text{TS}}$ values. The color map is truncated at 5 sigma with isocontours at 2 sigma intervals indicating values above this threshold.
- `tsmap_npred`: Map of best-fit source amplitude in counts.
- `tsmap_ts_hist`: Histogram of TS values for all points in the map. Overplotted is the reference distribution for chi-squared with one degree of freedom (expectation from Chernoff's theorem).



Configuration

The default configuration of the method is controlled with the `tsmap` section of the configuration file. The default configuration can be overridden by passing the option as a `kwargs` argument to the method.

Listing 1.29: `tsmap` Options

Option	De- fault	Description
<code>exclude</code>	None	List of sources that will be removed from the model when computing the TS map.
<code>loge_bounds</code>	None	Restrict the analysis to an energy range (<code>emin,emax</code>) in $\log_{10}(\text{E/MeV})$ that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either <code>emin/emax</code> are None then only an upper/lower bound on the energy range will be applied.
<code>make_plots</code>	False	Generate diagnostic plots.
<code>max_kernel</code>	3.0	Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy.
<code>model</code>	None	Dictionary defining the spatial/spectral properties of the test source. If <code>model</code> is None the test source will be a PointSource with an Index 2 power-law spectrum.
<code>multithread</code>	False	Split the calculation across all available cores.
<code>write_fits</code>	True	Write the output to a FITS file.
<code>write_npy</code>	True	Write the output dictionary to a numpy file.

Reference/API

`GTAnalysis.tsmap` (*prefix*='', ***kwargs*)

Generate a spatial TS map for a source component with properties defined by the `model` argument. The TS map will have the same geometry as the ROI. The output of this method is a dictionary containing `Map` objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

This method uses a simplified likelihood fitting implementation that only fits for the normalization of the test source. Before running this method it is recommended to first optimize the ROI model (e.g. by running `optimize()`).

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the TS map. (default : None)
- **log_bounds** (*list*) – Restrict the analysis to an energy range (*emin*,*emax*) in $\log_{10}(E/\text{MeV})$ that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either *emin*/*emax* are None then only an upper/lower bound on the energy range will be applied. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **max_kernel_radius** (*float*) – Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy. (default : 3.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across all available cores. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_numpy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

Returns `tsmap` – A dictionary containing the `Map` objects for TS and source amplitude.

Return type `dict`

TS Cube

Warning: This method requires Fermi Science Tools version 11-04-00 or later.

`tscube()` can be used generate both test statistic (TS) maps and bin-by-bin scans of the test source likelihood as a function of spatial pixel and energy bin (likelihood cubes). The implementation is based on the `gttscube` ST application which uses an efficient newton optimization algorithm for fitting the test source at each pixel in the ROI.

The TS map output has the same format as TS maps produced by `tsmap()` (see *TS Map* for further details). However while `tsmap()` fixes the background model, `tscube()` can also fit background normalization parameters when scanning the test source likelihood. This method makes no approximations in the evaluation of the likelihood and may be somewhat slower than `tsmap()` depending on the ROI dimensions and energy bounds.

For each spatial bin the method calculates the maximum likelihood test statistic given by

$$\text{TS} = 2 \sum_k \ln L(\mu, \hat{\theta}|n_k) - \ln L(0, \hat{\theta}|n_k)$$

where the summation index k runs over both spatial and energy bins, μ is the test source normalization parameter, and θ represents the parameters of the background model. Normalization parameters of the background model are refit at every test source position if they are free in the model. All other spectral parameters (indices etc.) are kept fixed.

Examples

The method is executed by providing a `model` dictionary argument that defines the spectrum and spatial morphology of the test source:

```
# Generate TS cube for a power-law point source with Index=2.0
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
cube = gta.tscube('fit1',model=model)

# Generate TS cube for a power-law point source with Index=2.0 and
# restricting the analysis to E > 3.16 GeV
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
cube = gta.tscube('fit1_emin35',model=model,erange=[3.5,None])

# Generate TS cubes for a power-law point source with Index=1.5, 2.0, and 2.5
model={'SpatialModel' : 'PointSource'}
cubes = []
for index in [1.5,2.0,2.5]:
    model['Index'] = index
    cubes += [gta.tsmap('fit1',model=model)]
```

In addition to generating a TS map, this method can also extract a test source likelihood profile as a function of energy at every position in the ROI (likelihood cube). This information is saved to the `SCANDATA` HDU of the output FITS file:

```
from astropy.table import Table
cube = gta.tscube('fit1',model=model, do_sed=True)
tab_scan = Table.read(cube['file'],'SCANDATA')
tab_ebounds = Table.read(cube['file'],'EBOUNDS')

eflux_scan = tab_ebounds['REF_EFLUX'][None,:,None]*tab_scan['norm_scan']

# Plot likelihood for pixel 400 and energy bin 2
plt.plot(eflux_scan[400,2],tab_scan['dloglike_scan'][400,2])
```

The likelihood profile cube can be used to evaluate the likelihood for a test source with an arbitrary spectral model at any position in the ROI. The `TSCube` and `CastroData` classes can be used to analyze a TS cube:

```
from fermipy.castro import TSCube
tscube = TSCube.create_from_fits('tscube.fits')
cd = tscube.castroData_from_ipix(400)

# Fit the likelihoods at pixel 400 with different spectral models
cd.test_spectra()
```

Configuration

The default configuration of the method is controlled with the `tscube` section of the configuration file. The default configuration can be overridden by passing the option as a *kwargs* argument to the method.

Listing 1.30: *tscube* Options

Option	De- fault	Description
cov_scale	-1.0	Scale factor to apply to broadband fitting cov. matrix in bin-by-bin fits (< 0 -> fixed)
cov_scale_bb	-1.0	Scale factor to apply to global fitting cov. matrix in broadband fits. (< 0 -> no prior)
do_sed	True	Compute the energy bin-by-bin fits
init_lambda	0	Initial value of damping parameter for newton step size calculation. A value of zero disables damping.
max_iter	30	Maximum number of iterations for the Newtons method fitter.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
nnorm	10	Number of points in the likelihood v. normalization scan
norm_sigma	5.0	Number of sigma to use for the scan range
remake_test_source	False	If true, recomputes the test source image (otherwise just shifts it)
st_scan_level	0	Level to which to do ST-based fitting (for testing)
tol	0.001	Critetia for fit convergence (estimated vertical distance to min < tol)
tol_type	0	Absoulte (0) or relative (1) criteria for convergence.

Reference/API

GTAnalysis.**tscube** (*prefix*='', ***kwargs*)

Generate a spatial TS map for a source component with properties defined by the `model` argument. This method uses the `gttscube` ST application for source fitting and will simultaneously fit the test source normalization as well as the normalizations of any background components that are currently free. The output of this method is a dictionary containing [Map](#) objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **do_sed** (*bool*) – Compute the energy bin-by-bin fits.
- **nnorm** (*int*) – Number of points in the likelihood v. normalization scan.
- **norm_sigma** (*float*) – Number of sigma to use for the scan range.
- **tol** (*float*) – Critetia for fit convergence (estimated vertical distance to min < tol).
- **tol_type** (*int*) – Absoulte (0) or relative (1) criteria for convergence.
- **max_iter** (*int*) – Maximum number of iterations for the Newton’s method fitter
- **remake_test_source** (*bool*) – If true, recomputes the test source image (otherwise just shifts it)
- **st_scan_level** (*int*) –
- **make_plots** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write a FITS file with the results of the analysis.

Returns `maps` – A dictionary containing the [Map](#) objects for TS and source amplitude.

Return type `dict`

Residual Map

`residmap()` calculates the residual between smoothed data and model maps. Whereas a TS map is only sensitive to positive deviations with respect to the model, `residmap()` is sensitive to both positive and negative residuals and therefore can be useful for assessing the model goodness-of-fit. The significance of the data/model residual at map position (i, j) is given by

$$\sigma_{ij}^2 = 2\text{sgn}(\tilde{n}_{ij} - \tilde{m}_{ij}) (\ln L_P(\tilde{n}_{ij}, \tilde{n}_{ij}) - \ln L_P(\tilde{n}_{ij}, \tilde{m}_{ij}))$$

$$\text{with } \tilde{m}_{ij} = \sum_k (m_k * f_k)_{ij} \quad \tilde{n}_{ij} = \sum_k (n_k * f_k)_{ij} \quad \ln L_P(n, m) = n \ln(m) - m$$

where n_k and m_k are the data and model maps at energy plane k and f_k is the convolution kernel. The convolution kernel is proportional to the counts expectation at a given pixel and normalized such that

$$f_{ijk} = s_{ijk} \left(\sum_{ijk} s_{ijk}^2 \right)^{-1}$$

where s is the expectation counts cube for a pure signal normalized to one.

Examples

The spatial and spectral properties of the convolution kernel are defined with the `model` dictionary argument. All source models are supported as well as a gaussian kernel (defined by setting `SpatialModel` to `Gaussian`).

```
# Generate residual map for a Gaussian kernel with Index=2.0 and
# radius (R_68) of 0.3 degrees
model = {'Index' : 2.0,
        'SpatialModel' : 'Gaussian', 'SpatialWidth' : 0.3 }
maps = gta.residmap('fit1', model=model)

# Generate residual map for a power-law point source with Index=2.0 for
# E > 3.16 GeV
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
maps = gta.residmap('fit1_emin35', model=model, erange=[3.5, None])

# Generate residual maps for a power-law point source with Index=1.5, 2.0, and 2.5
model={'SpatialModel' : 'PointSource'}
maps = []
for index in [1.5, 2.0, 2.5]:
    model['Index'] = index
    maps += [gta.residmap('fit1', model=model)]
```

`residmap()` returns a `maps` dictionary containing `Map` representations of the residual significance and amplitude as well as the smoothed data and model maps. The contents of the output dictionary are described in the following table.

Key	Type	Description
sigma	<code>Map</code>	Residual significance in sigma.
excess	<code>Map</code>	Residual amplitude in counts.
data	<code>Map</code>	Smoothed counts map.
model	<code>Map</code>	Smoothed model map.
files	dict	File paths of the FITS image files generated by this method.
src_dict	dict	Source dictionary with the properties of the convolution kernel.

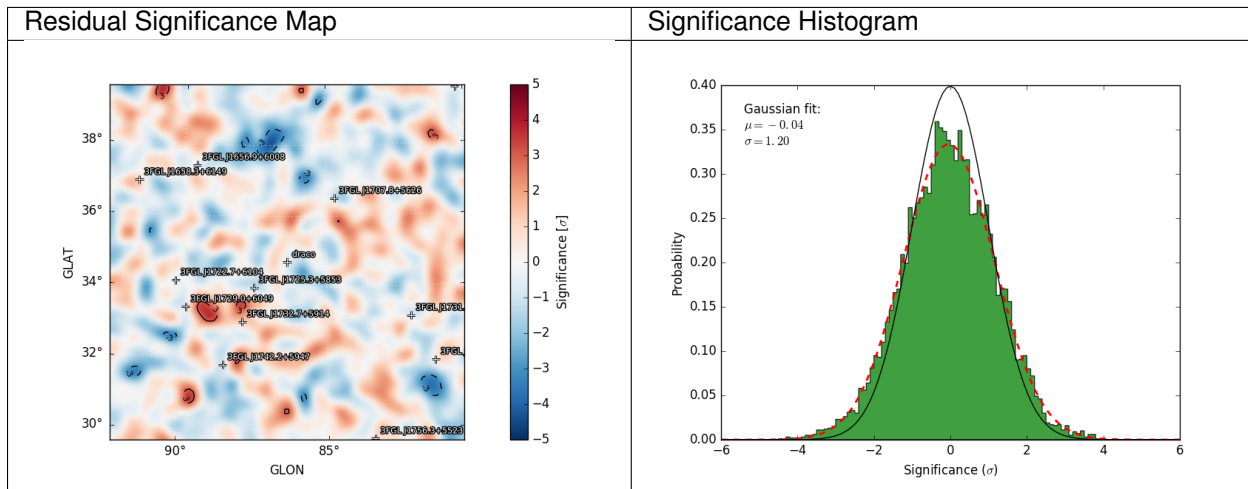
The `write_fits` and `write_npy` options can be used to write the output to a FITS or numpy file. All output files are prepended with the `prefix` argument.

Diagnostic plots can be generated by setting `make_plots=True` or by passing the output dictionary to `make_residmap_plots`:

```
maps = gta.residmap('fit1', model=model, make_plots=True)
gta.plotter.make_residmap_plots(maps, roi=gta.roi)
```

This will generate the following plots:

- `residmap_excess`: Smoothed excess map (data-model).
- `residmap_data`: Smoothed data map.
- `residmap_model`: Smoothed model map.
- `residmap_sigma`: Map of residual significance. The color map is truncated at -5 and 5 sigma with labeled isocontours at 2 sigma intervals indicating values outside of this range.
- `residmap_sigma_hist`: Histogram of significance values for all points in the map. Overplotted are distributions for the best-fit Gaussian and a unit Gaussian.



Configuration

The default configuration of the method is controlled with the `residmap` section of the configuration file. The default configuration can be overridden by passing the option as a `kwargs` argument to the method.

Listing 1.31: `residmap` Options

Option	De-fault	Description
<code>exclude</code>	None	List of sources that will be removed from the model when computing the residual map.
<code>loge_bound</code>	None	Restrict the analysis to an energy range (<code>emin,emax</code>) in $\log_{10}(E/\text{MeV})$ that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either <code>emin/emax</code> are None then only an upper/lower bound on the energy range will be applied.
<code>make_plots</code>	False	Generate diagnostic plots.
<code>model</code>	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
<code>write_fits</code>	True	Write the output to a FITS file.
<code>write_npy</code>	True	Write the output dictionary to a numpy file.

Reference/API

`GTAnalysis.residmap` (*prefix*='', ***kwargs*)

Generate 2-D spatial residual maps using the current ROI model and the convolution kernel defined with the `model` argument.

Parameters

- **prefix** (*str*) – String that will be prefixed to the output residual map files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the residual map. (default : None)
- **loge_bounds** (*list*) – Restrict the analysis to an energy range (*emin*,*emax*) in $\log_{10}(E/\text{MeV})$ that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either *emin*/*emax* are None then only an upper/lower bound on the energy range will be applied. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If `model` is None the test source will be a `PointSource` with an Index 2 power-law spectrum. (default : None)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_numpy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

Returns `maps` – A dictionary containing the `Map` objects for the residual significance and amplitude.

Return type `dict`

Source Finding

`find_sources()` is an iterative source-finding algorithm that uses peak detection on a TS map to find new source candidates. The procedure for adding new sources at each iteration is as follows:

- Generate a TS map for the test source model defined with the `model` argument.
- Identify peaks with $\sqrt{\text{TS}} > \sqrt{\text{ts_threshold}}$ and an angular distance of at least `min_separation` from a higher amplitude peak in the map.
- Order the peaks by TS and add a source at each peak starting from the highest TS peak. Set the source position by fitting a 2D parabola to the log-likelihood surface around the peak maximum. After adding each source, re-fit its spectral parameters.
- Add sources at the N highest peaks up to `N = sources_per_iter`.

Source finding is repeated up to `max_iter` iterations or until no peaks are found in a given iteration. Sources found by the method are added to the model and given designations `PS JXXXX.X+XXXX` according to their position in celestial coordinates.

Examples

```
model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
srcs = gta.find_sources(model=model, sqrt_ts_threshold=5.0,
                        min_separation=0.5)
```

The method for generating the TS maps can be controlled with the `tsmap_fitter` option. TS maps can be generated with either `tsmap()` or `tscube()`.

Reference/API

GTAnalysis.**find_sources** (*prefix*='', ***kwargs*)

An iterative source-finding algorithm that uses likelihood ratio (TS) maps of the region of interest to find new sources. After each iteration a new TS map is generated incorporating sources found in the previous iteration. The method stops when the number of iterations exceeds `max_iter` or no sources exceeding `sqrt_ts_threshold` are found.

Parameters

- **free_params** (*list*) – (default : None)
- **max_iter** (*int*) – Maximum number of source finding iterations. The source finder will continue adding sources until no additional peaks are found or the number of iterations exceeds this number. (default : 5)
- **min_separation** (*float*) – Minimum separation in degrees between sources detected in each iteration. The source finder will look for the maximum peak in the TS map within a circular region of this radius. (default : 1.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across all available cores. (default : False)
- **sources_per_iter** (*int*) – Maximum number of sources that will be added in each iteration. If the number of detected peaks in a given iteration is larger than this number, only the N peaks with the largest TS will be used as seeds for the current iteration. (default : 4)
- **sqrt_ts_threshold** (*float*) – Source threshold in sqrt(TS). Only peaks with sqrt(TS) exceeding this threshold will be used as seeds for new sources. (default : 5.0)
- **tmap_fitter** (*str*) – Set the method for generating the TS map. Valid options are tmap or tscube. (default : tmap)
- **tmap** (*dict*) – Keyword arguments dictionary for tmap method.
- **tscube** (*dict*) – Keyword arguments dictionary for tscube method.

Returns

- **peaks** (*list*) – List of peak objects.
- **sources** (*list*) – List of source objects.

Source Localization

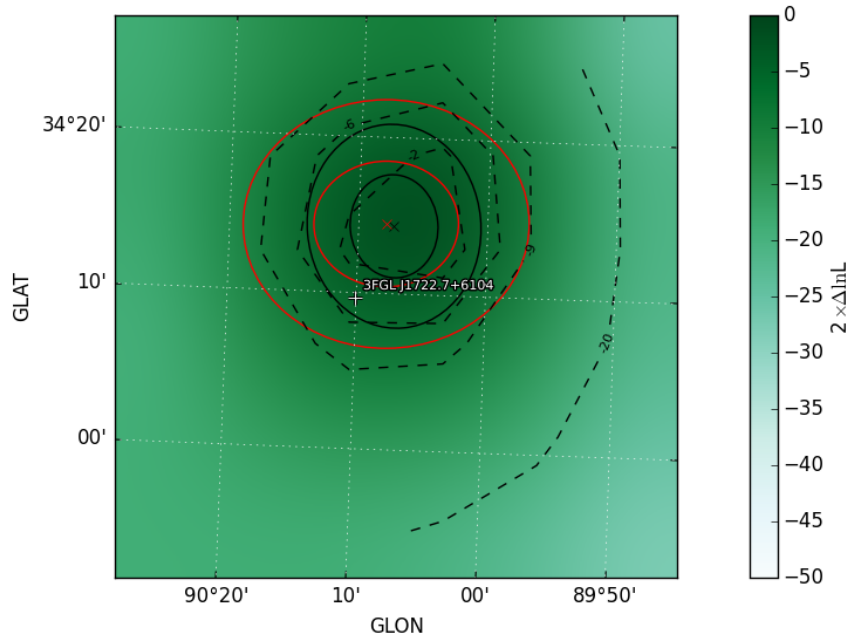
The `localize()` method can be used to spatially localize a source. Localization is performed by scanning the 2D likelihood surface in a local patch around the nominal source position. The current implementation of the localization analysis proceeds in two steps:

- **TS Map Scan:** Obtain a rough estimate of the source position by generating a fast TS Map of the region using the `tmap` method. In this step all background parameters are fixed to their nominal values.
- **Likelihood Scan:** Refine the position of the source by performing a scan of the likelihood surface in a box centered on the best-fit position found with the TS Map method. The size of the search region is set to encompass the 99% positional uncertainty contour. This method uses a full likelihood fit at each point in the likelihood scan and will re-fit all free parameters of the model.

The localization method is executed by passing the name of a source as its argument. The method returns a python dictionary with the best-fit source position and localization errors and also saves this information to the *localization* dictionary of the *Source* object.

```
>>> loc = gta.localize('3FGL J1722.7+6104')
>>> print(loc['ra'], loc['dec'], loc['r68'], loc['r95'])
(260.53164555483784, 61.04493807148745, 0.14384100879403075, 0.23213050350030126)
```

By default the method will save a plot to the working directory with a visualization of the localization contours. The black and red contours show the uncertainty ellipse derived from the TS Map and likelihood scan, respectively.



The default configuration for the localization analysis can be overridden by supplying one or more *kwargs*:

```
# Localize the source and update its properties in the model
# with the localized position
>>> o = gta.extension('sourceA', update=True)
```

The localization method will not profile over any background parameters that were free when the method was executed. One can free background parameters with the *free_background* parameter:

```
# Free a nearby source that may be partially degenerate with the
# source of interest
gta.free_norm('sourceB')
gta.localize('sourceA', free_background=True)
```

The contents of the output dictionary are described in the following table:

Listing 1.32: *localize* Output

Key	Type	Description
ra	float	Right ascension of best-fit position in deg.
dec	float	Declination of best-fit position in deg.
glon	float	Galactic Longitude of best-fit position in deg.
glat	float	Galactic Latitude of best-fit position in deg.
offset	float	Angular offset in deg between the old and new (localized) source positions.
sigma	float	1-sigma positional uncertainty in deg.
r68	float	68% positional uncertainty in deg.
r95	float	95% positional uncertainty in deg.
r99	float	99% positional uncertainty in deg.
sigmax	float	1-sigma uncertainty in deg in longitude.
sigmay	float	1-sigma uncertainty in deg in latitude.
sigma_semimajor	float	1-sigma uncertainty in deg along major axis of uncertainty ellipse.
sigma_semiminor	float	1-sigma uncertainty in deg along minor axis of uncertainty ellipse.
xpix	float	Longitude pixel coordinate of best-fit position.
ypix	float	Latitude pixel coordinate of best-fit position.
theta	float	Position angle of uncertainty ellipse.
eccentricity	float	Eccentricity of uncertainty ellipse defined as $\sqrt{1-b^2/a^2}$.
eccentricity2	float	Eccentricity of uncertainty ellipse defined as $\sqrt{a^2/b^2-1}$.
config	dict	Copy of the input parameters to this method.

Reference/API

GTAnalysis.**localize** (*name*, ****kwargs**)

Find the best-fit position of a source. Localization is performed in two steps. First a TS map is computed centered on the source with half-width set by `dtheta_max`. A fit is then performed to the maximum TS peak in this map. The source position is then further refined by scanning the likelihood in the vicinity of the peak found in the first step. The size of the scan region is set to encompass the 99% positional uncertainty contour as determined from the peak fit.

Parameters

- **name** (*str*) – Source name.
- **dtheta_max** (*float*) – Half-width of the search region in degrees used for the first pass of the localization search. (default : 0.5)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **nstep** (*int*) – Number of steps in longitude/latitude that will be taken when refining the source position. The bounds of the scan range are set to the 99% positional uncertainty as determined from the TS map peak fit. The total number of sampling points will be `nstep**2`. (default : 5)
- **update** (*bool*) – Update the source model with the best-fit position. (default : True)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)

- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns **localize** – Dictionary containing results of the localization analysis.

Return type *dict*

Phased Analysis

Fermipy provides several options to support analysis with selections on pulsar phase. The following examples assume that you already have a phased FT1 file that contains a PULSE_PHASE column with the pulsar phase for each event.

The following examples illustrates the settings for the *gtlike* and *selection* sections of the configuration file that would be used for a single-component ON- or OFF-phase analysis:

```
selection :
  emin : 100
  emax : 316227.76
  zmax  : 90
  evclass : 128
  evtype : 3
  tmin   : 239557414
  tmax   : 428903014
  target : '3FGL J0534.5+2201p'
  phasemin : 0.68
  phasemax : 1.00

gtlike :
  edisp : True
  irfs : 'P8R2_SOURCE_V6'
  edisp_disable : ['isodiff', 'galdiff']
  expscale : 0.32
```

The `gtlike.expscale` parameter defines the correction that should be applied to the nominal exposure to account for the phase selection defined by `selection.phasemin` and `selection.phasemax`. Normally this should be set to the size of the phase selection interval.

To perform a joint analysis of multiple phase selections you can use the *components* section to define separate ON- and OFF-phase components:

```
components:
- selection : {phasemin : 0.68, phasemax: 1.0}
  gtlike    : {expscale : 0.32, src_expscale : {'3FGL J0534.5+2201p':0.0}}
- selection : {phasemin : 0.0 , phasemax: 0.68}
  gtlike    : {expscale : 0.68, src_expscale : {'3FGL J0534.5+2201p':1.0}}
```

The `src_expscale` parameter can be used to define an exposure correction for individual sources. In this example it is used to zero the pulsar component for the OFF-phase selection.

Sensitivity Tools

The `fermipy-flux-sensitivity` script can be used to calculate the LAT detection threshold versus energy. Inputs are the galactic diffuse model and the livetime cube of the observation epoch. The `obs_time_yr` option can be used to rescale the livetime cube to a shorter or longer observation time.

```
$ fermipy-flux-sensitivity --glon=30 --glat=30 --output=flux.fits \
--ltcube=ltcube.fits --galdiff=gll_iem_v06.fits --event_class=P8R2_SOURCE_V6
```

If no livetime cube is provided then the sensitivity will be computed assuming an “ideal” survey-mode operation with uniform exposure over the whole sky and no Earth obscuration or deadtime. By default the flux sensitivity will be calculated for a TS threshold of 25 and at least 3 counts.

The output FITS file contains a table with the flux threshold in each energy bin.

```
from astropy.table import Table
tab = Table.read('flux.fits')
print(tab['e_min'], tab['e_max'], tab['flux'])
```

fermipy package

fermipy.config module

class fermipy.config.ConfigManager

Bases: `object`

static create (*configfile*)

Create a configuration dictionary from a yaml config file. This function will first populate the dictionary with defaults taken from pre-defined configuration files. The configuration dictionary is then updated with the user-defined configuration file. Any settings defined by the user will take precedence over the default settings.

static load (*path*)

class fermipy.config.ConfigSchema (*options=None, **kwargs*)

Bases: `object`

Class encapsulating a configuration schema.

add_option (*name, default_value, helpstr='', otype=None*)

add_section (*name, section*)

create_config (*config=None, validate=True, **kwargs*)

items ()

class fermipy.config.Configurable (*config, **kwargs*)

Bases: `object`

The base class provides common facilities like loading and saving configuration state.

config

Return the configuration dictionary of this class.

configdir

configure (*config, **kwargs*)

classmethod get_config ()

Return a default configuration dictionary for this class.

print_config (*logger, loglevel=None*)

schema

Return the configuration schema of this class.

write_config (*outfile*)

Write the configuration dictionary to an output file.

fermipy.config.cast_config (*config, defaults*)

`fermipy.config.create_default_config(schema)`

Create a configuration dictionary from a schema dictionary. The schema defines the valid configuration keys and their default values. Each element of `schema` should be a tuple/list containing (default value,docstring,type) or a dict containing a nested schema.

`fermipy.config.update_from_schema(cfg, cfgin, schema)`

Update configuration dictionary `cfg` with the contents of `cfgin` using the `schema` dictionary to determine the valid input keys.

Parameters

- **cfg** (*dict*) – Configuration dictionary to be updated.
- **cfgin** (*dict*) – New configuration dictionary that will be merged with `cfg`.
- **schema** (*dict*) – Configuration schema defining the valid configuration keys and their types.

Returns `cfgout`

Return type `dict`

`fermipy.config.validate_config(config, defaults, section=None)`

`fermipy.config.validate_from_schema(cfg, schema, section=None)`

`fermipy.config.validate_option(opt_name, opt_val, schema_type)`

fermipy.defaults module

`fermipy.defaults.make_default_dict(d)`

fermipy.gtanalysis module

class `fermipy.gtanalysis.GTAnalysis` (*config*, ***kwargs*)

Bases: `fermipy.config.Configurable`, `fermipy.sed.SEDGenerator`,
`fermipy.residmap.ResidMapGenerator`, `fermipy.tsmap.TSMapGenerator`,
`fermipy.tsmap.TSCubeGenerator`, `fermipy.sourcefind.SourceFind`,
`fermipy.extension.ExtensionFit`, `fermipy.lightcurve.LightCurve`

High-level analysis interface that manages a set of analysis component objects. Most of the functionality of the Fermipy package is provided through the methods of this class. The class constructor accepts a dictionary that defines the configuration for the analysis. Keyword arguments to the constructor can be used to override parameters in the configuration dictionary.

__delattr__

`x.__delattr__('name') <==> del x.name`

__format__ ()

default object formatter

__getattr__

`x.__getattr__('name') <==> x.name`

__hash__

__reduce__ ()

helper for pickle

__reduce_ex__ ()

helper for pickle

`__repr__`

`__setattr__`

`x.__setattr__('name', value) <==> x.name = value`

`__sizeof__()` → int

size of object in memory, in bytes

`__str__`

add_gauss_prior (*name*, *parName*, *mean*, *sigma*)

add_source (*name*, *src_dict*, *free=None*, *init_source=True*, *save_source_maps=True*,
use_pylike=True, *use_single_psf=False*, ***kwargs*)

Add a source to the ROI model. This function may be called either before or after `setup`.

Parameters

- **name** (*str*) – Source name.
- **src_dict** (dict or *Source* object) – Dictionary or source object defining the source properties (coordinates, spectral parameters, etc.).
- **free** (*bool*) – Initialize the source with a free normalization parameter.
- **use_pylike** (*bool*) – Create source maps with pyLikelihood.
- **use_single_psf** (*bool*) – Use the PSF model calculated for the ROI center. If false then a new model will be generated using the position of the source.

add_sources_from_roi (*names*, *roi*, *free=False*, ***kwargs*)

Add multiple sources to the current ROI model copied from another ROI model.

Parameters

- **names** (*list*) – List of str source names to add.
- **roi** (*ROIModel* object) – The roi model from which to add sources.
- **free** (*bool*) – Initialize the source with a free normalization paramter.

bowtie (*name*, *fd=None*, *loge=None*)

Generate a spectral uncertainty band (bowtie) for the given source. This will create an uncertainty band on the differential flux as a function of energy by propagating the errors on the global fit parameters. Note that this band only reflects the uncertainty for parameters that are currently free in the model.

Parameters

- **name** (*str*) – Source name.
- **fd** (*FluxDensity*) – Flux density object. If this parameter is None then one will be created.
- **loge** (*array-like*) – Sequence of energies in log10(E/MeV) at which the flux band will be evaluated.

cleanup ()

clone (*config*, ***kwargs*)

Make a clone of this analysis instance.

components

Return the list of analysis components.

config

Return the configuration dictionary of this class.

configdir

configure (*config*, ***kwargs*)

constrain_norms (*srcNames*, *cov_scale=1.0*)

Constrain the normalizations of one or more sources by adding gaussian priors with sigma equal to the parameter error times a scaling factor.

counts_map ()

Return a [Map](#) representation of the counts map.

Returns *map*

Return type [Map](#)

static create (*infile*, *config=None*)

Create a new instance of GTAnalysis from an analysis output file generated with `write_roi`. By default the new instance will inherit the configuration of the saved analysis instance. The configuration may be overridden by passing a configuration file path with the `config` argument.

Parameters

- **infile** (*str*) – Path to the ROI results file.
- **config** (*str*) – Path to a configuration file. This will override the configuration in the ROI results file.

create_roi_table ()

defaults = {'sourcefind': {'tmap_fitter': ('tmap', 'Set the method for generating the TS map. Valid options are tmap

delete_source (*name*, *save_template=True*, *delete_source_map=False*, *build_fixed_wts=True*, ***kwargs*)

Delete a source from the ROI model.

Parameters

- **name** (*str*) – Source name.
- **save_template** (*bool*) – Keep the SpatialMap FITS template associated with this source.
- **delete_source_map** (*bool*) – Delete the source map associated with this source from the source maps file.

Returns *src* – The deleted source object.

Return type [Model](#)

delete_sources (*cuts=None*, *distance=None*, *skydir=None*, *minmax_ts=None*, *minmax_npred=None*, *exclude=None*, *square=False*)

Delete sources in the ROI model satisfying the given selection criteria.

Parameters

- **cuts** (*dict*) – Dictionary of [min,max] selections on source properties.
- **distance** (*float*) – Cut on angular distance from `skydir`. If None then no selection will be applied.
- **skydir** ([SkyCoord](#)) – Reference sky coordinate for `distance` selection. If None then the distance selection will be applied with respect to the ROI center.
- **minmax_ts** (*list*) – Free sources that have TS in the range [min,max]. If either min or max are None then only a lower (upper) bound will be applied. If this parameter is none no selection will be applied.

- **minmax_npred** (*list*) – Free sources that have npred in the range [min,max]. If either min or max are None then only a lower (upper) bound will be applied. If this parameter is none no selection will be applied.
- **square** (*bool*) – Switch between applying a circular or square (ROI-like) selection on the maximum projected distance from the ROI center.

Returns *srcs* – A list of *Model* objects.

Return type *list*

energies

Return the energy bin edges in MeV.

enumbins

Return the number of energy bins.

extension (*name*, ***kwargs*)

Test this source for spatial extension with the likelihood ratio method (TS_ext). This method will substitute an extended spatial model for the given source and perform a one-dimensional scan of the spatial extension parameter over the range specified with the width parameters. The 1-D profile likelihood is then used to compute the best-fit value, upper limit, and TS for extension. The background parameters that will be simultaneously profiled when performing the spatial scan can be controlled with the *free_background* and *free_radius* options. By default the position of the source will be fixed to its current position. A simultaneous fit to position and extension can be performed by setting *fit_position* to True.

Parameters

- **name** (*str*) – Source name.
- **fit_position** (*bool*) – Perform a simultaneous fit to the source position and extension. (default : False)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **psf_scale_fn** (*tuple*) – Tuple of two vectors (logE,f) defining an energy-dependent PSF scaling function that will be applied when building spatial models for the source of interest. The tuple (logE,f) defines the fractional corrections f at the sequence of energies $\log E = \log_{10}(E/\text{MeV})$ where $f=0$ corresponds to no correction. The correction function $f(E)$ is evaluated by linearly interpolating the fractional correction factors f in $\log(E)$. The corrected PSF is given by $P'(x;E) = P(x/(1+f(E));E)$ where x is the angular separation. (default : None)
- **save_model_map** (*bool*) – Save model counts cubes for the best-fit model of extension. (default : False)
- **spatial_model** (*str*) – Spatial model that will be used to test the source extension. The spatial scale parameter of the model will be set such that the 68% containment radius of the model is equal to the width parameter. (default : RadialGaussian)
- **sqrt_ts_threshold** (*float*) – Threshold on $\sqrt{\text{TS}_{\text{ext}}}$ that will be applied when *update* is True. If None then no threshold is applied. (default : None)

- **update** (*bool*) – Update this source with the best-fit model for spatial extension if `TS_ext > tsext_threshold`. (default : False)
- **width** (*list*) – Sequence of values in degrees for the likelihood scan over spatial extension (68% containment radius). If this argument is None then the scan points will be determined from `width_min/width_max/width_nstep`. (default : None)
- **width_max** (*float*) – Maximum value in degrees for the likelihood scan over spatial extent. (default : 1.0)
- **width_min** (*float*) – Minimum value in degrees for the likelihood scan over spatial extent. (default : 0.01)
- **width_nstep** (*int*) – Number of scan points between `width_min` and `width_max`. Scan points will be spaced evenly on a logarithmic scale between `width_min` and `width_max`. (default : 21)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns `extension` – Dictionary containing results of the extension analysis. The same dictionary is also saved to the dictionary of this source under ‘extension’.

Return type `dict`

files

find_sources (*prefix='', **kwargs*)

An iterative source-finding algorithm that uses likelihood ratio (TS) maps of the region of interest to find new sources. After each iteration a new TS map is generated incorporating sources found in the previous iteration. The method stops when the number of iterations exceeds `max_iter` or no sources exceeding `sqrts_threshold` are found.

Parameters

- **free_params** (*list*) – (default : None)
- **max_iter** (*int*) – Maximum number of source finding iterations. The source finder will continue adding sources until no additional peaks are found or the number of iterations exceeds this number. (default : 5)
- **min_separation** (*float*) – Minimum separation in degrees between sources detected in each iteration. The source finder will look for the maximum peak in the TS map within a circular region of this radius. (default : 1.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across all available cores. (default : False)
- **sources_per_iter** (*int*) – Maximum number of sources that will be added in each iteration. If the number of detected peaks in a given iteration is larger than this number, only the N peaks with the largest TS will be used as seeds for the current iteration. (default : 4)
- **sqrts_threshold** (*float*) – Source threshold in $\sqrt{\text{TS}}$. Only peaks with $\sqrt{\text{TS}}$ exceeding this threshold will be used as seeds for new sources. (default : 5.0)
- **tmap_fitter** (*str*) – Set the method for generating the TS map. Valid options are `tmap` or `tscube`. (default : `tmap`)

- **tmap** (*dict*) – Keyword arguments dictionary for tmap method.
- **tscube** (*dict*) – Keyword arguments dictionary for tscube method.

