# Fermipy Documentation

***Release 0.10.0+dirty***

**Matthew Wood**

July 03, 2016

Contents


**1 Introduction** **1**

1.1 Documentation Contents . . . 1

1.1.1 Installation . . . 1

1.1.2 Quickstart Guide . . . 5

1.1.3 Configuration . . . 11

1.1.4 Output File . . . 20

1.1.5 ROI Optimization and Fitting . . . 22

1.1.6 Customizing the Model . . . 25

1.1.7 Advanced Analysis Methods . . . 27

1.1.8 fermipy package . . . 41

1.1.9 Changelog . . . 81

**2 Indices and tables** **83**

**Python Module Index** **85**

CHAPTER 1

# Introduction

This is the Fermipy documentation page. Fermipy is a set of python modules and scripts that automate analysis with the Fermi Science Tools. fermipy provides a configuration-file driven workflow in which the analysis parameters (data selection, IRFs, and ROI model) are defined in a user-specified YAML file. The analysis is controlled with a set of python classes that provide methods to execute various analysis tasks. For instruction on installing Fermipy see the *Installation* page. For a short introduction to using Fermipy see the *Quickstart Guide*.

## 1.1 Documentation Contents

### 1.1.1 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.

#### 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-201505
$ tar xzf ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.19-10-without-rootA.tar.g
$ export FERMI_DIR=ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.19-10-without-ro
$ 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 conda 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.

The `condainstall.sh` script can be used to install fermipy into an existing conda python installation or to create a minimal conda installation from scratch. In either case download and run the `condainstall.sh` installation script from the fermipy repository:

```
$ curl -OL https://raw.githubusercontent.com/fermiPy/fermipy/master/condainstall.sh
$ bash 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 conda installed and the `conda` command is in your path the script will use your existing installation. The script will create a separate environment for your fermipy installation called *fermi-env*.

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

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

This will both activate the *fermi-env* environment and set up your shell environment to run the Fermi Science Tools. The *fermi-env* python environment can be exited by running:

```
$ source deactivate
```

### 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 10-01-01). To manually override the ST version you can optionally 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` will get the latest tagged released available on the PyPi package respository. You can check your currently installed version of fermipy with `pip show`:

```
$ pip show fermipy
---
Metadata-Version: 2.0
Name: fermipy
```

```
Version: 0.6.7
Summary: A Python package for analysis of Fermi-LAT data
Home-page: https://github.com/fermiPy/fermipy
Author: The Fermipy developers
Author-email: fermipy.developers@gmail.com
License: BSD
Location: /home/vagrant/miniconda/envs/fermi-env/lib/python2.7/site-packages
Requires: wcsaxes, astropy, matplotlib, healpy, scipy, numpy, pyyaml
```

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
```

### Building from Source

These instructions describe how to install fermipy from its git source code repository using `setup.py`. Installing from source is necessary if you want to do local development or test features in an untagged release. Note that for non-expert users it is recommended to install fermipy with `pip` following the instructions above. First clone the fermipy repository:

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

To install the head of the master branch run `setup.py install` from the root of the source tree:

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

A useful option if you are doing active code development is to install your working copy as the local installation. This can be done by running `setup.py develop`:

```
# 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 use `git tag`:

```
$ git tag
0.4.0
0.5.0
0.5.1
0.5.2
0.5.3
0.5.4
0.6.0
0.6.1
```

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 enable the `Agg` backend at runtime:

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

However this backend does not support interactive plotting.

In some cases the setup.py script will fail to properly install the fermipy package dependecies. 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
```

### 1.1.2 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 FT1 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 use 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()*. The 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)
m1 = gta.tsmap('fit_model',model=model)
```

More documentation about these methods is available in the *Source Detection* page.

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')
```

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 / (cm2 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 respository. 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
```

### 1.1.3 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 blocks of the configuration.

The class configuration dictionary is initialized at the time of object creation by passing a dictionary or a path to YAML configuration file to the class constructor. Keyword arguments can be optionally 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*, *binnig*, 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 available 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 | Default | Description |
|---|---|---|
| `binsperdec` | 8 | Number of energy bins per decade. |
| `binsz` | 0.1 | Spatial bin size in degrees. |
| `coordsys` | CEL | Coordinate system of the spatial projection (CEL or GAL). |
| `enumbins` | None | Number of energy bins. If none this will be inferred from energy range and `binsperdec` parameter. |
| `hpx_ebin` | True | Include energy binning |
| `hpx_order` | 10 | Order of the map (int between 0 and 12, included) |
| `hpx_ordering_scheme` | RING | HEALPix Ordering Scheme |
| `npix` | None | Number of pixels. If none then this will be set from `roiwidth` and `binsz`. |
| `proj` | AIT | Spatial projection for WCS mode. |
| `projtype` | WCS | Projection mode (WCS or HPX). |
| `roiwidth` | 10.0 | Width of the ROI in degrees. The number of pixels in each spatial dimension will be set from `roiwidth` / `binsz` (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 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 can be defined as either a list or dictionary of dictionary elements 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. In this case files associated to each component will be named according to their order in the list (e.g. file_00.fits, file_01.fits, etc.).

```
# Component section for Front/Back analysis with list style
components:
  - { selection : { evtype : 1 } } # Front
  - { selection : { evtype : 2 } } # Back
```

This example illustrates how to define the components as a dictionary of dictionaries. In this case the files of a component will be appended with its corresponding key (e.g. file_front.fits, file_back.fits).

```
# Component section for Front/Back analysis with dictionary style
components:
  front : { selection : { evtype : 1 } } # Front
  back  : { 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 `gtmktime`. |
| `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 |
|---|---|---|
| fix_background | False | Fix any background parameters that are currently free in the model when performing the likelihood scan over extension. |
| spatial_model | RadialGaussian | Spatial model use for extension test. |
| sqrt_ts_threshold | None | Threshold on sqrt(TS_ext) that will be applied when update is True. If None then nothreshold is applied. |
| update | False | Update the source model with the best-fit spatial extension. |
| width | None | Parameter vector for scan over spatial extent. If none then the parameter vector will be set from width_min, width_max, and 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 steps for the spatial likelihood scan. |

### 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 | [r'\.fits$\|\.fit$\|\.xml$\|\.npy$', r'^srcmap_[0-9]+\.txt$'] | Stage files to the output directory that match at least one of the regular expressions in this list. This option only takes effect when usescratch is True. |
| savefits | True | Save intermediate FITS files. |
| scratchdir | /scratch | Path to the scratch directory. If usescratch is True then a temporary working directory will be created under this directory. |
| usescratch | False | Run analysis in a temporary working directory under scratchdir. |
| workdir | None | Path to the working directory. |
| workdir_regex | [r'\.fits$\|\.fit$\|\.xml$\|\.npy$'] | Stage files to the working directory that match at least one of the regular expressions in this list. This option only takes effect when usescratch 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 | Default | Description |
|---|---|---|
| `bexpmap` | 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. |
| `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 | |
| `srcmap` | None | |

### 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 1.10: 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 1.11: *model* Options

| Option | Default | Description |
|---|---|---|
| assoc_xmatch_colnames | ['3FGL_Name'] | Choose a set of association columns on which to cross-match catalogs. |
| catalogs | None | |
| diffuse | None | |
| extdir | None | Set a directory that will be searched for extended source FITS templates. Template files in this directory will take precendence 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 galactic IEM mapcube. |
| isodiff | None | Set the isotropic template. |
| 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 selection cut for inclusion of catalog sources in the model. Includes sources within a circle of this radius centered on the ROI. If this parameter is none then no selection is applied. This selection will be ORed with the src_roiwidth selection. |
| src_radius_roi | None | Half-width of src_roiwidth selection. This parameter can be used in lieu of src_roiwidth. |
| src_roiwidth | None | Width of square selection cut for inclusion of catalog sources in the model. Includes sources within a square region with side src_roiwidth centered on the ROI. If this parameter is none then no selection is applied. This selection will be ORed with the src_radius selection. |

#### optimizer

Listing 1.12: *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 1.13: *plotting* Options

| Option | Default | Description |
|---|---|---|
| `catalogs` | None | |
| `cmap` | ds9_b | Set the colormap for 2D plots. |
| `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 *Source Detection* page.

Listing 1.14: *residmap* Options

| Option | Default | Description |
|---|---|---|
| `loge_bounds` | None | Lower and upper energy bounds in log10(E/MeV). By default the calculation will be performed over the full analysis energy range. |
| `model` | None | Dictionary defining the properties of the test source. By default the test source will be a PointSource with an Index 2 power-law specturm. |

### roiopt

The options in *roiopt* control the default behavior of the `optimize` method. For more information about using this method see the *ROI Optimization and Fitting* page.

Listing 1.15: *roiopt* Options

| Option | Default | 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 1.16: *sed* Options

| Option | Default | Description |
|---|---|---|
| `bin_index` | 2.0 | Spectral index that will be use when fitting the energy distribution within an energy bin. |
| `cov_scale` | 3.0 | Scale factor that sets the strength of the prior on nuisance parameters when ‘‘fix_background‘‘=True. Setting this to None disables the prior. |
| `fix_background` | True | Fix background normalization parameters when fitting the source flux in each energy bin. If True background normalizations will be profiled with a prior on their value with strength set by `cov_scale`. |
| `ul_confidence` | 0.95 | Confidence level for upper limit calculation. |
| `use_local_index` | 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 `bin_index` will be used. |

### 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 1.17: *selection* Options

| Option | Default | 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)) |
| 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 precendence 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 Detection* page.

Listing 1.18: *sourcefind* Options

| Option | Default | Description |
|---|---|---|
| max_iter | 3 | Set the number of search iterations. |
| min_separation | 1.0 | Set the minimum separation in deg for sources added in each iteration. |
| model | None | Set the source model dictionary. By default the test source will be a PointSource with an Index 2 power-law specturm. |
| sources_per_iter | 3 | |
| sqrt_ts_threshold | 5.0 | Set the threshold on sqrt(TS). |
| tsmap_fitter | tsmap | Set the method for generating the TS map. |

#### tsmap

The options in *tsmap* control the default behavior of the `tsmap` method. For more information about using this method see the *Source Detection* page.

Listing 1.19: *tsmap* Options

| Option | Default | Description |
|---|---|---|
| loge_bounds | None | Lower and upper energy bounds in log10(E/MeV). By default the calculation will be performed over the full analysis energy range. |
| max_kernel_radius | 3.0 | |
| model | None | Dictionary defining the properties of the test source. |
| multithread | False | |

#### tscube

The options in *tscube* control the default behavior of the `tscube` method. For more information about using this method see the *Source Detection* page.

Listing 1.20: *tscube* Options

| Option | Default | 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. |
| max_iter | 30 | Maximum number of iterations for the Newtons method fitter. |
| model | None | Dictionary defining the properties of the test source. By default the test source will be a PointSource with an Index 2 power-law specturm. |
| 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. |

### 1.1.4 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 1.21: 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 *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. |

### 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 i |
| 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. |
| 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 ir |
| 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. |
| ts | float | Source test statistic. |
| loglike | float | Log-likelihood of the model evaluated at the best-fit normalization of the source. |
| dloglike_scan | ndarray | Delta Log-likelihood values for likelihood scan of source normalization. |
| eflux_scan | ndarray | Energy flux values for likelihood scan of source normalization. |
| flux_scan | ndarray | Flux values for likelihood scan of source normalization. |

Table 1.2 – continued from previous page

| Key | Type | Description |
|---|---|---|
| npred | float | Number of predicted counts from this source integrated over the analysis energy range. |
| params | dict | Dictionary of spectral parameters. |
| 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 *SED Analysis* for more information. |
| extension | dict | Output of extension analysis. See *Extension Fitting* for more information. |
| localize | dict | Output of localization analysis. See *Source Localization* for more information. |
| pivot_energy | float | Decorrelation energy in MeV. |
| flux | ndarray | Photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range |
| flux100 | ndarray | Photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV. |
| flux1000 | ndarray | Photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV. |
| flux10000 | ndarray | Photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV. |
| flux_ul95 | float | 95% CL upper limit on the photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range |
| flux100_ul95 | float | 95% CL upper limit on the photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV |
| flux1000_ul95 | float | 95% CL upper limit on the photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV. |
| flux10000_ul95 | float | 95% CL upper limit on the photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV. |
| eflux | ndarray | Energy flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range |
| eflux100 | ndarray | Energy flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV. |
| eflux1000 | ndarray | Energy flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV. |
| eflux10000 | ndarray | Energy flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV. |
| eflux_ul95 | float | 95% CL upper limit on the energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy |
| eflux100_ul95 | float | 95% CL upper limit on the energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 31 |
| eflux1000_ul95 | float | 95% CL upper limit on the energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 G |
| eflux10000_ul95 | float | 95% CL upper limit on the energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 |
| dfde | ndarray | Differential photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at the pivot energ |
| dfde100 | ndarray | Differential photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 100 MeV. |
| dfde1000 | ndarray | Differential photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 1 GeV. |
| dfde10000 | ndarray | Differential photon flux and uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 10 GeV. |
| dfde_index | ndarray | Logarithmic slope of the differential photon spectrum evaluated at the pivot energy. |
| dfde100_index | ndarray | Logarithmic slope of the differential photon spectrum evaluated at 100 MeV. |
| dfde1000_index | ndarray | Logarithmic slope of the differential photon spectrum evaluated evaluated at 1 GeV. |
| dfde10000_index | ndarray | Logarithmic slope of the differential photon spectrum evaluated at 10 GeV. |
| e2dfde | ndarray | E^2 times the differential photon flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) evaluated at the |
| e2dfde100 | ndarray | E^2 times the differential photon flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) evaluated at 100 |
| e2dfde1000 | ndarray | E^2 times the differential photon flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) evaluated at 1 G |
| e2dfde10000 | ndarray | E^2 times the differential photon flux and uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) evaluated at 10 G |

### 1.1.5 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)
2016-04-19 14:10:02 INFO     GTAnalysis.print_params():
 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 1.22: *fit* Output Dictionary

| Key | Type | Description |
|---|---|---|
| fit_quality | int | 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) |
| errors | ndarray | Vector of parameter errors (unscaled). |
| loglike | float | Post-fit log-likehood value. |
| correlation | ndarray | Correlation matrix between free parameters of the fit. |
| config | dict | Copy of input configuration to this method. |
| values | ndarray | Vector of best-fit parameter values (unscaled). |
| dloglike | float | Improvement in log-likehood value. |
| fit_status | int | Optimizer return code (0 = ok). |
| covariance | ndarray | Covariance matrix between free parameters of the fit. |
| edm | float | 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 charateristics 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 1.23: *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-likehood 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.