Returns

- **peaks** (*list*) – List of peak objects.
- **sources** (*list*) – List of source objects.

fit (*update=True, **kwargs*)

Run the likelihood optimization. This will execute a fit of all parameters that are currently free in the model and update the characteristics of the corresponding model components (TS, npred, etc.). The fit will be repeated N times (set with the `retries` parameter) until a fit quality greater than or equal to `min_fit_quality` and a fit status code of 0 is obtained. If the fit does not succeed after N retries then all parameter values will be reverted to their state prior to the execution of the fit.

Parameters

- **update** (*bool*) – Update the model dictionary for all sources with free parameters.
- **tol** (*float*) – Set the optimizer tolerance.
- **verbosity** (*int*) – Set the optimizer output level.
- **optimizer** (*str*) – Set the likelihood optimizer (e.g. MINUIT or NEWMINUIT).
- **retries** (*int*) – Set the number of times to rerun the fit when the fit quality is < 3.
- **min_fit_quality** (*int*) – Set the minimum fit quality. If the fit quality is smaller than this value then all model parameters will be restored to their values prior to the fit.
- **reoptimize** (*bool*) – Refit background sources when updating source properties (TS and likelihood profiles).

Returns fit – Dictionary containing diagnostic information from the fit (fit quality, parameter covariances, etc.).

Return type *dict*

fit_correlation ()

free_index (*name, free=True, **kwargs*)

Free/Fix index of a source.

Parameters

- **name** (*str*) – Source name.
- **free** (*bool*) – Choose whether to free (`free=True`) or fix (`free=False`).

free_norm (*name, free=True, **kwargs*)

Free/Fix normalization of a source.

Parameters

- **name** (*str*) – Source name.
- **free** (*bool*) – Choose whether to free (`free=True`) or fix (`free=False`).

free_parameter (*name, par, free=True*)

free_shape (*name, free=True, **kwargs*)

Free/Fix shape parameters of a source.

Parameters

- **name** (*str*) – Source name.

- **free** (*bool*) – Choose whether to free (free=True) or fix (free=False).

free_source (*name*, *free*=True, *pars*=None, ***kwargs*)

Free/Fix parameters of a source.

Parameters

- **name** (*str*) – Source name.
- **free** (*bool*) – Choose whether to free (free=True) or fix (free=False) source parameters.
- **pars** (*list*) – Set a list of parameters to be freed/fixed for this source. If none then all source parameters will be freed/fixed with the exception of those defined in the skip_pars list.

free_sources (*free*=True, *pars*=None, *cuts*=None, *distance*=None, *skydir*=None, *minmax_ts*=None, *minmax_npred*=None, *exclude*=None, *square*=False, ***kwargs*)

Free or fix sources in the ROI model satisfying the given selection. When multiple selections are defined, the selected sources will be those satisfying the logical AND of all selections (e.g. distance < X && minmax_ts[0] < ts < minmax_ts[1] && ...).

Parameters

- **free** (*bool*) – Choose whether to free (free=True) or fix (free=False) source parameters.
- **pars** (*list*) – Set a list of parameters to be freed/fixed for each source. If none then all source parameters will be freed/fixed. If pars='norm' then only normalization parameters will be freed.
- **cuts** (*dict*) – Dictionary of [min,max] selections on source properties.
- **distance** (*float*) – Cut on angular distance from skydir. If None then no selection will be applied.
- **skydir** (*SkyCoord*) – Reference sky coordinate for distance selection. If None then the distance selection will be applied with respect to the ROI center.
- **minmax_ts** (*list*) – Free sources that have TS in the range [min,max]. If either min or max are None then only a lower (upper) bound will be applied. If this parameter is none no selection will be applied.
- **minmax_npred** (*list*) – Free sources that have npred in the range [min,max]. If either min or max are None then only a lower (upper) bound will be applied. If this parameter is none no selection will be applied.
- **exclude** (*list*) – Names of sources that will be excluded from the selection.
- **square** (*bool*) – Switch between applying a circular or square (ROI-like) selection on the maximum projected distance from the ROI center.

Returns *srcs* – A list of *Model* objects.

Return type *list*

free_sources_by_name (*names*, *free*=True, *pars*=None, ***kwargs*)

Free all sources with names matching *names*.

Parameters

- **names** (*list*) – List of source names.
- **free** (*bool*) – Choose whether to free (free=True) or fix (free=False) source parameters.
- **pars** (*list*) – Set a list of parameters to be freed/fixed for each source. If none then all source parameters will be freed/fixed. If pars='norm' then only normalization parameters will be freed.

Returns `srcs` – A list of `Model` objects.

Return type `list`

generate_model (*model_name=None*)

Generate model maps for all components. `model_name` should be a unique identifier for the model. If `model_name` is `None` then the model maps will be generated using the current parameters of the ROI.

get_config ()

Return a default configuration dictionary for this class.

get_free_param_vector ()

get_free_source_params (*name*)

get_norm (*name*)

get_params (*freeonly=False*)

get_source_dnde (*name*)

Return differential flux distribution of a source. For sources with FileFunction spectral type this returns the internal differential flux array.

Returns

- **loge** (`ndarray`) – Array of energies at which the differential flux is evaluated ($\log_{10}(E/\text{MeV})$).
- **dnde** (`ndarray`) – Array of differential flux values ($\text{cm}^{-2} \text{ s}^{-1} \text{ MeV}^{-1}$) evaluated at energies in `loge`.

get_source_name (*name*)

Return the name of a source as it is defined in the pyLikelihood model object.

get_sources (*cuts=None, distance=None, skydir=None, minmax_ts=None, minmax_npred=None, exclude=None, square=False*)

Retrieve list of sources in the ROI satisfying the given selections.

Returns `srcs` – A list of `Model` objects.

Return type `list`

get_src_model (*name, paramsonly=False, reoptimize=False, npts=None, **kwargs*)

Compose a dictionary for a source with the current best-fit parameters.

Parameters

- **name** (`str`) –
- **paramsonly** (`bool`) – Skip computing TS and likelihood profile.
- **reoptimize** (`bool`) – Re-fit background parameters in likelihood scan.
- **npts** (`int`) – Number of points for likelihood scan.

Returns `src_dict`

Return type `dict`

lightcurve (*name, **kwargs*)

Generate a lightcurve for the named source. The function will complete the basic analysis steps for each bin and perform a likelihood fit for each bin. Extracted values (along with errors) are Integral Flux, spectral model, Spectral index, TS value, pred. # of photons.

Parameters

- **name** (`str`) – source name

- **binsz** (*float*) – Set the lightcurve bin size in seconds. (default : 86400.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_params** (*list*) – Set the parameters of the source of interest that will be re-fit in each time bin. If this list is empty then all parameters will be freed. (default : None)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **free_sources** (*list*) – List of sources to be freed. These sources will be added to the list of sources satisfying the free_radius selection. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **nbins** (*int*) – Set the number of lightcurve bins. The total time range will be evenly split into this number of time bins. (default : None)
- **time_bins** (*list*) – Set the lightcurve bin edge sequence in MET. This option takes precedence over binsz and nbins. (default : None)
- **use_local_ltcube** (*bool*) – (default : True)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

Returns **LightCurve** – Dictionary containing output of the LC analysis

Return type `dict`

like

Return the global likelihood object.

load_roi (*infile*, *reload_sources=False*)

This function reloads the analysis state from a previously saved instance generated with `write_roi`.

Parameters

- **infile** (*str*) –
- **reload_sources** (*bool*) – Regenerate source maps for non-diffuse sources.

load_xml (*xmlfile*)

Load model definition from XML.

Parameters **xmlfile** (*str*) – Name of the input XML file.

localize (*name*, ***kwargs*)

Find the best-fit position of a source. Localization is performed in two steps. First a TS map is computed centered on the source with half-width set by `dtheta_max`. A fit is then performed to the maximum TS peak in this map. The source position is then further refined by scanning the likelihood in the vicinity of the peak found in the first step. The size of the scan region is set to encompass the 99% positional uncertainty contour as determined from the peak fit.

Parameters

- **name** (*str*) – Source name.
- **dtheta_max** (*float*) – Half-width of the search region in degrees used for the first pass of the localization search. (default : 0.5)

- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **nstep** (*int*) – Number of steps in longitude/latitude that will be taken when refining the source position. The bounds of the scan range are set to the 99% positional uncertainty as determined from the TS map peak fit. The total number of sampling points will be $nstep*2$. (default : 5)
- **update** (*bool*) – Update the source model with the best-fit position. (default : True)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns *localize* – Dictionary containing results of the localization analysis.

Return type *dict*

log_energies

Return the energy bin edges in $\log_{10}(E/\text{MeV})$.

loge_bounds

Current analysis energy bounds in $\log_{10}(E/\text{MeV})$.

loglevel

Return the default loglevel.

make_plots (*prefix*, *mcube_map=None*, ***kwargs*)

Make diagnostic plots using the current ROI model.

model_counts_map (*name=None*, *exclude=None*)

Return the model counts map for a single source, a list of sources, or for the sum of all sources in the ROI. The exclude parameter can be used to exclude one or more components when generating the model map.

Parameters

- **name** (*str or list of str*) – Parameter controlling the set of sources for which the model counts map will be calculated. If name=None the model map will be generated for all sources in the ROI.
- **exclude** (*str or list of str*) – List of sources that will be excluded when calculating the model map.

Returns *map*

Return type *Map*

model_counts_spectrum (*name*, *logemin=None*, *logemax=None*, *summed=False*)

Return the predicted number of model counts versus energy for a given source and energy range. If summed=True return the counts spectrum summed over all components otherwise return a list of model spectra.

npix

Return the number of energy bins.

optimize (**kwargs)

Iteratively optimize the ROI model. The optimization is performed in three sequential steps:

- Free the normalization of the N largest components (as determined from NPred) that contain a fraction `npred_frac` of the total predicted counts in the model and perform a simultaneous fit of the normalization parameters of these components.
- Individually fit the normalizations of all sources that were not included in the first step in order of their `npred` values. Skip any sources that have `NPred < npred_threshold`.
- Individually fit the shape and normalization parameters of all sources with `TS > shape_ts_threshold` where `TS` is determined from the first two steps of the ROI optimization.

To ensure that the model is fully optimized this method can be run multiple times.

Parameters

- **npred_frac** (*float*) – Threshold on the fractional number of counts in the N largest components in the ROI. This parameter determines the set of sources that are fit in the first optimization step.
- **npred_threshold** (*float*) – Threshold on the minimum number of counts of individual sources. This parameter determines the sources that are fit in the second optimization step.
- **shape_ts_threshold** (*float*) – Threshold on source `TS` used for determining the sources that will be fit in the third optimization step.
- **max_free_sources** (*int*) – Maximum number of sources that will be fit simultaneously in the first optimization step.
- **skip** (*list*) – List of str source names to skip while optimizing.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

outdir

Return the analysis output directory.

plotter

Return the plotter instance.

print_config (*logger*, *loglevel=None*)

print_model (*loglevel=20*)

print_params (*allpars=False*, *loglevel=20*)

Print information about the model parameters (values, errors, bounds, scale).

print_roi (*loglevel=20*)

Print information about the spectral and spatial properties of the ROI (sources, diffuse components).

profile (*name*, *parName*, *logemin=None*, *logemax=None*, *reoptimize=False*, *xvals=None*, *npts=None*, *savestate=True*, **kwargs)

Profile the likelihood for the given source and parameter.

Parameters

- **name** (*str*) – Source name.
- **parName** (*str*) – Parameter name.
- **reoptimize** (*bool*) – Re-fit nuisance parameters at each step in the scan. Note that enabling this option will only re-fit parameters that were free when the method was executed.

Returns `Inlprofile` – Dictionary containing results of likelihood scan.

Return type `dict`

profile_norm (*name*, *logemin=None*, *logemax=None*, *reoptimize=False*, *xvals=None*, *npts=None*, *fix_shape=True*, *savestate=True*, ***kwargs*)

Profile the normalization of a source.

Parameters

- **name** (*str*) – Source name.
- **reoptimize** (*bool*) – Re-optimize free parameters in the model at each point in the profile likelihood scan.

projtype

Return the type of projection to use

reload_source (*name*, *init_source=True*)

Delete and reload a source in the model. This will update the spatial model of this source to the one defined in the XML model.

reload_sources (*names*, *init_source=True*)

remove_prior (*name*, *parName*)

remove_priors ()

Clear all priors.

residmap (*prefix=''*, ***kwargs*)

Generate 2-D spatial residual maps using the current ROI model and the convolution kernel defined with the `model` argument.

Parameters

- **prefix** (*str*) – String that will be prefixed to the output residual map files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the residual map. (default : None)
- **log_bounds** (*list*) – Restrict the analysis to an energy range (*emin*, *emax*) in $\log_{10}(E/\text{MeV})$ that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either *emin*/*emax* are None then only an upper/lower bound on the energy range will be applied. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_numpy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

Returns `maps` – A dictionary containing the `Map` objects for the residual significance and amplitude.

Return type `dict`

roi

Return the ROI object.

scale_parameter (*name*, *par*, *scale*)

schema

Return the configuration schema of this class.

sed (*name*, ***kwargs*)

Generate a spectral energy distribution (SED) for a source. This function will fit the normalization of the source in each energy bin. By default the SED will be generated with the analysis energy bins but a custom binning can be defined with the `log_e_bins` parameter.

Parameters

- **name** (*str*) – Source name.
- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **log_e_bins** (*ndarray*) – Sequence of energies in $\log_{10}(E/\text{MeV})$ defining the edges of the energy bins. If this argument is `None` then the analysis energy bins will be used. The energies in this sequence must align with the bin edges of the underlying analysis instance.
- **bin_index** (*float*) – Spectral index that will be used when fitting the energy distribution within an energy bin. (default : 2.0)
- **cov_scale** (*float*) – Scale factor that sets the strength of the prior on nuisance parameters that are free. Setting this to `None` disables the prior. (default : 3.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If `True` then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : `False`)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If `None` then no sources will be freed. (default : `None`)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : `False`)
- **ul_confidence** (*float*) – Confidence level for flux upper limit. (default : 0.95)
- **use_local_index** (*bool*) – Use a power-law approximation to the shape of the global spectrum in each bin. If this is `false` then a constant index set to `bin_index` will be used. (default : `False`)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : `True`)
- **write_numpy** (*bool*) – Write the output dictionary to a numpy file. (default : `True`)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns `sed` – Dictionary containing output of the SED analysis.

Return type `dict`

set_edisp_flag (*name*, *flag=True*)

Enable or disable the energy dispersion correction for the given source.

set_energy_range (*logemin*, *logemax*)

Set the energy bounds of the analysis. This restricts the evaluation of the likelihood to the data that falls in this range. Input values will be rounded to the closest bin edge value. If either argument is `None` then the lower or upper bound of the analysis instance will be used.

Parameters

- **logemin** (*float*) – Lower energy bound in $\log_{10}(E/\text{MeV})$.
- **logemax** (*float*) – Upper energy bound in $\log_{10}(E/\text{MeV})$.

Returns `eminmax` – Minimum and maximum energy in $\log_{10}(E/\text{MeV})$.

Return type `array`

set_free_param_vector (*free*)

set_log_level (*level*)

set_norm (*name, value, update_source=True*)

set_norm_bounds (*name, bounds*)

set_norm_scale (*name, value*)

set_parameter (*name, par, value, true_value=True, scale=None, bounds=None, update_source=True*)

Update the value of a parameter. Parameter bounds will automatically be adjusted to encompass the new parameter value.

Parameters

- **name** (*str*) – Source name.
- **par** (*str*) – Parameter name.
- **value** (*float*) – Parameter value. By default this argument should be the unscaled (True) parameter value.
- **scale** (*float*) – Parameter scale (optional). Value argument is interpreted with respect to the scale parameter if it is provided.
- **update_source** (*bool*) – Update the source dictionary for the object.

set_parameter_bounds (*name, par, bounds*)

Set the bounds of a parameter.

Parameters

- **name** (*str*) – Source name.
- **par** (*str*) – Parameter name.
- **bounds** (*list*) – Upper and lower bound.

set_parameter_scale (*name, par, scale*)

Update the scale of a parameter while keeping its value constant.

set_source_dnde (*name, dnde, update_source=True*)

Set the differential flux distribution of a source with the FileFunction spectral type.

Parameters

- **name** (*str*) – Source name.
- **dnde** (*ndarray*) – Array of differential flux values ($\text{cm}^{-2} \text{ s}^{-1} \text{ MeV}^{-1}$).

set_source_morphology (*name, **kwargs*)

Set the spatial model of a source.

Parameters

- **name** (*str*) – Source name.
- **spatial_model** (*str*) – Spatial model name (PointSource, RadialGaussian, etc.).
- **spatial_pars** (*dict*) – Dictionary of spatial parameters (optional).
- **use_cache** (*bool*) – Generate the spatial model by interpolating the cached source map.
- **use_pylike** (*bool*) –

set_source_spectrum (*name*, *spectrum_type*='PowerLaw', *spectrum_pars*=None, *update_source*=True)

Set the spectral model of a source. This function can be used to change the spectral type of a source or modify its spectral parameters. If called with *spectrum_type*='FileFunction' and *spectrum_pars*=None, the source spectrum will be replaced with a FileFunction with the same differential flux distribution as the original spectrum.

Parameters

- **name** (*str*) – Source name.
- **spectrum_type** (*str*) – Spectrum type (PowerLaw, etc.).
- **spectrum_pars** (*dict*) – Dictionary of spectral parameters (optional).
- **update_source** (*bool*) – Recompute all source characteristics (flux, TS, NPred) using the new spectral model of the source.

setup (*init_sources*=True, *overwrite*=False, ***kwargs*)

Run pre-processing for each analysis component and construct a joint likelihood object. This function performs the following tasks: data selection (gtselect, gtmktime), data binning (gtbin), and model generation (gtexpcube2, gtsrcmaps).

Parameters

- **init_sources** (*bool*) – Choose whether to compute properties (flux, TS, etc.) for individual sources.
- **overwrite** (*bool*) – Run all pre-processing steps even if the output file of that step is present in the working directory. By default this function will skip any steps for which the output file already exists.

simulate_roi (*name*=None, *randomize*=True, *restore*=False)

Generate a simulation of the ROI using the current best-fit model and replace the data counts cube with this simulation. The simulation is created by generating an array of Poisson random numbers with expectation values drawn from the model cube of the binned analysis instance. This function will update the counts cube both in memory and in the source map file. The counts cube can be restored to its original state by calling this method with *restore* = True.

Parameters

- **name** (*str*) – Name of the model component to be simulated. If None then the whole ROI will be simulated.
- **restore** (*bool*) – Restore the data counts cube to its original state.

simulate_source (*src_dict*=None)

Inject simulated source counts into the data.

Parameters **src_dict** (*dict*) – Dictionary defining the spatial and spectral properties of the source that will be injected.

stage_input ()

Copy input files to working directory.

stage_output ()

Copy data products to final output directory.

tmax

Return the MET time for the end of the observation.

tmin

Return the MET time for the start of the observation.

tscube (*prefix*='', ***kwargs*)

Generate a spatial TS map for a source component with properties defined by the `model` argument. This method uses the `gttscube` ST application for source fitting and will simultaneously fit the test source normalization as well as the normalizations of any background components that are currently free. The output of this method is a dictionary containing `Map` objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **do_sed** (*bool*) – Compute the energy bin-by-bin fits.
- **nnorm** (*int*) – Number of points in the likelihood v. normalization scan.
- **norm_sigma** (*float*) – Number of sigma to use for the scan range.
- **tol** (*float*) – Criteria for fit convergence (estimated vertical distance to min < tol).
- **tol_type** (*int*) – Absolute (0) or relative (1) criteria for convergence.
- **max_iter** (*int*) – Maximum number of iterations for the Newton's method fitter
- **remake_test_source** (*bool*) – If true, recomputes the test source image (otherwise just shifts it)
- **st_scan_level** (*int*) –
- **make_plots** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write a FITS file with the results of the analysis.

Returns `maps` – A dictionary containing the `Map` objects for TS and source amplitude.

Return type `dict`

tsmap (*prefix*='', ***kwargs*)

Generate a spatial TS map for a source component with properties defined by the `model` argument. The TS map will have the same geometry as the ROI. The output of this method is a dictionary containing `Map` objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

This method uses a simplified likelihood fitting implementation that only fits for the normalization of the test source. Before running this method it is recommended to first optimize the ROI model (e.g. by running `optimize()`).

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the TS map. (default : None)
- **log_bounds** (*list*) – Restrict the analysis to an energy range (`emin,emax`) in $\log_{10}(E/\text{MeV})$ that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either `emin/emax` are None then only an upper/lower bound on the energy range will be applied. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)

- **max_kernel_radius** (*float*) – Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy. (default : 3.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across all available cores. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

Returns **tmap** – A dictionary containing the [Map](#) objects for TS and source amplitude.

Return type **dict**

unzero_source (*name*, ***kwargs*)

update_source (*name*, *paramsonly=False*, *reoptimize=False*, ***kwargs*)

Update the dictionary for this source.

Parameters

- **name** (*str*) –
- **paramsonly** (*bool*) –
- **reoptimize** (*bool*) – Re-fit background parameters in likelihood scan.

workdir

Return the analysis working directory.

write_config (*outfile*)

Write the configuration dictionary to an output file.

write_fits (*fitsfile*)

write_model_map (*model_name*, *name=None*)

Save the counts model map to a FITS file.

Parameters

- **model_name** (*str*) – String that will be append to the name of the output file.
- **name** (*str*) – Name of the component.

write_roi (*outfile=None*, *save_model_map=False*, *fmt='npz'*, ***kwargs*)

Write current state of the analysis to a file. This method writes an XML model definition, a ROI dictionary, and a FITS source catalog file. A previously saved analysis state can be reloaded from the ROI dictionary file with the [load_roi](#) method.

Parameters

- **outfile** (*str*) – String prefix of the output files. The extension of this string will be stripped when generating the XML, YAML and npy filenames.
- **make_plots** (*bool*) – Generate diagnostic plots.
- **save_model_map** (*bool*) – Save the current counts model to a FITS file.
- **fmt** (*str*) – Set the output file format (yaml or npy).

write_xml (*xmlfile*)

Save current model definition as XML file.

Parameters **xmlfile** (*str*) – Name of the output XML file.

zero_source (*name*, ***kwargs*)

fermipy.logger module

class `fermipy.logger.Logger`

Bases: `object`

This class provides helper functions which facilitate creating instances of the built-in logger class.

static get (*name*, *logfile*, *loglevel=10*)

Create a python logger instance.

Parameters **name** (*str*) – Logger name.

static setup (*config=None*, *logfile=None*)

This method sets up the default configuration of the logger. Once this method is called all subsequent instances Logger instances will inherit this configuration.

class `fermipy.logger.StreamLogger` (*name='stdout'*, *logfile=None*, *quiet=True*)

Bases: `object`

File-like object to log stdout/stderr using the `logging` module.

close()

flush()

write (*msg*, *level=10*)

`fermipy.logger.log_level` (*level*)

This is a function that returns a python like level from a HEASOFT like level.

fermipy.roi_model module

class `fermipy.roi_model.CompositeSource` (*name*, *data*)

Bases: `fermipy.roi_model.Model`

diffuse

nested_sources

write_xml (*root*)

class `fermipy.roi_model.IsoSource` (*name*, *data*)

Bases: `fermipy.roi_model.Model`

diffuse

filefunction

write_xml (*root*)

class `fermipy.roi_model.MapCubeSource` (*name*, *data*)

Bases: `fermipy.roi_model.Model`

diffuse

mapcube

write_xml (*root*)

```
class fermipy.roi_model.Model(name, data)
```

Bases: `object`

Base class for point-like and diffuse source components. This class is a container for spectral and spatial parameters as well as other source properties such as TS, Npred, and location within the ROI.

add_name(name)

add_to_table(tab)

assoc

check_cuts(cuts)

static create_from_dict(src_dict, roi_skydir=None, rescale=False)

data

get_catalog_dict()

get_norm()

items()

name

names

params

psf_scale_fn

set_name(name, names=None)

set_psf_scale_fn(fn)

set_spectral_pars(spectral_pars)

spatial_pars

spectral_pars

update_data(d)

update_from_source(src)

update_spectral_pars(spectral_pars)

```
class fermipy.roi_model.ROIModel(config=None, **kwargs)
```

Bases: `fermipy.config.Configurable`

This class is responsible for managing the ROI model (both sources and diffuse components). Source catalogs can be read from either FITS or XML files. Individual components are represented by instances of `Model` and can be accessed by name using the bracket operator.

- Create an ROI with all 3FGL sources and print a summary of its contents:

```
>>> skydir = astropy.coordinates.SkyCoord(0.0,0.0,unit='deg')
>>> roi = ROIModel({'catalogs' : ['3FGL'],'src_roiwidth' : 10.0},skydir=skydir)
>>> print(roi)
```

name	SpatialModel	SpectrumType	offset	ts	npred
3FGL J2357.3-0150	PointSource	PowerLaw	1.956	nan	0.0
3FGL J0006.2+0135	PointSource	PowerLaw	2.232	nan	0.0
3FGL J0016.3-0013	PointSource	PowerLaw	4.084	nan	0.0
3FGL J0014.3-0455	PointSource	PowerLaw	6.085	nan	0.0

- Print a summary of an individual source

```
>>> print(roi['3FGL J0006.2+0135'])
Name          : 3FGL J0006.2+0135
Associations   : ['3FGL J0006.2+0135']
RA/DEC         :      1.572/      1.585
GLON/GLAT      :      100.400/     -59.297
TS             : nan
Npred          : nan
Flux           :      nan +/-      nan
EnergyFlux     :      nan +/-      nan
SpatialModel   : PointSource
SpectrumType   : PowerLaw
Spectral Parameters
Index          :      -2 +/-      nan
Scale          :      1000 +/-      nan
Prefactor      :      1e-12 +/-      nan
```

- Get the SkyCoord for a source

```
>>> dir = roi['SourceA'].skydir
```

- Loop over all sources and print their names

```
>>> for s in roi.sources: print(s.name)
3FGL J2357.3-0150
3FGL J0006.2+0135
3FGL J0016.3-0013
3FGL J0014.3-0455
```

clear()

Clear the contents of the ROI.

copy_source (*name*)

static create (*selection, config, **kwargs*)

Create an ROIModel instance.

static create_from_position (*skydir, config, **kwargs*)

Create an ROIModel instance centered on a sky direction.

Parameters

- **skydir** (*SkyCoord*) – Sky direction on which the ROI will be centered.
- **config** (*dict*) – Model configuration dictionary.

static create_from_roi_data (*datafile*)

Create an ROI model.

static create_from_source (*name, config, **kwargs*)

Create an ROI centered on the given source.

create_param_table ()

static create_roi_from_ft1 (*ft1file, config*)

Create an ROI model by extracting the sources coordinates form an FT1 file.

create_source (*name, src_dict, build_index=True, merge_sources=True, rescale=True*)

Add a new source to the ROI model from a dictionary or an existing source object.

Parameters

- **name** (*str*) –
- **src_dict** (dict or *Source*) –

Returns *src***Return type** *Source***create_source_table** ()**create_table** (*names=None*)

Create an astropy Table object with the contents of the ROI model.

defaults = {'catalogs': (None, ',', <type 'list'>), 'src_roiwidth': (None, 'Width of square region in degrees centered on the source')}**delete_sources** (*srcs*)**diffuse_sources****get_nearby_sources** (*name, distance, min_dist=None, square=False*)**get_source_by_name** (*name*)

Return a single source in the ROI with the given name. The input name string can match any of the strings in the names property of the source object. Case and whitespace are ignored when matching name strings. If no sources are found or multiple sources then an exception is thrown.

Parameters **name** (*str*) – Name string.**Returns** *srcs* – A source object.**Return type** *Model***get_sources** (*skydir=None, distance=None, cuts=None, minmax_ts=None, minmax_npred=None, exclude=None, square=False, coordsys='CEL'*)

Retrieve list of source objects satisfying the following selections:

- Angular separation from **skydir** or ROI center (if **skydir** is None) less than **distance**.
- Cuts on source properties defined in **cuts** list.
- TS and Npred in range specified by **minmax_ts** and **minmax_npred**.

Sources can be excluded from the selection by adding their name to the **exclude** list.**Returns** *srcs* – List of source objects.**Return type** *list***get_sources_by_name** (*name*)

Return a list of sources in the ROI matching the given name. The input name string can match any of the strings in the names property of the source object. Case and whitespace are ignored when matching name strings.

Parameters **name** (*str*) –**Returns** *srcs* – A list of *Model* objects.**Return type** *list***get_sources_by_position** (*skydir, dist, min_dist=None, square=False, coordsys='CEL'*)

Retrieve sources within a certain angular distance of a sky coordinate. This function supports two types of geometric selections: circular (**square=False**) and square (**square=True**). The circular selection finds all sources with a given angular distance of the target position. The square selection finds sources within an ROI-like region of size $R \times R$ where $R = 2 \times \text{dist}$.

Parameters

- **skydir** (*SkyCoord*) – Sky direction with respect to which the selection will be applied.
- **dist** (*float*) – Maximum distance in degrees from the sky coordinate.
- **square** (*bool*) – Choose whether to apply a circular or square selection.
- **coordsys** (*str*) – Coordinate system to use when applying a selection with square=True.

get_sources_by_property (*pname, pmin, pmax=None*)

has_source (*name*)

load (***kwargs*)

Load both point source and diffuse components.

load_diffuse_srcs ()

load_existing_catalog (*cat, **kwargs*)

Load sources from an existing catalog object.

Parameters **cat** (*Catalog*) – Catalog object.

load_fits_catalog (*name, **kwargs*)

Load sources from a FITS catalog file.

Parameters **name** (*str*) – Catalog name or path to a catalog FITS file.

load_source (*src, build_index=True, merge_sources=True, **kwargs*)

Load a single source.

Parameters

- **src** (*Source*) – Source object that will be added to the ROI.
- **merge_sources** (*bool*) – When a source matches an existing source in the model update that source with the properties of the new source.
- **build_index** (*bool*) – Re-make the source index after loading this source.

load_sources (*sources*)

Delete all sources in the ROI and load the input source list.

load_xml (*xmlfile, **kwargs*)

Load sources from an XML file.

match_source (*src*)

Look for source or sources in the model that match the given source. Sources are matched by name and any association columns defined in the `assoc_xmatch_columns` parameter.

point_sources

projection

set_projection (*proj*)

skydir

Return the sky direction corresponding to the center of the ROI.

sources

src_name_cols = ['Source_Name', 'ASSOC', 'ASSOC1', 'ASSOC2', 'ASSOC_GAM', '1FHL_Name', '2FGL_Name',

write_fits (*fitsfile*)

Write the ROI model to a FITS file.

write_xml (*xmlfile*)

Save the ROI model as an XML file.

class `fermipy.roi_model.Source` (*name, data, radec=None*)

Bases: `fermipy.roi_model.Model`

Class representation of a source (non-diffuse) model component. A source object serves as a container for the properties of that source (position, spatial/spectral parameters, TS, etc.) as derived in the current analysis. Most properties of a source object can be accessed with the bracket operator:

Return the TS of this source >>> print src['ts']

Get a skycoord representation of the source position >>> print src.skydir

associations

static create_from_dict (*src_dict, roi_skydir=None, rescale=False*)

Create a source object from a python dictionary.

Parameters **src_dict** (*dict*) – Dictionary defining the properties of the source.

static create_from_xml (*root, extdir=None*)

Create a Source object from an XML node.

Parameters

- **root** (*Element*) – XML node containing the source.
- **extdir** (*str*) – Path to the extended source archive.

static create_from_xmlfile (*xmlfile, extdir=None*)

Create a Source object from an XML file.

Parameters

- **xmlfile** (*str*) – Path to XML file.
- **extdir** (*str*) – Path to the extended source archive.

data

diffuse

extended

radec

separation (*src*)

set_position (*skydir*)

Set the position of the source.

Parameters **skydir** (*SkyCoord*) –

set_radec (*ra, dec*)

set_roi_direction (*roidir*)

set_roi_projection (*proj*)

set_spatial_model (*spatial_model, spatial_pars*)

skydir

Return a SkyCoord representation of the source position.

Returns **skydir**

Return type *SkyCoord*

update_data (*d*)

write_xml (*root*)

Write this source to an XML node.

`fermipy.roi_model.create_source_table` (*scan_shape*)

Create an empty source table.

Returns *tab*

Return type `Table`

`fermipy.roi_model.get_dist_to_edge` (*skydir*, *lon*, *lat*, *width*, *coordsys*='CEL')

`fermipy.roi_model.get_linear_dist` (*skydir*, *lon*, *lat*, *coordsys*='CEL')

`fermipy.roi_model.get_skydir_distance_mask` (*src_skydir*, *skydir*, *dist*, *min_dist*=None, *square*=False, *coordsys*='CEL')

Retrieve sources within a certain angular distance of an (ra,dec) coordinate. This function supports two types of geometric selections: circular (*square*=False) and square (*square*=True). The circular selection finds all sources with a given angular distance of the target position. The square selection finds sources within an ROI-like region of size $R \times R$ where $R = 2 \times \text{dist}$.

Parameters

- **src_skydir** (`SkyCoord`) – Array of sky directions.
- **skydir** (`SkyCoord`) – Sky direction with respect to which the selection will be applied.
- **dist** (*float*) – Maximum distance in degrees from the sky coordinate.
- **square** (*bool*) – Choose whether to apply a circular or square selection.
- **coordsys** (*str*) – Coordinate system to use when applying a selection with *square*=True.

`fermipy.roi_model.get_true_params_dict` (*pars_dict*)

`fermipy.roi_model.spectral_pars_from_catalog` (*cat*)

Create spectral parameters from 3FGL catalog columns.

fermipy.utils module

`fermipy.utils.angle_to_cartesian` (*lon*, *lat*)

Convert spherical coordinates to cartesian unit vectors.

`fermipy.utils.apply_minmax_selection` (*val*, *val_minmax*)

`fermipy.utils.arg_to_list` (*arg*)

`fermipy.utils.center_to_edge` (*center*)

`fermipy.utils.collect_dirs` (*path*, *max_depth*=1, *followlinks*=True)

Recursively find directories under the given path.

`fermipy.utils.convolve2d_disk` (*fn*, *r*, *sig*, *nstep*=200)

Evaluate the convolution $f'(r) = f(r) * g(r)$ where $f(r)$ is azimuthally symmetric function in two dimensions and g is a step function given by:

$$g(r) = H(1-r/s)$$

Parameters

- **fn** (*function*) – Input function that takes a single radial coordinate parameter.
- **r** (`ndarray`) – Array of points at which the convolution is to be evaluated.

- **sig** (*float*) – Radius parameter of the step function.
- **nstep** (*int*) – Number of sampling point for numeric integration.

`fermipy.utils.convolve2d_gauss (fn, r, sig, nstep=200)`

Evaluate the convolution $f'(r) = f(r) * g(r)$ where $f(r)$ is azimuthally symmetric function in two dimensions and g is a 2D gaussian with standard deviation s given by:

$$g(r) = 1/(2*\pi*s^2) \text{Exp}[-r^2/(2*s^2)]$$

Parameters

- **fn** (*function*) – Input function that takes a single radial coordinate parameter.
- **r** (*ndarray*) – Array of points at which the convolution is to be evaluated.
- **sig** (*float*) – Width parameter of the gaussian.
- **nstep** (*int*) – Number of sampling point for numeric integration.

`fermipy.utils.cov_to_correlation (cov)`

`fermipy.utils.create_dict (d0, **kwargs)`

`fermipy.utils.create_hpx_disk_region_string (skyDir, coordsys, radius, inclusive=0)`

`fermipy.utils.create_model_name (src)`

Generate a name for a source object given its spatial/spectral properties.