### 1.1.6 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 three types of pre-defined spatial models which can be defined by setting the `SpatialModel` property: PointSource (the default), RadialDisk, and RadialGaussian. 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: '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 }
```

#### 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.

### 1.1.7 Advanced Analysis Methods

fermipy provides several advanced analysis methods that are documented in the following pages:

#### 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 `fix_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 (`fix_background` = True). Setting `fix_background` to False 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.

The default configuration of *sed()* is defined with the *sed* section of the configuration file:

Listing 1.24: *sed* Options

| Option | Default | Description |
|---|---|---|
| bin_index | 2.0 | Spectral index that will be use when fitting the energy distribution within an energy bin. |
| cov_scale | 3.0 | Scale factor that sets the strength of the prior on nuisance parameters when ‘‘fix_background‘‘=True. Setting this to None disables the prior. |
| fix_background | True | Fix background normalization parameters when fitting the source flux in each energy bin. If True background normalizations will be profiled with a prior on their value with strength set by cov_scale. |
| ul_confidence | 0.95 | Confidence level for upper limit calculation. |
| use_local_index | 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 bin_index will be used. |

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',fix_background=False,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()*:

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_DFDE_E_MIN | Differential flux of the reference model evaluated at the lower bin edge ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) |
| SED | REF_DFDE_E_MAX | Differential flux of the reference model evaluated at the upper bin edge ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) |
| SED | REF_FLUX | Flux of the reference model in each bin ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| SED | REF_EFLUX | Energy flux of the reference model in each bin ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| SED | REF_DFDE | Differential flux of the reference model evaluated at the bin center ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) |
| 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 ref 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 | DFDE | Central value of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$). |
| MODEL_FLUX | DFDE_LO | Lower 1-sigma bound of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$). |
| MODEL_FLUX | DFDE_HI | Upper 1-sigma bound of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$). |
| MODEL_FLUX | DFDE_ERR | Symmetric error of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$). |
| MODEL_FLUX | DFDE_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 |
|---|---|---|
| logemin | ndarray | Lower edges of SED energy bins (log10(E/MeV)). |
| logemax | ndarray | Upper edges of SED energy bins (log10(E/MeV)). |
| logectr | ndarray | Centers of SED energy bins (log10(E/MeV)). |
| emin | ndarray | Lower edges of SED energy bins (MeV). |
| emax | ndarray | Upper edges of SED energy bins (MeV). |
| ectr | ndarray | Centers of SED energy bins (MeV). |

Table 1.3 -

| Key | Type | Description |
|---|---|---|
| ref_flux | ndarray | Flux of the reference model in each bin ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| ref_eflux | ndarray | Energy flux of the reference model in each bin ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| ref_dfde | ndarray | Differential flux of the reference model evaluated at the bin center ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) |
| ref_dfde_emin | ndarray | Differential flux of the reference model evaluated at the lower bin edge ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) |
| ref_dfde_emax | ndarray | Differential flux of the reference model evaluated at the upper bin edge ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) |
| ref_e2dfde | ndarray | E^2 x the differential flux of the reference model evaluated at the bin center ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$ |
| 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 ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| eflux | ndarray | Energy flux in each bin ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| dfde | ndarray | Differential flux in each bin ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$). |
| e2dfde | ndarray | E^2 x the differential flux in each bin ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$). |
| dfde_err | ndarray | 1-sigma error on dfde evaluated from likelihood curvature. |
| dfde_err_lo | ndarray | Lower 1-sigma error on dfde evaluated from the profile likelihood (MINOS errors). |
| dfde_err_hi | ndarray | Upper 1-sigma error on dfde evaluated from the profile likelihood (MINOS errors). |
| dfde_ul95 | ndarray | 95% CL upper limit on dfde evaluated from the profile likelihood (MINOS errors). |
| dfde_ul | ndarray | Upper limit on dfde evaluated from the profile likelihood using a CL = ul_confidence. |
| e2dfde_err | ndarray | 1-sigma error on e2dfde evaluated from likelihood curvature. |
| e2dfde_err_lo | ndarray | Lower 1-sigma error on e2dfde evaluated from the profile likelihood (MINOS errors). |
| e2dfde_err_hi | ndarray | Upper 1-sigma error on e2dfde evaluated from the profile likelihood (MINOS errors). |
| e2dfde_ul95 | ndarray | 95% CL upper limit on e2dfde evaluated from the profile likelihood (MINOS errors). |
| e2dfde_ul | ndarray | Upper limit on e2dfde evaluated from the profile likelihood using a CL = 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. |
| lnlprofile | dict | Likelihood scan for each energy bin. |
| norm_scan | ndarray | Array of NxM normalization values for the profile likelihood scan in N energy bins and M sc |
| dloglike_scan | ndarray | Array of NxM delta-loglikelihood values for the profile likelihood scan in N energy bins and |
| loglike_scan | ndarray | Array of NxM loglikelihood values for the profile likelihood scan in N energy bins and M sca |
| params | dict | Dictionary of best-fit spectral parameters with 1-sigma uncertainties. |
| 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 para |
| config | dict | Copy of input configuration to this method. |

### 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 loge_bins parameter.

**Parameters**

- **name** (*str*) – Source name.

- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **loge_bins** (ndarray) – Sequence of energies in log10(E/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 underyling analysis instance.
- **bin_index** (*float*) – Spectral index that will be use when fitting the energy distribution within an energy bin.
- **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.
- **fix_background** (*bool*) – Fix background components when fitting the flux normalization in each energy bin. If fix_background=False then all background parameters that are currently free in the fit will be profiled. By default fix_background=True.
- **ul_confidence** (*float*) – Set the confidence level that will be used for the calculation of flux upper limits in each energy bin.
- **cov_scale** (*float*) – Scaling factor that will be applied when setting the gaussian prior on the normalization of free background sources. If this parameter is None then no gaussian prior will be applied.
- **write_fits** (*bool*) – Write a FITS file containing the SED analysis results.
- **write_npy** (*bool*) – Write a numpy file with the contents of the output dictionary.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

**Returns** **sed** – Dictionary containing output of the SED analysis. This dictionary is also saved to the 'sed' dictionary of the *Source* instance.

**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 evaluated by a likelihood profile scan over the source width. Currently this method supports two models for extension: a 2D Gaussian (*GaussianSource*) or a 2D disk (*DiskSource*).

The default configuration of *extension()* is defined in the *extension* section of the configuration file:

Listing 1.26: *extension* Options

| Option | Default | Description |
|---|---|---|
| fix_background | False | Fix any background parameters that are currently free in the model when performing the likelihood scan over extension. |
| spatial_model | RadialGaussian | Spatial model use for extension test. |
| sqrt_ts_threshold | None | Threshold on sqrt(TS_ext) that will be applied when update is True. If None then nothreshold is applied. |
| update | False | Update the source model with the best-fit spatial extension. |
| width | None | Parameter vector for scan over spatial extent. If none then the parameter vector will be set from width_min, width_max, and 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 steps for the spatial likelihood scan. |

At runtime the default settings for the extension analysis can be overriden 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='DiskSource')
```