Parameters **src** (*Source*) – A source object.

Returns **name** – A source name.

Return type *str*

`fermipy.utils.create_source_name (skydir, floor=True, prefix='PS')`

`fermipy.utils.create_xml_element (root, name, attrib)`

`fermipy.utils.dot_prod (xyz0, xyz1)`

Compute the dot product between two cartesian vectors where the second dimension contains the vector components.

`fermipy.utils.edge_to_center (edges)`

`fermipy.utils.edge_to_width (edges)`

`fermipy.utils.eq2gal (ra, dec)`

`fermipy.utils.extend_array (edges, binsz, lo, hi)`

Extend an array to encompass lo and hi values.

`fermipy.utils.find_function_root (fn, x0, xb, delta=0.0)`

Find the root of a function: $f(x)+\delta$ in the interval encompassed by x_0 and x_b .

Parameters

- **fn** (*function*) – Python function.
- **x0** (*float*) – Fixed bound for the root search. This will either be used as the lower or upper bound depending on the relative value of x_b .
- **xb** (*float*) – Upper or lower bound for the root search. If a root is not found in the interval $[x_0, x_b]/[x_b, x_0]$ this value will be increased/decreased until a change in sign is found.

`fermipy.utils.find_rows_by_string (tab, names, colnames=['assoc'])`

Find the rows in a table `tab` that match at least one of the strings in `names`. This method ignores whitespace and case when matching strings.

Parameters

- **tab** (`astropy.table.Table`) – Table that will be searched.
- **names** (`list`) – List of strings.
- **colname** (`str`) – Name of the table column that will be searched for matching string.

Returns **mask** – Boolean mask for rows with matching strings.

Return type `ndarray`

`fermipy.utils.fit_parabola(z, ix, iy, dpix=3, zmin=None)`

Fit a parabola to a 2D numpy array. This function will fit a parabola with the functional form described in [parabola](#) to a 2D slice of the input array `z`. The fit region encompasses pixels that are within `dpix` of the pixel coordinate `(iz,iy)` OR that have a value relative to the peak value greater than `zmin`.

Parameters

- **z** (`ndarray`) –
- **ix** (`int`) – X index of center pixel of fit region in array `z`.
- **iy** (`int`) – Y index of center pixel of fit region in array `z`.
- **dpix** (`int`) – Max distance from center pixel of fit region.
- **zmin** (`float`) –

`fermipy.utils.fits_reccarray_to_dict(table)`

Convert a FITS reccarray to a python dictionary.

`fermipy.utils.format_filename(outdir, basename, prefix=None, extension=None)`

`fermipy.utils.gal2eq(l, b)`

`fermipy.utils.get_parameter_limits(xval, loglike, ul_confidence=0.95, tol=0.001)`

Compute upper/lower limits, peak position, and 1-sigma errors from a 1-D likelihood function. This function uses the delta-loglikelihood method to evaluate parameter limits by searching for the point at which the change in the log-likelihood value with respect to the maximum equals a specific value. A parabolic spline fit to the log-likelihood values is used to improve the accuracy of the calculation.

Parameters

- **xval** (`ndarray`) – Array of parameter values.
- **loglike** (`ndarray`) – Array of log-likelihood values.
- **ul_confidence** (`float`) – Confidence level to use for limit calculation.
- **tol** (`float`) – Tolerance parameter for spline.

`fermipy.utils.init_matplotlib_backend(backend=None)`

This function initializes the matplotlib backend. When no DISPLAY is available the backend is automatically set to 'Agg'.

Parameters **backend** (`str`) – matplotlib backend name.

`fermipy.utils.interpolate_function_min(x, y)`

`fermipy.utils.is_fits_file(path)`

`fermipy.utils.isstr(s)`

String instance testing method that works under both Python 2.X and 3.X. Returns true if the input is a string.

`fermipy.utils.join_strings(strings, sep='_')`

`fermipy.utils.load_data (infile, workdir=None)`
 Load python data structure from either a YAML or numpy file.

`fermipy.utils.load_npy (infile)`

`fermipy.utils.load_xml_elements (root, path)`

`fermipy.utils.load_yaml (infile, **kwargs)`

`fermipy.utils.lonlat_to_xyz (lon, lat)`

`fermipy.utils.make_cdisk_kernel (psf, sigma, npix, cdelt, xpix, ypix, psf_scale_fn=None, normalize=False)`
 Make a kernel for a PSF-convolved 2D disk.

Parameters

- **psf** (PSFModel) –
- **sigma** (*float*) – 68% containment radius in degrees.

`fermipy.utils.make_cgauss_kernel (psf, sigma, npix, cdelt, xpix, ypix, psf_scale_fn=None, normalize=False)`
 Make a kernel for a PSF-convolved 2D gaussian.

Parameters

- **psf** (PSFModel) –
- **sigma** (*float*) – 68% containment radius in degrees.

`fermipy.utils.make_disk_kernel (radius, npix=501, cdelt=0.01, xpix=None, ypix=None)`
 Make kernel for a 2D disk.

Parameters **radius** (*float*) – Disk radius in deg.

`fermipy.utils.make_gaussian_kernel (sigma, npix=501, cdelt=0.01, xpix=None, ypix=None)`
 Make kernel for a 2D gaussian.

Parameters **sigma** (*float*) – Standard deviation in degrees.

`fermipy.utils.make_pixel_distance (shape, xpix=None, ypix=None)`
 Fill a 2D array with dimensions `shape` with the distance of each pixel from a reference direction (`xpix,ypix`) in pixel coordinates. Pixel coordinates are defined such that (0,0) is located at the center of the corner pixel.

`fermipy.utils.make_psf_kernel (psf, npix, cdelt, xpix, ypix, psf_scale_fn=None, normalize=False)`
 Generate a kernel for a point-source.

Parameters

- **psf** (PSFModel) –
- **npix** (*int*) – Number of pixels in X and Y dimensions.
- **cdelt** (*float*) – Pixel size in degrees.

`fermipy.utils.match_regex_list (patterns, string)`
 Perform a regex match of a string against a list of patterns. Returns true if the string matches at least one pattern in the list.

`fermipy.utils.merge_dict (d0, d1, add_new_keys=False, append_arrays=False)`
 Recursively merge the contents of python dictionary `d0` with the contents of another python dictionary, `d1`.

Parameters

- **d0** (*dict*) – The input dictionary.
- **d1** (*dict*) – Dictionary to be merged with the input dictionary.

- **add_new_keys** (*str*) – Do not skip keys that only exist in d1.
- **append_arrays** (*bool*) – If an element is a numpy array set the value of that element by concatenating the two arrays.

`fermipy.utils.met_to_mjd` (*time*)

“Convert mission elapsed time to mean julian date.

`fermipy.utils.mkdir` (*dir*)

`fermipy.utils.onesided_cl_to_dlnl` (*cl*)

Compute the delta-loglikelihood values that corresponds to an upper limit of the given confidence level.

Parameters *cl* (*float*) – Confidence level.

Returns *dlnl* – Delta-loglikelihood value with respect to the maximum of the likelihood function.

Return type *float*

`fermipy.utils.onesided_dlnl_to_cl` (*dlnl*)

Compute the confidence level that corresponds to an upper limit with a given change in the loglikelihood value.

Parameters *dlnl* (*float*) – Delta-loglikelihood value with respect to the maximum of the likelihood function.

Returns *cl* – Confidence level.

Return type *float*

`fermipy.utils.overlap_slices` (*large_array_shape*, *small_array_shape*, *position*)

Modified version of `overlap_slices`.

Get slices for the overlapping part of a small and a large array.

Given a certain position of the center of the small array, with respect to the large array, tuples of slices are returned which can be used to extract, add or subtract the small array at the given position. This function takes care of the correct behavior at the boundaries, where the small array is cut of appropriately.

Parameters

- **large_array_shape** (*tuple*) – Shape of the large array.
- **small_array_shape** (*tuple*) – Shape of the small array.
- **position** (*tuple*) – Position of the small array’s center, with respect to the large array. Coordinates should be in the same order as the array shape.

Returns

- **slices_large** (*tuple of slices*) – Slices in all directions for the large array, such that `large_array[slices_large]` extracts the region of the large array that overlaps with the small array.
- **slices_small** (*slice*) – Slices in all directions for the small array, such that `small_array[slices_small]` extracts the region that is inside the large array.

`fermipy.utils.parabola` (*xy*, *amplitude*, *x0*, *y0*, *sx*, *sy*, *theta*)

Evaluate a 2D parabola given by:

$$f(x,y) = f_0 - (1/2) * \text{delta}^T * R * \text{Sigma} * R^T * \text{delta}$$

where

$$\text{delta} = [(x - x_0), (y - y_0)]$$

and *R* is the matrix for a 2D rotation by angle *theta* and *Sigma* is the covariance matrix:

Sigma = $[[1/\sigma_x^2, 0], [0, 1/\sigma_y^2]]$

Parameters

- **xy** (*tuple*) – Tuple containing x and y arrays for the values at which the parabola will be evaluated.
- **amplitude** (*float*) – Constant offset value.
- **x0** (*float*) – Centroid in x coordinate.
- **y0** (*float*) – Centroid in y coordinate.
- **sx** (*float*) – Standard deviation along first axis (x-axis when theta=0).
- **sy** (*float*) – Standard deviation along second axis (y-axis when theta=0).
- **theta** (*float*) – Rotation angle in radians.

Returns **vals** – Values of the parabola evaluated at the points defined in the xy input tuple.

Return type *ndarray*

`fermipy.utils.path_to_xmlpath(path)`

`fermipy.utils.poly_to_parabola(coeff)`

`fermipy.utils.prettify_xml(elem)`

Return a pretty-printed XML string for the Element.

`fermipy.utils.project(lon0, lat0, lon1, lat1)`

This function performs a stereographic projection on the unit vector (lon1,lat1) with the pole defined at the reference unit vector (lon0,lat0).

`fermipy.utils.rebin_map(k, nebin, npix, rebin)`

`fermipy.utils.resolve_file_path(path, **kwargs)`

`fermipy.utils.resolve_file_path_list(pathlist, workdir, prefix='', randomize=False)`

Resolve the path of each file name in the file pathlist and write the updated paths to a new file.

`fermipy.utils.resolve_path(path, workdir=None)`

`fermipy.utils.scale_parameter(p)`

`fermipy.utils.separation_cos_angle(lon0, lat0, lon1, lat1)`

Evaluate the cosine of the angular separation between two direction vectors.

`fermipy.utils.split_bin_edges(edges, npts=2)`

Subdivide an array of bins by splitting each bin into npts subintervals.

Parameters

- **edges** (*ndarray*) – Bin edge array.
- **npts** (*int*) – Number of intervals into which each bin will be subdivided.

Returns **edges** – Subdivided bin edge array.

Return type *ndarray*

`fermipy.utils.strip_suffix(filename, suffix)`

`fermipy.utils.sum_bins(x, dim, npts)`

`fermipy.utils.tolist(x)`

convenience function that takes in a nested structure of lists and dictionaries and converts everything to its base objects. This is useful for dumping a file to yaml.

1.numpy arrays into python lists

```
>>> type(tolist(np.asarray(123))) == int
True
>>> tolist(np.asarray([1,2,3])) == [1,2,3]
True
```

2.numpy strings into python strings.

```
>>> tolist([np.asarray('cat')]) == ['cat']
True
```

3.an ordered dict to a dict

```
>>> ordered=OrderedDict(a=1, b=2)
>>> type(tolist(ordered)) == dict
True
```

4.converts unicode to regular strings

```
>>> type(u'a') == str
False
>>> type(tolist(u'a')) == str
True
```

5.converts numbers & bools in strings to real representation, (i.e. '123' -> 123)

```
>>> type(tolist(np.asarray('123')))) == int
True
>>> type(tolist('123')) == int
True
>>> tolist('False') == False
True
```

`fermipy.utils.twosided_cl_to_dlnl(cl)`

Compute the delta-loglikelihood value that corresponds to a two-sided interval of the given confidence level.

Parameters `cl` (*float*) – Confidence level.

Returns `dlnl` – Delta-loglikelihood value with respect to the maximum of the likelihood function.

Return type *float*

`fermipy.utils.twosided_dlnl_to_cl(dlnl)`

Compute the confidence level that corresponds to a two-sided interval with a given change in the loglikelihood value.

Parameters `dlnl` (*float*) – Delta-loglikelihood value with respect to the maximum of the likelihood function.

Returns `cl` – Confidence level.

Return type *float*

`fermipy.utils.unicode_representer(dumper, uni)`

`fermipy.utils.unicode_to_str(args)`

`fermipy.utils.update_bounds(val, bounds)`

`fermipy.utils.update_keys(input_dict, key_map)`

`fermipy.utils.val_to_bin(edges, x)`
Convert axis coordinate to bin index.

`fermipy.utils.val_to_bin_bounded(edges, x)`
Convert axis coordinate to bin index.

`fermipy.utils.val_to_edge(edges, x)`
Convert axis coordinate to bin index.

`fermipy.utils.val_to_pix(center, x)`

`fermipy.utils.write_yaml(o, outfile, **kwargs)`

`fermipy.utils.xmlpath_to_path(path)`

`fermipy.utils.xyz_to_lonlat(*args)`

fermipy.plotting module

class `fermipy.plotting.AnalysisPlotter` (*config*, ***kwargs*)

Bases: `fermipy.config.Configurable`

defaults = {'catalogs': (None, '', <type 'list'>), 'format': ('png', '', <type 'str'>), 'log_bounds': (None, '', <type 'list'>)

make_components_plots (*gta*, *mcube_maps*, *prefix*, *log_bounds=None*, ***kwargs*)

make_extension_plots (*ext*, *roi=None*, ***kwargs*)

make_localization_plots (*loc*, *roi=None*, ***kwargs*)

make_residmap_plots (*maps*, *roi=None*, ***kwargs*)

Make plots from the output of `residmap`.

Parameters

- **maps** (*dict*) – Output dictionary of `residmap`.
- **roi** (`ROIModel`) – ROI Model object. Generate markers at the positions of the sources in this ROI.
- **zoom** (*float*) – Crop the image by this factor. If None then no crop is applied.

make_roi_plots (*gta*, *mcube_map*, ***kwargs*)

Make various diagnostic plots for the 1D and 2D counts/model distributions.

Parameters **prefix** (*str*) – Prefix that will be appended to all filenames.

make_sed_plots (*sed*, ***kwargs*)

make_tsmmap_plots (*maps*, *roi=None*, ***kwargs*)

Make plots from the output of `tsmap` or `tscube`. This method generates a 2D sky map for the best-fit test source in sqrt(TS) and Npred.

Parameters

- **maps** (*dict*) – Output dictionary of `tsmap` or `tscube`.
- **roi** (`ROIModel`) – ROI Model object. Generate markers at the positions of the sources in this ROI.
- **zoom** (*float*) – Crop the image by this factor. If None then no crop is applied.

run (*gta*, *mcube_map*, ***kwargs*)

Make all plots.

```

class fermipy.plotting.ExtensionPlotter(src, roi, suffix, workdir, loge_bounds=None)
    Bases: object

    plot(iaxis)

class fermipy.plotting.ImagePlotter(data, proj, mapping=None)
    Bases: object

    plot(subplot=111, cmap='magma', **kwargs)

    projtype

class fermipy.plotting.ROIPlotter(data_map, **kwargs)
    Bases: fermipy.config.Configurable

    cmap

    static create_from_fits(fitsfile, roi, **kwargs)

    data

    defaults = {'graticule_radii': (None, ',', <type 'list'>), 'label_ts_threshold': (0.0, ',', <type 'float'>), 'catalogs': (None, ',', <type 'list'>)}

    draw_circle(radius, **kwargs)

    static get_data_projection(data, axes, iaxis, xmin=-1, xmax=1, loge_bounds=None)

    plot(**kwargs)

    plot_catalog(catalog)

    plot_projection(iaxis, **kwargs)

    plot_roi(roi, **kwargs)

    plot_sources(skydir, labels, plot_kwargs, text_kwargs, **kwargs)

    proj

    projtype

    static setup_projection_axis(iaxis, loge_bounds=None)

    zoom(zoom)

class fermipy.plotting.SEDPlotter(sed)
    Bases: object

    static annotate(sed, xy=(0.05, 0.93), **kwargs)

    static get_ylimits(sed)

    plot(showlnl=False, **kwargs)

    static plot_flux_points(sed, **kwargs)

    static plot_lnlscan(sed, **kwargs)

    static plot_model(model_flux, **kwargs)

    static plot_resid(src, model_flux, **kwargs)

    static plot_sed(sed, showlnl=False, **kwargs)
        Render a plot of a spectral energy distribution.

```

Parameters

- **showlnl** (`bool`) – Overlay a map of the delta-loglikelihood values vs. flux in each energy bin.

- **cmap** (*str*) – Colormap that will be used for the delta-loglikelihood map.
- **llhcut** (*float*) – Minimum delta-loglikelihood value.
- **ul_ts_threshold** (*float*) – TS threshold that determines whether the MLE or UL is plotted in each energy bin.

sed

```
fermipy.plotting.annotate(**kwargs)
fermipy.plotting.get_xerr(sed)
fermipy.plotting.load_bluered_cmap()
fermipy.plotting.load_ds9_cmap()
fermipy.plotting.make_counts_spectrum_plot(o, roi, energies, imfile, **kwargs)
fermipy.plotting.plot_error_ellipse(fit, xy, cdelt, **kwargs)
fermipy.plotting.plot_markers(lon, lat, **kwargs)
fermipy.plotting.truncate_colormap(cmap, minval=0.0, maxval=1.0, n=256)
```

Function that extracts a subset of a colormap.

fermipy.sed module

Utilities for dealing with SEDs

Many parts of this code are taken from `dsphs/like/lnlfn.py` by Matthew Wood <mdwood@slac.stanford.edu> and Alex Drlica-Wagner <kadrlica@slac.stanford.edu>

```
class fermipy.sed.SEDGenerator
    Bases: object
```

Mixin class that provides SED functionality to *GTAnalysis*.

```
sed(name, **kwargs)
```

Generate a spectral energy distribution (SED) for a source. This function will fit the normalization of the source in each energy bin. By default the SED will be generated with the analysis energy bins but a custom binning can be defined with the `log_e_bins` parameter.

Parameters

- **name** (*str*) – Source name.
- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **log_e_bins** (*ndarray*) – Sequence of energies in $\log_{10}(E/\text{MeV})$ defining the edges of the energy bins. If this argument is `None` then the analysis energy bins will be used. The energies in this sequence must align with the bin edges of the underlying analysis instance.
- **{options}** –
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns `sed` – Dictionary containing output of the SED analysis.

Return type `dict`

fermipy.sourcefind module

class `fermipy.sourcefind.SourceFind`

Bases: `object`

Mixin class which provides source-finding functionality to *GTAnalysis*.

find_sources (*prefix*='', ***kwargs*)

An iterative source-finding algorithm that uses likelihood ratio (TS) maps of the region of interest to find new sources. After each iteration a new TS map is generated incorporating sources found in the previous iteration. The method stops when the number of iterations exceeds `max_iter` or no sources exceeding `sqrts_threshold` are found.

Parameters

- **{options}** –
- **tmap** (*dict*) – Keyword arguments dictionary for tmap method.
- **tscube** (*dict*) – Keyword arguments dictionary for tscube method.

Returns

- **peaks** (*list*) – List of peak objects.
- **sources** (*list*) – List of source objects.

localize (*name*, ***kwargs*)

Find the best-fit position of a source. Localization is performed in two steps. First a TS map is computed centered on the source with half-width set by `dtheta_max`. A fit is then performed to the maximum TS peak in this map. The source position is then further refined by scanning the likelihood in the vicinity of the peak found in the first step. The size of the scan region is set to encompass the 99% positional uncertainty contour as determined from the peak fit.

Parameters

- **name** (*str*) – Source name.
- **{options}** –
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

Returns **localize** – Dictionary containing results of the localization analysis.

Return type `dict`

fermipy.spectrum module

class `fermipy.spectrum.DMFitFunction` (*params*, *chan*='bb', *jfactor*=1e+19, *tablepath*=None)

Bases: `fermipy.spectrum.SpectralFunction`

Class that evaluates the spectrum for a DM particle of a given mass, channel, cross section, and J-factor. The parameterization is given by:

$$F(x) = 1 / (8 * \pi) * (1/\text{mass}^2) * \text{sigmav} * J * dN/dE(E, \text{mass}, i)$$

where the `params` array should be defined with:

- `params[0]` : `sigmav`
- `params[1]` : `mass`

Note that this class assumes that mass and J-factor are provided in units of GeV and GeV² cm⁻⁵ while energies are defined in MeV.

chan

Return the channel string.

chan_code

Return the channel code.

channel_index_mapping = {1: 8, 2: 6, 3: 3, 4: 1, 5: 2, 6: 7, 7: 4, 8: 5, 9: 0, 10: 10, 11: 11, 12: 9}

channel_name_mapping = {1: ['e+e-', 'ee'], 2: ['mu+mu-', 'mumu', 'musrc'], 3: ['tau+tau-', 'tautau', 'tausrc'], 4: ['b

channel_rev_map = {'mumu': 2, 'zz': 8, 'gluons': 6, 'ee': 1, 'bbsrc': 4, 'gg': 6, 'wwsrc': 7, 'bb-bar': 4, 'tautau': 3, 'm

static channels ()

Return all available DMFit channel strings

static nparam ()

class `fermipy.spectrum.LogParabola` (*params=None, scale=1.0, extra_params=None*)

Bases: `fermipy.spectrum.SpectralFunction`

Class that evaluates a function with the parameterization:

$$F(x) = p_0 * (x/x_s)^{(p_1 - p_2 * \log(x/x_s))}$$

where x_s is a scale parameter. The `params` array should be defined with:

- `params[0]` : Prefactor (p_0)

- `params[1]` : Index (p_1)

- `params[2]` : Curvature (p_2)

static nparam ()

class `fermipy.spectrum.PLExpCutoff` (*params=None, scale=1.0, extra_params=None*)

Bases: `fermipy.spectrum.SpectralFunction`

Class that evaluates a function with the parameterization:

$$F(x) = p_0 * (x/x_s)^{(p_1 - p_2 * \log(x/x_s))}$$

where x_s is the scale parameter. The `params` array should be defined with:

- `params[0]` : Prefactor (p_0)

- `params[1]` : Index (p_1)

- `params[2]` : Curvature (p_2)

static log_to_params (*params*)

static nparam ()

static params_to_log (*params*)

class `fermipy.spectrum.PowerLaw` (*params=None, scale=1.0, extra_params=None*)

Bases: `fermipy.spectrum.SpectralFunction`

Class that evaluates a power-law function with the parameterization:

$$F(x) = p_0 * (x/x_s)^{p_1}$$

where x_s is the scale parameter. The `params` array should be defined with:

- `params[0]` : Prefactor (p_0)

- `params[1]` : Index (p_1)

classmethod `eval_eflux` (*emin, emax, params, scale=1.0, extra_params=None*)

```

classmethod eval_flux (emin, emax, params, scale=1.0, extra_params=None)

static eval_norm (scale, index, emin, emax, flux)

static nparam ()

class fermipy.spectrum.SEDEFfluxFunctor (sfn, emin, emax)
    Bases: fermipy.spectrum.SEDFunctor

    Functor that computes the energy flux of a source in a pre-defined sequence of energy bins.

class fermipy.spectrum.SEDFluxFunctor (sfn, emin, emax)
    Bases: fermipy.spectrum.SEDFunctor

    Functor that computes the flux of a source in a pre-defined sequence of energy bins.

class fermipy.spectrum.SEDFunctor (sfn, emin, emax)
    Bases: object

    Functor object that wraps a SpectralFunction and computes the normalization of the model in a sequence
    of SED energy bins. The evaluation method of this class accepts a single vector for the parameters of the model.
    This class serves as an object that can be passed to likelihood optimizers.

    emax
    emin
    params
    scale
    spectral_fn

class fermipy.spectrum.SpectralFunction (params, scale=1.0, extra_params=None)
    Bases: object

    Base class for spectral models. Spectral models inheriting from this class should implement at a minimum an
    _eval_dnde method which evaluates the differential flux at a given energy.

    classmethod create_eflux_functor (emin, emax, params=None, scale=1.0, extra_params=None)

    classmethod create_flux_functor (emin, emax, params=None, scale=1.0, extra_params=None)

    classmethod create_from_eflux (params, emin, emax, eflux, scale=1.0)
        Create a spectral function instance given its energy flux.

    classmethod create_from_flux (params, emin, emax, flux, scale=1.0)
        Create a spectral function instance given its flux.

    classmethod create_functor (spec_type, func_type, emin, emax, params=None, scale=1.0, extra_params=None)

    dnde (x, params=None)
        Evaluate differential flux.

    dnde_deriv (x, params=None)
        Evaluate derivative of the differential flux with respect to E.

    e2dnde (x, params=None)
        Evaluate E^2 times differential flux.

    e2dnde_deriv (x, params=None)
        Evaluate derivative of E^2 times differential flux with respect to E.

    ednde (x, params=None)
        Evaluate E times differential flux.

```

ednde_deriv (*x*, *params*=None)

Evaluate derivative of E times differential flux with respect to E.

eflux (*emin*, *emax*, *params*=None)

Evaluate the integral energy flux.

classmethod eval_dnde (*x*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_dnde_deriv (*x*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_e2dnde (*x*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_e2dnde_deriv (*x*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_ednde (*x*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_ednde_deriv (*x*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_eflux (*emin*, *emax*, *params*, *scale*=1.0, *extra_params*=None)

classmethod eval_flux (*emin*, *emax*, *params*, *scale*=1.0, *extra_params*=None)

extra_params

Dictionary containing additional parameters needed for evaluation of the function.

flux (*emin*, *emax*, *params*=None)

Evaluate the integral flux.

log_params

Return transformed parameter vector in which norm and scale parameters are converted to log10.

params

Return parameter vector of the function.

scale

`fermipy.spectrum.cast_args(x)`

`fermipy.spectrum.cast_params(params)`

fermipy.skymap module

class `fermipy.skymap.HpxMap` (*counts*, *hpx*)

Bases: `fermipy.skymap.Map_Base`

Representation of a 2D or 3D counts map using HEALPix.

convert_to_cached_wcs (*hpx_in*, *sum_ebins*=False, *normalize*=True)

Make a WCS object and convert HEALPix data into WCS projection

Parameters

- **hpx_in** (*ndarray*) – HEALPix input data
- **sum_ebins** (*bool*) – sum energy bins over energy bins before reprojection
- **normalize** (*bool*) – True -> preserve integral by splitting HEALPix values between bins
- (**WCS object**, **np.ndarray()** with **reprojected data**) (*returns*) –

static create_from_fits (*fitsfile*, ***kwargs*)

static create_from_hdu (*hdu, ebins*)

Creates and returns an HpxMap object from a FITS HDU.

hdu : The FITS ebins : Energy bin edges [optional]

static create_from_hdulist (*hdulist, **kwargs*)

Creates and returns an HpxMap object from a FITS HDUList

extname : The name of the HDU with the map data ebounds : The name of the HDU with the energy bin data

create_image_hdu (*name=None, **kwargs*)

get_map_values (*lons, lats, ibin=None*)

Return the indices in the flat array corresponding to a set of coordinates

Parameters

- **lons** (*array-like*) – ‘Longitudes’ (RA or GLON)
- **lats** (*array-like*) – ‘Latitudes’ (DEC or GLAT)
- **ibin** (*int or array-like*) – Extract data only for a given energy bin. None -> extract data for all bins

Returns *vals* – Values of pixels in the flattened map, np.nan used to flag coords outside of map

Return type numpy.ndarray((n))

get_pixel_indices (*lats, lons*)

Return the indices in the flat array corresponding to a set of coordinates

get_pixel_skydirs ()

Get a list of sky coordinates for the centers of every pixel.

hpx

interpolate (*lon, lat, egy=None, interp_log=True*)

Interpolate map values.

Parameters *interp_log* (*bool*) – Interpolate the z-coordinate in logspace.

make_wcs_from_hpx (*sum_ebins=False, proj='CAR', oversample=2, normalize=True*)

Make a WCS object and convert HEALPix data into WCS projection

NOTE: this re-calculates the mapping, if you have already calculated the mapping it is much faster to use `convert_to_cached_wcs()` instead

Parameters

- **sum_ebins** (*bool*) – sum energy bins over energy bins before reprojecting
- **proj** (*str*) – WCS-projection
- **oversample** (*int*) – Oversampling factor for WCS map
- **normalize** (*bool*) – True -> preserve integral by splitting HEALPix values between bins
- **(WCS object, np.ndarray() with reprojected data)** (*returns*) –

sum_over_energy ()

Reduce a counts cube to a counts map

swap_scheme ()

ud_grade (*order, preserve_counts=False*)

class `fermipy.skymap.Map` (*counts, wcs, ebins=None*)

Bases: `fermipy.skymap.Map_Base`

Representation of a 2D or 3D counts map using WCS.

static create (*skydir, cdelt, npix, coordsys='CEL', projection='AIT'*)

static create_from_fits (*fitsfile, **kwargs*)

static create_from_hdu (*hdu, wcs*)

create_image_hdu (*name=None, **kwargs*)

create_primary_hdu ()

get_map_values (*lons, lats, ibin=None*)

Return the map values corresponding to a set of coordinates.

Parameters

- **lons** (*array-like*) – ‘Longitudes’ (RA or GLON)
- **lats** (*array-like*) – ‘Latitudes’ (DEC or GLAT)
- **ibin** (*int or array-like*) – Extract data only for a given energy bin. None -> extract data for all bins

Returns **vals** – Values of pixels in the flattened map, np.nan used to flag coords outside of map

Return type `numpy.ndarray((n))`

get_pixel_indices (*lons, lats, ibin=None*)

Return the indices in the flat array corresponding to a set of coordinates

Parameters

- **lons** (*array-like*) – ‘Longitudes’ (RA or GLON)
- **lats** (*array-like*) – ‘Latitudes’ (DEC or GLAT)
- **ibin** (*int or array-like*) – Extract data only for a given energy bin. None -> extract data for all energy bins.

Returns **pixcrd** – Pixel indices along each dimension of the map.

Return type `list`

get_pixel_skydirs ()

Get a list of sky coordinates for the centers of every pixel.

interpolate (*lon, lat, eby=None*)

ipix_swap_axes (*ipix, colwise=False*)

Return the transposed pixel index from the pixel xy coordinates

if colwise is True (False) this assumes the original index was in column wise scheme

ipix_to_xypix (*ipix, colwise=False*)

Return array multi-dimensional pixel indices from flattened index.

Parameters **colwise** (*bool*) – Use column-wise pixel indexing.

npix

pix_center

Return the ROI center in pixel coordinates.

pix_size

Return the pixel size along the two image dimensions.

skydir

Return the sky coordinate of the image center.

sum_over_energy()

Reduce a 3D counts cube to a 2D counts map

wcs**width**

Return the dimensions of the image.

xypix_to_ipix(xypix, colwise=False)

Return the flattened pixel indices from an array multi-dimensional pixel indices.

Parameters

- **xypix** (*list*) – List of pixel indices in the order (LON,LAT,ENERGY).
- **colwise** (*bool*) – Use column-wise pixel indexing.

class fermipy.skymap.**Map_Base**(*counts*)

Bases: `object`

Abstract representation of a 2D or 3D counts map.

counts**data****get_map_values(lons, lats, ibin=None)**

Return the map values corresponding to a set of coordinates.

get_pixel_indices(lats, lons)

Return the indices in the flat array corresponding to a set of coordinates

get_pixel_skydirs()

Get a list of sky coordinates for the centers of every pixel.

interpolate(lon, lat, egy=None)

Return the interpolated map values corresponding to a set of coordinates.

sum_over_energy()

Reduce a counts cube to a counts map by summing over the energy planes

fermipy.skymap.**make_coadd_hpx**(*maps, hpx, shape*)

fermipy.skymap.**make_coadd_map**(*maps, proj, shape*)

fermipy.skymap.**make_coadd_wcs**(*maps, wcs, shape*)

fermipy.skymap.**read_map_from_fits**(*fitsfile, extname=None*)

fermipy.castro module

Utilities for dealing with ‘castro data’, i.e., 2D table of likelihood values.

Castro data can be tabulated in terms of a variety of variables. The most common example is probably a simple SED, where we have the likelihood as a function of Energy and Energy Flux.

However, we could easily convert to the likelihood as a function of other variables, such as the Flux normalization and the spectral index, or the mass and cross-section of a putative dark matter particle.

class `fermipy.castro.CastroData` (*norm_vals, nll_vals, refSpec, norm_type*)

Bases: `fermipy.castro.CastroData_Base`

This class wraps the data needed to make a “Castro” plot, namely the log-likelihood as a function of normalization for a series of energy bins.

static create_from_fits (*fitsfile, norm_type='eflux', hdu_scan='SCANDATA', hdu_energies='EBOUNDS', irow=None*)

Create a CastroData object from a tscube FITS file.

Parameters

- **fitsfile** (*str*) – Name of the fits file
- **norm_type** (*str*) – Type of normalization to use. Valid options are:
 - `norm` : Normalization w.r.t. to test source
 - `flux` : Flux of the test source ($\text{ph cm}^{-2} \text{s}^{-1}$)
 - `eflux` : Energy Flux of the test source ($\text{MeV cm}^{-2} \text{s}^{-1}$)
 - `npred` : Number of predicted photons (Not implemented)
 - `dnde` : Differential flux of the test source ($\text{ph cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$)
- **hdu_scan** (*str*) – Name of the FITS HDU with the scan data
- **hdu_energies** (*str*) – Name of the FITS HDU with the energy binning and normalization data
- **irow** (*int or None*) – If none, then this assumes that there is a single row in the scan data table Otherwise, this specifies which row of the table to use

Returns castro

Return type `CastroData`

static create_from_flux_points (*txtfile*)

Create a Castro data object from a text file containing a sequence of differential flux points.

static create_from_sedfile (*fitsfile, norm_type='eflux'*)

Create a CastroData object from an SED fits file

Parameters

- **fitsfile** (*str*) – Name of the fits file
- **norm_type** (*str*) – Type of normalization to use, options are:
 - `norm` : Normalization w.r.t. to test source
 - `flux` : Flux of the test source ($\text{ph cm}^{-2} \text{s}^{-1}$)
 - `eflux` : Energy Flux of the test source ($\text{MeV cm}^{-2} \text{s}^{-1}$)
 - `npred` : Number of predicted photons (Not implemented)
 - `dnde` : Differential flux of the test source ($\text{ph cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$)

Returns castro

Return type `CastroData`

static create_from_stack (*shape, components, ylimits, weights=None*)

Combine the log-likelihoods from a number of components.

Parameters

- **shape** (*tuple*) – The shape of the return array
- **components** (*[~fermipy.castro.CastroData_Base]*) – The components to be stacked
- **weights** (*array-like*) –

Returns *castro*

Return type *CastroData*

static create_from_tables (*norm_type='eflux', tab_s='SCANDATA', tab_e='EBOUNDS'*)

Create a CastroData object from two tables

Parameters

- **norm_type** (*str*) – Type of normalization to use. Valid options are:
 - norm : Normalization w.r.t. to test source
 - flux : Flux of the test source ($\text{ph cm}^{-2} \text{s}^{-1}$)
 - eflux: Energy Flux of the test source ($\text{MeV cm}^{-2} \text{s}^{-1}$)
 - npred: Number of predicted photons (Not implemented)
 - dnde : Differential flux of the test source ($\text{ph cm}^{-2} \text{s}^{-1} \text{MeV}^{-1}$)
- **tab_s** (*str*) – table scan data
- **tab_e** (*str*) – table energy binning and normalization data

Returns *castro*

Return type *CastroData*

create_functor (*specType, initPars=None, scale=1000.0*)

Create a functor object that computes normalizations in a sequence of energy bins for a given spectral model.

Parameters

- **specType** (*str*) – The type of spectrum to use. This can be a string corresponding to the spectral model class name or a *SpectralFunction* object.
- **initPars** (*ndarray*) – Arrays of parameter values with which the spectral function will be initialized.
- **scale** (*float*) – The ‘pivot energy’ or energy scale to use for the spectrum

Returns *fn* – A functor object.