By default the extension method will profile over any background parameters that were free when the method was executed. One can optionally fix all background parameters with the *fix_background* parameter:

```
# Free a nearby source that maybe be partially degenerate with the
# source of interest
gta.free_norm('sourceB')

# Normalization of SourceB will be refit when testing the extension
# of sourceA
gta.extension('sourceA')

# Fix all background parameters when testing the extension
# of sourceA
gta.extension('sourceA',fix_background=True)
```

The results of the extension analysis are written to a dictionary which is the return value of the extension method. This dictionary is also written to the *extension* dictionary of the corresponding source and will also be saved in the output file generated by *write_roi()*.

```
ext = gta.extension('sourceA')

ext = gta.roi['sourceA']
```

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

Listing 1.27: *extension* Output Dictionary

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

#### 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. Any background parameters that are free will also be simultaneously profiled in the likelihood scan.

**Parameters**

- **name** (*str*) – Source name.
- **spatial_model** (*str*) – Spatial model that will be used to test the source extension. The spatial scale parameter of the respective model will be set such that the 68% containment radius of the model is equal to the width parameter. The following spatial models are supported:
  - RadialDisk : Azimuthally symmetric 2D disk.
  - RadialGaussian : Azimuthally symmetric 2D gaussian.
- **width_min** (*float*) – Minimum value in degrees for the spatial extension scan.
- **width_max** (*float*) – Maximum value in degrees for the spatial extension scan.
- **width_nstep** (*int*) – Number of scan points between width_min and width_max. Scan points will be spaced evenly on a logarithmic scale between log(width_min) and log(width_max).
- **width** (*array-like*) – Sequence of values in degrees for the spatial extension scan. If this argument is None then the scan points will be determined from width_min/width_max/width_nstep.
- **fix_background** (*bool*) – Fix all background sources when performing the extension fit.
- **update** (*bool*) – Update this source with the best-fit model for spatial extension if TS_ext > tsext_threshold.
- **sqrt_ts_threshold** (*float*) – Threshold on sqrt(TS_ext) that will be applied when update is true. If None then no threshold will be applied.
- **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

### Source Detection

fermipy provides several methods for source detection that can be used to look for unmodeled sources as well as evaluate the fit quality of the model. These methods are

- *TS Map*: *tsmap()* generates a test statistic (TS) map for a new source centered at each spatial bin in the ROI.
- *TS Cube*: *tscube()* generates 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*: `residmap()` generates a residual map by evaluating the difference between smoothed data and model maps (residual) at each spatial bin in the ROI.
- *Source Finding*: `find_sources()` is an iterative source-finding algorithim that adds new sources to the ROI by looking for peaks in the TS map.

Additional information about using each of these methods is provided in the sections below.

### TS Map

`tsmap()` performs a likelihood ratio test for an additional source at the center of each spatial bin of the ROI. The methodology is similar to that of the `gttsmap` ST application but with a simplified source fitting implementation that significantly speeds up the calculation. For each spatial bin the method calculates the maximum likelihood test statistic given by

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

where the summation index $k$ runs over both spatial and energy bins, $\mu$ is the test source normalization parameter, and $\theta$ represents the parameters of the background model. Unlike `gttsmap`, the likelihood fitting implementation used by `tsmap()` only fits for the normalization of the test source and does not re-fit parameters of the background model. The properties of the test source (spectrum and spatial morphology) are controlled with the `model` dictionary argument. The syntax for defining the test source properties follows the same conventions as `add_source()` as illustrated in the following examples.

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

If running interactively, the `multithread` option can be enabled to split the calculation across all available cores. However it is not recommended to use this option when running in a cluster environment.

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

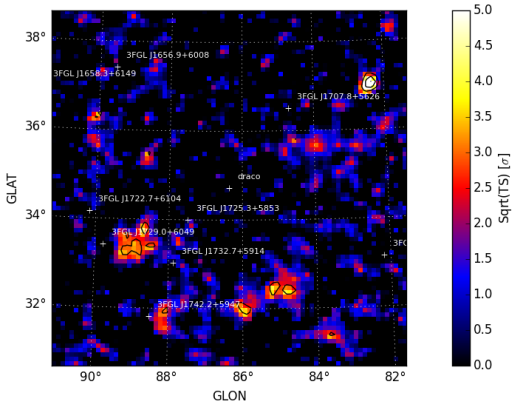
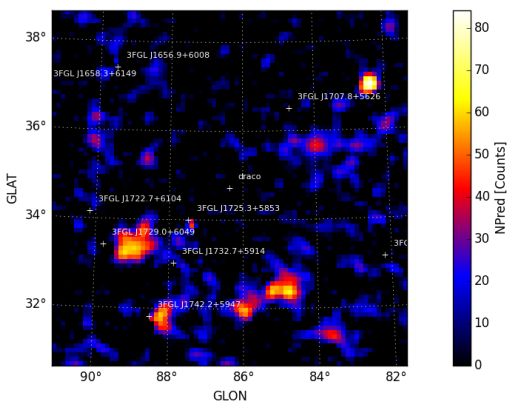
`tsmap()` returns a `maps` dictionary containing `Map` representations of the TS and NPred of the best-fit test source at each position.

```
>>> model = {'Index' : 2.0, 'SpatialModel' : 'PointSource'}
>>> maps = gta.tsmap('fit1',model=model)
>>> print(maps.keys())
[u'file', u'name', u'sqrt_ts', u'ts', u'src_dict', u'npred', u'amplitude']
```

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

| Key | Type | Description |
|---|---|---|
| amplitude | `Map` | Best-fit test source amplitude expressed in terms of the spectral prefactor. |
| npred | `Map` | Best-fit test source amplitude expressed in terms of the total model counts (Npred). |
| ts | `Map` | Test source TS (twice the logLike difference between null and alternate hypothese). |
| sqrt_ts | `Map` | 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. |

Maps are also written as both FITS and rendered image files to the analysis working directory. All output files are prepended with the `prefix` argument. Sample images for `sqrt_ts` and `npred` generated by *`tsmap()`* are shown below. A colormap threshold for the `sqrt_ts` image is applied at 5 sigma with iscontours at 2 sigma intervals (3,5,7,9, ...) indicating values above this threshold.

| Sqrt(TS) | NPred |
|---|---|
|  |  |



**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 (FITS and rendered images).
- **`model`** (*`dict`*) – Dictionary defining the properties of the test source.
- **`exclude`** (*`str or list of str`*) – Source or sources that will be removed from the model when computing the TS map.
- **`loge_bounds`** (*`list`*) – 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.
- **`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. The default

value is 3 degrees.