Return type *SEDFunctor*

nE

Return the number of energy bins. This is also the number of x-axis bins.

refSpec

Return a *ReferenceSpec* with the spectral data

spectrum_loglike (*specType, params, scale=1000.0*)

return the log-likelihood for a particular spectrum

Parameters

- **specTypes** (*str*) – The type of spectrum to try
- **params** (*array-like*) – The spectral parameters

- **scale** (*float*) – The energy scale or ‘pivot’ energy

test_spectra (*spec_types=None*)

Test different spectral types against the SED represented by this `CastroData`.

Parameters **spec_types** (*[str, ...]*) – List of spectral types to try

Returns

retDict – A dictionary of dictionaries. The top level dictionary is keyed by `spec_type`. The sub-dictionaries each contain:

- “Function” : *SpectralFunction*
- “Result” : tuple with the output of `scipy.optimize.fmin`
- “Spectrum” : *ndarray* with best-fit spectral values
- “ScaleEnergy” : float, the ‘pivot energy’ value
- “TS” : float, the TS for the best-fit spectrum

Return type *dict*

class `fermipy.castro.CastroData_Base` (*norm_vals, nll_vals, norm_type*)

Bases: *object*

This class wraps the data needed to make a “Castro” plot, namely the log-likelihood as a function of normalization.

In this case the x-axes and y-axes are generic Sub-classes can implement particular axes choices (e.g., EFlux v. Energy)

TS_spectrum (*spec_vals*)

Calculate and the TS for a given set of spectral values.

__call__ (*x*)

Return the negative log-likelihood for an array of values, summed over the energy bins

Parameters **x** (*ndarray*) – Array of N x M values

Returns **nll_val** – Array of negative log-likelihood values.

Return type *ndarray*

__getitem__ (*i*)

return the `LnLFn` object for the *i*th energy bin

build_scandata_table ()

chi2_vals (*x*)

Compute the difference in the log-likelihood between the MLE in each energy bin and the normalization predicted by a global best-fit model. This array can be summed to get a goodness-of-fit `chi2` for the model.

Parameters **x** (*ndarray*) – An array of normalizations derived from a global fit to all energy bins.

Returns **chi2_vals** – An array of `chi2` values for each energy bin.

Return type *ndarray*

derivative (*x, der=1*)

Return the derivative of the log-likelihood summed over the energy bins

Parameters

- **x** (*ndarray*) – Array of N x M values

- **der** (*int*) – Order of the derivate

Returns **der_val** – Array of negative log-likelihood values.

Return type `ndarray`

fitNorm_v2 (*specVals*)

Fit the normalization given a set of spectral values that define a spectral shape.

This version uses `scipy.optimize.fmin`.

Parameters

- **specVals** (*an array of (nebin values that define a spectral shape)*) –
- **xlims** (*fit limits*) –

Returns **norm** – Best-fit normalization value

Return type `float`

fitNormalization (*specVals, xlims*)

Fit the normalization given a set of spectral values that define a spectral shape

This version is faster, and solves for the root of the derivatvie

Parameters

- **specVals** (*an array of (nebin values that define a spectral shape)*) –
- **xlims** (*fit limits*) –
- **the best-fit normalization value** (*returns*) –

fit_spectrum (*specFunc, initPars, freePars=None*)

Fit for the free parameters of a spectral function

Parameters

- **specFunc** (*SpectralFunction*) – The Spectral Function
- **initPars** (`ndarray`) – The initial values of the parameters
- **freePars** (`ndarray`) – Boolean array indicating which parameters should be free in the fit.

Returns

- **params** (`ndarray`) – Best-fit parameters.
- **spec_vals** (`ndarray`) – The values of the best-fit spectral model in each energy bin.
- **ts_spec** (*float*) – The TS of the best-fit spectrum
- **chi2_vals** (`ndarray`) – Array of chi-squared values for each energy bin.
- **chi2_spec** (*float*) – Global chi-squared value for the sum of all energy bins.
- **pval_spec** (*float*) – p-value of chi-squared for the best-fit spectrum.

fn_mles ()

returns the summed likelihood at the maximum likelihood estimate

Note that simply sums the maximum likelihood values at each bin, and does not impose any sort of constrain between bins

getIntervals (*alpha*)

Evaluate the two-sided intervals corresponding to a C.L. of (1-alpha)%.

Parameters **alpha** (*float*) – limit confidence level.

Returns

- **limit_vals_hi** (*ndarray*) – An array of lower limit values.
- **limit_vals_lo** (*ndarray*) – An array of upper limit values.

getLimits (*alpha, upper=True*)

Evaluate the limits corresponding to a C.L. of (1-alpha)%.

Parameters

- **alpha** (*float*) – limit confidence level.
- **upper** (*bool*) – upper or lower limits.
- **an array of values, one for each energy bin** (*returns*) –

mles ()

return the maximum likelihood estimates for each of the energy bins

nll_null

Return the negative log-likelihood for the null-hypothesis

norm_derivative (*spec, norm*)

norm_type

Return the normalization type flag

nx

Return the number of profiles

ny

Return the number of profiles

static stack_nll (*shape, components, ylims, weights=None*)

Combine the log-likelihoods from a number of components.

Parameters

- **shape** (*tuple*) – The shape of the return array
- **components** (*CastroData_Base*) – The components to be stacked
- **weights** (*array-like*) –

Returns

- **norm_vals** (*'numpy.ndarray'*) – N X M array of Normalization values
- **nll_vals** (*'numpy.ndarray'*) – N X M array of log-likelihood values

ts_vals ()

returns test statistic values for each energy bin

class fermipy.castro.Interpolator (*x, y*)

Bases: *object*

Helper class for interpolating a 1-D function from a set of tabulated values.

Safely deals with overflows and underflows

__call__(*x*)
Return the interpolated values for an array of inputs
x : the inputs
Note that if any *x* value is outside the interpolation ranges this will return a linear extrapolation based on the slope at the endpoint

derivative(*x*, *der=1*)
return the derivative a an array of input values
x : the inputs *der* : the order of derivative

x
return the *x* values used to construct the split

xmax
return the maximum value over which the spline is defined

xmin
return the minimum value over which the spline is defined

y
return the *y* values used to construct the split

class fermipy.castro.LnLFn(*x*, *y*, *norm_type=0*)
Bases: `object`
Helper class for interpolating a 1-D log-likelihood function from a set of tabulated values.

TS()
return the Test Statistic

fn_mle()
return the function value at the maximum likelihood estimate

getDeltaLogLike(*dlnl*, *upper=True*)
Find the point at which the log-likelihood changes by a given value with respect to its value at the MLE.

getInterval(*alpha*)
Evaluate the interval corresponding to a C.L. of (1-alpha)%.
Parameters *alpha* (*limit confidence level.*) –

getLimit(*alpha*, *upper=True*)
Evaluate the limits corresponding to a C.L. of (1-alpha)%.
Parameters
• *alpha* (*limit confidence level.*) –
• *upper* (*upper or lower limits.*) –

interp
return the underlying Interpolator object

mle()
return the maximum likelihood estimate
This will return the cached value, if it exists

norm_type
Return a string specifying the quantity used for the normalization. This isn't actually used in this class, but it is carried so that the class is self-describing. The possible values are open-ended.

```
class fermipy.castro.ReferenceSpec(emin, emax, ref_dnde, ref_flux, ref_eflux, ref_npred,  
                                   eref=None)
```

Bases: `object`

This class encapsulates data for a reference spectrum.

Parameters

- **ne** (`int`) – Number of energy bins
- **ebins** (`ndarray`) – Array of bin edges.
- **emin** (`ndarray`) – Array of lower bin edges.
- **emax** (`ndarray`) – Array of upper bin edges.
- **bin_widths** (`ndarray`) – Array of energy bin widths.
- **eref** (`ndarray`) – Array of reference energies. Typically these are the geometric mean of the energy bins
- **ref_dnde** (`ndarray`) – Array of differential photon flux values.
- **ref_flux** (`ndarray`) – Array of integral photon flux values.
- **ref_eflux** (`ndarray`) – Array of integral energy flux values.
- **ref_npred** (`ndarray`) – Array of predicted number of photons in each energy bin.

bin_widths

build_ebound_table()

Build and return an EBOUNDS table with the encapsulated data.

static create_from_table (*tab_e*)

Parameters **tab_e** (`Table`) – EBOUNDS table.

ebins

emax

emin

eref

log_ebins

nE

ref_dnde

ref_eflux

return the energy flux values

ref_flux

return the flux values

ref_npred

return the number of predicted events

```
class fermipy.castro.SpecData(ref_spec, norm, norm_err)
```

Bases: `fermipy.castro.ReferenceSpec`

This class encapsulates spectral analysis results (best-fit normalizations, errors, etc.), energy binning, and reference spectrum definition.

Parameters

- **norm**(*ndarray*) –
- **norm_err**(*ndarray*) –
- **flux**(*ndarray*) – Array of integral photon flux values.
- **eflux**(*ndarray*) – Array of integral energy flux values.
- **dnde**(*ndarray*) – Differential flux values
- **dnde_err**(*ndarray*) – Uncertainties on differential flux values
- **e2dnde**(*ndarray*) – Differential flux values scaled by E^2
- **e2dnde_err**(*ndarray*) – Uncertainties on differential flux values scaled by E^2

build_spec_table()

static create_from_table(*tab*)

dnde

dnde_err

e2dnde

e2dnde_err

eflux

flux

norm

norm_err

class `fermipy.castro.TSCube`(*tmap*, *normmap*, *tscube*, *normcube*, *norm_vals*, *nll_vals*, *refSpec*, *norm_type*)

Bases: `object`

A class wrapping a TSCube, which is a collection of CastroData objects for a set of directions.

This class wraps a combination of:

- Pixel data,
- Pixel x Energy bin data,
- Pixel x Energy Bin x Normalization scan point data

castroData_from_ipix(*ipix*, *colwise=False*)

Build a CastroData object for a particular pixel

castroData_from_pix_xy(*xy*, *colwise=False*)

Build a CastroData object for a particular pixel

static create_from_fits(*fitsfile*, *norm_type='flux'*)

Build a TSCube object from a fits file created by gttscribe :param fitsfile: Path to the tscube FITS file.

Parameters *norm_type* (*str*) – String specifying the quantity used for the normalization

find_and_refine_peaks(*threshold*, *min_separation=1.0*, *use_cumul=False*)

Run a simple peak-finding algorithm, and fit the peaks to paraboloids to extract their positions and error ellipses.

Parameters

- **threshold**(*float*) – Peak threshold in TS.

- **min_separation** (*float*) – Radius of region size in degrees. Sets the minimum allowable separation between peaks.
- **use_cumul** (*bool*) – If true, used the cumulative TS map (i.e., the TS summed over the energy bins) instead of the TS Map from the fit to and index=2 powerlaw.

Returns **peaks** – List of dictionaries containing the location and amplitude of each peak. Output of `find_peaks`

Return type *list*

find_sources (*threshold*, *min_separation=1.0*, *use_cumul=False*, *output_peaks=False*, *output_castro=False*, *output_specInfo=False*, *output_src_dicts=False*, *output_srcs=False*)

nE

return the number of energy bins

nN

return the number of sample points in each energy bin

normcube

return the Cube of the normalization value per pixel / energy bin

normmap

return the Map of the Best-fit normalization value

nvals

Return the number of values in the tscube

refSpec

Return the Spectral Data object

test_spectra_of_peak (*peak*, *spec_types=None*)

Test different spectral types against the SED represented by the `CastroData` corresponding to a single pixel in this `TSCube`

Parameters **spec_types** (*[str, ...]*) – List of spectral types to try

Returns

- **castro** (*CastroData*) – The castro data object for the pixel corresponding to the peak
- **test_dict** (*dict*) – The dictionary returned by `test_spectra`

ts_cumul

return the Map of the cumulative TestStatistic value per pixel (summed over energy bin)

tscube

return the Cube of the TestStatistic value per pixel / energy bin

tmap

return the Map of the TestStatistic value

`fermipy.castro.build_source_dict` (*src_name*, *peak_dict*, *spec_dict*, *spec_type*)

`fermipy.castro.convert_sed_cols` (*tab*)

Cast SED column names to lowercase.

fermipy.tsmap module

class `fermipy.tsmap.TSCubeGenerator`

Bases: `object`

tscube (*prefix*='', ***kwargs*)

Generate a spatial TS map for a source component with properties defined by the `model` argument. This method uses the `gttscube` ST application for source fitting and will simultaneously fit the test source normalization as well as the normalizations of any background components that are currently free. The output of this method is a dictionary containing `Map` objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **do_sed** (*bool*) – Compute the energy bin-by-bin fits.
- **nnorm** (*int*) – Number of points in the likelihood v. normalization scan.
- **norm_sigma** (*float*) – Number of sigma to use for the scan range.
- **tol** (*float*) – Criteria for fit convergence (estimated vertical distance to min < tol).
- **tol_type** (*int*) – Absolute (0) or relative (1) criteria for convergence.
- **max_iter** (*int*) – Maximum number of iterations for the Newton's method fitter
- **remake_test_source** (*bool*) – If true, recomputes the test source image (otherwise just shifts it)
- **st_scan_level** (*int*) –
- **make_plots** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write a FITS file with the results of the analysis.

Returns `maps` – A dictionary containing the `Map` objects for TS and source amplitude.

Return type `dict`

`class fermipy.tsmap.TSMapGenerator`

Bases: `object`

Mixin class for `GTAnalysis` that generates TS maps.

tsmap (*prefix*='', ***kwargs*)

Generate a spatial TS map for a source component with properties defined by the `model` argument. The TS map will have the same geometry as the ROI. The output of this method is a dictionary containing `Map` objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

This method uses a simplified likelihood fitting implementation that only fits for the normalization of the test source. Before running this method it is recommended to first optimize the ROI model (e.g. by running `optimize()`).

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files.
- **{options}** –

Returns `tsmap` – A dictionary containing the `Map` objects for TS and source amplitude.

Return type `dict`

`fermipy.tsmap.cash` (*counts*, *model*)

Compute the Poisson log-likelihood function.

`fermipy.tsmap.convert_tscube(infile, outfile)`

`fermipy.tsmap.convert_tscube_old(infile, outfile)`

Convert between old and new TSCube formats.

`fermipy.tsmap.extract_array(array_large, array_small, position)`

`fermipy.tsmap.extract_images_from_tscube(infile, outfile)`

Extract data from table HDUs in TSCube file and convert them to FITS images

`fermipy.tsmap.extract_large_array(array_large, array_small, position)`

`fermipy.tsmap.extract_small_array(array_small, array_large, position)`

`fermipy.tsmap.f_cash(x, counts, bkg, model)`

Wrapper for cash statistics, that defines the model function.

Parameters

- **x** (*float*) – Model amplitude.
- **counts** (*ndarray*) – Count map slice, where model is defined.
- **bkg** (*ndarray*) – Background map slice, where model is defined.
- **model** (*ndarray*) – Source template (multiplied with exposure).

`fermipy.tsmap.f_cash_sum(x, counts, bkg, model, bkg_sum=0, model_sum=0)`

`fermipy.tsmap.poisson_log_like(counts, model)`

Compute the Poisson log-likelihood function for the given counts and model arrays.

`fermipy.tsmap.truncate_array(array1, array2, position)`

Truncate array1 by finding the overlap with array2 when the array1 center is located at the given position in array2.

fermipy.residmap module

class `fermipy.residmap.ResidMapGenerator`

Bases: `object`

Mixin class for *GTAnalysis* that generates spatial residual maps from the difference of data and model maps smoothed with a user-defined spatial/spectral template. The map of residual significance can be interpreted in the same way as a TS map (the likelihood of a source at the given location).

residmap (*prefix=''*, ***kwargs*)

Generate 2-D spatial residual maps using the current ROI model and the convolution kernel defined with the `model` argument.

Parameters

- **prefix** (*str*) – String that will be prefixed to the output residual map files.
- **{options}** –

Returns `maps` – A dictionary containing the Map objects for the residual significance and amplitude.

Return type `dict`

`fermipy.residmap.convolve_map(m, k, cpix, threshold=0.001, imin=0, imax=None)`

Perform an energy-dependent convolution on a sequence of 2-D spatial maps.

Parameters

- **m** (`ndarray`) – 3-D map containing a sequence of 2-D spatial maps. First dimension should be energy.
- **k** (`ndarray`) – 3-D map containing a sequence of convolution kernels (PSF) for each slice in **m**. This map should have the same dimension as **m**.
- **cpix** (`list`) – Indices of kernel reference pixel in the two spatial dimensions.
- **threshold** (`float`) – Kernel amplitude
- **imin** (`int`) – Minimum index in energy dimension.
- **imax** (`int`) – Maximum index in energy dimension.

`fermipy.residmap.get_source_kernel(gta, name, kernel=None)`

Get the PDF for the given source.

`fermipy.residmap.poisson_lnl(nc, mu)`

fermipy.lightcurve module

class `fermipy.lightcurve.LightCurve`

Bases: `object`

lightcurve (`name, **kwargs`)

Generate a lightcurve for the named source. The function will complete the basic analysis steps for each bin and perform a likelihood fit for each bin. Extracted values (along with errors) are Integral Flux, spectral model, Spectral index, TS value, pred. # of photons.

Parameters

- **name** (`str`) – source name
- **{options}** –

Returns `LightCurve` – Dictionary containing output of the LC analysis

Return type `dict`

Module contents

`fermipy.get_st_version()`

Get the version string of the ST release.

`fermipy.test(package=None, test_path=None, args=None, plugins=None, verbose=False, pastebin=None, remote_data=False, pep8=False, pdb=False, coverage=False, open_files=False, **kwargs)`

Run the tests using `py.test`. A proper set of arguments is constructed and passed to `pytest.main`.

Parameters

- **package** (`str, optional`) – The name of a specific package to test, e.g. ‘io.fits’ or ‘utils’. If nothing is specified all default tests are run.
- **test_path** (`str, optional`) – Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.
- **args** (`str, optional`) – Additional arguments to be passed to `pytest.main` in the `args` keyword argument.
- **plugins** (`list, optional`) – Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

- **verbose** (*bool, optional*) – Convenience option to turn on verbose output from `py.test`. Passing True is the same as specifying `'-v'` in args.
- **pastebin** (`{'failed', 'all', None}`, *optional*) – Convenience option for turning on `py.test` pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.
- **remote_data** (*bool, optional*) – Controls whether to run tests marked with `@remote_data`. These tests use online data and are not run by default. Set to True to run these tests.
- **pep8** (*bool, optional*) – Turn on PEP8 checking via the `pytest-pep8` plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in args.
- **pdb** (*bool, optional*) – Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in args.
- **coverage** (*bool, optional*) – Generate a test coverage report. The result will be placed in the directory `htmlcov`.
- **open_files** (*bool, optional*) – Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Works only on platforms with a working `lsof` command.
- **parallel** (*int, optional*) – When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use all the cores on the machine. Requires the `pytest-xdist` plugin installed. Only available when using Astropy 0.3 or later.
- **kwargs** – Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

fermipy.jobs subpackage

fermipy.jobs.chain module

Utilities to execute command line applications.

The main class is `Link`, which wraps a single command line application.

The `Chain` class inherits from `Link` and allow chaining together several applications into a single object.

class `fermipy.jobs.chain.Chain` (*linkname, links, **kwargs*)

Bases: `fermipy.jobs.chain.Link`

An object tying together a series of applications into a single application.

This class keep track of the arguments to pass to the applications as well as input and output files.

Note that this class is itself a `Link`. This allows you to write a python module that implements a chain and also has a `__main__` function to allow it to be called from the shell.

Parameters `argmapper` (*function or None*) – Function that maps input options (in `self._options`) to the format that is passed to the links in the chains. If None, then no mapping is applied. This is useful if you want to build a complicated set of options from a few inputs.

__getitem__ (*key*)

Return the `Link` whose linkname is key

add_link (*link*)

Append link to this *Chain*

argmapper

Return the argument mapping function, if exists

get_jobs (*recursive=True*)

Return a dictionary with all the jobs

If recursive is True this will include jobs from internal *Link*

links

Return the list of links

map_arguments (*args*)

Map arguments to options.

This will use self._argmapper if it is defined.

Parameters

- **args** (dict or Namespace) – If a namespace is given, it will be cast to dict
- **dict** (*Returns*) –

missing_input_files ()

Return a dictionary of the missing input files and *Link* they are associated with

missing_output_files ()

Return a dictionary of the missing output files and *Link* they are associated with

print_summary (*stream=<open file '<stdout>', mode 'w'>, indent='', recurse_level=2*)

Print a summary of the activity done by this *Chain*.

Parameters

- **stream** (*file*) – Stream to print to
- **indent** (*str*) – Indentation at start of line
- **recurse_level** (*int*) – Number of recursion levels to print

run (*stream=<open file '<stdout>', mode 'w'>, dry_run=False*)

run_chain (*stream=<open file '<stdout>', mode 'w'>, dry_run=False, stage_files=True*)

Run all the links in the chain

Parameters

- **stream** (*file*) – Stream to print to
- **dry_run** (*bool*) – Print commands but do not run them
- **stage_files** (*bool*) – Stage files to and from the scratch area

update_args (*override_args*)

Update the argument used to invoke the application

Note that this will also update the dictionary of input and output files.

Parameters **override_args** (*dict*) – dictionary passed to the links

class fermipy.jobs.chain.**Link** (*linkname, **kwargs*)

Bases: *object*

A wrapper for a command line application.

This class keeps track for the arguments to pass to the application as well as input and output files.

This can be used either with other Link to build a chain, or as as standalone wrapper to pass configuration to the application.

Parameters

- **appname** (*str*) – Name of the application
- **args** (*dict*) – Up-to-date dictionary with the arguments that will be passed to the application
- **defaults** (*dict*) – Dictionary with defaults values for the arguments
- **options** (*dict*) – Dictionary with the options that we are allowed to set and default values
- **mapping** (*dict*) – Dictionary remapping keys in options to arguments sent to the application This is useful when two ScienceTools use different names for what is effectively the same parameter
- **files** (*FileDict*) – Object that keeps track of input and output files
- **jobs** (*OrderedDict*) – Dictionary mapping keys to *JobDetails*

arg_names

Return the list of arg names

check_input_files (*return_found=True, return_missing=True*)

Check if input files exist.

Return two lists: (found, missing)

check_output_files (*return_found=True, return_missing=True*)

Check if output files exist.

Return two lists: (found, missing)

command_template ()

Build and return a string that can be used as a template invoking this chain from the command line.

The actual command can be obtained by using `self.command_template().format(**self.args)`

create_job_details (*key, job_config, logfile, status*)

Create a *JobDetails* for a single job

Parameters

- **key** (*str*) – Key used to identify this particular job
- **job_config** (*dict*) – Dictionary with arguments passed to this particular job
- **logfile** (*str*) – Name of the associated log file
- **status** (*int*) – Current status of the job
- **JobDetails** (*Returns*) –

fill_argparser (*parser*)

Fill an `argparser.ArgumentParser` with the options from this chain

finalize (*dry_run=False*)

Remove / compress files as requested

formatted_command ()

Build and return the formatted command for this *Link*.

This is exactly the command as called from the Unix command line.

get_failed_jobs()

Return a dictionary with the subset of jobs that are marked as failed

get_jobs (*recursive=True*)

Return a dictionary with all the jobs

If recursive is True this will include jobs from internal [Link](#)

make_argv()

Generate the vector of arguments for this [Link](#).

This is exactly the 'argv' generated for the command as called from the Unix command line.

map_scratch_files (*file_dict*)

Build and return the mapping for copying files to and from scratch area

missing_input_files()

Make and return a dictionary of the missing input files.

This returns a dictionary mapping filepath to list of links that use the file as input.

missing_output_files()

Make and return a dictionary of the missing output files.

This returns a dictionary mapping filepath to list of links that product the file as output.

pre_run_checks (*stream=<open file '<stdout>', mode 'w'>, dry_run=False*)

Do some checks before running this link

This checks if input and output files are present.

If input files are missing this will raise `OSError` if `dry_run` is False If all output files are present this return False.

Parameters

- **stream** (*file*) – Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it
- **bool** (*Returns*) – True if it is ok to proceed with running the link

print_summary (*stream=<open file '<stdout>', mode 'w'>, indent='', recurse_level=2*)

Print a summary of the activity done by this [Link](#).

Parameters

- **stream** (*file*) – Stream to print to
- **indent** (*str*) – Indentation at start of line
- **recurse_level** (*int*) – Number of recursion levels to print

register_job (*key, job_config, logfile, status*)

Create a `JobDetails` for this link and add it to the self.jobs dictionary.

Parameters

- **key** (*str*) – Key used to identify this particular job
- **job_config** (*dict*) – Dictionary with arguments passed to this particular job
- **logfile** (*str*) – Name of the associated log file
- **status** (*int*) – Current status of the job
- **JobDetails** (*Returns*) –

run (*stream*=<open file '<stdout>', mode 'w'>, *dry_run*=False)

Runs this link.

This version is intended to be overwritten by sub-classes so as to provide a single function that behaves the same for all version of [Link](#)

Parameters

- **stream** (*file*) – Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it

run_argparser (*argv*)

Initialize a link with a set of arguments using an `argparser.ArgumentParser`

run_command (*stream*=<open file '<stdout>', mode 'w'>, *dry_run*=False)

Runs the command for this link. This method can be overridden by sub-classes to invoke a different command

Parameters

- **stream** (*file*) – Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it

run_link (*stream*=<open file '<stdout>', mode 'w'>, *dry_run*=False, *stage_files*=True)

Runs this link.

This checks if input and output files are present.

If input files are missing this will raise `OSError` if *dry_run* is False If all output files are present this will skip execution.

Parameters

- **stream** (*file*) – Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it
- **stage_files** (*bool*) – Stage files to and from the scratch area

set_file_stage (*file_stage*)

Set this link to use a `FileStageManager` to copy files to and from a scratch area

stage_input_files (*file_mapping*, *dry_run*=True)

Stage the input files to the scratch area and adjust the arguments accordingly

stage_output_files (*file_mapping*, *dry_run*=True)

Stage the input files to the scratch area and adjust the arguments accordingly

update_args (*override_args*)

Update the argument used to invoke the application

Note that this will also update the dictionary of input and output files.

Parameters **override_args** (*dict*) – Dictionary of arguments to override the current values

update_file_args (*file_mapping*)

Adjust the arguments to deal with staging files to the scratch area

update_options (*input_dict*)

Update the values in self.options

Parameters

- **input_dict** (*dict*) – Dictionary with argument key : value pairings

- **values into self._options** (*Inserts*) –

update_sub_file_dict (*sub_files*)
Update a file dict with information from self

`fermipy.jobs.chain.add_argument` (*parser, dest, info*)
Add an argument to an `argparse.ArgumentParser` object

`fermipy.jobs.chain.check_files` (*filelist, file_stage_manager=None, return_found=True, return_missing=True*)
Check that all files in a list exist
Return two lists: (found, missing)

`fermipy.jobs.chain.convert_dict_to_option_dict` (*input_dict*)
Convert a simple key-value dictionary to a dictionary of options tuples

`fermipy.jobs.chain.convert_option_dict_to_dict` (*option_dict*)
Convert a dictionary of options tuples to a simple key-value dictionary

`fermipy.jobs.chain.convert_value_to_option_tuple` (*value, helpstr=None*)
Convert a value to a tuple of the form expected by `Link.options`
Returns (value, helpstr, type(value))

`fermipy.jobs.chain.extract_arguments` (*args, defaults, mapping*)
Extract a set of arguments from a large dictionary

Parameters

- **args** (*dict*) – Dictionary with the arguments values to use
- **defaults** (*dict*) – Dictionary with all the argument to extract, and default values for each
- **mapping** (*dict*) – Dictionary mapping key in defaults to key in args This is useful: 1) when two applications use different names for what is effectively the same parameter 2) when you want to build a chain with multiple instances of the same application and pass different argument values to the different instances
- **dict filled with the arguments to pass to gtapp** (*Returns*) –

fermipy.jobs.file_archive module

Classes and utilites to keep track of files associated to an analysis.

The main class is `FileArchive`, which keep track of all the files associated to an analysis.

The `FileHandle` helper class encapsulates information on a particular file.

class `fermipy.jobs.file_archive.FileArchive` (***kwargs*)
Bases: `object`

Class that keeps track of the status of files used in an analysis

Parameters

- **table_file** (*str*) – Path to the file used to persist this `FileArchive`
- **table** (`astropy.table.Table`) – Persistent representation of this `FileArchive`
- **cache** (`OrderedDict`) – Transient representation of this `FileArchive`
- **base_path** (*str*) – Base file path for all files in this `FileArchive`

`__getitem__` (*key*)

Return the *FileHandle* whose linkname is key

`base_path`

Return the base file path for all files in this *FileArchive*

`static build_archive` (***kwargs*)

Return the singleton *FileArchive* instance, building it if needed

`cache`

Return the transient representation of this *FileArchive*

`static get_archive` ()

Return the singleton *FileArchive* instance

`get_file_ids` (*file_list*, *creator=None*, *status=0*, *file_dict=None*)

Get or create a list of file ids based on file names

Parameters

- `file_list` (*list*) – The paths to the file
- `creator` (*int*) – A unique key for the job that created these files
- `status` (*FileStatus*) – Enumeration giving current status of files
- `file_dict` (*FileDict*) – Mask giving flags set on this file
- `list of integers` (*Returns*) –

`get_file_paths` (*id_list*)

Get a list of file paths based on a set of ids

Parameters

- `id_list` (*list*) – List of integer file keys
- `list of file paths` (*Returns*) –

`get_handle` (*filepath*)

Get the *FileHandle* object associated to a particular file

`register_file` (*filepath*, *creator*, *status=0*, *flags=0*)

Register a file in the archive.

If the file already exists, this raises a *KeyError*

Parameters

- `filepath` (*str*) – The path to the file
- `creator` (*int*) – A unique key for the job that created this file
- `status` (*FileStatus*) – Enumeration giving current status of file
- `flags` (*FileFlags*) – Enumeration giving flags set on this file
- `FileHandle` (*Returns*) –

`table`

Return the persistent representation of this *FileArchive*

`table_file`

Return the path to the file used to persist this *FileArchive*

update_file (*filepath*, *creator*, *status*)

Update a file in the archive

If the file does not exists, this raises a `KeyError`

Parameters

- **filepath** (*str*) – The path to the file
- **creator** (*int*) – A unique key for the job that created this file
- **status** (*FileStatus*) – Enumeration giving current status of file
- **FileHandle** (*Returns*) –

update_file_status ()

Update the status of all the files in the archive

write_table_file (*table_file=None*)

Write the table to self._table_file

class fermipy.jobs.file_archive.**FileDict** (***kwargs*)

Bases: `object`

Small class to keep track of files used & created by a link.

Parameters

- **file_args** (*dict*) – Dictionary mapping argument [str] to *FileFlags* enum
- **file_dict** (*dict*) – Dictionary mapping file path [str] to *FileFlags* enum

chain_input_files

Return a list of the input files needed by this chain.

For `Link` sub-classes this will return only those files that were not created by any internal `Link`

chain_output_files

Return a list of the all the output files produced by this link.

For `Link` sub-classes this will return only those files that were not marked as internal files or marked for removal.

gzip_files

Return a list of the files compressed by this link.

This returns all files that were explicitly marked for compression.

input_files

Return a list of the input files needed by this link.

For `Link` sub-classes this will return the union of all the input files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

input_files_to_stage

Return a list of the input files needed by this link.

For `Link` sub-classes this will return the union of all the input files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

internal_files

Return a list of the intermediate files produced by this link.

This returns all files that were explicitly marked as internal files.

items()

Return iterator over self.file_dict

latch_file_info(args)

Extract the file paths from a set of arguments

output_files

Return a list of the output files produced by this link.

For `Link` sub-classes this will return the union of all the output files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

output_files_to_stage

Return a list of the input files needed by this link.

For `Link` sub-classes this will return the union of all the input files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

print_chain_summary(stream=<open file '<stdout>', mode 'w'>, indent='')

Print a summary of the files in this file dict.

This version uses chain_input_files and chain_output_files to count the input and output files.

print_summary(stream=<open file '<stdout>', mode 'w'>, indent='')

Print a summary of the files in this file dict.

This version explicitly counts the union of all input and output files.

temp_files

Return a list of the temporary files produced by this link.

This returns all files that were explicitly marked for removal.

update(file_dict)

Update self with values from a dictionary mapping file path [str] to *FileFlags* enum

class fermipy.jobs.file_archive.**FileFlags**

Bases: `object`

Bit masks to indicate file types

gz_mask = 8

in_ch_mask = 23

in_stage_mask = 33

input_mask = 1

internal_mask = 16

no_flags = 0

out_ch_mask = 22

out_stage_mask = 34

output_mask = 2

rm_mask = 4

rmint_mask = 20

stageable = 32

class fermipy.jobs.file_archive.**FileHandle** (**kwargs)
 Bases: `object`

Class to keep track of information about a file file.

Parameters

- **key** (*int*) – Unique id for this particular file
- **creator** (*int*) – Unique id for the job that created this file
- **timestamp** (*int*) – File creation time cast as an int
- **status** (*FileStatus*) – Enum giving current status of file
- **flags** (*FileFlags*) – Mask giving flags set on this file
- **path** (*str*) – Path to file

append_to_table (*table*)
 Add this instance as a row on a `astropy.table.Table`

check_status (*basepath=None*)
 Check on the status of this particular file

static create_from_row (*table_row*)
 Build and return a `FileHandle` from an `astropy.table.Row`

static make_dict (*table*)
 Build and return a dict of `FileHandle` from an `astropy.table.Table`
 The dictionary is keyed by `FileHandle.key`, which is a unique integer for each file

static make_table (*file_dict*)
 Build and return an `astropy.table.Table` to store `FileHandle`

update_table_row (*table, row_idx*)
 Update the values in an `astropy.table.Table` for this instances

class fermipy.jobs.file_archive.**FileStageManager** (*scratchdir, workdir*)
 Bases: `object`

Small class to deal with staging files to and from a scratch area

construct_scratch_path (*dirname, basename*)
 Construct and return a path in the scratch area.
 This will be <self.scratchdir>/<dirname>/<basename>

static copy_from_scratch (*file_mapping, dry_run=True*)
 Copy output files from scratch area

static copy_to_scratch (*file_mapping, dry_run=True*)
 Copy input files to scratch area

get_scratch_path (*local_file*)
 Construct and return a path in the scratch area from a local file.

static make_scratch_dirs (*file_mapping, dry_run=True*)
 Make any directories need in the scratch area

map_files (*local_files*)
 Build a dictionary mapping local paths to scratch paths.

Parameters

- **local_files** (*list*) – List of filenames to be mapped to scratch area
- **dict** (*Returns*) – Mapping local_file : fullpath of scratch file

split_local_path (*local_file*)

Split the local path into a directory name and a file name

If local_file is in self.workdir or a subdirectory of it, the directory will consist of the relative path from workdir.

If local_file is not in self.workdir, directory will be empty.

Returns (dirname, basename)

class fermipy.jobs.file_archive.**FileStatus**

Bases: *object*

Enumeration of file status types

exists = 2

expected = 1

missing = 3

no_file = 0

superseded = 4

temp_removed = 5

fermipy.jobs.file_archive.**get_timestamp**()

Get the current time as an integer

fermipy.jobs.file_archive.**get_unique_match** (*table, colname, value*)

Get the row matching value for a particular column. If exactly one row matches, return index of that row, Otherwise raise KeyError.

fermipy.jobs.file_archive.**main_browse**()

Entry point for command line use for browsing a FileArchive

fermipy.jobs.job_archive module

Classes and utilites to keep track the various jobs that comprise an analysis pipeline.

The main class is *JobArchive*, which keep track of all the jobs associated to an analysis.

The *JobDetails* helper class encapsulates information on a instance of running a job.

class fermipy.jobs.job_archive.**JobArchive** (***kwargs*)

Bases: *object*

Class that keeps of all the jobs associated to an analysis.