- **make_plots** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write a FITS file.
- **write_npy** (*bool*) – Write a numpy file.

**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 *tsmap()* fits for positive excesses with respect to the current 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

$$\sigma_{ij}^2 = 2\mathrm{sgn}(\tilde{n}_{ij} - \tilde{m}_{ij}) \left(\ln L_P(\tilde{n}_{ij}, \tilde{n}_{ij}) - \ln L_P(\tilde{n}_{ij}, \tilde{m}_{ij})\right)$$
$$\text{with} \quad \tilde{m}_{ij} = (m * k)_{ij} \quad \tilde{n}_{ij} = (n * k)_{ij} \quad \ln L_P(n, m) = n \ln(m) - m$$

where $n$ and $m$ are the data and model maps and $k$ is the convolution kernel. The spatial and spectral properties of the convolution kernel are defined with the `model` argument. All source models are supported as well as a gaussian kernel (defined by setting *SpatialModel* to *Gaussian*). The following examples illustrate how to run the method with different spatial kernels.

```
# 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])

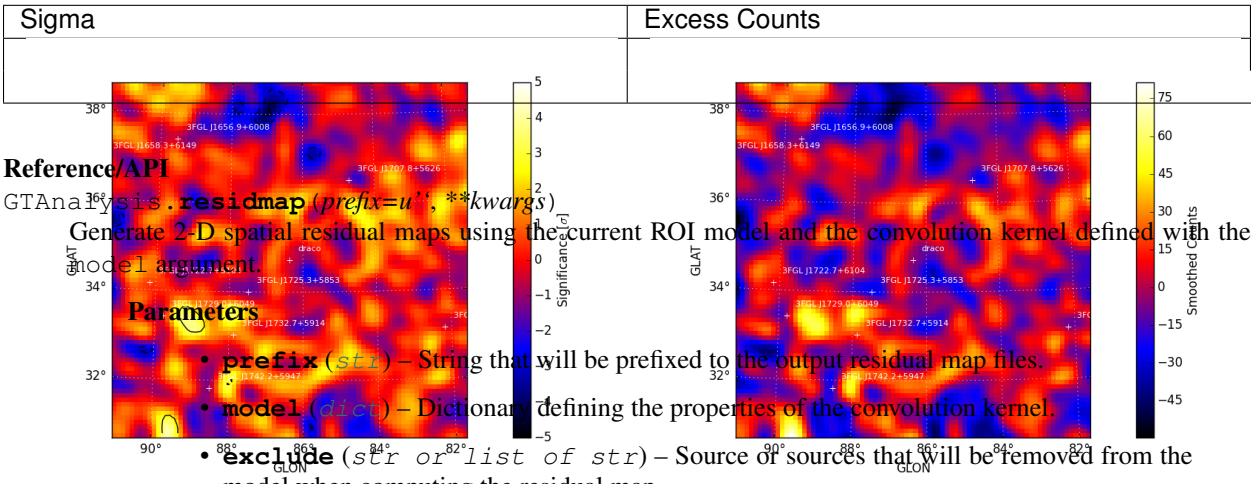
# Generate residual maps for a power-law point source with Index=1.5, 2.0, and 2.5
model={'SpatialModel' : 'PointSource'}
maps = []
for index in [1.5,2.0,2.5]:
    model['Index'] = index
    maps += [gta.residmap('fit1',model=model)]
```

*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 | `Map` | Residual significance in sigma. |
| excess | `Map` | Residual amplitude in counts. |
| data | `Map` | Smoothed counts map. |
| model | `Map` | 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. |

Maps are also written as both FITS and rendered image files to the analysis working directory. All output files are prepended with the `prefix` argument. Sample images for `sigma` and `excess` generated by *residmap()* are

shown below. A colormap threshold for the `sigma` image is applied at both -5 and 5 sigma with iscontours at 2 sigma intervals (-5, -3, 3, 5, 7, 9, ...) indicating values above and below this threshold.




**Reference/API**

`GTAnalysis.`**`residmap`**(*prefix=u''*, ***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.
- **`model`** (*`dict`*) – Dictionary defining the properties of the convolution kernel.
- **`exclude`** (*`str or list of str`*) – Source or sources that will be removed from the model when computing the residual map.
- **`loge_bounds`** (*`list`*) – 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`** (*`bool`*) – Write image files.
- **`write_fits`** (*`bool`*) – Write FITS files.

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

**Return type** dict

### TS Cube

**Warning:** This method is experimental and is not supported by the current public release of the Fermi STs.

`GTAnalysis.`**`tscube`**(*prefix=u''*, ***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

### Source Finding

**Warning:** This method is experimental and still under development. API changes are likely to occur in future releases.

*find_sources()* is an iterative source-finding algorithm that uses peak detection on the TS map to find the locations of new sources.

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

An iterative source-finding algorithm.

**Parameters**

- **model** (*dict*) – Dictionary defining the properties of the test source. This is the model that will be used for generating TS maps.
- **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.
- **min_separation** (*float*) – Minimum separation in degrees of 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.
- **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.
- **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.
- **tsmap_fitter** (*str*) – Set the method used internally for generating TS maps. Valid options:
  - tsmap
  - tscube
- **tsmap** (*dict*) – Keyword arguments dictionary for tsmap method.
- **tscube** (*dict*) – Keyword arguments dictionary for tscube method.

**Returns**

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

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

### Source Localization

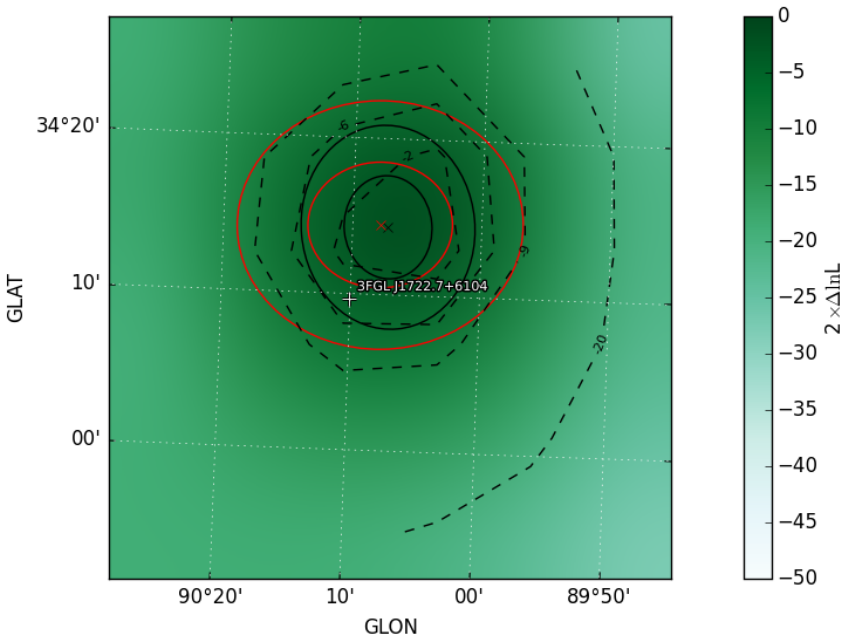
The `localize()` method can be used to spatially localize a source. Localization is performed by scanning the 2D likelihood surface in a local patch around the nominal source position. The current implementation of the localization analysis proceeds in two steps:

- **TS Map Scan**: Obtain a rough estimate of the source position by generating a fast TS Map of the region using the `tsmap` 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 overriden 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 profile over any background parameters that were free when the method was executed. One can fix all background parameters with the *fix_background* parameter:

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

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

Listing 1.28: *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*) – Maximum offset in RA/DEC in deg from the nominal source position that will be used to define the boundaries of the TS map search region.
- **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.
- **fix_background** (*bool*) – Fix background parameters when fitting the source position.

- **update** (*bool*) – Update the model for this source with the best-fit position. If newname=None this will overwrite the existing source map of this source with one corresponding to its new location.
- **newname** (*str*) – Name that will be assigned to the relocalized source when update=True. If newname is None then the existing source name will be used.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

**Returns** **localize** – Dictionary containing results of the localization analysis. This dictionary is also saved to the dictionary of this source in 'localize'.

**Return type** dict

### 1.1.8 fermipy package

#### Submodules

#### 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.**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*)

**write_config**(*outfile*)
Write the configuration dictionary to an output file.

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

fermipy.config.**create_default_config**(*defaults*)
Create a configuration dictionary from a defaults dictionary. The defaults dictionary defines valid configuration keys with default values and docstrings. Each dictionary element should be a tuple or list containing (default value,docstring,type).

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

### 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.SourceFinder*

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

**__getattribute__**
x.__getattribute__('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=False*, *init_source=True*, *save_source_maps=True*, ***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.

**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**()

**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 overriden 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.

**defaults = {u'sourcefind': {u'max_iter': (3, u'Set the number of search iterations.', <type 'int'>), u'min_separation': (**

**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*) – Delete 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*, *square=False*, *exclude_diffuse=True*)

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

**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. Any background parameters that are free will also be simultaneously profiled in the likelihood scan.

**Parameters**

- **name** (*str*) – Source name.
- **spatial_model** (*str*) – Spatial model that will be used to test the source extension. The spatial scale parameter of the respective model will be set such that the 68% containment radius of the model is equal to the width parameter. The following spatial models are supported:
  - RadialDisk : Azimuthally symmetric 2D disk.
  - RadialGaussian : Azimuthally symmetric 2D gaussian.
- **width_min** (*float*) – Minimum value in degrees for the spatial extension scan.
- **width_max** (*float*) – Maximum value in degrees for the spatial extension scan.
- **width_nstep** (*int*) – Number of scan points between width_min and width_max. Scan points will be spaced evenly on a logarithmic scale between log(width_min) and log(width_max).

- **width** (*array-like*) – Sequence of values in degrees for the spatial extension scan. If this argument is None then the scan points will be determined from width_min/width_max/width_nstep.
- **fix_background** (*bool*) – Fix all background sources when performing the extension fit.
- **update** (*bool*) – Update this source with the best-fit model for spatial extension if TS_ext > `tsext_threshold`.
- **sqrt_ts_threshold** (*float*) – Threshold on sqrt(TS_ext) that will be applied when `update` is true. If None then no threshold will be applied.
- **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

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

An iterative source-finding algorithm.

**Parameters**

- **model** (*dict*) – Dictionary defining the properties of the test source. This is the model that will be used for generating TS maps.
- **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.
- **min_separation** (*float*) – Minimum separation in degrees of 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.
- **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.
- **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.
- **tsmap_fitter** (*str*) – Set the method used internally for generating TS maps. Valid options:
  - tsmap
  - tscube
- **tsmap** (*dict*) – Keyword arguments dictionary for tsmap 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 charateristics 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*, *square=False*, *exclude_diffuse=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.
- **square** (*bool*) – Switch between applying a circular or square (ROI-like) selection on the maximum projected distance from the ROI center.
- **exclude_diffuse** (*bool*) – Exclude diffuse sources.

**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_dfde** (*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 (log10(E/MeV)).
- **dfde** (ndarray) – Array of differential flux values (cm^{-2} s^{-1} 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*, *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

**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*) – Maximum offset in RA/DEC in deg from the nominal source position that will be used to define the boundaries of the TS map search region.

- **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.
- **fix_background** (*bool*) – Fix background parameters when fitting the source position.
- **update** (*bool*) – Update the model for this source with the best-fit position. If newname=None this will overwrite the existing source map of this source with one corresponding to its new location.
- **newname** (*str*) – Name that will be assigned to the relocalized source when update=True. If newname is None then the existing source name will be used.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

**Returns** **localize** – Dictionary containing results of the localization analysis. This dictionary is also saved to the dictionary of this source in 'localize'.

**Return type** dict

**log_energies**
Return the energy bin edges in log10(E/MeV).

**loge_bounds**
Current analysis energy bounds in log10(E/MeV).

**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.

**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** **lnlprofile** – 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*)
Delete and reload a source in the model. This will refresh the spatial model of this source to the one defined in the XML model.

**remove_prior**(*name*, *parName*)

**remove_priors**()
Clear all priors.

**residmap**(*prefix=u''*, ***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.
- **model** (*dict*) – Dictionary defining the properties of the convolution kernel.
- **exclude** (*str or list of str*) – Source or sources that will be removed from the model when computing the residual map.
- **loge_bounds** (*list*) – 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** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write FITS files.

**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*)

**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 `loge_bins` parameter.

**Parameters**

- **name** (*str*) – Source name.
- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **loge_bins** (*ndarray*) – Sequence of energies in log10(E/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 underyling analysis instance.
- **bin_index** (*float*) – Spectral index that will be use when fitting the energy distribution within an energy bin.

- **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.
- **fix_background** (*bool*) – Fix background components when fitting the flux normalization in each energy bin. If fix_background=False then all background parameters that are currently free in the fit will be profiled. By default fix_background=True.
- **ul_confidence** (*float*) – Set the confidence level that will be used for the calculation of flux upper limits in each energy bin.
- **cov_scale** (*float*) – Scaling factor that will be applied when setting the gaussian prior on the normalization of free background sources. If this parameter is None then no gaussian prior will be applied.
- **write_fits** (*bool*) – Write a FITS file containing the SED analysis results.
- **write_npy** (*bool*) – Write a numpy file with the contents of the output dictionary.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

**Returns** **sed** – Dictionary containing output of the SED analysis. This dictionary is also saved to the 'sed' dictionary of the *Source* instance.

**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 log10(E/MeV).
- **logemax** (*float*) – Upper energy bound in log10(E/MeV).

**Returns** **eminmax** – Minimum and maximum energy in log10(E/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_dfde**(*name*, *dfde*, *update_source=True*)
Set the differential flux distribution of a source with the FileFunction spectral type.

**Parameters**

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

**set_source_spectrum**(*name*, *spectrum_type=u'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*)
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.