Parameters

- **table_file** (*str*) – Path to the file used to persist this *JobArchive*
- **table** (*astropy.table.Table*) – Persistent representation of this *JobArchive*
- **table_ids** (*astropy.table.Table*) – Ancillary table with information about file ids
- **file_archive** (*FileArchive*) – Archive with infomation about all this files used and produced by this analysis

```

__getitem__(fullkey)
    Return the JobDetails matching fullkey

static build_archive (**kwargs)
    Return the singleton JobArchive instance, building it if needed

static build_temp_job_archive ()
    Build and return a JobArchive using default locations of persistent files.

cache
    Return the transient representation of this JobArchive

file_archive
    Return the FileArchive with information about all the files used and produced by this analysis

static get_archive ()
    Return the singleton JobArchive instance

get_details (jobname, jobkey)
    Get the JobDetails associated to a particular job instance

make_job_details (row_idx)
    Create a JobDetails from an astropy.table.Row

register_job (job_details)
    Register a job in this JobArchive

register_job_from_link (link, key, **kwargs)
    Register a job in the JobArchive from a Link object

register_jobs (job_dict)
    Register a bunch of jobs in this archive

table
    Return the persistent representation of this JobArchive

table_file
    Return the path to the file used to persist this JobArchive

table_ids
    Return the persistent representation of the ancillary info of this JobArchive

update_job (job_details)
    Update a job in the JobArchive

update_job_status (checker_func)
    Update the status of all the jobs in the archive

write_table_file (job_table_file=None, file_table_file=None)
    Write the table to self._table_file

```

```

class fermipy.jobs.job_archive.JobDetails (**kwargs)
    Bases: object

```

A simple structure to keep track of the details of each of the sub-process jobs.

Parameters

- **dbkey** (*int*) – A unique key to identify this job
- **jobname** (*str*) – A name used to identify this job
- **jobkey** (*str*) – A string to identify this instance of the job
- **appname** (*str*) – The executable invoked to run the job

- **logfile** (*str*) – The logfile for this job, may be used to check for success/ failure
- **job_config** (*dict*) – A dictionary with the arguments for the job
- **parent_id** (*int*) – Unique key identifying the parent job
- **infile_ids** (*list of int*) – Keys to identify input files to this job
- **outfile_ids** (*list of int*) – Keys to identify output files from this job
- **rmfile_ids** (*list of int*) – Keys to identify temporary files removed by this job
- **intfile_ids** (*list of int*) – Keys to identify internal files
- **status** (*int*) – Current job status, one of the enums above

append_to_tables (*table, table_ids*)

Add this instance as a row on a `astropy.table.Table`

check_status_logfile (*checker_func*)

Check on the status of this particular job using the logfile

static create_from_row (*table_row, table_id_array*)

Create a `JobDetails` from an `astropy.table.Row`

fullkey

Return the fullkey for this job fullkey = <jobkey>@<jobname>

get_file_ids (*file_archive, creator=None, status=0*)

Fill the file id arrays from the file lists

Parameters

- **file_archive** (`FileArchive`) – Used to look up file ids
- **creator** (*int*) – A unique key for the job that created these file
- **status** (`FileStatus`) – Enumeration giving current status these files

get_file_paths (*file_archive, file_id_array*)

Get the full paths of the files used by this object from the the id arrays

static make_dict (*table, table_ids*)

Build a dictionary map int to `JobDetails` from an `astropy.table.Table`

static make_fullkey (*jobkey, jobname*)

Combine jobname and jobkey to make a unique key fullkey = <jobkey>@<jobname>

static make_tables (*job_dict*)

Build and return an `astropy.table.Table` to store 'JobDetails

static split_fullkey (*fullkey*)

Split fullkey to make extract jobname, jobkey fullkey = <jobkey>@<jobname>

update_table_row (*table, row_idx*)

Add this instance as a row on a `astropy.table.Table`

class `fermipy.jobs.job_archive.JobStatus`

Bases: `object`

Enumeration of job status types

done = 3

failed = 4

no_job = 0

pending = 1

running = 2

`fermipy.jobs.job_archive.get_matches(table, colname, value)`

Get the rows matching a value for a particular column.

`fermipy.jobs.job_archive.main_browse()`

Entry point for command line use for browsing a JobArchive

fermipy.jobs.gtlink module

Utilities to chain together a series of ScienceTools apps

class `fermipy.jobs.gtlink.Gtlink(linkname, **kwargs)`

Bases: `fermipy.jobs.chain.Link`

A wrapper for a single ScienceTools application

This class keeps track for the arguments to pass to the application as well as input and output files.

This can be used either with other `Link` to build a `Chain`, or as as standalone wrapper to pass configuration to the application.

See help for `chain.Link` for additional details

command_template()

Build and return a string that can be used as a template invoking this chain from the command line.

The actual command can be obtained by using `self.command_template().format(**self.args)`

get_gtapp()

Returns a `GtApp` object that will run this `Link`

run_command(stream=<open file '<stdout>', mode 'w'>, dry_run=False)

Runs the command for this link. This method can be overridden by sub-classes to invoke a different command

Parameters

- **stream(file)** – Must have ‘write’ function
- **dry_run(bool)** – Print command but do not run it

update_args(override_args)

Update the argument used to invoke the application

See help for `chain.Link` for details

This calls the base class function then fills the parameters of the `GtApp` object

`fermipy.jobs.gtlink.build_gtapp(appname, **kwargs)`

Build an object that can run ScienceTools application

Parameters

- **appname(str)** – Name of the application (e.g., `gtbin`)
- **kwargs(arguments used to invoke the application)** –
- **GtApp.GtApp object that will run the application in question**
(Returns) –

`fermipy.jobs.gtlink.extract_parameters(pil, keys=None)`

Extract and return parameter names and values from a `pil` object

Parameters

- **pil** (*Pil object*) –
- **keys** (*list*) – List of parameter names, if None, extract all parameters

Returns **out_dict** – Dictionary with parameter name, value pairs

Return type *dict*

`fermipy.jobs.gtlink.run_gtapp(gtapp, stream, dry_run, **kwargs)`

Runs one on the ScienceTools apps

Taken from `fermipy.gtanalysis.run_gtapp` by Matt Wood

Parameters

- **gtapp** (*GtApp.GtApp object*) – The application (e.g., `gtbin`)
- **stream** (*stream object*) – Must have ‘write’ function
- **dry_run** (*bool*) – Print command but do not run it
- **kwargs** (*arguments used to invoke the application*) –

`fermipy.jobs.gtlink.update_gtapp(gtapp, **kwargs)`

Update the parameters of the object that can run ScienceTools applications

Parameters

- **gtapp** (*GtApp.GtApp*) – Object that will run the application in question
- **kwargs** (*arguments used to invoke the application*) –

fermipy.jobs.lsf_impl module

Implementation of `ScatterGather` class for dealing with LSF batch jobs

class `fermipy.jobs.lsf_impl.LsfScatterGather(**kwargs)`

Bases: `fermipy.jobs.scatter_gather.ScatterGather`

Implmentation of `ScatterGather` that uses LSF

check_job (*job_details*)

Check the status of a single job

Returns str, one of ‘Pending’, ‘Running’, ‘Done’, ‘Failed’

default_options = {‘jobs_per_cycle’: (20, ‘Maximum number of jobs to submit in each cycle.’, <type ‘int’>), ‘time_p

dispatch_job_hook (*link, key, job_config, logfile*)

Send a single job to the LSF batch

Parameters

- **link** (*fermipy.jobs.chain.Link*) – The link used to invoke the command we are running
- **key** (*str*) – A string that identifies this particular instance of the job
- **job_config** (*dict*) – A dictionary with the arguments for the job. Used with the `self.command_template` job template
- **logfile** (*str*) – The logfile for this job, may be used to check for success/ failure

submit_jobs (*link, job_dict=None*)

Submit all the jobs in `job_dict`

```
fermipy.jobs.lsf_impl.build_bsub_command(command_template, lsf_args)
```

Build and return a lsf batch command template

The structure will be ‘bsub -s <key> <value> <command_template>’ where <key> and <value> refer to items in lsf_args

```
fermipy.jobs.lsf_impl.build_sg_from_link(link, config_maker, **kwargs)
```

Build a ScatterGather that will run multiple instance of a single link

```
fermipy.jobs.lsf_impl.check_log(logfile, exited='Exited with exit code', successful='Successfully completed')
```

Check a log file to determine status of LSF job

Often logfile doesn’t exist because the job hasn’t begun to run. It is unclear what you want to do in that case...

Parameters

- **logfile** (*str*) – String with path to logfile
- **exited** (*str*) – Value to check for in existing logfile for exit with failure
- **successful** (*str*) – Value to check for in existing logfile for success
- **str**, one of ‘Pending’, ‘Running’, ‘Done’, ‘Failed’ (*Returns*) –

```
fermipy.jobs.lsf_impl.get_lsf_status()
```

Count and print the number of jobs in various LSF states

fermipy.jobs.scatter_gather module

Abstract interface for parallel execution of multiple jobs.

The main class is *ScatterGather*, which can submit many instances of a job with different configurations.

The *ConfigMaker* abstract helper class is used to generate configurations.

```
class fermipy.jobs.scatter_gather.ConfigMaker(link, **kwargs)
```

Bases: *object*

Abstract helper class to build configuration dictionaries for parallel jobs.

Sub-classes will need to:

Define options by passing a dictionary of option tuples to the c’tor. This will take a form something like:

```
options=dict(string_opt=("default", "Some string", str), float_opt=(3.0, "Some float", float),
             list_opt=(None, "Some list", list))
```

```
add_arguments(parser, action)
```

Hook to add arguments to an *argparse.ArgumentParser*

```
add_options(option_dict)
```

Add options into an option dictionary

```
build_job_configs(args)
```

Hook to build job configurations

Sub-class implementation should return three dictionaries:

input_config [dict] Configuration options passed to initializers

job_configs [dict] Dictionary of dictionaries passed to parallel jobs

output_config [dict] Configuration options passed to gatherer

make_base_config (*args*)

Hook to build a baseline job configuration

Parameters *args* (*dict*) – Command line arguments, see `add_arguments`

class `fermipy.jobs.scatter_gather.ScatterGather` (***kwargs*)

Bases: `fermipy.jobs.chain.Link`

Abstract base class to dispatch several jobs in parallel and collect and merge the results.

__call__ (*argv*)

Parses command line arguments and runs the requested action

Parameters

- **argv** (*list-like*) – List of command line arguments
- **str with status** (*Returns*) –

base_config

Return the baseline job configuration

build_configs (*args*)

Build the configuration objects.

This invokes the `ConfigMaker` to build the configurations

build_job_dict ()

Build a dictionary of `JobDetails` objects for the internal `Link`

check_job (*job_details*)

Check the status of a specific job

check_status ()

Check on the status of all the jobs in job dict.

Returns

- **running** (*bool*) – True if jobs are still running
- **failed** (*bool*) – True if any jobs have failed

config_maker

Return the object used to translate arguments

default_gather_logfile = 'gather.log'

default_init_logfile = 'init.log'

default_options = {'dry_run': (False, 'Print commands, but do not execute them', <type 'bool'>), 'action': ('run', 'A

default_prefix_logfile = 'scatter'

dispatch_job (*link, key*)

Function to dispatch a single job

Parameters

- **link** (*Link*) – Link object that sends the job
- **key** (*str*) – Key used to identify this particular job
- **JobDetails object** (*Returns*) –

dispatch_job_hook (*link, key, job_config, logfile*)

Hook to dispatch a single job

gather_link

Return the `Link` object used the scatter phase of processing

gather_results()

Run the gather `Link` using `self._output_config` as input

Returns a `JobStatus` enum

get_jobs(*recursive=True*)

Return a dictionary with all the jobs

If recursive is `True` this will include jobs from all internal `Link`

initialize()

Run the initialization `Link` using `self._input_config` as input.

Returns a `JobStatus` enum

initialize_link

Return the `Link` object used to initial the processing

invoke(*argv*)

Invoke this object to preform a particular action

Parameters

- **argv** (*list-like*) – List of command line arguments, passed to helper classes
- **str with status** (*Returns*) –

job_archive

Return the `JobArchive` object

no_batch

Return the value of the `no_batch` flag

print_failed(*stream=<open file '<stderr>', mode 'w'>*)

Print list of the failed jobs

print_summary(*stream=<open file '<stdout>', mode 'w'>, indent='', recurse_level=2*)

Print a summary of the activity done by this `Link`.

Parameters

- **stream** (*file*) – Stream to print to
- **indent** (*str*) – Indentation at start of line
- **recurse_level** (*int*) – Number of recursion levels to print

print_update(*stream=<open file '<stdout>', mode 'w'>*)

Print an update about the current number of jobs running

resubmit()

Function to resubmit failed jobs and collect results

run(*stream=<open file '<stdout>', mode 'w'>, dry_run=False*)**run_argparser(*argv*)**

Initialize self with a set of arguments

run_jobs()

Function to dipatch jobs and collect results

scatter_link

Return the `Link` object used the scatter phase of processing

submit_jobs (*link*, *job_dict=None*)

Run the `Link` with all of the items `job_dict` as input.

If `job_dict` is `None`, the `job_dict` will be take from `link.jobs`

Returns a `JobStatus` enum

update_args (*override_args*)

Update the arguments used to invoke the application

Note that this will also update the dictionary of input and output files

Parameters `override_args` (*dict*) – dictionary of arguments to override the current values

`fermipy.jobs.scatter_gather.clean_job` (*logfile*, *outfiles*, *dry_run=False*)

Removes log file and files created by failed jobs.

If `dry_run` is `True`, print name of files to be removed, but do not remove them.

`fermipy.jobs.scatter_gather.remove_file` (*filepath*, *dry_run=False*)

Remove the file at `filepath`

Catches exception if the file does not exist.

If `dry_run` is `True`, print name of file to be removed, but do not remove it.

Module contents

fermipy.diffuse subpackage

fermipy.diffuse.binning module

Small helper class to represent the binning used for a single component of a summed likelihood in diffuse analysis

class `fermipy.diffuse.binning.Component` (***kwargs*)

Bases: `object`

Small helper class to represent the binning used for a single component of a summed likelihood in diffuse analysis

Parameters

- **log_e_min** (*float*) – Log base 10 of minimum energy for this component
- **log_e_max** (*float*) – Log base 10 of maximum energy for this component
- **enumbins** (*int*) – Number of energy bins for this component
- **zmax** (*float*) – Maximum zenith angle cube for this component in degrees
- **hpx_order** (*int*) – HEALPix order to use for this component

static build_from_energy_dict (*ebin_name*, *input_dict*)

Build a list of components from a dictionary for a single energy range

static build_from_yamlfile (*yamlfile*)

Build a list of components from a yaml file

static build_from_yamlstr (*yamlstr*)

Build a list of components from a yaml string

emax

Maximum energy for this component

emin
Minimum energy for this component

evtype
Event type bit mask for this component

make_key (*format_str*)
Make a key to identify this component
format_str is formatted using object `__dict__`

fermipy.diffuse.defaults module

Analysis framework for all-sky diffuse emission fitting

fermipy.diffuse.name_policy module

Handle the naming conventions for composite likelihood analysis

```
class fermipy.diffuse.name_policy.NameFactory (**kwargs)
    Bases: object

    Helper class to define file names and keys consistently.

    angprofile (**kwargs)
        return the file name for sun or moon angular profiles

    angprofile_format = 'templates/profile_{sourcekey}.fits'

    bexpcube (**kwargs)
        return the name of a binned exposure cube file

    bexpcube_format = 'bexp_cubes/bexpcube_{dataset}_{component}_{coordsys}_{irf_ver}.fits'

    bexpcube_moon (**kwargs)
        return the name of a binned exposure cube file

    bexpcube_sun (**kwargs)
        return the name of a binned exposure cube file

    bexpcubemoon_format = 'bexp_cubes/bexpcube_{dataset}_{component}_{irf_ver}_moon.fits'

    bexpcubesun_format = 'bexp_cubes/bexpcube_{dataset}_{component}_{irf_ver}_sun.fits'

    catalog_split_yaml (**kwargs)
        return the name of a catalog split yaml file

    catalog_split_yaml_format = 'models/{sourcekey}.yaml'

    ccube (**kwargs)
        return the name of a counts cube file

    ccube_format = 'counts_cubes/ccube_{dataset}_{component}_{coordsys}.fits'

    comp_srcmdl_xml (**kwargs)
        return the name of a source model file

    comp_srcmdl_xml_format = 'analysis/model_{modelkey}/srcmdl_{modelkey}_{component}.xml'

    component (**kwargs)
        Return a key that specifies data the sub-selection

    component_format = '{zcut}_{ebin}_{psftype}'
```

dataset (***kwargs*)
Return a key that specifies the data selection

dataset_format = '{data_pass}_{data_ver}_{data_time}_{evclass}'

diffuse_template (***kwargs*)
return the file name for other diffuse map templates

diffuse_template_format = 'templates/template_{sourcekey}.fits'

ft1file (***kwargs*)
return the name of the input ft1 file list

ft1file_format = '{dataset}_{zcut}.lst'

ft2file (***kwargs*)
return the name of the input ft2 file list

ft2file_format = 'ft2_{data_time}.lst'

fullpath (***kwargs*)
Return a full path name for a given file

fullpath_format = '{basedir}/{localpath}'

galprop_gasmap (***kwargs*)
return the file name for Galprop input gasmaps

galprop_gasmap_format = 'gasmap/{sourcekey}_{projtype}_{galprop_run}_{maptype}.fits.gz'

galprop_ringkey (***kwargs*)
return the sourcekey for galprop input maps : specifies the component and ring

galprop_ringkey_format = '{source_name}_{ringkey}'

galprop_rings_yaml (***kwargs*)
return the name of a galprop rings merging yaml file

galprop_rings_yaml_format = 'models/galprop_rings_{galkey}.yaml'

galprop_sourcekey (***kwargs*)
return the sourcekey for merged galprop maps : specifies the merged component and merging scheme

galprop_sourcekey_format = '{source_name}_{galpropkey}'

generic (*input_string*, ***kwargs*)
return a generic filename for a given dataset and component

irfs (***kwargs*)
Get the name of IFRs associated with a particular dataset

ltcube (***kwargs*)
return the name of a livetime cube file

ltcube_format = 'lt_cubes/ltcube_{data_time}_{zcut}.fits'

ltcube_moon (***kwargs*)
return the name of a livetime cube file

ltcube_sun (***kwargs*)
return the name of a livetime cube file

ltcubemoon_format = 'sunmoon/ltcube_{data_time}_{zcut}_moon.fits'

ltcubesun_format = 'sunmoon/ltcube_{data_time}_{zcut}_sun.fits'

make_filenames (***kwargs*)
Make a dictionary of filenames for various types

master_srcmdl_xml (***kwargs*)
return the name of a source model file

master_srcmdl_xml_format = 'analysis/model_{modelkey}/srcmdl_{modelkey}_master.xml'

mcube (***kwargs*)
return the name of a model cube file

mcube_format = 'model_cubes/mcube_{sourcekey}_{dataset}_{component}_{coordsys}_{irf_ver}.fits'

merged_gasmap (***kwargs*)
return the file name for Galprop merged gasmaps

merged_gasmap_format = 'merged_gasmaps/{sourcekey}_{projtype}_{maptype}.fits.gz'

merged_sourcekey (***kwargs*)
return the sourcekey for merged sets of point sources : specifies the catalog and merging rule

merged_sourcekey_format = '{catalog}_{rulekey}'

merged_srcmaps (***kwargs*)
return the name of a source map file

merged_srcmaps_format = 'analysis/model_{modelkey}/srcmaps_{dataset}_{component}_{coordsys}_{irf_ver}.fits'

model_yaml (***kwargs*)
return the name of a model yaml file

model_yaml_format = 'models/model_{modelkey}.yaml'

nested_srcmdl_xml (***kwargs*)
return the file name for source model xml files of nested sources

nested_srcmdl_xml_format = 'srcmdls/{sourcekey}_sources.xml'

residual_cr (***kwargs*)
Return the name of the residual CR analysis output files

residual_cr_format = 'residual_cr/residual_cr_{dataset}_{component}_{coordsys}_{irf_ver}.fits'

sourcekey (***kwargs*)
Return a key that specifies the name and version of a source or component

sourcekey_format = '{source_name}_{source_ver}'

spectral_template (***kwargs*)
return the file name for spectral templates

spectral_template_format = 'templates/spectral_{sourcekey}.txt'

srcmaps (***kwargs*)
return the name of a source map file

srcmaps_format = 'srcmaps/srcmaps_{sourcekey}_{dataset}_{component}_{coordsys}_{irf_ver}.fits'

srcmdl_xml (***kwargs*)
return the file name for source model xml files

srcmdl_xml_format = 'srcmdls/{sourcekey}.xml'

stamp (***kwargs*)
Return the path for a stamp file for a scatter gather job

stamp_format = 'stamps/{linkname}.stamp'

```
template_sunmoon (**kwargs)  
    return the file name for sun or moon template files  
  
templatesunmoon_format = 'templates/template_{sourcekey}_{zcut}.fits'  
  
update_base_dict (yamlfile)  
    Update the values in baseline dictionary used to resolve names
```

fermipy.diffuse.spectral module

Classes and utilities that manage spectral model specific to diffuse analyses

```
class fermipy.diffuse.spectral.SpectralLibrary (spectral_dict)  
    Bases: object  
  
    A small helper class that serves as an alias dictionary for spectral models  
  
    __getitem__ (key)  
        Get an item from the dictionary  
  
    static create_from_yaml (yamlfile)  
        Create the dictionary for a yaml file  
  
    static create_from_yamlstr (yamlstr)  
        Create the dictionary for a yaml file  
  
    update (spectral_dict)  
        Update the dictionary
```

fermipy.diffuse.source_factory module

Classes and utilities that create fermipy source objects

```
class fermipy.diffuse.source_factory.SourceFactory  
    Bases: object  
  
    Small helper class to build and keep track of sources  
  
    add_sources (source_info_dict)  
        Add all of the sources in source_info_dict to this factory  
  
    static build_catalog (catalog_type, catalog_file, catalog_extdir, **kwargs)  
        Build a fermipy.catalog.Catalog object  
  
        Parameters  
        • catalog_type (str) – Specifies catalog type, options include 2FHL | 3FGL | 4FGLP  
        • catalog_file (str) – FITS file with catalog tables  
        • catalog_extdir (str) – Path to directory with extended source templates  
  
    static copy_selected_sources (roi, source_names)  
        Build and return a fermipy.roi_model.ROIModel object by copying selected sources from another  
        such object  
  
    static make_fermipy_roi_model_from_catalogs (cataloglist)  
        Build and return a fermipy.roi_model.ROIModel object from a list of  
        fermipy.catalog.Catalog objects
```

static make_roi (*sources*={})

Build and return a *fermipy.roi_model.ROIModel* object from a dict with information about the sources

source_info_dict

Return the dictionary of source_info objects used to build sources

sources

Return the dictionary of sources

fermipy.diffuse.source_factory.make_catalog_sources (*catalog_roi_model*,
source_names)

Construct and return dictionary of sources that are a subset of sources in *catalog_roi_model*.

Parameters

- **catalog_roi_model** (dict or *fermipy.roi_model.ROIModel*) – Input set of sources
- **source_names** (*list*) – Names of sources to extract
- **dict mapping source_name to fermipy.roi_model.Source object**
(Returns) –

fermipy.diffuse.source_factory.make_composite_source (*name*, *cata-*
log_roi_model, *spectrum*,
nested_source_names)

Construct and return a *fermipy.roi_model.CompositeSource* object

fermipy.diffuse.source_factory.make_isotropic_source (*name*, *Spectrum_Filename*,
spectrum)

Construct and return a *fermipy.roi_model.IsoSource* object

fermipy.diffuse.source_factory.make_mapcube_source (*name*, *Spatial_Filename*, *spec-*
trum)

Construct and return a *fermipy.roi_model.MapCubeSource* object

fermipy.diffuse.source_factory.make_point_source (*name*, *src_dict*)

Construct and return a *fermipy.roi_model.Source* object

fermipy.diffuse.source_factory.make_sources (*comp_key*, *comp_dict*)

Make dictionary mapping component keys to a source or set of sources

Parameters

- **comp_key** (*str*) – Key used to access sources
- **comp_dict** (*dict*) – Information used to build sources
- **OrderedDict maping comp_key to fermipy.roi_model.Source**
(return) –

fermipy.diffuse.catalog_src_manager module

Classes and utilities that manage catalog sources

class *fermipy.diffuse.catalog_src_manager.CatalogComponentChain* (*linkname*)

Bases: *fermipy.jobs.chain.Chain*

Small class to build srcmaps for diffuse components

default_options = {'sources': (None, 'Path to yaml file defining catalog components.', <type 'str'>), 'make_xml': (Fa

run_argparser (*argv*)

Initialize a link with a set of arguments using argparse

class fermipy.diffuse.catalog_src_manager.**CatalogSourceManager** (**kwargs)

Bases: `object`

Small helper class to keep track of how we deal with catalog sources

This keeps track of two dictionaries

One of the dictionaries is keyed by catalog name, and contains information about complete catalogs `catalog_comp_info_dicts[catalog_name] : model_component.CatalogInfo`

The other dictionary is keyed by `[{catalog_name}_{split_ver}][{split_key}]` Where: `{catalog_name}` is something like '3FGL' `{split_ver}` is something like 'v00' and specifies how to divide sources in the catalog `{split_key}` refers to a specific sub-selection of sources

`split_comp_info_dicts[splitkey] : model_component.ModelComponentInfo`

build_catalog_info (*catalog_info*)

Build a CatalogInfo object

catalog_comp_info_dict (*catkey*)

Return the roi_model for an entire catalog

catalog_components (*catalog_name, split_ver*)

Return the set of merged components for a particular split key

catalogs ()

Return the list of full catalogs used

make_catalog_comp_info (*full_cat_info, split_key, rule_key, rule_val, sources*)

Make the information about a single merged component

Parameters

- **full_cat_info** (`_model_component.CatalogInfo`) – Information about the full catalog
- **split_key** (*str*) – Key identifying the version of the splitting used
- **rule_key** (*str*) – Key identifying the specific rule for this component
- **rule_val** (*list*) – List of the cuts used to define this component
- **sources** (*list*) – List of the names of the sources in this component
- **CompositeSourceInfo** or **CatalogSourcesInfo** (*Returns*) –

make_catalog_comp_info_dict (*catalog_sources*)

Make the information about the catalog components

Parameters **catalog_sources** (*dict*) – Dictionary with catalog source definitions

Returns

- **catalog_ret_dict** (*dict*) – Dictionary mapping `catalog_name` to `model_component.CatalogInfo`
- **split_ret_dict** (*dict*) – Dictionary mapping `sourcekey` to `model_component.ModelComponentInfo`

read_catalog_info_yaml (*splitkey*)

Read the yaml file for a particular split key

split_comp_info (*catalog_name, split_ver, split_key*)
Return the info for a particular split key

split_comp_info_dict (*catalog_name, split_ver*)
Return the information about a particular scheme for how to handle catalog sources

splitkeys ()
Return the list of catalog split keys used

`fermipy.diffuse.catalog_src_manager.create_chain_catalog_comps (**kwargs)`
Create and return a *CatalogComponentChain* object

`fermipy.diffuse.catalog_src_manager.main_chain ()`
Entry point for command line use for single job

`fermipy.diffuse.catalog_src_manager.make_catalog_comp_dict (**kwargs)`
Build and return the information about the catalog components

`fermipy.diffuse.catalog_src_manager.make_mask (cat_table, cut)`
Mask a bit mask selecting the rows that pass a selection

`fermipy.diffuse.catalog_src_manager.mask_extended (cat_table)`
Remove rows representing extended sources from a catalog table

`fermipy.diffuse.catalog_src_manager.select_extended (cat_table)`
Select only rows representing extended sources from a catalog table

`fermipy.diffuse.catalog_src_manager.select_sources (cat_table, cuts)`
Select only rows passing a set of cuts from catalog table

fermipy.diffuse.diffuse_src_manager module

Classes and utilities that manage the diffuse emission background models

class `fermipy.diffuse.diffuse_src_manager.DiffuseComponentChain` (*linkname*)
Bases: *fermipy.jobs.chain.Chain*

Small class to build srcmaps for diffuse components

default_options = {'make_xml': (False, 'Make XML files for diffuse components', <type 'bool'>), 'data': (None, 'Pa

run_argparser (*argv*)

Initialize a link with a set of arguments using argparser

class `fermipy.diffuse.diffuse_src_manager.DiffuseModelManager` (***kwargs*)
Bases: *object*

Small helper class to keep track of diffuse component templates

This keeps track of the 'diffuse component information' dictionary

This keyed by: key = {source_name}_{source_ver} Where: {source_name} is something like 'loopI'
{source_ver} is something like v00

The dictionary is `diffuse_comp_info_dict[key] -> model_component.ModelComponentInfo`

Note that some components (those that represent moving sources or are selection dependent) will have a sub-dictionary of `diffuse_comp_info_dict` object for each sub-component

The components are defined in a file called `config/diffuse_components.yaml`

diffuse_comp_info (*sourcekey*)

Return the Component info associated to a particular key

make_diffuse_comp_info (*source_name*, *source_ver*, *diffuse_dict*, *components=None*,
comp_key=None)

Make a dictionary mapping the merged component names to list of template files

Parameters

- **source_name** (*str*) – Name of the source
- **source_ver** (*str*) – Key identifying the version of the source
- **diffuse_dict** (*dict*) – Information about this component
- **comp_key** (*str*) – Used when we need to keep track of sub-components, i.e., for moving and selection dependent sources.
- **model_component.ModelComponentInfo** **or** (*Returns*) –
- **model_component.IsoComponentInfo** –

make_diffuse_comp_info_dict (*diffuse_sources*, *components*)

Make a dictionary mapping from diffuse component to information about that component

Parameters

- **diffuse_sources** (*dict*) – Dictionary with diffuse source definitions
- **components** (*dict*) – Dictionary with event selection definitions, needed for selection dependent diffuse components

Returns **ret_dict** – Dictionary mapping sourcekey to `model_component.ModelComponentInfo`

Return type *dict*

make_template_name (*model_type*, *sourcekey*)

Make the name of a template file for particular component

Parameters

- **model_type** (*str*) – Type of model to use for this component
- **sourcekey** (*str*) – Key to identify this component
- **filename** **or** **None** **if component does not require a template file** (*Returns*) –

make_xml_name (*sourcekey*)

Make the name of an xml file for a model definition of a single component

Parameters **sourcekey** (*str*) – Key to identify this component

static read_diffuse_component_yaml (*yamlfile*)

Read the yaml file for the diffuse components

sourcekeys ()

Return the list of source keys

class `fermipy.diffuse.diffuse_src_manager.GalpropMapManager` (***kwargs*)

Bases: `object`

Small helper class to keep track of Galprop gasmaps

This keeps track of two types of dictionaries. Both are keyed by: `key = {source_name}_{ring}_{galkey}`

Where: `{source_name}` is something like ‘merged_C0’ `{ring}` is the ring index `{galkey}` is a key specifying which version of galprop rings to use.

The two dictionaries are: `ring_dict[key] = model_component.GalpropMergedRingInfo` `diffuse_comp_info_dict[key] = model_component.ModelComponentInfo`

The dictionaries are defined in files called. `models/galprop_rings_{galkey}.yaml`

diffuse_comp_info_dicts (*galkey*)

Return the components info dictionary for a particular galprop key

galkeys ()

Return the list of galprop keys used

make_diffuse_comp_info (*merged_name, galkey*)

Make the information about a single merged component

Parameters

- **merged_name** (*str*) – The name of the merged component
- **galkey** (*str*) – A short key identifying the galprop parameters
- **odel_component.ModelComponentInfo** (*Returns*) –

make_diffuse_comp_info_dict (*galkey*)

Make a dictionary mapping from merged component to information about that component

Parameters **galkey** (*str*) – A short key identifying the galprop parameters

make_merged_name (*source_name, galkey, fullpath*)

Make the name of a gasmap file for a set of merged rings

Parameters

- **source_name** (*str*) – The galprop component, used to define path to gasmap files
- **galkey** (*str*) – A short key identifying the galprop parameters
- **fullpath** (*bool*) – Return the full path name

make_ring_dict (*galkey*)

Make a dictionary mapping the merged component names to list of template files

Parameters

- **galkey** (*str*) – Unique key for this ring dictionary
- **model_component.GalpropMergedRingInfo** (*Returns*) –

make_ring_filelist (*sourcekeys, rings, galprop_run*)

Make a list of all the template files for a merged component

Parameters

- **sourcekeys** (*list-like of str*) – The names of the components to merge
- **rings** (*list-like of int*) – The indices of the rings to merge
- **galprop_run** (*str*) – String identifying the galprop parameters

make_ring_filename (*source_name, ring, galprop_run*)

Make the name of a gasmap file for a single ring

Parameters

- **source_name** (*str*) – The galprop component, used to define path to gasmap files
- **ring** (*int*) – The ring index
- **galprop_run** (*str*) – String identifying the galprop parameters

make_xml_name (*source_name, galkey, fullpath*)

Make the name of an xml file for a model definition for a set of merged rings

Parameters

- **source_name** (*str*) – The galprop component, used to define path to gasmap files
- **galkey** (*str*) – A short key identifying the galprop parameters
- **fullpath** (*bool*) – Return the full path name

merged_components (*galkey*)

Return the set of merged components for a particular galprop key

read_galprop_rings_yaml (*galkey*)

Read the yaml file for a particular galprop key

ring_dict (*galkey*)

Return the ring dictionary for a particular galprop key

`fermipy.diffuse.diffuse_src_manager.create_chain_diffuse_comps (**kwargs)`

Create and return a *DiffuseComponentChain* object

`fermipy.diffuse.diffuse_src_manager.main_chain()`

Entry point for command line use for single job

`fermipy.diffuse.diffuse_src_manager.make_diffuse_comp_info_dict (**kwargs)`

Build and return the information about the diffuse components

`fermipy.diffuse.diffuse_src_manager.make_ring_dicts (**kwargs)`

Build and return the information about the Galprop rings

fermipy.diffuse.model_component module

Small helper classes that wrap information about model components

class `fermipy.diffuse.model_component.CatalogInfo (**kwargs)`

Bases: *object*

Information about a source catalog

Parameters

- **catalog_name** (*str*) – The name given to the merged component, e.g., merged_CO or merged_HI
- **catalog_file** (*str*) – Fits file with catalog data
- **catalog_extdir** (*str*) – Directory with extended source templates
- **catalog_type** (*str*) – Identifies the format of the catalog fits file: e.g., ‘3FGL’ or ‘4FGLP’
- **catalog** (`fermipy.catalog.Catalog`) – Catalog object
- **roi_model** (`fermipy.roi_model.ROIModel`) – Fermipy object describing all the catalog sources
- **srcmdl_name** (*str*) – Name of xml file with the catalog source model

update (***kwargs*)

Update data members from keyword arguments

class `fermipy.diffuse.model_component.CatalogSourcesInfo (**kwargs)`

Bases: `fermipy.diffuse.model_component.ModelComponentInfo`

Information about a model component consisting of sources from a catalog

Parameters

- **source_names** (*list*) – The names of the nested sources
- **catalog_info** (*model_component.CatalogInfo* or *None*) – Information about the catalog containing the nested sources
- **roi_model** (*fermipy.roi_model.ROIModel*) – Fermipy object describing the nested sources

class *fermipy.diffuse.model_component.CompositeSourceInfo* (**kwargs)
 Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component represented by a CompositeSource

Parameters

- **source_names** (*list*) – The names of the nested sources
- **catalog_info** (*model_component.CatalogInfo* or *None*) – Information about the catalog containing the nested sources
- **roi_model** (*fermipy.roi_model.ROIModel*) – Fermipy object describing the nested sources

class *fermipy.diffuse.model_component.GalpropMergedRingInfo* (**kwargs)
 Bases: *object*

Information about a set of Merged Galprop Rings

Parameters

- **source_name** (*str*) – The name given to the merged component, e.g., merged_CO or merged_HI
- **ring** (*int*) – The index of the merged ring
- **sourcekey** (*str*) – Key that identifies this component, e.g., merged_CO_1, or merged_HI_3
- **galkey** (*str*) – Key that identifies how to merge the galprop rings, e.g., ‘ref’
- **galprop_run** (*str*) – Key that identifies the galprop run used to make the input rings
- **files** (*str*) – List of files of the input gasmap files
- **merged_gasmap** (*str*) – Filename for the merged gasmap

update (**kwargs)

Update data members from keyword arguments

class *fermipy.diffuse.model_component.IsoComponentInfo* (**kwargs)
 Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component represented by a IsoSource

Parameters **Spectral_Filename** (*str*) – Name of the template file for the spatial model

class *fermipy.diffuse.model_component.MapCubeComponentInfo* (**kwargs)
 Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component represented by a MapCubeSource

Parameters **Spatial_Filename** (*str*) – Name of the template file for the spatial model

class `fermipy.diffuse.model_component.ModelComponentInfo` (***kwargs*)

Bases: `object`

Information about a model component

Parameters

- **source_name** (*str*) – The name given to the component, e.g., `loop_I` or `moon`
- **source_ver** (*str*) – Key to indentify the model version of the source, e.g., `v00`
- **sourcekey** (*str*) – Key that identifies this component, e.g., `loop_I_v00` or `moon_v00`
- **model_type** (*str*) – Type of model, ‘MapCubeSource’ | ‘IsoSource’ | ‘Composite-Source’ | ‘Catalog’ | ‘PointSource’
- **srcmdl_name** (*str*) – Name of the xml file with the xml just for this component
- **moving** (*bool*) – Flag for moving sources (i.e., the sun and moon)
- **selection_dependent** (*bool*) – Flag for selection dependent sources (i.e., the residual cosmic ray model)
- **components** (*dict*) – Sub-dictionary of `ModelComponentInfo` objects for moving and selection_dependent sources
- **comp_key** (*str*) – Component key for this component of moving and selection_dependent sources

add_component_info (*compinfo*)

Add sub-component specific information to a particular data selection

Parameters **compinfo** (`ModelComponentInfo` object) – Sub-component being added

clone_and_merge_sub (*key*)

Clones self and merges clone with sub-component specific information

Parameters

- **key** (*str*) – Key specifying which sub-component
- **ModelComponentInfo object** (*Returns*) –

get_component_info (*comp*)

Return the information about sub-component specific to a particular data selection

Parameters

- **comp** (`binning.Component` object) – Specifies the sub-component
- **ModelComponentInfo object** (*Returns*) –

update (***kwargs*)

Update data members from keyword arguments

class `fermipy.diffuse.model_component.PointSourceInfo` (***kwargs*)

Bases: `fermipy.diffuse.model_component.ModelComponentInfo`

Information about a model component represented by a PointSource

fermipy.diffuse.model_manager module

Classes and utilities that manage fitting models for diffuse analyses

class `fermipy.diffuse.model_manager.ModelComponent` (***kwargs*)

Bases: `object`

Small helper class to tie a ModelComponentInfo to a spectrum

class `fermipy.diffuse.model_manager.ModelInfo` (***kwargs*)

Bases: `object`

Small helper class to keep track of a single fitting model

`__getitem__` (*key*)

Return a single model info by name

`component_names`

Return the list of name of the components

`items` ()

Return the key, value pairs of model components

`make_model_rois` (*components, name_factory*)

Make the fermipy roi_model objects for each of a set of binning components

`make_srcmap_manifest` (*components, name_factory*)

Build a yaml file that specifies how to make the srcmap files for a particular model

Parameters

- **components** (*list*) – The binning components used in this analysis
- **name_factory** (*NameFactory*) – Object that handles naming conventions
- **a dictionary that contains information about where to find the** (*Returns*) –
- **maps for each component of the model** (*source*) –

class `fermipy.diffuse.model_manager.ModelManager` (***kwargs*)

Bases: `object`

Small helper class to create fitting models and manager XML files for fermipy

This class contains a ‘library’, which is a dictionary of all the source components:

specifically it maps:

`sourcekey` : `model_component.ModelComponentInfo`

csm

Return the CatalogSourceManager

dmm

Return the DiffuseModelManager

static `get_sub_comp_info` (*source_info, comp*)

Build and return information about a sub-component for a particular selection

gmm

Return the GalpropMapManager

make_fermipy_config_yaml (*modelkey, components, data, **kwargs*)

Build a fermipy top-level yaml configuration file

Parameters

- **modelkey** (*str*) – Key used to identify this particular model
- **components** (*list*) – The binning components used in this analysis

- **data** (*str*) – Path to file containing dataset definition

make_library (*diffuse_yaml, catalog_yaml, binning_yaml*)

Build up the library of all the components

Parameters

- **diffuse_yaml** (*str*) – Name of the yaml file with the library of diffuse component definitions
- **catalog_yaml** (*str*) – Name of the yaml file with the library of catalog split definitions
- **binning_yaml** (*str*) – Name of the yaml file with the binning definitions

make_model_info (*modelkey*)

Build a dictionary with the information for a particular model.

Parameters

- **modelkey** (*str*) – Key used to identify this particular model
- **ModelInfo** (*Return*) –

make_srcmap_manifest (*modelkey, components, data*)

Build a yaml file that specifies how to make the srcmap files for a particular model

Parameters

- **modelkey** (*str*) – Key used to identify this particular model
- **components** (*list*) – The binning components used in this analysis
- **data** (*str*) – Path to file containing dataset definition

read_model_yaml (*modelkey*)

Read the yaml file for the diffuse components

`fermipy.diffuse.model_manager.make_library(**kwargs)`

Build and return a ModelManager object and fill the associated model library

fermipy.diffuse.job_library module

Module to collect configuration to run specific jobs

class `fermipy.diffuse.job_library.ConfigMaker_Gtexpcube2` (*link, **kwargs*)

Bases: `fermipy.jobs.scatter_gather.ConfigMaker`

Small class to generate configurations for gtexpcube2

This takes the following arguments: `-comp` : binning component definition yaml file `-data` : dataset definition yaml file `-irf_ver` : IRF versions string (e.g., 'V6') `-coordsys` : Coordinate system ['GAL' | 'CEL'] `-hpx_order`: HEALPix order parameter

build_job_configs (*args*)

Hook to build job configurations

default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'comp': (None, 'Path to yaml file defining

class `fermipy.diffuse.job_library.ConfigMaker_SrcmapsCatalog` (*link, **kwargs*)

Bases: `fermipy.jobs.scatter_gather.ConfigMaker`

Small class to generate configurations for gtsrcmaps for catalog sources

This takes the following arguments: `--comp` : binning component definition yaml file `--data` : dataset definition yaml file `--irf_ver` : IRF versions string (e.g., 'V6') `--sources` : Yaml file with input source model definitions `--make_xml` : Write xml files for the individual components

build_job_configs (*args*)

Hook to build job configurations

default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'sources': (None, 'Path to yaml file defining sources')}

class `fermipy.diffuse.job_library.ConfigMaker_SumRings` (*link*, ***kwargs*)

Bases: `fermipy.jobs.scatter_gather.ConfigMaker`

Small class to generate configurations for fermipy-coadd to sum galprop ring gasmaps

This takes the following arguments: `--diffuse` : Diffuse model component definition yaml file `--outdir` : Output directory

build_job_configs (*args*)

Hook to build job configurations

default_options = {'diffuse': (None, 'Path to yaml file defining diffuse components.', <type 'str'>), 'outdir': (None, 'Path to output directory')}

class `fermipy.diffuse.job_library.ConfigMaker_Vstack` (*link*, ***kwargs*)

Bases: `fermipy.jobs.scatter_gather.ConfigMaker`

Small class to generate configurations for fermipy-vstack to merge source maps

This takes the following arguments: `--comp` : binning component definition yaml file `--data` : dataset definition yaml file `--irf_ver` : IRF versions string (e.g., 'V6') `--diffuse` : Diffuse model component definition yaml file

build_job_configs (*args*)

Hook to build job configurations

default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'comp': (None, 'Path to yaml file defining components')}

class `fermipy.diffuse.job_library.ConfigMaker_healview` (*link*, ***kwargs*)

Bases: `fermipy.jobs.scatter_gather.ConfigMaker`

Small class to generate configurations for fermipy-healview to display source maps

This takes the following arguments: `--comp` : binning component definition yaml file `--data` : dataset definition yaml file `--irf_ver` : IRF versions string (e.g., 'V6') `--diffuse` : Diffuse model component definition yaml file

build_job_configs (*args*)

Hook to build job configurations

default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'comp': (None, 'Path to yaml file defining components')}

`fermipy.diffuse.job_library.create_link_fermipy_coadd` (***kwargs*)

Make a `fermipy.jobs.Link` object to run fermipy-coadd

`fermipy.diffuse.job_library.create_link_fermipy_healview` (***kwargs*)

Make a `fermipy.jobs.Link` object to run fermipy-healview

`fermipy.diffuse.job_library.create_link_fermipy_vstack` (***kwargs*)

Make a `fermipy.jobs.Link` object to run fermipy-vstack

`fermipy.diffuse.job_library.create_link_gtexpcube2` (***kwargs*)

Make a `fermipy.jobs.Gtlink` object to run gtexpcube2

`fermipy.diffuse.job_library.create_link_gtsrcmaps` (***kwargs*)

Make a `fermipy.jobs.Gtlink` object to run gtsrcmaps

`fermipy.diffuse.job_library.create_sg_gtexpcube2` (***kwargs*)

Build and return a `ScatterGather` object that can invoke gtexpcube2

```
fermipy.diffuse.job_library.create_sg_gtsrcmaps_catalog(**kwargs)
    Build and return a ScatterGather object that can invoke gtsrcmaps for catalog sources

fermipy.diffuse.job_library.create_sg_healview_diffuse(**kwargs)
    Build and return a ScatterGather object that can invoke fermipy-healview

fermipy.diffuse.job_library.create_sg_sum_ring_gasmaps(**kwargs)
    Build and return a ScatterGather object that can invoke fermipy-coadd

fermipy.diffuse.job_library.create_sg_vstack_diffuse(**kwargs)
    Build and return a ScatterGather object that can invoke fermipy-vstack

fermipy.diffuse.job_library.invoke_sg_gtexpcube2()
    Entry point for command line use for dispatching batch jobs

fermipy.diffuse.job_library.invoke_sg_gtsrcmaps_catalog()
    Entry point for command line use for dispatching batch jobs

fermipy.diffuse.job_library.invoke_sg_healview_diffuse()
    Entry point for command line use for dispatching batch jobs

fermipy.diffuse.job_library.invoke_sg_sum_ring_gasmaps()
    Entry point for command line use for dispatching batch jobs

fermipy.diffuse.job_library.invoke_sg_vstack_diffuse()
    Entry point for command line use for dispatching batch jobs
```

fermipy.diffuse.gt_assemble_model module

Merge source maps to build composite sources

```
class fermipy.diffuse.gt_assemble_model.ConfigMaker_AssembleModel(link,
                                                                **kwargs)
```

Bases: *fermipy.jobs.scatter_gather.ConfigMaker*

Small class to generate configurations for this script

Parameters

- **--comp** (binning component definition yaml file)–
- **--data** (dataset definition yaml file)–
- **--hpx_order** (Maximum HEALPix order to use)–
- **--irf_ver** (IRF versions string (e.g., 'V6'))–
- **args** (Names of models to assemble source maps for)–

```
build_job_configs(args)
```

Hook to build job configurations

```
default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'hpx_order': (7, 'Maximum HEALPIX or
```

```
class fermipy.diffuse.gt_assemble_model.GtAssembleModel(**kwargs)
```

Bases: *object*

Small class to assemble source map files for fermipy analysis.

This is useful for re-merging after parallelizing source map creation.

```
static append_hdus(hdulist, srcmap_file, source_names, hpx_order)
```

Append HEALPix maps to a list

Parameters

- **hdulist** (*list*) – The list being appended to
- **srcmap_file** (*str*) – Path to the file containing the HDUs
- **source_names** (*list of str*) – Names of the sources to extract from srcmap_file
- **hpx_order** (*int*) – Maximum order for maps

static assemble_component (*compname, compinfo, hpx_order*)
Assemble the source map file for one binning component

Parameters

- **compname** (*str*) – The key for this component (e.g., E0_PSF3)
- **compinfo** (*dict*) – Information about this component
- **hpx_order** (*int*) – Maximum order for maps

static copy_ccube (*ccube, outsrcmap, hpx_order*)
Copy a counts cube into outsrcmap file reducing the HEALPix order to hpx_order if needed.

default_options = {'comp': (None, 'Path to yaml file defining binning.', <type 'str'>), 'hpx_order': (7, 'Maximum H

static make_link (***kwargs*)
Make a `fermipy.jobs.Link` object to run `GtAssembleModel`

static make_parser ()
Make an argument parser for this class

static open_outsrcmap (*outsrcmap*)
Open and return the outsrcmap file in append mode

run (*argv*)
Assemble the source map file for one binning component FIXME

class `fermipy.diffuse.gt_assemble_model.GtInitModel` (***kwargs*)
Bases: `object`

Small class to preprate files fermipy analysis.

Specifically this create the srcmap_manifest and fermipy_config_yaml files

default_options = {'sources': (None, 'Path to yaml file defining catalog components.', <type 'str'>), 'comp': (None, 'I

static make_link (***kwargs*)
Make a `fermipy.jobs.Link` object to run `GtAssembleModel`

static make_parser ()
Make an argument parser for this class

run (*argv*)
Assemble the source map file for one binning component FIXME

`fermipy.diffuse.gt_assemble_model.create_link_assemble_model` (***kwargs*)
Build and return a `Link` object that can invoke `GtAssembleModel`

`fermipy.diffuse.gt_assemble_model.create_sg_assemble_model` (***kwargs*)
Build and return a `ScatterGather` object that can invoke this script

`fermipy.diffuse.gt_assemble_model.main_batch` ()
Entry point for command line use for dispatching batch jobs

`fermipy.diffuse.gt_assemble_model.main_init` ()
Entry point for command line use for init job

`fermipy.diffuse.gt_assemble_model.main_single()`
 Entry point for command line use for single job

fermipy.diffuse.gt_coadd_split module

Prepare data for diffuse all-sky analysis

class `fermipy.diffuse.gt_coadd_split.CoaddSplit` (*linkname, comp_dict=None*)
 Bases: `fermipy.jobs.chain.Chain`

Small class to merge counts cubes for a series of binning components

run_argparser (*argv*)
 Initialize a link with a set of arguments using argparser

update_links (*comp_dict*)
 Build the links in this chain from the binning specification

`fermipy.diffuse.gt_coadd_split.create_chain_coadd_split(**kwargs)`
 Build and return a `CoaddSplit` object

`fermipy.diffuse.gt_coadd_split.main()`
 Entry point for command line use

fermipy.diffuse.gt_merge_srcmaps module

Merge source maps to build composite sources

class `fermipy.diffuse.gt_merge_srcmaps.ConfigMaker_MergeSrcmaps` (*link, **kwargs*)
 Bases: `fermipy.jobs.scatter_gather.ConfigMaker`

Small class to generate configurations for this script

This adds the following arguments: `-comp` : binning component definition yaml file `-data` : dataset definition yaml file `-irf_ver` : IRF versions string (e.g., 'V6') `-sources` : Catalog model component definition yaml file

build_job_configs (*args*)
 Hook to build job configurations

default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'sources': (None, 'Path to yaml file defin

class `fermipy.diffuse.gt_merge_srcmaps.GtMergeSourceMaps` (***kwargs*)
 Bases: `object`

Small class to merge source maps for composite sources.

This is useful for parallelizing source map creation.

NULL_MODEL = 'srcmdls/null.xml'

default_options = {'irfs': ('CALDB', 'Instrument response functions', <type 'str'>), 'expcube': (None, 'Input Liveti

static make_link (***kwargs*)
 Make a `fermipy.jobs.Link` object to run 'GtMergeSourceMaps

static make_parser ()
 Make an argument parser for this class

run (*argv*)
 Run this analysis

`fermipy.diffuse.gt_merge_srcmaps.create_link_merge_srcmaps(**kwargs)`
 Build and return a `Link` object that can invoke `GtAssembleModel`

```
fermipy.diffuse.gt_merge_srcmaps.create_sg_merge_srcmaps (**kwargs)
```

Build and return a ScatterGather object that can invoke this script

```
fermipy.diffuse.gt_merge_srcmaps.main_batch()
```

Entry point for command line use for dispatching batch jobs

```
fermipy.diffuse.gt_merge_srcmaps.main_single()
```

Entry point for command line use for single job

fermipy.diffuse.gt_split_and_bin module

Prepare data for diffuse all-sky analysis

```
class fermipy.diffuse.gt_split_and_bin.ConfigMaker_SplitAndBin(chain, gather,
                                                             **kwargs)
```

Bases: *fermipy.jobs.scatter_gather.ConfigMaker*

Small class to generate configurations for SplitAndBin

```
build_job_configs (args)
```

Hook to build job configurations

```
default_options = {'coordsys': ('GAL', 'Coordinate system of the spatial projection (CEL or GAL).', <type 'str'>), 'l'
```

```
make_base_config (args)
```

Hook to build a baseline job configuration

Parameters *args* (*dict*) – Command line arguments, see *add_arguments*

```
class fermipy.diffuse.gt_split_and_bin.SplitAndBin(linkname, comp_dict=None)
```

Bases: *fermipy.jobs.chain.Chain*

Small class to split and bin data according to some user-provided specification

```
run_argparser (argv)
```

Initialize a link with a set of arguments using argparser

```
update_links (comp_dict)
```

Build the links in this chain from the binning specification

```
fermipy.diffuse.gt_split_and_bin.create_chain_split_and_bin (**kwargs)
```

Make a *fermipy.jobs.SplitAndBin*

```
fermipy.diffuse.gt_split_and_bin.create_inputlist (arglist)
```

Read lines from a file and makes a list of file names.

Removes whitespace and lines that start with '#' Recursively read all files with the extension '.lst'

```
fermipy.diffuse.gt_split_and_bin.create_sg_split_and_bin (**kwargs)
```

Build and return a *fermipy.jobs.ScatterGather* object that can invoke this script

```
fermipy.diffuse.gt_split_and_bin.main_batch()
```

Entry point for command line use for dispatching batch jobs

```
fermipy.diffuse.gt_split_and_bin.main_single()
```

Entry point for command line use for single job

```
fermipy.diffuse.gt_split_and_bin.readlines (arg)
```

Read lines from a file into a list.

Removes whitespace and lines that start with '#'

fermipy.diffuse.gt_srcmap_partial module

Run gtsrcmaps for a single energy plane for a single source

This is useful to parallize the production of the source maps

```
class fermipy.diffuse.gt_srcmap_partial.ConfigMaker_SrcmapPartial (link,  
                                                                **kwargs)
```

Bases: *fermipy.jobs.scatter_gather.ConfigMaker*

Small class to generate configurations for this script

This adds the following arguments: `-comp` : binning component definition yaml file `-data` : dataset definition yaml file `-irf_ver` : IRF versions string (e.g., 'V6') `-diffuse` : Diffuse model component definition yaml file `-make_xml` : Write xml files for the individual components

build_job_configs (*args*)

Hook to build job configurations

default_options = {'irf_ver': ('V6', 'Version of IRFs to use.', <type 'str'>), 'make_xml': (False, 'Write xml files need

```
class fermipy.diffuse.gt_srcmap_partial.GtSrcmapPartial (**kwargs)
```

Bases: *object*

Small class to create srcmaps for only once source in a model, and optionally for only some of the energy layers.

This is useful for parallelizing source map creation.

NULL_MODEL = 'srcmdls/null.xml'

default_options = {'irfs': ('CALDB', 'Instrument response functions', <type 'str'>), 'expcube': (None, 'Input Livetime

run (*argv*)

Run this analysis

```
fermipy.diffuse.gt_srcmap_partial.create_link_srcmap_partial (**kwargs)
```

Build and return a *Link* object that can invoke *GtAssembleModel*

```
fermipy.diffuse.gt_srcmap_partial.create_sg_srcmap_partial (**kwargs)
```

Build and return a *ScatterGather* object that can invoke this script

```
fermipy.diffuse.gt_srcmap_partial.main_batch ()
```

Entry point for command line use for dispatching batch jobs

```
fermipy.diffuse.gt_srcmap_partial.main_single ()
```

Entry point for command line use for single job

fermipy.diffuse.residual_cr module

Compute the residual cosmic-ray contamination

```
class fermipy.diffuse.residual_cr.ConfigMaker_ResidualCR (link, **kwargs)
```

Bases: *fermipy.jobs.scatter_gather.ConfigMaker*

Small class to generate configurations for this script

build_job_configs (*args*)

Hook to build job configurations

default_options = {'data_dirty': (None, 'Path to yaml file defining dataset for "clean" selection', <type 'str'>), 'full_c

```
class fermipy.diffuse.residual_cr.ResidualCRAnalysis (**kwargs)
```

Bases: *object*

```
default_options = {'bexpcube_clean': (None, 'Input exposure cube for clean event class.', <type 'str'>), 'ccube_clean': (None, 'Input cleaned cube for clean event class.', <type 'str'>)}  
run(argv)  
    Run this analysis
```

Bases: *fermipy.jobs.chain.Chain*

run_argparser (<i>argv</i>)	Initialize a link with a set of arguments using argparse
--------------------------------------	--

Build and return a *ResidualCRChain* object

Build and return a `Link` object that can invoke *ResidualCRAnalysis*

Build and return a ScatterGather object that can invoke this script

```
fermipy.diffuse.residual_cr.main_chain()
```

Energy point for running the entire Cosmic-ray analysis

Entry point for command line use for single job

Changelog

0.13.0 (01/16/2017)

- ## 1.2. Documentation Contents

0.12.0 (11/20/2016)

- Add support for phased analysis (#87). `gtlike.expscale` and `gtlike.src_expscale` can be used to apply a constant exposure correction to a whole component or individual sources within a component. See *Phased Analysis* for examples.
- Add script and tools for calculating flux sensitivity (#88 and #95). The `fermipy-flux-sensitivity` script evaluates both the differential and integral flux sensitivity for a given TS threshold and minimum number of detected counts. See *Sensitivity Tools* for examples.
- Add `fermipy-healview` script for generating images of healpix maps and cubes.
- Improvements to HPX-related classes and utilities.
- Refactoring in `irfs` module to support development of new validation tools.
- Improvements to configuration handling to allow parameter validation when updating configuration at runtime.
- Add lightcurve method (#80). See *Light Curves* for documentation.
- Change convention for flux arrays in source object. Values and uncertainties are now stored in separate arrays (e.g. `flux` and `flux_err`).
- Add *Docker-based installation* instructions. This can be used to run the RHEL6 SLAC ST builds on any machine that supports Docker (e.g. OSX Yosemite or later).
- Adopt changes to column name conventions in SED format. All column names are now lowercase.

0.11.0 (08/24/2016)

- Add support for weighted likelihood fits (supported in ST 11-03-00 or later). Weights maps can be specified with the `wmap` parameter in `gtlike`.
- Implemented performance improvements in `tmap` including switching to newton's method for step-size calculation and masking of empty pixels (see #79).
- Ongoing development and refactoring of classes for dealing with `CastroData` (binned likelihood profiles).
- Added `reload_sources` method for faster recomputation of source maps.
- Fixed sign error in localization plotting method that gave wrong orientation for error ellipse..
- Refactored classes in `spectrum` and simplified interface for doing spectral fits (see #69).
- Added `DMFitFunction` spectral model class in `spectrum` (see #66). This uses the same lookup tables as the ST `DMFitFunction` class but provides a pure python implementation which can be used independently of the STs.

0.10.0 (07/03/2016)

- Implement support for more spectral models (`DMFitFunction`, `EblAtten`, `FileFunction`, `Gaussian`).
- New options (`outdir_regex` and `workdir_regex`) for fine-grained control over input/output file staging.
- Add `offset_roi_edge` to source dictionary. Defined as the distance from the source position to the edge of the ROI (< 0 = inside the ROI, > 0 = outside the ROI).
- Add new variables in `fit` output (`edm`, `fit_status`).
- Add new package scripts (`fermipy-collect-sources`, `fermipy-cluster-sources`).
- Various refactoring and improvements in code for dealing with castro data.

- Add `MODEL_FLUX` and `PARAMS` HDUs to SED FITS file. Many new elements added SED output dictionary.
- Support NEWTON fitter with the same interface as MINUIT and NEWMINUIT. Running `fit` with `optimizer = NEWTON` will use the NEWTON fitter where applicable (only free norms) and MINUIT otherwise. The `optimizer` argument to `sed`, `extension`, and `localize` can be used to override the default optimizer at runtime. Note that the NEWTON fitter is only supported by ST releases *after* 11-01-01.

0.9.0 (05/25/2016)

- Bug fixes and various refactoring in TSCube and CastroData. Classes for reading and manipulating bin-by-bin likelihoods are now moved to the `castro` module.
- Rationalized naming conventions for energy-related variables. Properties and method arguments with units of the logarithm of the energy now consistently contain `log` in the name.
 - `energies` now returns bin energies in MeV (previously it returned logarithmic energies). `log_energies` can be used to access logarithmic bin energies.
 - Changed `erange` parameter to `log_bounds` in the methods that accept an energy range.
 - Changed the units of `emin`, `ectr`, and `emax` in the sed output dictionary to MeV.
- Add more columns to the FITS source catalog file generated by `write_roi`. All float and string values in the source dictionary are now automatically included in the FITS file. Parameter values, errors, and names are written to the `param_values`, `param_errors`, and `param_names` vector columns.
- Add package script for dispatching batch jobs to LSF (`fermipy-dispatch`).
- Fixed some bugs related to handling of unicode strings.

0.8.0 (05/18/2016)

- Added new variables to source dictionary:
 - Likelihood scan of source normalization (`dloglike_scan`, `eflux_scan`, `flux_scan`).
 - Source localization errors (`pos_sigma`, `pos_sigma_semimajor`, `pos_sigma_semiminor`, `pos_r68`, `pos_r95`, `pos_r99`, `pos_angle`). These are automatically filled when running `localize` or `find_sources`.
- Removed camel-case in some source variable names.
- Add `cacheft1` option to `data` disable caching FT1 files. Cacheing is still enabled by default.
- Support FITS file format for preliminary releases of the 4FGL catalog.
- Add `__future__` statements throughout to ensure forward-compatibility with python3.
- Reorganize utility modules including those for manipulation of WCS and healpix images.
- Various improvements and refactoring in `localize`. This method now moved to the `sourcefind` module.
- Add new global parameter `llscan_pts` in `gtlike` to define the number of likelihood evaluation points.
- Write output of `sed` to a FITS file in the Likelihood SED format. More information about the Likelihood SED format is available on this [page](#).
- Write ROI model to a FITS file when calling `write_roi`. This file contains a BINTABLE with one row per source and uses the same column names as the 3FGL catalog file to describe spectral parameterizations. Note that this file currently only contains a subset of the information available in the numpy output file.
- Reorganize classes and methods in `sed` for manipulating and fitting bin-by-bin likelihoods. Spectral functions moved to a dedicated `spectrum` module.

- Write return dictionary to a numpy file in *residmap* and *tmap*.

Indices and tables

f

`fermipy`, 105
`fermipy.castro`, 93
`fermipy.config`, 53
`fermipy.defaults`, 54
`fermipy.diffuse`, 145
`fermipy.diffuse.binning`, 124
`fermipy.diffuse.catalog_src_manager`, 129
`fermipy.diffuse.defaults`, 125
`fermipy.diffuse.diffuse_src_manager`, 131
`fermipy.diffuse.gt_assemble_model`, 140
`fermipy.diffuse.gt_coadd_split`, 142
`fermipy.diffuse.gt_merge_srcmaps`, 142
`fermipy.diffuse.gt_split_and_bin`, 143
`fermipy.diffuse.gt_srcmap_partial`, 144
`fermipy.diffuse.job_library`, 138
`fermipy.diffuse.model_component`, 134
`fermipy.diffuse.model_manager`, 136
`fermipy.diffuse.name_policy`, 125
`fermipy.diffuse.residual_cr`, 144
`fermipy.diffuse.source_factory`, 128
`fermipy.diffuse.spectral`, 128
`fermipy.jobs`, 124
`fermipy.jobs.chain`, 106
`fermipy.jobs.file_archive`, 111
`fermipy.jobs.gtlink`, 119
`fermipy.jobs.job_archive`, 116
`fermipy.jobs.lsf_impl`, 120
`fermipy.jobs.scatter_gather`, 121
`fermipy.lightcurve`, 105
`fermipy.logger`, 71
`fermipy.plotting`, 84
`fermipy.residmap`, 104
`fermipy.roi_model`, 71
`fermipy.sed`, 86
`fermipy.skymap`, 90
`fermipy.sourcefind`, 87
`fermipy.spectrum`, 87
`fermipy.tsmmap`, 102
`fermipy.utils`, 77

Symbols

- `__call__()` (fermipy.castro.CastroData_Base method), 96
 - `__call__()` (fermipy.castro.Interpolator method), 98
 - `__call__()` (fermipy.jobs.scatter_gather.ScatterGather method), 122
 - `__delattr__` (fermipy.gtanalysis.GTAnalysis attribute), 54
 - `__format__()` (fermipy.gtanalysis.GTAnalysis method), 54
 - `__getattr__` (fermipy.gtanalysis.GTAnalysis attribute), 54
 - `__getitem__()` (fermipy.castro.CastroData_Base method), 96
 - `__getitem__()` (fermipy.diffuse.model_manager.ModelInfo method), 137
 - `__getitem__()` (fermipy.diffuse.spectral.SpectralLibrary method), 128
 - `__getitem__()` (fermipy.jobs.chain.Chain method), 106
 - `__getitem__()` (fermipy.jobs.file_archive.FileArchive method), 111
 - `__getitem__()` (fermipy.jobs.job_archive.JobArchive method), 116
 - `__hash__` (fermipy.gtanalysis.GTAnalysis attribute), 54
 - `__reduce__()` (fermipy.gtanalysis.GTAnalysis method), 54
 - `__reduce_ex__()` (fermipy.gtanalysis.GTAnalysis method), 54
 - `__repr__` (fermipy.gtanalysis.GTAnalysis attribute), 54
 - `__setattr__` (fermipy.gtanalysis.GTAnalysis attribute), 55
 - `__sizeof__()` (fermipy.gtanalysis.GTAnalysis method), 55
 - `__str__` (fermipy.gtanalysis.GTAnalysis attribute), 55
- ## A
- `add_argument()` (in module fermipy.jobs.chain), 111
 - `add_arguments()` (fermipy.jobs.scatter_gather.ConfigMaker method), 121
 - `add_component_info()` (fermipy.diffuse.model_component.ModelComponentInfo method), 136
 - `add_gauss_prior()` (fermipy.gtanalysis.GTAnalysis method), 55
 - `add_link()` (fermipy.jobs.chain.Chain method), 106
 - `add_name()` (fermipy.roi_model.Model method), 72
 - `add_option()` (fermipy.config.ConfigSchema method), 53
 - `add_options()` (fermipy.jobs.scatter_gather.ConfigMaker method), 121
 - `add_section()` (fermipy.config.ConfigSchema method), 53
 - `add_source()` (fermipy.gtanalysis.GTAnalysis method), 55
 - `add_sources()` (fermipy.diffuse.source_factory.SourceFactory method), 128
 - `add_sources_from_roi()` (fermipy.gtanalysis.GTAnalysis method), 55
 - `add_to_table()` (fermipy.roi_model.Model method), 72
 - AnalysisPlotter (class in fermipy.plotting), 84
 - `angle_to_cartesian()` (in module fermipy.utils), 77
 - `angprofile()` (fermipy.diffuse.name_policy.NameFactory method), 125
 - `angprofile_format` (fermipy.diffuse.name_policy.NameFactory attribute), 125
 - `annotate()` (fermipy.plotting.SEDPlotter static method), 85
 - `annotate()` (in module fermipy.plotting), 86
 - `append_hdus()` (fermipy.diffuse.gt_assemble_model.GtAssembleModel static method), 140
 - `append_to_table()` (fermipy.jobs.file_archive.FileHandle method), 115
 - `append_to_tables()` (fermipy.jobs.job_archive.JobDetails method), 118
 - `apply_minmax_selection()` (in module fermipy.utils), 77
 - `arg_names` (fermipy.jobs.chain.Link attribute), 108
 - `arg_to_list()` (in module fermipy.utils), 77
 - `argmapper` (fermipy.jobs.chain.Chain attribute), 107
 - `assemble_component()` (fermipy.diffuse.gt_assemble_model.GtAssembleModel static method), 141
 - `assoc` (fermipy.roi_model.Model attribute), 72
 - `associations` (fermipy.roi_model.Source attribute), 76
- ## B
- `base_config` (fermipy.jobs.scatter_gather.ScatterGather attribute), 122

base_path (fermipy.jobs.file_archive.FileArchive attribute), 112

bexpcube() (fermipy.diffuse.name_policy.NameFactory method), 125

bexpcube_format (fermipy.diffuse.name_policy.NameFactory attribute), 125

bexpcube_moon() (fermipy.diffuse.name_policy.NameFactory method), 125

bexpcube_sun() (fermipy.diffuse.name_policy.NameFactory method), 125

bexpcubemoon_format (fermipy.diffuse.name_policy.NameFactory attribute), 125

bexpcubesun_format (fermipy.diffuse.name_policy.NameFactory attribute), 125

bin_widths (fermipy.castro.ReferenceSpec attribute), 100

bowtie() (fermipy.gtanalysis.GTAnalysis method), 55

build_archive() (fermipy.jobs.file_archive.FileArchive static method), 112

build_archive() (fermipy.jobs.job_archive.JobArchive static method), 117

build_bsub_command() (in module fermipy.jobs.lsf_impl), 121

build_catalog() (fermipy.diffuse.source_factory.SourceFactory static method), 128

build_catalog_info() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130

build_configs() (fermipy.jobs.scatter_gather.ScatterGather method), 122

build_ebound_table() (fermipy.castro.ReferenceSpec method), 100

build_from_energy_dict() (fermipy.diffuse.binning.Component static method), 124

build_from_yamlfile() (fermipy.diffuse.binning.Component static method), 124

build_from_yamlstr() (fermipy.diffuse.binning.Component static method), 124

build_gtapp() (in module fermipy.jobs.gtlink), 119

build_job_configs() (fermipy.diffuse.gt_assemble_model.ConfigMaker_AssembleModel method), 140

build_job_configs() (fermipy.diffuse.gt_merge_srcmaps.ConfigMaker_MergeSrcmaps method), 142

build_job_configs() (fermipy.diffuse.gt_split_and_bin.ConfigMaker_SplitAndBin method), 143

build_job_configs() (fermipy.diffuse.gt_srcmap_partial.ConfigMaker_SrcmapPartial method), 144

build_job_configs() (fermipy.diffuse.job_library.ConfigMaker_Gtexpcube2 method), 138

build_job_configs() (fermipy.diffuse.job_library.ConfigMaker_healview method), 139

build_job_configs() (fermipy.diffuse.job_library.ConfigMaker_SrcmapsCatalog method), 139

build_job_configs() (fermipy.diffuse.job_library.ConfigMaker_SumRings method), 139

build_job_configs() (fermipy.diffuse.job_library.ConfigMaker_Vstack method), 139

build_job_configs() (fermipy.diffuse.residual_cr.ConfigMaker_ResidualCR method), 144

build_job_configs() (fermipy.jobs.scatter_gather.ConfigMaker method), 121

build_job_dict() (fermipy.jobs.scatter_gather.ScatterGather method), 122

build_scandata_table() (fermipy.castro.CastroData_Base method), 96

build_sg_from_link() (in module fermipy.jobs.lsf_impl), 121

build_source_dict() (in module fermipy.castro), 102

build_spec_table() (fermipy.castro.SpecData method), 101

build_temp_job_archive() (fermipy.jobs.job_archive.JobArchive static method), 117