**tscube**(*prefix=u''*, ***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

**tsmap**(*prefix=u''*, ***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 (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **exclude** (*str or list of str*) – Source or sources that will be removed from the model when computing the TS map.
- **loge_bounds** (*list*) – 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.
- **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. The default value is 3 degrees.
- **make_plots** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write a FITS file.
- **write_npy** (*bool*) – Write a numpy file.

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

**Return type** dict

**unzero_source**(*name*)

**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_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=u'npy'*, ***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*)

### 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*)

**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.**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=None*)
Bases: object

Base class for source objects. This class is a container for both spectral and spatial parameters as well as other source properties such as TS, Npred, and location within the ROI.

**add_name**(*name*)

**assoc**

**check_cuts**(*cuts*)

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

**data**

**get_catalog_dict**()

**get_norm**()

**items**()

**name**

**names**

**params**

**set_name**(*name*, *names=None*)

**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']
```

•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
```

**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.

**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*)
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_table**()

**defaults = {'logfile': (None, u'', <type 'str'>), u'catalogs': (None, u'', <type 'list'>), u'src_roiwidth': (None, u'Width of**

**delete_sources**(*srcs*)

**diffuse_sources**

**get_nearby_sources**(*name*, *dist*, *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*, *square=False*, *exclude_diffuse=False*, *coordsys=u'CEL'*)
Retrieve list of sources satisfying the given selections.

**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=u'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 x R where R = 2 x 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_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 = [u'Source_Name', u'ASSOC', u'ASSOC1', u'ASSOC2', u'ASSOC_GAM', u'1FHL_Name', u'2FGL**

**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*)
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.

**data**

**diffuse**

**extended**

**load_from_catalog** ()
Load spectral parameters from catalog values.

**radec**

**separation** (*src*)

**set_position** (*skydir*)
Set the position of the source.

**Parameters** **skydir** (SkyCoord) –

**set_roi_direction** (*roidir*)

**set_roi_projection**(*proj*)

**set_spatial_model**(*spatial_model*, *spatial_width=None*)

**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=u'CEL'*)

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

fermipy.roi_model.**get_params_dict**(*pars_dict*)

fermipy.roi_model.**get_skydir_distance_mask**(*src_skydir*, *skydir*, *dist*, *min_dist=None*, *square=False*, *coordsys=u'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 x R where R = 2 x 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.**resolve_file_path**(*path*, ***kwargs*)

### fermipy.utils module

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

fermipy.utils.**cl_to_dlnl**(*cl*)
Compute the delta-log-likehood corresponding to an upper limit of the given confidence level.

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 gaussian given by:

g(r) = 1/(2*pi*s^2) 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_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*)

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

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.**extract_mapcube_region**(*infile*, *skydir*, *outfile*, *maphdu=0*)
Extract a region out of an all-sky mapcube file.

**Parameters**

- **infile** (*str*) – Path to mapcube file.
- **skydir** (SkyCoord) –

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 x0 and xb.

**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 xb.
- **xb** (*float*) – Upper or lower bound for the root search. If a root is not found in the interval [x0,xb]/[xb,x0] this value will be increased/decreased until a change in sign is found.

fermipy.utils.**fit_parabola**(*z*, *ix*, *iy*, *dpix=2*, *zmin=None*)

fermipy.utils.**fits_recarray_to_dict**(*table*)
Convert a FITS recarray 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*, *delta_tol=0.05*)
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.
- **delta_tol** (*float*) – Mask sequential points in the profile that have a log-likelihood differences smaller than this threshold.

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

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=u'_'*)

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*, *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*, *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**(*sigma*, *npix=501*, *cdelt=0.01*, *xpix=0.0*, *ypix=0.0*)
Make kernel for a 2D disk.

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

fermipy.utils.**make_gaussian_kernel**(*sigma*, *npix=501*, *cdelt=0.01*, *xpix=0.0*, *ypix=0.0*)
Make kernel for a 2D gaussian.

**Parameters** **sigma** (*float*) – 68% containment radius in degrees.

fermipy.utils.**make_pixel_offset**(*npix*, *xpix=0.0*, *ypix=0.0*)
Make a 2D array with the distance of each pixel from a reference direction in pixel coordinates. Pixel coordinates are defined such that (0,0) is located at the center of the coordinate grid.

fermipy.utils.**make_psf_kernel**(*psf*, *npix*, *cdelt*, *xpix*, *ypix*, *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.**mkdir**(*dir*)

fermipy.utils.**parabola**((*x*, *y*), *amplitude*, *x0*, *y0*, *sx*, *sy*, *theta*)

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_path**(*path*, *workdir=None*)

fermipy.utils.**scale_parameter**(*p*)

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

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 dupming 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 represntation, (i.e. '123' -> 123)

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

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.**write_fits_image**(*data*, *wcs*, *outfile*)

fermipy.utils.**write_hpx_image**(*data*, *hpx*, *outfile*, *extname=u'SKYMAP'*)

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

fermipy.utils.**xmlpath_to_path**(*path*)

fermipy.utils.**xyz_to_lonlat**(**args*)

### 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> 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 loge_bins parameter.

**Parameters**

- **name** (*str*) – Source name.
- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **loge_bins** (ndarray) – Sequence of energies in log10(E/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 underyling analysis instance.
- **bin_index** (*float*) – Spectral index that will be use when fitting the energy distribution within an energy bin.
- **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.
- **fix_background** (*bool*) – Fix background components when fitting the flux normalization in each energy bin. If fix_background=False then all background parameters that are currently free in the fit will be profiled. By default fix_background=True.
- **ul_confidence** (*float*) – Set the confidence level that will be used for the calculation of flux upper limits in each energy bin.
- **cov_scale** (*float*) – Scaling factor that will be applied when setting the gaussian prior on the normalization of free background sources. If this parameter is None then no gaussian prior will be applied.
- **write_fits** (*bool*) – Write a FITS file containing the SED analysis results.
- **write_npy** (*bool*) – Write a numpy file with the contents of the output dictionary.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

**Returns** **sed** – Dictionary containing output of the SED analysis. This dictionary is also saved to the 'sed' dictionary of the *Source* instance.

**Return type** dict

### fermipy.sourcefind module

**class** fermipy.sourcefind.**SourceFinder**
Bases: object

Mixin class which provides source-finding functionality to *GTAnalysis*.

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

An iterative source-finding algorithm.

**Parameters**

- **model** (*dict*) – Dictionary defining the properties of the test source. This is the model that will be used for generating TS maps.
- **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.
- **min_separation** (*float*) – Minimum separation in degrees of 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.
- **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.
- **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.
- **tsmap_fitter** (*str*) – Set the method used internally for generating TS maps. Valid options:
  - tsmap
  - tscube
- **tsmap** (*dict*) – Keyword arguments dictionary for tsmap 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.
- **dtheta_max** (*float*) – Maximum offset in RA/DEC in deg from the nominal source position that will be used to define the boundaries of the TS map search region.
- **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.
- **fix_background** (*bool*) – Fix background parameters when fitting the source position.