C

cache (fermipy.jobs.file_archive.FileArchive attribute), 112

cache (fermipy.jobs.job_archive.JobArchive attribute), 117

cash() (in module fermipy.tsmapi), 103

cast_args() (in module fermipy.spectrum), 90

cast_config() (in module fermipy.config), 53

cast_params() (in module fermipy.spectrum), 90

CastroData (class in fermipy.castro), 93

CastroData_Base (class in fermipy.castro), 96

castroData_from_ipix() (fermipy.castro.TSCube method), 101

castroData_from_pix_xy() (fermipy.castro.TSCube method), 101

castro_comp_info_dict() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130

[catalog_components\(\)](#) (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130
[catalog_split_yaml\(\)](#) (fermipy.diffuse.name_policy.NameFactory method), 125
[catalog_split_yaml_format](#) (fermipy.diffuse.name_policy.NameFactory attribute), 125
[CatalogComponentChain](#) (class in fermipy.diffuse.catalog_src_manager), 129
[CatalogInfo](#) (class in fermipy.diffuse.model_component), 134
[catalogs\(\)](#) (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130
[CatalogSourceManager](#) (class in fermipy.diffuse.catalog_src_manager), 130
[CatalogSourcesInfo](#) (class in fermipy.diffuse.model_component), 134
[ccube\(\)](#) (fermipy.diffuse.name_policy.NameFactory method), 125
[ccube_format](#) (fermipy.diffuse.name_policy.NameFactory attribute), 125
[center_to_edge\(\)](#) (in module fermipy.utils), 77
[Chain](#) (class in fermipy.jobs.chain), 106
[chain_input_files](#) (fermipy.jobs.file_archive.FileDict attribute), 113
[chain_output_files](#) (fermipy.jobs.file_archive.FileDict attribute), 113
[chan](#) (fermipy.spectrum.DMFitFunction attribute), 87
[chan_code](#) (fermipy.spectrum.DMFitFunction attribute), 88
[channel_index_mapping](#) (fermipy.spectrum.DMFitFunction attribute), 88
[channel_name_mapping](#) (fermipy.spectrum.DMFitFunction attribute), 88
[channel_rev_map](#) (fermipy.spectrum.DMFitFunction attribute), 88
[channels\(\)](#) (fermipy.spectrum.DMFitFunction static method), 88
[check_cuts\(\)](#) (fermipy.roi_model.Model method), 72
[check_files\(\)](#) (in module fermipy.jobs.chain), 111
[check_input_files\(\)](#) (fermipy.jobs.chain.Link method), 108
[check_job\(\)](#) (fermipy.jobs.lsf_impl.LsfScatterGather method), 120
[check_job\(\)](#) (fermipy.jobs.scatter_gather.ScatterGather method), 122
[check_log\(\)](#) (in module fermipy.jobs.lsf_impl), 121
[check_output_files\(\)](#) (fermipy.jobs.chain.Link method), 108
[check_status\(\)](#) (fermipy.jobs.file_archive.FileHandle method), 115
[check_status\(\)](#) (fermipy.jobs.scatter_gather.ScatterGather method), 122
[check_status_logfile\(\)](#) (fermipy.jobs.job_archive.JobDetails method), 118
[chi2_vals\(\)](#) (fermipy.castro.CastroData_Base method), 96
[clean_job\(\)](#) (in module fermipy.jobs.scatter_gather), 124
[cleanup\(\)](#) (fermipy.gtanalysis.GTAnalysis method), 55
[clear\(\)](#) (fermipy.roi_model.ROIModel method), 73
[clone\(\)](#) (fermipy.gtanalysis.GTAnalysis method), 55
[clone_and_merge_sub\(\)](#) (fermipy.diffuse.model_component.ModelComponentInfo method), 136
[close\(\)](#) (fermipy.logger.StreamLogger method), 71
[cmap](#) (fermipy.plotting.ROIPlotter attribute), 85
[CoaddSplit](#) (class in fermipy.diffuse.gt_coadd_split), 142
[collect_dirs\(\)](#) (in module fermipy.utils), 77
[command_template\(\)](#) (fermipy.jobs.chain.Link method), 108
[command_template\(\)](#) (fermipy.jobs.gtlink.Gtlink method), 119
[comp_srcmdl_xml\(\)](#) (fermipy.diffuse.name_policy.NameFactory method), 125
[comp_srcmdl_xml_format](#) (fermipy.diffuse.name_policy.NameFactory attribute), 125
[Component](#) (class in fermipy.diffuse.binning), 124
[component\(\)](#) (fermipy.diffuse.name_policy.NameFactory method), 125
[component_format](#) (fermipy.diffuse.name_policy.NameFactory attribute), 125
[component_names](#) (fermipy.diffuse.model_manager.ModelInfo attribute), 137
[components](#) (fermipy.gtanalysis.GTAnalysis attribute), 55
[CompositeSource](#) (class in fermipy.roi_model), 71
[CompositeSourceInfo](#) (class in fermipy.diffuse.model_component), 135
[config](#) (fermipy.config.Configurable attribute), 53
[config](#) (fermipy.gtanalysis.GTAnalysis attribute), 55
[config_maker](#) (fermipy.jobs.scatter_gather.ScatterGather attribute), 122
[configdir](#) (fermipy.config.Configurable attribute), 53
[configdir](#) (fermipy.gtanalysis.GTAnalysis attribute), 55
[ConfigMaker](#) (class in fermipy.jobs.scatter_gather), 121
[ConfigMaker_AssembleModel](#) (class in fermipy.diffuse.gt_assemble_model), 140
[ConfigMaker_Gtexpcube2](#) (class in fermipy.diffuse.job_library), 138
[ConfigMaker_healview](#) (class in fermipy.diffuse.job_library), 139

ConfigMaker_MergeSrcmaps (class in mipy.diffuse.gt_merge_srcmaps), 142
 ConfigMaker_ResidualCR (class in mipy.diffuse.residual_cr), 144
 ConfigMaker_SplitAndBin (class in mipy.diffuse.gt_split_and_bin), 143
 ConfigMaker_SrcmapPartial (class in mipy.diffuse.gt_srcmap_partial), 144
 ConfigMaker_SrcmapsCatalog (class in mipy.diffuse.job_library), 138
 ConfigMaker_SumRings (class in mipy.diffuse.job_library), 139
 ConfigMaker_Vstack (class in mipy.diffuse.job_library), 139
 ConfigManager (class in fermipy.config), 53
 ConfigSchema (class in fermipy.config), 53
 Configurable (class in fermipy.config), 53
 configure() (fermipy.config.Configurable method), 53
 configure() (fermipy.gtanalysis.GTAnalysis method), 56
 constrain_norms() (fermipy.gtanalysis.GTAnalysis method), 56
 construct_scratch_path() (fermipy.jobs.file_archive.FileStageManager method), 115
 convert_dict_to_option_dict() (in module fermipy.jobs.chain), 111
 convert_option_dict_to_dict() (in module fermipy.jobs.chain), 111
 convert_sed_cols() (in module fermipy.castro), 102
 convert_to_cached_wcs() (fermipy.skymap.HpxMap method), 90
 convert_tscube() (in module fermipy.tsmmap), 103
 convert_tscube_old() (in module fermipy.tsmmap), 104
 convert_value_to_option_tuple() (in module fermipy.jobs.chain), 111
 convolve2d_disk() (in module fermipy.utils), 77
 convolve2d_gauss() (in module fermipy.utils), 78
 convolve_map() (in module fermipy.residmap), 104
 copy_ccube() (fermipy.diffuse.gt_assemble_model.GtAssembleModel static method), 141
 copy_from_scratch() (fermipy.jobs.file_archive.FileStageManager static method), 115
 copy_selected_sources() (fermipy.diffuse.source_factory.SourceFactory static method), 128
 copy_source() (fermipy.roi_model.ROIModel method), 73
 copy_to_scratch() (fermipy.jobs.file_archive.FileStageManager static method), 115
 counts (fermipy.skymap.Map_Base attribute), 93
 counts_map() (fermipy.gtanalysis.GTAnalysis method), 56
 cov_to_correlation() (in module fermipy.utils), 78
 create() (fermipy.config.ConfigManager static method), 53
 create() (fermipy.gtanalysis.GTAnalysis static method), 56
 create() (fermipy.roi_model.ROIModel static method), 73
 create() (fermipy.skymap.Map static method), 92
 create_chain_catalog_comps() (in module fermipy.diffuse.catalog_src_manager), 131
 create_chain_coadd_split() (in module fermipy.diffuse.gt_coadd_split), 142
 create_chain_diffuse_comps() (in module fermipy.diffuse.diffuse_src_manager), 134
 create_chain_residual_cr() (in module fermipy.diffuse.residual_cr), 145
 create_chain_split_and_bin() (in module fermipy.diffuse.gt_split_and_bin), 143
 create_config() (fermipy.config.ConfigSchema method), 53
 create_default_config() (in module fermipy.config), 53
 create_dict() (in module fermipy.utils), 78
 create_efflux_func() (fermipy.spectrum.SpectralFunction class method), 89
 create_flux_func() (fermipy.spectrum.SpectralFunction class method), 89
 create_from_dict() (fermipy.roi_model.Model static method), 72
 create_from_dict() (fermipy.roi_model.Source static method), 76
 create_from_efflux() (fermipy.spectrum.SpectralFunction class method), 89
 create_from_fits() (fermipy.castro.CastroData static method), 94
 create_from_fits() (fermipy.castro.TSCube static method), 101
 create_from_fits() (fermipy.plotting.ROIPlotter static method), 85
 create_from_fits() (fermipy.skymap.HpxMap static method), 90
 create_from_fits() (fermipy.skymap.Map static method), 92
 create_from_flux() (fermipy.spectrum.SpectralFunction class method), 89
 create_from_flux_points() (fermipy.castro.CastroData static method), 94
 create_from_hdu() (fermipy.skymap.HpxMap static method), 90
 create_from_hdu() (fermipy.skymap.Map static method), 92
 create_from_hdulist() (fermipy.skymap.HpxMap static method), 91
 create_from_position() (fermipy.roi_model.ROIModel static method), 73

[create_from_roi_data\(\)](#) (fermipy.roi_model.ROIModel static method), [73](#)
[create_from_row\(\)](#) (fermipy.jobs.file_archive.FileHandle static method), [115](#)
[create_from_row\(\)](#) (fermipy.jobs.job_archive.JobDetails static method), [118](#)
[create_from_sedfile\(\)](#) (fermipy.castro.CastroData static method), [94](#)
[create_from_source\(\)](#) (fermipy.roi_model.ROIModel static method), [73](#)
[create_from_stack\(\)](#) (fermipy.castro.CastroData static method), [94](#)
[create_from_table\(\)](#) (fermipy.castro.ReferenceSpec static method), [100](#)
[create_from_table\(\)](#) (fermipy.castro.SpecData static method), [101](#)
[create_from_tables\(\)](#) (fermipy.castro.CastroData static method), [95](#)
[create_from_xml\(\)](#) (fermipy.roi_model.Source static method), [76](#)
[create_from_xmlfile\(\)](#) (fermipy.roi_model.Source static method), [76](#)
[create_from_yaml\(\)](#) (fermipy.diffuse.spectral.SpectralLibrary static method), [128](#)
[create_from_yamlstr\(\)](#) (fermipy.diffuse.spectral.SpectralLibrary static method), [128](#)
[create_functor\(\)](#) (fermipy.castro.CastroData method), [95](#)
[create_functor\(\)](#) (fermipy.spectrum.SpectralFunction class method), [89](#)
[create_hpx_disk_region_string\(\)](#) (in module fermipy.utils), [78](#)
[create_image_hdu\(\)](#) (fermipy.skymap.HpxMap method), [91](#)
[create_image_hdu\(\)](#) (fermipy.skymap.Map method), [92](#)
[create_inputlist\(\)](#) (in module fermipy.diffuse.gt_split_and_bin), [143](#)
[create_job_details\(\)](#) (fermipy.jobs.chain.Link method), [108](#)
[create_link_assemble_model\(\)](#) (in module fermipy.diffuse.gt_assemble_model), [141](#)
[create_link_fermipy_coadd\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_link_fermipy_healview\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_link_fermipy_vstack\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_link_gtexpcube2\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_link_gtsrcmaps\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_link_merge_srcmaps\(\)](#) (in module fermipy.diffuse.gt_merge_srcmaps), [142](#)
[create_link_residual_cr\(\)](#) (in module fermipy.diffuse.residual_cr), [145](#)
[create_link_srcmap_partial\(\)](#) (in module fermipy.diffuse.gt_srcmap_partial), [144](#)
[create_model_name\(\)](#) (in module fermipy.utils), [78](#)
[create_param_table\(\)](#) (fermipy.roi_model.ROIModel method), [73](#)
[create_primary_hdu\(\)](#) (fermipy.skymap.Map method), [92](#)
[create_roi_from_ft1\(\)](#) (fermipy.roi_model.ROIModel static method), [73](#)
[create_roi_table\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [56](#)
[create_sg_assemble_model\(\)](#) (in module fermipy.diffuse.gt_assemble_model), [141](#)
[create_sg_gtexpcube2\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_sg_gtsrcmaps_catalog\(\)](#) (in module fermipy.diffuse.job_library), [139](#)
[create_sg_healview_diffuse\(\)](#) (in module fermipy.diffuse.job_library), [140](#)
[create_sg_merge_srcmaps\(\)](#) (in module fermipy.diffuse.gt_merge_srcmaps), [143](#)
[create_sg_residual_cr\(\)](#) (in module fermipy.diffuse.residual_cr), [145](#)
[create_sg_split_and_bin\(\)](#) (in module fermipy.diffuse.gt_split_and_bin), [143](#)
[create_sg_srcmap_partial\(\)](#) (in module fermipy.diffuse.gt_srcmap_partial), [144](#)
[create_sg_sum_ring_gasmapi\(\)](#) (in module fermipy.diffuse.job_library), [140](#)
[create_sg_vstack_diffuse\(\)](#) (in module fermipy.diffuse.job_library), [140](#)
[create_source\(\)](#) (fermipy.roi_model.ROIModel method), [73](#)
[create_source_name\(\)](#) (in module fermipy.utils), [78](#)
[create_source_table\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[create_source_table\(\)](#) (in module fermipy.roi_model), [77](#)
[create_table\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[create_xml_element\(\)](#) (in module fermipy.utils), [78](#)
[csm](#) (fermipy.diffuse.model_manager.ModelManager attribute), [137](#)

D

[data](#) (fermipy.plotting.ROIPlotter attribute), [85](#)
[data](#) (fermipy.roi_model.Model attribute), [72](#)
[data](#) (fermipy.roi_model.Source attribute), [76](#)
[data](#) (fermipy.skymap.Map_Base attribute), [93](#)
[dataset\(\)](#) (fermipy.diffuse.name_policy.NameFactory method), [125](#)
[dataset_format](#) (fermipy.diffuse.name_policy.NameFactory attribute), [126](#)

default_gather_logfile (fermipy.jobs.scatter_gather.ScatterGather attribute), 122

default_init_logfile (fermipy.jobs.scatter_gather.ScatterGather attribute), 122

default_options (fermipy.diffuse.catalog_src_manager.CatalogSrcManager attribute), 129

default_options (fermipy.diffuse.diffuse_src_manager.DiffuseSrcManager attribute), 131

default_options (fermipy.diffuse.gt_assemble_model.ConfigMaker_GtAssembleModel attribute), 140

default_options (fermipy.diffuse.gt_assemble_model.GtAssembleModel attribute), 141

default_options (fermipy.diffuse.gt_assemble_model.GtInitModel attribute), 141

default_options (fermipy.diffuse.gt_merge_srcmaps.ConfigMaker_MergeSrcMaps attribute), 142

default_options (fermipy.diffuse.gt_merge_srcmaps.GtMergeSrcMaps attribute), 142

default_options (fermipy.diffuse.gt_split_and_bin.ConfigMaker_SplitAndBin attribute), 143

default_options (fermipy.diffuse.gt_srcmap_partial.ConfigMaker_GtSrcmapPartial attribute), 144

default_options (fermipy.diffuse.gt_srcmap_partial.GtSrcmapPartial attribute), 144

default_options (fermipy.diffuse.job_library.ConfigMaker_Gtexpcube2 attribute), 138

default_options (fermipy.diffuse.job_library.ConfigMaker_healview attribute), 139

default_options (fermipy.diffuse.job_library.ConfigMaker_SrcmapsCatalog attribute), 139

default_options (fermipy.diffuse.job_library.ConfigMaker_SumRings attribute), 139

default_options (fermipy.diffuse.job_library.ConfigMaker_Vispect attribute), 139

default_options (fermipy.diffuse.residual_cr.ConfigMaker_ResidualCR attribute), 144

default_options (fermipy.diffuse.residual_cr.ResidualCRAnalysis attribute), 145

default_options (fermipy.jobs.lsf_impl.LsfScatterGather attribute), 120

default_options (fermipy.jobs.scatter_gather.ScatterGather attribute), 122

default_prefix_logfile (fermipy.jobs.scatter_gather.ScatterGather attribute), 122

defaults (fermipy.gtanalysis.GTAnalysis attribute), 56

defaults (fermipy.plotting.AnalysisPlotter attribute), 84

defaults (fermipy.plotting.ROIPlotter attribute), 85

defaults (fermipy.roi_model.ROIModel attribute), 74

delete_source() (fermipy.gtanalysis.GTAnalysis method), 56

delete_sources() (fermipy.gtanalysis.GTAnalysis method), 56

delete_sources() (fermipy.roi_model.ROIModel method), 74

derivative() (fermipy.castro.CastroData_Base method), 96

derivative() (fermipy.castro.Interpolator method), 99

diffuse_comp_info() (fermipy.roi_model.CompositeSource attribute), 71

diffuse_comp_info_dicts() (fermipy.roi_model.IsoSource attribute), 71

diffuse_sources (fermipy.roi_model.MapCubeSource attribute), 71

diffuse_template() (fermipy.roi_model.Source attribute), 76

diffuse_template() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager attribute), 131

diffuse_template() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager attribute), 133

diffuse_template() (fermipy.name_policy.NameFactory method), 126

diffuse_template() (fermipy.name_policy.NameFactory attribute), 126

DiffuseComponentChain (class in fermipy.diffuse.diffuse_src_manager), 131

DiffuseModelManager (class in fermipy.diffuse.diffuse_src_manager), 131

dispatch_job() (fermipy.jobs.scatter_gather.ScatterGather method), 122

dispatch_job_hook() (fermipy.jobs.lsf_impl.LsfScatterGather method), 120

dispatch_job_hook() (fermipy.jobs.scatter_gather.ScatterGather method), 122

DMFitFunction (class in fermipy.spectrum), 87

dnnde (fermipy.castro.SpecData attribute), 101

dnnde() (fermipy.spectrum.SpectralFunction method), 89

dnnde_deriv() (fermipy.spectrum.SpectralFunction method), 89

dnnde_err (fermipy.castro.SpecData attribute), 101

done (fermipy.jobs.job_archive.JobStatus attribute), 118

dot_prod() (in module fermipy.utils), 78

draw_circle() (fermipy.plotting.ROIPlotter method), 85

E

e2dnnde (fermipy.castro.SpecData attribute), 101

e2dnnde() (fermipy.spectrum.SpectralFunction method), 89

- e2dnde_deriv() (fermipy.spectrum.SpectralFunction method), 89
 e2dnde_err (fermipy.castro.SpecData attribute), 101
 ebins (fermipy.castro.ReferenceSpec attribute), 100
 edge_to_center() (in module fermipy.utils), 78
 edge_to_width() (in module fermipy.utils), 78
 ednde() (fermipy.spectrum.SpectralFunction method), 89
 ednde_deriv() (fermipy.spectrum.SpectralFunction method), 89
 eflux (fermipy.castro.SpecData attribute), 101
 eflux() (fermipy.spectrum.SpectralFunction method), 90
 emax (fermipy.castro.ReferenceSpec attribute), 100
 emax (fermipy.diffuse.binning.Component attribute), 124
 emax (fermipy.spectrum.SEDFunctor attribute), 89
 emin (fermipy.castro.ReferenceSpec attribute), 100
 emin (fermipy.diffuse.binning.Component attribute), 124
 emin (fermipy.spectrum.SEDFunctor attribute), 89
 energies (fermipy.gtanalysis.GTAnalysis attribute), 57
 enumbins (fermipy.gtanalysis.GTAnalysis attribute), 57
 eq2gal() (in module fermipy.utils), 78
 eref (fermipy.castro.ReferenceSpec attribute), 100
 eval_dnde() (fermipy.spectrum.SpectralFunction class method), 90
 eval_dnde_deriv() (fermipy.spectrum.SpectralFunction class method), 90
 eval_e2dnde() (fermipy.spectrum.SpectralFunction class method), 90
 eval_e2dnde_deriv() (fermipy.spectrum.SpectralFunction class method), 90
 eval_ednde() (fermipy.spectrum.SpectralFunction class method), 90
 eval_ednde_deriv() (fermipy.spectrum.SpectralFunction class method), 90
 eval_eflux() (fermipy.spectrum.PowerLaw class method), 88
 eval_eflux() (fermipy.spectrum.SpectralFunction class method), 90
 eval_flux() (fermipy.spectrum.PowerLaw class method), 88
 eval_flux() (fermipy.spectrum.SpectralFunction class method), 90
 eval_norm() (fermipy.spectrum.PowerLaw static method), 89
 evttype (fermipy.diffuse.binning.Component attribute), 125
 exists (fermipy.jobs.file_archive.FileStatus attribute), 116
 expected (fermipy.jobs.file_archive.FileStatus attribute), 116
 extend_array() (in module fermipy.utils), 78
 extended (fermipy.roi_model.Source attribute), 76
 extension() (fermipy.gtanalysis.GTAnalysis method), 57
 ExtensionPlotter (class in fermipy.plotting), 84
 extra_params (fermipy.spectrum.SpectralFunction attribute), 90
 extract_arguments() (in module fermipy.jobs.chain), 111
 extract_array() (in module fermipy.tsmmap), 104
 extract_images_from_tscube() (in module fermipy.tsmmap), 104
 extract_large_array() (in module fermipy.tsmmap), 104
 extract_parameters() (in module fermipy.jobs.gtlink), 119
 extract_small_array() (in module fermipy.tsmmap), 104
- ## F
- f_cash() (in module fermipy.tsmmap), 104
 f_cash_sum() (in module fermipy.tsmmap), 104
 failed (fermipy.jobs.job_archive.JobStatus attribute), 118
 fermipy (module), 105
 fermipy.castro (module), 93
 fermipy.config (module), 53
 fermipy.defaults (module), 54
 fermipy.diffuse (module), 145
 fermipy.diffuse.binning (module), 124
 fermipy.diffuse.catalog_src_manager (module), 129
 fermipy.diffuse.defaults (module), 125
 fermipy.diffuse.diffuse_src_manager (module), 131
 fermipy.diffuse.gt_assemble_model (module), 140
 fermipy.diffuse.gt_coadd_split (module), 142
 fermipy.diffuse.gt_merge_srcmaps (module), 142
 fermipy.diffuse.gt_split_and_bin (module), 143
 fermipy.diffuse.gt_srcmap_partial (module), 144
 fermipy.diffuse.job_library (module), 138
 fermipy.diffuse.model_component (module), 134
 fermipy.diffuse.model_manager (module), 136
 fermipy.diffuse.name_policy (module), 125
 fermipy.diffuse.residual_cr (module), 144
 fermipy.diffuse.source_factory (module), 128
 fermipy.diffuse.spectral (module), 128
 fermipy.jobs (module), 124
 fermipy.jobs.chain (module), 106
 fermipy.jobs.file_archive (module), 111
 fermipy.jobs.gtlink (module), 119
 fermipy.jobs.job_archive (module), 116
 fermipy.jobs.lsf_impl (module), 120
 fermipy.jobs.scatter_gather (module), 121
 fermipy.lightcurve (module), 105
 fermipy.logger (module), 71
 fermipy.plotting (module), 84
 fermipy.residmap (module), 104
 fermipy.roi_model (module), 71
 fermipy.sed (module), 86
 fermipy.skymap (module), 90
 fermipy.sourcefind (module), 87
 fermipy.spectrum (module), 87
 fermipy.tsmmap (module), 102
 fermipy.utils (module), 77
 file_archive (fermipy.jobs.job_archive.JobArchive attribute), 117
 FileArchive (class in fermipy.jobs.file_archive), 111

FileDict (class in fermipy.jobs.file_archive), 113
 FileFlags (class in fermipy.jobs.file_archive), 114
 filefunction (fermipy.roi_model.IsoSource attribute), 71
 FileHandle (class in fermipy.jobs.file_archive), 115
 files (fermipy.gtanalysis.GTAnalysis attribute), 58
 FileStageManager (class in fermipy.jobs.file_archive), 115
 FileStatus (class in fermipy.jobs.file_archive), 116
 fill_argparser() (fermipy.jobs.chain.Link method), 108
 finalize() (fermipy.jobs.chain.Link method), 108
 find_and_refine_peaks() (fermipy.castro.TSCube method), 101
 find_function_root() (in module fermipy.utils), 78
 find_rows_by_string() (in module fermipy.utils), 78
 find_sources() (fermipy.castro.TSCube method), 102
 find_sources() (fermipy.gtanalysis.GTAnalysis method), 58
 find_sources() (fermipy.sourcefind.SourceFind method), 87
 fit() (fermipy.gtanalysis.GTAnalysis method), 59
 fit_correlation() (fermipy.gtanalysis.GTAnalysis method), 59
 fit_parabola() (in module fermipy.utils), 79
 fit_spectrum() (fermipy.castro.CastroData_Base method), 97
 fitNorm_v2() (fermipy.castro.CastroData_Base method), 97
 fitNormalization() (fermipy.castro.CastroData_Base method), 97
 fits_recarray_to_dict() (in module fermipy.utils), 79
 flush() (fermipy.logger.StreamLogger method), 71
 flux (fermipy.castro.SpecData attribute), 101
 flux() (fermipy.spectrum.SpectralFunction method), 90
 fn_mle() (fermipy.castro.LnLFn method), 99
 fn_mles() (fermipy.castro.CastroData_Base method), 97
 format_filename() (in module fermipy.utils), 79
 formatted_command() (fermipy.jobs.chain.Link method), 108
 free_index() (fermipy.gtanalysis.GTAnalysis method), 59
 free_norm() (fermipy.gtanalysis.GTAnalysis method), 59
 free_parameter() (fermipy.gtanalysis.GTAnalysis method), 59
 free_shape() (fermipy.gtanalysis.GTAnalysis method), 59
 free_source() (fermipy.gtanalysis.GTAnalysis method), 60
 free_sources() (fermipy.gtanalysis.GTAnalysis method), 60
 free_sources_by_name() (fermipy.gtanalysis.GTAnalysis method), 60
 ft1file() (fermipy.diffuse.name_policy.NameFactory method), 126
 ft1file_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
 ft2file() (fermipy.diffuse.name_policy.NameFactory

method), 126
 ft2file_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
 fullkey (fermipy.jobs.job_archive.JobDetails attribute), 118
 fullpath() (fermipy.diffuse.name_policy.NameFactory method), 126
 fullpath_format (fermipy.diffuse.name_policy.NameFactory attribute), 126

G

gal2eq() (in module fermipy.utils), 79
 galkeys() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
 galprop_gasmap() (fermipy.diffuse.name_policy.NameFactory method), 126
 galprop_gasmap_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
 galprop_ringkey() (fermipy.diffuse.name_policy.NameFactory method), 126
 galprop_ringkey_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
 galprop_rings_yaml() (fermipy.diffuse.name_policy.NameFactory method), 126
 galprop_rings_yaml_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
 galprop_sourcekey() (fermipy.diffuse.name_policy.NameFactory method), 126
 galprop_sourcekey_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
 GalpropMapManager (class in fermipy.diffuse.diffuse_src_manager), 132
 GalpropMergedRingInfo (class in fermipy.diffuse.model_component), 135
 gather_link (fermipy.jobs.scatter_gather.ScatterGather attribute), 122
 gather_results() (fermipy.jobs.scatter_gather.ScatterGather method), 123
 generate_model() (fermipy.gtanalysis.GTAnalysis method), 61
 generic() (fermipy.diffuse.name_policy.NameFactory method), 126
 get() (fermipy.logger.Logger static method), 71
 get_archive() (fermipy.jobs.file_archive.FileArchive static method), 112
 get_archive() (fermipy.jobs.job_archive.JobArchive static method), 117

[get_catalog_dict\(\)](#) (fermipy.roi_model.Model method), [72](#)
[get_component_info\(\)](#) (fermipy.diffuse.model_component.ModelComponent method), [136](#)
[get_config\(\)](#) (fermipy.config.Configurable class method), [53](#)
[get_config\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_data_projection\(\)](#) (fermipy.plotting.ROIPlotter static method), [85](#)
[get_details\(\)](#) (fermipy.jobs.job_archive.JobArchive method), [117](#)
[get_dist_to_edge\(\)](#) (in module fermipy.roi_model), [77](#)
[get_failed_jobs\(\)](#) (fermipy.jobs.chain.Link method), [108](#)
[get_file_ids\(\)](#) (fermipy.jobs.file_archive.FileArchive method), [112](#)
[get_file_ids\(\)](#) (fermipy.jobs.job_archive.JobDetails method), [118](#)
[get_file_paths\(\)](#) (fermipy.jobs.file_archive.FileArchive method), [112](#)
[get_file_paths\(\)](#) (fermipy.jobs.job_archive.JobDetails method), [118](#)
[get_free_param_vector\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_free_source_params\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_gtapp\(\)](#) (fermipy.jobs.gtlink.Gtlink method), [119](#)
[get_handle\(\)](#) (fermipy.jobs.file_archive.FileArchive method), [112](#)
[get_jobs\(\)](#) (fermipy.jobs.chain.Chain method), [107](#)
[get_jobs\(\)](#) (fermipy.jobs.chain.Link method), [109](#)
[get_jobs\(\)](#) (fermipy.jobs.scatter_gather.ScatterGather method), [123](#)
[get_linear_dist\(\)](#) (in module fermipy.roi_model), [77](#)
[get_lsf_status\(\)](#) (in module fermipy.jobs.lsf_impl), [121](#)
[get_map_values\(\)](#) (fermipy.skymap.HpxMap method), [91](#)
[get_map_values\(\)](#) (fermipy.skymap.Map method), [92](#)
[get_map_values\(\)](#) (fermipy.skymap.Map_Base method), [93](#)
[get_matches\(\)](#) (in module fermipy.jobs.job_archive), [119](#)
[get_nearby_sources\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[get_norm\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_norm\(\)](#) (fermipy.roi_model.Model method), [72](#)
[get_parameter_limits\(\)](#) (in module fermipy.utils), [79](#)
[get_params\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_pixel_indices\(\)](#) (fermipy.skymap.HpxMap method), [91](#)
[get_pixel_indices\(\)](#) (fermipy.skymap.Map method), [92](#)
[get_pixel_indices\(\)](#) (fermipy.skymap.Map_Base method), [93](#)
[get_pixel_skydirs\(\)](#) (fermipy.skymap.HpxMap method), [91](#)
[get_pixel_skydirs\(\)](#) (fermipy.skymap.Map method), [92](#)
[get_pixel_skydirs\(\)](#) (fermipy.skymap.Map_Base method), [93](#)
[get_scratch_path\(\)](#) (fermipy.jobs.file_archive.FileStageManager method), [115](#)
[get_skydir_distance_mask\(\)](#) (in module fermipy.roi_model), [77](#)
[get_source_by_name\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[get_source_dnde\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_source_kernel\(\)](#) (in module fermipy.residmap), [105](#)
[get_source_name\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_sources\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_sources\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[get_sources_by_name\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[get_sources_by_position\(\)](#) (fermipy.roi_model.ROIModel method), [74](#)
[get_sources_by_property\(\)](#) (fermipy.roi_model.ROIModel method), [75](#)
[get_src_model\(\)](#) (fermipy.gtanalysis.GTAnalysis method), [61](#)
[get_st_version\(\)](#) (in module fermipy), [105](#)
[get_sub_comp_info\(\)](#) (fermipy.diffuse.model_manager.ModelManager static method), [137](#)
[get_timestamp\(\)](#) (in module fermipy.jobs.file_archive), [116](#)
[get_true_params_dict\(\)](#) (in module fermipy.roi_model), [77](#)
[get_unique_match\(\)](#) (in module fermipy.jobs.file_archive), [116](#)
[get_xerr\(\)](#) (in module fermipy.plotting), [86](#)
[get_ylimits\(\)](#) (fermipy.plotting.SEDPlotter static method), [85](#)
[getDeltaLogLike\(\)](#) (fermipy.castro.LnLFn method), [99](#)
[getInterval\(\)](#) (fermipy.castro.LnLFn method), [99](#)
[getIntervals\(\)](#) (fermipy.castro.CastroData_Base method), [97](#)
[getLimit\(\)](#) (fermipy.castro.LnLFn method), [99](#)
[getLimits\(\)](#) (fermipy.castro.CastroData_Base method), [98](#)
[gmm](#) (fermipy.diffuse.model_manager.ModelManager attribute), [137](#)
[GTAnalysis](#) (class in fermipy.gtanalysis), [54](#)
[GtAssembleModel](#) (class in fermipy.diffuse.gt_assemble_model), [140](#)
[GtInitModel](#) (class in fermipy.diffuse.gt_assemble_model), [141](#)
[Gtlink](#) (class in fermipy.jobs.gtlink), [119](#)
[GtMergeSourceMaps](#) (class in fermipy.jobs.gtlink), [119](#)

mipy.diffuse.gt_merge_srcmaps), 142
 GtSrcmapPartial (class in fermipy.diffuse.gt_srcmap_partial), 144
 gz_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 gzip_files (fermipy.jobs.file_archive.FileDict attribute), 113

H

has_source() (fermipy.roi_model.ROIModel method), 75
 hpx (fermipy.skymap.HpxMap attribute), 91
 HpxMap (class in fermipy.skymap), 90

I

ImagePlotter (class in fermipy.plotting), 85
 in_ch_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 in_stage_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 init_matplotlib_backend() (in module fermipy.utils), 79
 initialize() (fermipy.jobs.scatter_gather.ScatterGather method), 123
 initialize_link (fermipy.jobs.scatter_gather.ScatterGather attribute), 123
 input_files (fermipy.jobs.file_archive.FileDict attribute), 113
 input_files_to_stage (fermipy.jobs.file_archive.FileDict attribute), 113
 input_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 internal_files (fermipy.jobs.file_archive.FileDict attribute), 113
 internal_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 interp (fermipy.castro.LnLFn attribute), 99
 interpolate() (fermipy.skymap.HpxMap method), 91
 interpolate() (fermipy.skymap.Map method), 92
 interpolate() (fermipy.skymap.Map_Base method), 93
 interpolate_function_min() (in module fermipy.utils), 79
 Interpolator (class in fermipy.castro), 98
 invoke() (fermipy.jobs.scatter_gather.ScatterGather method), 123
 invoke_sg_gtexpcube2() (in module fermipy.diffuse.job_library), 140
 invoke_sg_gsrcmaps_catalog() (in module fermipy.diffuse.job_library), 140
 invoke_sg_healview_diffuse() (in module fermipy.diffuse.job_library), 140
 invoke_sg_sum_ring_gasmapi() (in module fermipy.diffuse.job_library), 140
 invoke_sg_vstack_diffuse() (in module fermipy.diffuse.job_library), 140
 ipix_swap_axes() (fermipy.skymap.Map method), 92
 ipix_to_xypix() (fermipy.skymap.Map method), 92

irfs() (fermipy.diffuse.name_policy.NameFactory method), 126
 is_fits_file() (in module fermipy.utils), 79
 IsoComponentInfo (class in fermipy.diffuse.model_component), 135
 IsoSource (class in fermipy.roi_model), 71
 isstr() (in module fermipy.utils), 79
 items() (fermipy.config.ConfigSchema method), 53
 items() (fermipy.diffuse.model_manager.ModelInfo method), 137
 items() (fermipy.jobs.file_archive.FileDict method), 114
 items() (fermipy.roi_model.Model method), 72

J

job_archive (fermipy.jobs.scatter_gather.ScatterGather attribute), 123
 JobArchive (class in fermipy.jobs.job_archive), 116
 JobDetails (class in fermipy.jobs.job_archive), 117
 JobStatus (class in fermipy.jobs.job_archive), 118
 join_strings() (in module fermipy.utils), 79

L

latch_file_info() (fermipy.jobs.file_archive.FileDict method), 114
 LightCurve (class in fermipy.lightcurve), 105
 lightcurve() (fermipy.gtanalysis.GTAnalysis method), 61
 lightcurve() (fermipy.lightcurve.LightCurve method), 105
 like (fermipy.gtanalysis.GTAnalysis attribute), 62
 Link (class in fermipy.jobs.chain), 107
 links (fermipy.jobs.chain.Chain attribute), 107
 LnLFn (class in fermipy.castro), 99
 load() (fermipy.config.ConfigManager static method), 53
 load() (fermipy.roi_model.ROIModel method), 75
 load_bluered_cmap() (in module fermipy.plotting), 86
 load_data() (in module fermipy.utils), 79
 load_diffuse_srcs() (fermipy.roi_model.ROIModel method), 75
 load_ds9_cmap() (in module fermipy.plotting), 86
 load_existing_catalog() (fermipy.roi_model.ROIModel method), 75
 load_fits_catalog() (fermipy.roi_model.ROIModel method), 75
 load_npy() (in module fermipy.utils), 80
 load_roi() (fermipy.gtanalysis.GTAnalysis method), 62
 load_source() (fermipy.roi_model.ROIModel method), 75
 load_sources() (fermipy.roi_model.ROIModel method), 75
 load_xml() (fermipy.gtanalysis.GTAnalysis method), 62
 load_xml() (fermipy.roi_model.ROIModel method), 75
 load_xml_elements() (in module fermipy.utils), 80
 load_yaml() (in module fermipy.utils), 80
 localize() (fermipy.gtanalysis.GTAnalysis method), 62
 localize() (fermipy.sourcefind.SourceFind method), 87

- log_ebins (fermipy.castro.ReferenceSpec attribute), 100
- log_energies (fermipy.gtanalysis.GTAnalysis attribute), 63
- log_level() (in module fermipy.logger), 71
- log_params (fermipy.spectrum.SpectralFunction attribute), 90
- log_to_params() (fermipy.spectrum.PLEXPcutoff static method), 88
- loge_bounds (fermipy.gtanalysis.GTAnalysis attribute), 63
- Logger (class in fermipy.logger), 71
- loglevel (fermipy.gtanalysis.GTAnalysis attribute), 63
- LogParabola (class in fermipy.spectrum), 88
- lonlat_to_xyz() (in module fermipy.utils), 80
- LsfScatterGather (class in fermipy.jobs.lsf_impl), 120
- ltcube() (fermipy.diffuse.name_policy.NameFactory method), 126
- ltcube_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
- ltcube_moon() (fermipy.diffuse.name_policy.NameFactory method), 126
- ltcube_sun() (fermipy.diffuse.name_policy.NameFactory method), 126
- ltcubemoon_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
- ltcubesun_format (fermipy.diffuse.name_policy.NameFactory attribute), 126
- ## M
- main() (in module fermipy.diffuse.gt_coadd_split), 142
- main_batch() (in module fermipy.diffuse.gt_assemble_model), 141
- main_batch() (in module fermipy.diffuse.gt_merge_srcmaps), 143
- main_batch() (in module fermipy.diffuse.gt_split_and_bin), 143
- main_batch() (in module fermipy.diffuse.gt_srcmap_partial), 144
- main_batch() (in module fermipy.diffuse.residual_cr), 145
- main_browse() (in module fermipy.jobs.file_archive), 116
- main_browse() (in module fermipy.jobs.job_archive), 119
- main_chain() (in module fermipy.diffuse.catalog_src_manager), 131
- main_chain() (in module fermipy.diffuse.diffuse_src_manager), 134
- main_chain() (in module fermipy.diffuse.residual_cr), 145
- main_init() (in module fermipy.diffuse.gt_assemble_model), 141
- main_single() (in module fermipy.diffuse.gt_assemble_model), 141
- main_single() (in module fermipy.diffuse.gt_merge_srcmaps), 143
- main_single() (in module fermipy.diffuse.gt_split_and_bin), 143
- main_single() (in module fermipy.diffuse.gt_srcmap_partial), 144
- main_single() (in module fermipy.diffuse.residual_cr), 145
- make_argv() (fermipy.jobs.chain.Link method), 109
- make_base_config() (fermipy.diffuse.gt_split_and_bin.ConfigMaker_SplitAndBin method), 143
- make_base_config() (fermipy.jobs.scatter_gather.ConfigMaker method), 121
- make_catalog_comp_dict() (in module fermipy.diffuse.catalog_src_manager), 131
- make_catalog_comp_info() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130
- make_catalog_comp_info_dict() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130
- make_catalog_sources() (in module fermipy.diffuse.source_factory), 129
- make_cdisk_kernel() (in module fermipy.utils), 80
- make_cgauss_kernel() (in module fermipy.utils), 80
- make_coadd_hpx() (in module fermipy.skymap), 93
- make_coadd_map() (in module fermipy.skymap), 93
- make_coadd_wcs() (in module fermipy.skymap), 93
- make_components_plots() (fermipy.plotting.AnalysisPlotter method), 84
- make_composite_source() (in module fermipy.diffuse.source_factory), 129
- make_counts_spectrum_plot() (in module fermipy.plotting), 86
- make_default_dict() (in module fermipy.defaults), 54
- make_dict() (fermipy.jobs.file_archive.FileHandle static method), 115
- make_dict() (fermipy.jobs.job_archive.JobDetails static method), 118
- make_diffuse_comp_info() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager method), 131
- make_diffuse_comp_info() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
- make_diffuse_comp_info_dict() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager method), 132
- make_diffuse_comp_info_dict() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
- make_diffuse_comp_info_dict() (in module fermipy.diffuse.diffuse_src_manager), 133

mipy.diffuse.diffuse_src_manager), 134
 make_disk_kernel() (in module fermipy.utils), 80
 make_extension_plots() (fermipy.plotting.AnalysisPlotter method), 84
 make_fermipy_config_yaml() (fermipy.diffuse.model_manager.ModelManager method), 137
 make_fermipy_roi_model_from_catalogs() (fermipy.diffuse.source_factory.SourceFactory static method), 128
 make_filenames() (fermipy.diffuse.name_policy.NameFactory method), 126
 make_fullkey() (fermipy.jobs.job_archive.JobDetails static method), 118
 make_gaussian_kernel() (in module fermipy.utils), 80
 make_isotropic_source() (in module fermipy.diffuse.source_factory), 129
 make_job_details() (fermipy.jobs.job_archive.JobArchive method), 117
 make_key() (fermipy.diffuse.binning.Component method), 125
 make_library() (fermipy.diffuse.model_manager.ModelManager method), 138
 make_library() (in module fermipy.diffuse.model_manager), 138
 make_link() (fermipy.diffuse.gt_assemble_model.GtAssembleModel static method), 141
 make_link() (fermipy.diffuse.gt_assemble_model.GtInitModel static method), 141
 make_link() (fermipy.diffuse.gt_merge_srcmaps.GtMergeSourceMaps static method), 142
 make_localization_plots() (fermipy.plotting.AnalysisPlotter method), 84
 make_mapcube_source() (in module fermipy.diffuse.source_factory), 129
 make_mask() (in module fermipy.diffuse.catalog_src_manager), 131
 make_merged_name() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
 make_model_info() (fermipy.diffuse.model_manager.ModelManager method), 138
 make_model_rois() (fermipy.diffuse.model_manager.ModelInfo method), 137
 make_parser() (fermipy.diffuse.gt_assemble_model.GtAssembleModel static method), 141
 make_parser() (fermipy.diffuse.gt_assemble_model.GtInitModel static method), 141
 make_parser() (fermipy.diffuse.gt_merge_srcmaps.GtMergeSourceMaps static method), 142
 make_pixel_distance() (in module fermipy.utils), 80
 make_plots() (fermipy.gtanalysis.GTAnalysis method), 63
 make_point_source() (in module fermipy.diffuse.source_factory), 129
 make_psf_kernel() (in module fermipy.utils), 80
 make_residmap_plots() (fermipy.plotting.AnalysisPlotter method), 84
 make_ring_dict() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
 make_ring_dicts() (in module fermipy.diffuse.diffuse_src_manager), 134
 make_ring_filelist() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
 make_ring_filename() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
 make_roi() (fermipy.diffuse.source_factory.SourceFactory static method), 128
 make_roi_plots() (fermipy.plotting.AnalysisPlotter method), 84
 make_scratch_dirs() (fermipy.jobs.file_archive.FileStageManager static method), 115
 make_sed_plots() (fermipy.plotting.AnalysisPlotter method), 84
 make_sources() (in module fermipy.diffuse.source_factory), 129
 make_srcmap_manifest() (fermipy.diffuse.model_manager.ModelInfo method), 137
 make_srcmap_manifest() (fermipy.diffuse.model_manager.ModelManager method), 138
 make_table() (fermipy.jobs.file_archive.FileHandle static method), 115
 make_tables() (fermipy.jobs.job_archive.JobDetails static method), 118
 make_template_name() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager method), 132
 make_tsmmap_plots() (fermipy.plotting.AnalysisPlotter method), 84
 make_wcs_from_hpx() (fermipy.skymap.HpxMap method), 91
 make_xml_name() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager method), 132
 make_xml_name() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 133
 map_arguments() (fermipy.skymap.HpxMap in fermipy.skymap), 91
 map_arguments() (fermipy.jobs.chain.Chain method), 107

Map_Base (class in fermipy.skymap), 93
 map_files() (fermipy.jobs.file_archive.FileStageManager method), 115
 map_scratch_files() (fermipy.jobs.chain.Link method), 109
 mapcube (fermipy.roi_model.MapCubeSource attribute), 71
 MapCubeComponentInfo (class in fermipy.diffuse.model_component), 135
 MapCubeSource (class in fermipy.roi_model), 71
 mask_extended() (in module fermipy.diffuse.catalog_src_manager), 131
 master_srcmdl_xml() (fermipy.diffuse.name_policy.NameFactory method), 127
 master_srcmdl_xml_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 match_regex_list() (in module fermipy.utils), 80
 match_source() (fermipy.roi_model.ROIModel method), 75
 mcube() (fermipy.diffuse.name_policy.NameFactory method), 127
 mcube_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 merge_dict() (in module fermipy.utils), 80
 merged_components() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 134
 merged_gasmap() (fermipy.diffuse.name_policy.NameFactory method), 127
 merged_gasmap_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 merged_sourcekey() (fermipy.diffuse.name_policy.NameFactory method), 127
 merged_sourcekey_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 merged_srcmaps() (fermipy.diffuse.name_policy.NameFactory method), 127
 merged_srcmaps_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 met_to_mjd() (in module fermipy.utils), 81
 missing (fermipy.jobs.file_archive.FileStatus attribute), 116
 missing_input_files() (fermipy.jobs.chain.Chain method), 107
 missing_input_files() (fermipy.jobs.chain.Link method), 109
 missing_output_files() (fermipy.jobs.chain.Chain method), 107
 missing_output_files() (fermipy.jobs.chain.Link method), 109
 mkdir() (in module fermipy.utils), 81
 mle() (fermipy.castro.LnLFn method), 99
 mles() (fermipy.castro.CastroData_Base method), 98
 Model (class in fermipy.roi_model), 71
 model_counts_map() (fermipy.gtanalysis.GTAnalysis method), 63
 model_counts_spectrum() (fermipy.gtanalysis.GTAnalysis method), 63
 model_yaml() (fermipy.diffuse.name_policy.NameFactory method), 127
 model_yaml_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 ModelComponent (class in fermipy.diffuse.model_manager), 136
 ModelComponentInfo (class in fermipy.diffuse.model_component), 135
 ModelInfo (class in fermipy.diffuse.model_manager), 137
 ModelManager (class in fermipy.diffuse.model_manager), 137

N

name (fermipy.roi_model.Model attribute), 72
 NameFactory (class in fermipy.diffuse.name_policy), 125
 names (fermipy.roi_model.Model attribute), 72
 nE (fermipy.castro.CastroData attribute), 95
 nE (fermipy.castro.ReferenceSpec attribute), 100
 nE (fermipy.castro.TSCube attribute), 102
 nested_sources (fermipy.roi_model.CompositeSource attribute), 71
 nested_srcmdl_xml() (fermipy.diffuse.name_policy.NameFactory method), 127
 nested_srcmdl_xml_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 nll_null (fermipy.castro.CastroData_Base attribute), 98
 nN (fermipy.castro.TSCube attribute), 102
 no_batch (fermipy.jobs.scatter_gather.ScatterGather attribute), 123
 no_file (fermipy.jobs.file_archive.FileStatus attribute), 116
 no_flags (fermipy.jobs.file_archive.FileFlags attribute), 114
 no_job (fermipy.jobs.job_archive.JobStatus attribute), 118
 norm (fermipy.castro.SpecData attribute), 101
 norm_derivative() (fermipy.castro.CastroData_Base method), 98
 norm_err (fermipy.castro.SpecData attribute), 101

- norm_type (fermipy.castro.CastroData_Base attribute), 98
- norm_type (fermipy.castro.LnLFn attribute), 99
- normcube (fermipy.castro.TSCube attribute), 102
- normmap (fermipy.castro.TSCube attribute), 102
- nparam() (fermipy.spectrum.DMFitFunction static method), 88
- nparam() (fermipy.spectrum.LogParabola static method), 88
- nparam() (fermipy.spectrum.PLEXPcutoff static method), 88
- nparam() (fermipy.spectrum.PowerLaw static method), 89
- npix (fermipy.gtanalysis.GTAnalysis attribute), 63
- npix (fermipy.skymap.Map attribute), 92
- NULL_MODEL (fermipy.diffuse.gt_merge_srcmaps.GtMergeSourceMaps attribute), 142
- NULL_MODEL (fermipy.diffuse.gt_srcmap_partial.GtSrcmapPartial attribute), 144
- nvals (fermipy.castro.TSCube attribute), 102
- nx (fermipy.castro.CastroData_Base attribute), 98
- ny (fermipy.castro.CastroData_Base attribute), 98
- ## O
- onesided_cl_to_dlnl() (in module fermipy.utils), 81
- onesided_dlnl_to_cl() (in module fermipy.utils), 81
- open_outsrcmap() (fermipy.diffuse.gt_assemble_model.GtAssembleModel static method), 141
- optimize() (fermipy.gtanalysis.GTAnalysis method), 63
- out_ch_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
- out_stage_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
- outdir (fermipy.gtanalysis.GTAnalysis attribute), 64
- output_files (fermipy.jobs.file_archive.FileDict attribute), 114
- output_files_to_stage (fermipy.jobs.file_archive.FileDict attribute), 114
- output_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
- overlap_slices() (in module fermipy.utils), 81
- ## P
- parabola() (in module fermipy.utils), 81
- params (fermipy.roi_model.Model attribute), 72
- params (fermipy.spectrum.SEDFunction attribute), 89
- params (fermipy.spectrum.SpectralFunction attribute), 90
- params_to_log() (fermipy.spectrum.PLEXPcutoff static method), 88
- path_to_xmlpath() (in module fermipy.utils), 82
- pending (fermipy.jobs.job_archive.JobStatus attribute), 118
- pix_center (fermipy.skymap.Map attribute), 92
- pix_size (fermipy.skymap.Map attribute), 92
- PLEXPcutoff (class in fermipy.spectrum), 88
- plot() (fermipy.plotting.ExtensionPlotter method), 85
- plot() (fermipy.plotting.ImagePlotter method), 85
- plot() (fermipy.plotting.ROIPlotter method), 85
- plot() (fermipy.plotting.SEDPlotter method), 85
- plot_catalog() (fermipy.plotting.ROIPlotter method), 85
- plot_error_ellipse() (in module fermipy.plotting), 86
- plot_flux_points() (fermipy.plotting.SEDPlotter static method), 85
- plot_inlscan() (fermipy.plotting.SEDPlotter static method), 85
- plot_markers() (in module fermipy.plotting), 86
- plot_model() (fermipy.plotting.SEDPlotter static method), 85
- plot_projection() (fermipy.plotting.ROIPlotter method), 85
- plot_resid() (fermipy.plotting.SEDPlotter static method), 85
- plot_roi() (fermipy.plotting.ROIPlotter method), 85
- plot_sed() (fermipy.plotting.SEDPlotter static method), 85
- plot_sources() (fermipy.plotting.ROIPlotter method), 85
- plotter (fermipy.gtanalysis.GTAnalysis attribute), 64
- point_sources (fermipy.roi_model.ROIModel attribute), 75
- PointSourceInfo (class in fermipy.diffuse.model_component), 136
- poisson_inl() (in module fermipy.residmap), 105
- poisson_log_like() (in module fermipy.tsmmap), 104
- poly_to_parabola() (in module fermipy.utils), 82
- PowerLaw (class in fermipy.spectrum), 88
- pre_run_checks() (fermipy.jobs.chain.Link method), 109
- prettify_xml() (in module fermipy.utils), 82
- print_chain_summary() (fermipy.jobs.file_archive.FileDict method), 114
- print_config() (fermipy.config.Configurable method), 53
- print_config() (fermipy.gtanalysis.GTAnalysis method), 64
- print_failed() (fermipy.jobs.scatter_gather.ScatterGather method), 123
- print_model() (fermipy.gtanalysis.GTAnalysis method), 64
- print_params() (fermipy.gtanalysis.GTAnalysis method), 64
- print_roi() (fermipy.gtanalysis.GTAnalysis method), 64
- print_summary() (fermipy.jobs.chain.Chain method), 107
- print_summary() (fermipy.jobs.chain.Link method), 109
- print_summary() (fermipy.jobs.file_archive.FileDict method), 114
- print_summary() (fermipy.jobs.scatter_gather.ScatterGather method), 123
- print_update() (fermipy.jobs.scatter_gather.ScatterGather method), 123

- profile() (fermipy.gtanalysis.GTAnalysis method), 64
 profile_norm() (fermipy.gtanalysis.GTAnalysis method), 65
 proj (fermipy.plotting.ROIPlotter attribute), 85
 project() (in module fermipy.utils), 82
 projection (fermipy.roi_model.ROIModel attribute), 75
 projtype (fermipy.gtanalysis.GTAnalysis attribute), 65
 projtype (fermipy.plotting.ImagePlotter attribute), 85
 projtype (fermipy.plotting.ROIPlotter attribute), 85
 psf_scale_fn (fermipy.roi_model.Model attribute), 72
- ## R
- radec (fermipy.roi_model.Source attribute), 76
 read_catalog_info_yaml() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130
 read_diffuse_component_yaml() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager static method), 132
 read_galprop_rings_yaml() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 134
 read_map_from_fits() (in module fermipy.skymap), 93
 read_model_yaml() (fermipy.diffuse.model_manager.ModelManager method), 138
 readlines() (in module fermipy.diffuse.gt_split_and_bin), 143
 rebin_map() (in module fermipy.utils), 82
 ref_dnde (fermipy.castro.ReferenceSpec attribute), 100
 ref_efflux (fermipy.castro.ReferenceSpec attribute), 100
 ref_flux (fermipy.castro.ReferenceSpec attribute), 100
 ref_npred (fermipy.castro.ReferenceSpec attribute), 100
 ReferenceSpec (class in fermipy.castro), 99
 refSpec (fermipy.castro.CastroData attribute), 95
 refSpec (fermipy.castro.TSCube attribute), 102
 register_file() (fermipy.jobs.file_archive.FileArchive method), 112
 register_job() (fermipy.jobs.chain.Link method), 109
 register_job() (fermipy.jobs.job_archive.JobArchive method), 117
 register_job_from_link() (fermipy.jobs.job_archive.JobArchive method), 117
 register_jobs() (fermipy.jobs.job_archive.JobArchive method), 117
 reload_source() (fermipy.gtanalysis.GTAnalysis method), 65
 reload_sources() (fermipy.gtanalysis.GTAnalysis method), 65
 remove_file() (in module fermipy.jobs.scatter_gather), 124
 remove_prior() (fermipy.gtanalysis.GTAnalysis method), 65
 remove_priors() (fermipy.gtanalysis.GTAnalysis method), 65
 residmap() (fermipy.gtanalysis.GTAnalysis method), 65
 residmap() (fermipy.residmap.ResidMapGenerator method), 104
 ResidMapGenerator (class in fermipy.residmap), 104
 residual_cr() (fermipy.diffuse.name_policy.NameFactory method), 127
 residual_cr_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
 ResidualCRAnalysis (class in fermipy.diffuse.residual_cr), 144
 ResidualCRChain (class in fermipy.diffuse.residual_cr), 145
 resolve_file_path() (in module fermipy.utils), 82
 resolve_file_path_list() (in module fermipy.utils), 82
 resolve_path() (in module fermipy.utils), 82
 resubmit() (fermipy.jobs.scatter_gather.ScatterGather method), 123
 ring_dict() (fermipy.diffuse.diffuse_src_manager.GalpropMapManager method), 134
 rm_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 rmint_mask (fermipy.jobs.file_archive.FileFlags attribute), 114
 roi (fermipy.gtanalysis.GTAnalysis attribute), 65
 ROIModel (class in fermipy.roi_model), 72
 ROIPlotter (class in fermipy.plotting), 85
 run() (fermipy.diffuse.gt_assemble_model.GtAssembleModel method), 141
 run() (fermipy.diffuse.gt_assemble_model.GtInitModel method), 141
 run() (fermipy.diffuse.gt_merge_srcmaps.GtMergeSourceMaps method), 142
 run() (fermipy.diffuse.gt_srcmap_partial.GtSrcmapPartial method), 144
 run() (fermipy.diffuse.residual_cr.ResidualCRAnalysis method), 145
 run() (fermipy.jobs.chain.Chain method), 107
 run() (fermipy.jobs.chain.Link method), 109
 run() (fermipy.jobs.scatter_gather.ScatterGather method), 123
 run() (fermipy.plotting.AnalysisPlotter method), 84
 run_argparser() (fermipy.diffuse.catalog_src_manager.CatalogComponentChain method), 129
 run_argparser() (fermipy.diffuse.diffuse_src_manager.DiffuseComponentChain method), 131
 run_argparser() (fermipy.diffuse.gt_coadd_split.CoaddSplit method), 142
 run_argparser() (fermipy.diffuse.gt_split_and_bin.SplitAndBin method), 143
 run_argparser() (fermipy.diffuse.residual_cr.ResidualCRChain method), 145

run_argparser() (fermipy.jobs.chain.Link method), 110
run_argparser() (fermipy.jobs.scatter_gather.ScatterGather method), 123
run_chain() (fermipy.jobs.chain.Chain method), 107
run_command() (fermipy.jobs.chain.Link method), 110
run_command() (fermipy.jobs.gtlink.Gtlink method), 119
run_gtapp() (in module fermipy.jobs.gtlink), 120
run_jobs() (fermipy.jobs.scatter_gather.ScatterGather method), 123
run_link() (fermipy.jobs.chain.Link method), 110
running (fermipy.jobs.job_archive.JobStatus attribute), 119

S

scale (fermipy.spectrum.SEDFunctor attribute), 89
scale (fermipy.spectrum.SpectralFunction attribute), 90
scale_parameter() (fermipy.gtanalysis.GTAnalysis method), 65
scale_parameter() (in module fermipy.utils), 82
scatter_link (fermipy.jobs.scatter_gather.ScatterGather attribute), 123
ScatterGather (class in fermipy.jobs.scatter_gather), 122
schema (fermipy.config.Configurable attribute), 53
schema (fermipy.gtanalysis.GTAnalysis attribute), 65
sed (fermipy.plotting.SEDPlotter attribute), 86
sed() (fermipy.gtanalysis.GTAnalysis method), 65
sed() (fermipy.sed.SEDGenerator method), 86
SEDEFuxFunctor (class in fermipy.spectrum), 89
SEDEFuxFunctor (class in fermipy.spectrum), 89
SEDFunctor (class in fermipy.spectrum), 89
SEDGenerator (class in fermipy.sed), 86
SEDPlotter (class in fermipy.plotting), 85
select_extended() (in module fermipy.diffuse.catalog_src_manager), 131
select_sources() (in module fermipy.diffuse.catalog_src_manager), 131
separation() (fermipy.roi_model.Source method), 76
separation_cos_angle() (in module fermipy.utils), 82
set_edisp_flag() (fermipy.gtanalysis.GTAnalysis method), 66
set_energy_range() (fermipy.gtanalysis.GTAnalysis method), 66
set_file_stage() (fermipy.jobs.chain.Link method), 110
set_free_param_vector() (fermipy.gtanalysis.GTAnalysis method), 66
set_log_level() (fermipy.gtanalysis.GTAnalysis method), 67
set_name() (fermipy.roi_model.Model method), 72
set_norm() (fermipy.gtanalysis.GTAnalysis method), 67
set_norm_bounds() (fermipy.gtanalysis.GTAnalysis method), 67
set_norm_scale() (fermipy.gtanalysis.GTAnalysis method), 67

set_parameter() (fermipy.gtanalysis.GTAnalysis method), 67
set_parameter_bounds() (fermipy.gtanalysis.GTAnalysis method), 67
set_parameter_scale() (fermipy.gtanalysis.GTAnalysis method), 67
set_position() (fermipy.roi_model.Source method), 76
set_projection() (fermipy.roi_model.ROIModel method), 75
set_psf_scale_fn() (fermipy.roi_model.Model method), 72
set_radec() (fermipy.roi_model.Source method), 76
set_roi_direction() (fermipy.roi_model.Source method), 76
set_roi_projection() (fermipy.roi_model.Source method), 76
set_source_dnde() (fermipy.gtanalysis.GTAnalysis method), 67
set_source_morphology() (fermipy.gtanalysis.GTAnalysis method), 67
set_source_spectrum() (fermipy.gtanalysis.GTAnalysis method), 67
set_spatial_model() (fermipy.roi_model.Source method), 76
set_spectral_pars() (fermipy.roi_model.Model method), 72
setup() (fermipy.gtanalysis.GTAnalysis method), 68
setup() (fermipy.logger.Logger static method), 71
setup_projection_axis() (fermipy.plotting.ROIPlotter static method), 85
simulate_roi() (fermipy.gtanalysis.GTAnalysis method), 68
simulate_source() (fermipy.gtanalysis.GTAnalysis method), 68
skydir (fermipy.roi_model.ROIModel attribute), 75
skydir (fermipy.roi_model.Source attribute), 76
skydir (fermipy.skymap.Map attribute), 93
Source (class in fermipy.roi_model), 76
source_info_dict (fermipy.diffuse.source_factory.SourceFactory attribute), 129
SourceFactory (class in fermipy.diffuse.source_factory), 128
SourceFind (class in fermipy.sourcefind), 87
sourcekey() (fermipy.diffuse.name_policy.NameFactory method), 127
sourcekey_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
sourcekeys() (fermipy.diffuse.diffuse_src_manager.DiffuseModelManager method), 132
sources (fermipy.diffuse.source_factory.SourceFactory attribute), 129
sources (fermipy.roi_model.ROIModel attribute), 75
spatial_pars (fermipy.roi_model.Model attribute), 72
SpecData (class in fermipy.castro), 100

- spectral_fn (fermipy.spectrum.SEDFunctor attribute), 89
- spectral_pars (fermipy.roi_model.Model attribute), 72
- spectral_pars_from_catalog() (in module fermipy.roi_model), 77
- spectral_template() (fermipy.diffuse.name_policy.NameFactory method), 127
- spectral_template_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
- SpectralFunction (class in fermipy.spectrum), 89
- SpectralLibrary (class in fermipy.diffuse.spectral), 128
- spectrum_loglike() (fermipy.castro.CastroData method), 95
- split_bin_edges() (in module fermipy.utils), 82
- split_comp_info() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 130
- split_comp_info_dict() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 131
- split_fullkey() (fermipy.jobs.job_archive.JobDetails static method), 118
- split_local_path() (fermipy.jobs.file_archive.FileStageManager method), 116
- SplitAndBin (class in fermipy.diffuse.gt_split_and_bin), 143
- splitkeys() (fermipy.diffuse.catalog_src_manager.CatalogSourceManager method), 131
- src_name_cols (fermipy.roi_model.ROIModel attribute), 75
- srcmaps() (fermipy.diffuse.name_policy.NameFactory method), 127
- srcmaps_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
- srcmdl_xml() (fermipy.diffuse.name_policy.NameFactory method), 127
- srcmdl_xml_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
- stack_nll() (fermipy.castro.CastroData_Base static method), 98
- stage_input() (fermipy.gtanalysis.GTAnalysis method), 68
- stage_input_files() (fermipy.jobs.chain.Link method), 110
- stage_output() (fermipy.gtanalysis.GTAnalysis method), 68
- stage_output_files() (fermipy.jobs.chain.Link method), 110
- stageable (fermipy.jobs.file_archive.FileFlags attribute), 114
- stamp() (fermipy.diffuse.name_policy.NameFactory method), 127
- stamp_format (fermipy.diffuse.name_policy.NameFactory attribute), 127
- StreamLogger (class in fermipy.logger), 71
- strip_suffix() (in module fermipy.utils), 82
- submit_jobs() (fermipy.jobs.lsf_impl.LsfScatterGather method), 120
- submit_jobs() (fermipy.jobs.scatter_gather.ScatterGather method), 123
- sum_bins() (in module fermipy.utils), 82
- sum_over_energy() (fermipy.skymap.HpxMap method), 91
- sum_over_energy() (fermipy.skymap.Map method), 93
- sum_over_energy() (fermipy.skymap.Map_Base method), 93
- superseded (fermipy.jobs.file_archive.FileStatus attribute), 116
- sunmoon() (fermipy.skymap.HpxMap method), 91
- ## T
- table (fermipy.jobs.job_archive.JobArchive attribute), 117
- table_file (fermipy.jobs.file_archive.FileArchive attribute), 112
- table_file (fermipy.jobs.job_archive.JobArchive attribute), 117
- table_files (fermipy.jobs.job_archive.JobArchive attribute), 117
- temp_files (fermipy.jobs.file_archive.FileDict attribute), 114
- temp_removed (fermipy.jobs.file_archive.FileStatus attribute), 116
- template_sunmoon() (fermipy.diffuse.name_policy.NameFactory method), 127
- templatesunmoon_format (fermipy.diffuse.name_policy.NameFactory attribute), 128
- test() (in module fermipy), 105
- test_spectra() (fermipy.castro.CastroData method), 96
- test_spectra_of_peak() (fermipy.castro.TSCube method), 102
- tmax (fermipy.gtanalysis.GTAnalysis attribute), 68
- tmin (fermipy.gtanalysis.GTAnalysis attribute), 68
- tolist() (in module fermipy.utils), 82
- truncate_array() (in module fermipy.tsmmap), 104
- truncate_colormap() (in module fermipy.plotting), 86
- TS() (fermipy.castro.LnLFn method), 99
- ts_cumul (fermipy.castro.TSCube attribute), 102
- TS_spectrum() (fermipy.castro.CastroData_Base method), 96
- ts_vals() (fermipy.castro.CastroData_Base method), 98
- TSCube (class in fermipy.castro), 101
- tscube (fermipy.castro.TSCube attribute), 102

tscube() (fermipy.gtanalysis.GTAnalysis method), 68
 tscube() (fermipy.tsmap.TSCubeGenerator method), 102
 TSCubeGenerator (class in fermipy.tsmap), 102
 tsmap (fermipy.castro.TSCube attribute), 102
 tsmap() (fermipy.gtanalysis.GTAnalysis method), 69
 tsmap() (fermipy.tsmap.TSMapGenerator method), 103
 TSMapGenerator (class in fermipy.tsmap), 103
 twosided_cl_to_dlnl() (in module fermipy.utils), 83
 twosided_dlnl_to_cl() (in module fermipy.utils), 83

U

ud_grade() (fermipy.skymap.HpxMap method), 91
 unicode_representer() (in module fermipy.utils), 83
 unicode_to_str() (in module fermipy.utils), 83
 unzero_source() (fermipy.gtanalysis.GTAnalysis method), 70
 update() (fermipy.diffuse.model_component.CatalogInfo method), 134
 update() (fermipy.diffuse.model_component.GalpropMergedCatalogInfo method), 135
 update() (fermipy.diffuse.model_component.ModelComponent method), 136
 update() (fermipy.diffuse.spectral.SpectralLibrary method), 128
 update() (fermipy.jobs.file_archive.FileDict method), 114
 update_args() (fermipy.jobs.chain.Chain method), 107
 update_args() (fermipy.jobs.chain.Link method), 110
 update_args() (fermipy.jobs.gtlink.Gtlink method), 119
 update_args() (fermipy.jobs.scatter_gather.ScatterGather method), 124
 update_base_dict() (fermipy.diffuse.name_policy.NameFactory method), 128
 update_bounds() (in module fermipy.utils), 83
 update_data() (fermipy.roi_model.Model method), 72
 update_data() (fermipy.roi_model.Source method), 76
 update_file() (fermipy.jobs.file_archive.FileArchive method), 112
 update_file_args() (fermipy.jobs.chain.Link method), 110
 update_file_status() (fermipy.jobs.file_archive.FileArchive method), 113
 update_from_schema() (in module fermipy.config), 54
 update_from_source() (fermipy.roi_model.Model method), 72
 update_gtapp() (in module fermipy.jobs.gtlink), 120
 update_job() (fermipy.jobs.job_archive.JobArchive method), 117
 update_job_status() (fermipy.jobs.job_archive.JobArchive method), 117
 update_keys() (in module fermipy.utils), 83
 update_links() (fermipy.diffuse.gt_coadd_split.CoaddSplit method), 142

update_links() (fermipy.diffuse.gt_split_and_bin.SplitAndBin method), 143
 update_options() (fermipy.jobs.chain.Link method), 110
 update_source() (fermipy.gtanalysis.GTAnalysis method), 70
 update_spectral_pars() (fermipy.roi_model.Model method), 72
 update_sub_file_dict() (fermipy.jobs.chain.Link method), 111
 update_table_row() (fermipy.jobs.file_archive.FileHandle method), 115
 update_table_row() (fermipy.jobs.job_archive.JobDetails method), 118

V

val_to_bin() (in module fermipy.utils), 84
 val_to_bin_bounded() (in module fermipy.utils), 84
 val_to_edge() (in module fermipy.utils), 84
 val_to_index() (in module fermipy.utils), 84
 validate_config() (in module fermipy.config), 54
 validate_from_schema() (in module fermipy.config), 54
 validate_option() (in module fermipy.config), 54

W

wcs (fermipy.skymap.Map attribute), 93
 width (fermipy.skymap.Map attribute), 93
 workdir (fermipy.gtanalysis.GTAnalysis attribute), 70
 write() (fermipy.logger.StreamLogger method), 71
 write_config() (fermipy.config.Configurable method), 53
 write_config() (fermipy.gtanalysis.GTAnalysis method), 70
 write_fits() (fermipy.gtanalysis.GTAnalysis method), 70
 write_fits() (fermipy.roi_model.ROIModel method), 75
 write_model_map() (fermipy.gtanalysis.GTAnalysis method), 70
 write_roi() (fermipy.gtanalysis.GTAnalysis method), 70
 write_table_file() (fermipy.jobs.file_archive.FileArchive method), 113
 write_table_file() (fermipy.jobs.job_archive.JobArchive method), 117
 write_xml() (fermipy.gtanalysis.GTAnalysis method), 70
 write_xml() (fermipy.roi_model.CompositeSource method), 71
 write_xml() (fermipy.roi_model.IsoSource method), 71
 write_xml() (fermipy.roi_model.MapCubeSource method), 71
 write_xml() (fermipy.roi_model.ROIModel method), 75
 write_xml() (fermipy.roi_model.Source method), 77
 write_yaml() (in module fermipy.utils), 84

X

x (fermipy.castro.Interpolator attribute), 99
 xmax (fermipy.castro.Interpolator attribute), 99
 xmin (fermipy.castro.Interpolator attribute), 99

`xmlpath_to_path()` (in module `fermipy.utils`), [84](#)
`xypix_to_ipix()` (`fermipy.skymap.Map` method), [93](#)
`xyz_to_lonlat()` (in module `fermipy.utils`), [84](#)

Y

`y` (`fermipy.castro.Interpolator` attribute), [99](#)

Z

`zero_source()` (`fermipy.gtanalysis.GTAnalysis` method),
[71](#)
`zoom()` (`fermipy.plotting.ROIPlotter` method), [85](#)