- **update** (*bool*) – Update the model for this source with the best-fit position. If newname=None this will overwrite the existing source map of this source with one corresponding to its new location.
- **newname** (*str*) – Name that will be assigned to the relocalized source when update=True. If newname is None then the existing source name will be used.
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

**Returns** **localize** – Dictionary containing results of the localization analysis. This dictionary is also saved to the dictionary of this source in 'localize'.

**Return type** dict

fermipy.sourcefind.**estimate_pos_and_err_parabolic**(*tsvals*)

**Solve for the position and uncertainty of source in one dimension** assuming that you are near the maximum and the errors are parabolic

**Parameters** **tsvals** (ndarray) – The TS values at the maximum TS, and for each pixel on either side

**Returns**

**Return type** The position and uncertainty of the source, in pixel units w.r.t. the center of the maximum pixel

fermipy.sourcefind.**find_peaks**(*input_map*, *threshold*, *min_separation=0.5*)

Find peaks in a 2-D map object that have amplitude larger than `threshold` and lie a distance at least `min_separation` from another peak of larger amplitude. The implementation of this method uses `maximum_filter`.

**Parameters**

- **input_map** (Map) –
- **threshold** (*float*) –
- **min_separation** (*float*) – Radius of region size in degrees. Sets the minimum allowable separation between peaks.

**Returns** **peaks** – List of dictionaries containing the location and amplitude of each peak.

**Return type** list

fermipy.sourcefind.**fit_error_ellipse**(*tsmap*, *xy=None*, *dpix=3*)

Fit a positional uncertainty ellipse from a TS map.

**Parameters**

- **tsmap** (*Map*) –
- **xy** (*tuple*) –

fermipy.sourcefind.**refine_peak**(*tsmap*, *pix*)

Solve for the position and uncertainty of source assuming that you are near the maximum and the errors are parabolic

**Parameters** **tsmap** (ndarray) – Array with the TS data.

**Returns**

**Return type** The position and uncertainty of the source, in pixel units w.r.t. the center of the maximum pixel

### 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

hpx_in : HEALPix input data sum_ebins : bool, sum energy bins over energy bins before reprojecting normalize : True -> perserve integral by splitting HEALPix values between bins

returns (WCS object, np.ndarray() with reprojected data)

**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*, *extname=u'SKYMAP'*, *ebounds=u'EBOUNDS'*)
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

**hpx**

**make_wcs_from_hpx**(*sum_ebins=False*, *proj=u'CAR'*, *oversample=2*, *normalize=True*)
Make a WCS object and convert HEALPix data into WCS projection

sum_ebins : bool, sum energy bins over energy bins before reprojecting proj : WCS-projection oversample : Oversampling factor for WCS map normalize : True -> perserve integral by splitting HEALPix values between bins

returns (WCS object, np.ndarray() with reprojected data)

NOTE: this re-calculates the mapping, if you have already calculated the

mapping it is much faster to use convert_to_cached_wcs() instead

**class** fermipy.skymap.**Map**(*counts*, *wcs*)
Bases: *fermipy.skymap.Map_Base*

Representation of a 2D or 3D counts map using WCS.

**static create**(*skydir*, *cdelt*, *npix*, *coordsys=u'CEL'*, *projection=u'AIT'*)

**static create_from_fits**(*fitsfile*, ***kwargs*)

**static create_from_hdu**(*hdu*, *wcs*)

**create_image_hdu**(*name=None*)

**create_primary_hdu**()

**get_map_values**(*lons*, *lats*)
Return the indices in the flat array corresponding to a set of coordinates

**Parameters**

- **lons** (*array-like*) – 'Longitudes' (RA or GLON)
- **lats** (*array-like*) – 'Latitidues' (DEC or GLAT)

**Returns**

- **vals** (*numpy.ndarray((n))*)
- *Values of pixels in the flattened map, np.nan used to flag coords outside of map*

**get_pixel_indices**(*lons*, *lats*)
Return the indices in the flat array corresponding to a set of coordinates

**Parameters**

- **lons** (*array-like*) – 'Longitudes' (RA or GLON)
- **lats** (*array-like*) – 'Latitidues' (DEC or GLAT)

**Returns**

- **idxs** (*numpy.ndarray((n),'i')*)
- *Indices of pixels in the flattened map, -1 used to flag coords outside of map*

**get_pixel_skydirs**()

**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 the pixel xy coordinates from the pixel index

if colwise is True (False) this uses columnwise (rowwise) indexing

**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.

**xy_pix_to_ipix**(*xypix*, *colwise=False*)
Return the pixel index from the pixel xy coordinates

if colwise is True (False) this uses columnwise (rowwise) indexing

**class** fermipy.skymap.**Map_Base**(*counts*)
Bases: object

Abstract representation of a 2D or 3D counts map.

**counts**

**get_pixel_indices**(*lats*, *lons*)

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 tabluated 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*, *specData*, *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=u'EFLUX'*, *hdu_scan=u'SCANDATA'*, *hdu_energies=u'EBOUNDS'*, *irow=None*)
Create a CastroData object from a 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 ( ph cm^-2 s^-1 ) EFLUX: Energy Flux of the test source ( MeV cm^-2 s^-1 ) NPRED: Number of predicted photons (Not implemented) DFDE : Differential flux of the test source ( ph cm^-2 s^-1 MeV^-1 ) EDFDE: Differential energy flux of the test source ( MeV cm^-2 s^-1 MeV^- ) (Not Implemented)
- **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**

**Return type** A '~fermipy.castro.CastroData' object

**static create_from_sedfile**(*fitsfile*, *norm_type=u'EFLUX'*)
Create a CastroData object from an SED fits file

**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 ( ph cm^-2 s^-1 ) EFLUX: Energy Flux of the test source ( MeV cm^-2 s^-1 ) NPRED: Number of predicted photons (Not implemented) DFDE : Differential flux of the test source ( ph cm^-2 s^-1 MeV^-1 ) EDFDE: Differential energy flux of the test source ( MeV cm^-2 s^-1 MeV^- ) (Not Implemented)

**Returns**

A '~fermipy.castro.CastroData' object

**static create_from_stack**(*shape*, *components*, *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**

**Return type** A '~fermipy.castro.CastroData' object

**static create_from_tables**(*norm_type=u'EFLUX'*, *tab_s=u'SCANDATA'*, *tab_e=u'EBOUNDS'*)
Create a CastroData object from two tables

**Parameters**

- **norm_type** (str) – Type of normalization to use, options are: NORM : Normalization w.r.t. to test source FLUX : Flux of the test source ( ph cm^-2 s^-1 ) EFLUX: Energy Flux of the test source ( MeV cm^-2 s^-1 ) NPRED: Number of predicted photons (Not implemented) DFDE : Differential flux of the test source ( ph cm^-2 s^-1 MeV^-1 ) EDFDE: Differential energy flux of the test source ( MeV cm^-2 s^-1 MeV^- ) (Not Implemented)
- **tab_s** (str) – table scan data
- **tab_e** (str) – table energy binning and normalization data

**Returns**

**Return type** A '~fermipy.castro.CastroData' object

**create_functor**(*specType*, *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. 'PowerLaw','LogParabola','PLExpCutoff' are implemented.
- **scale** (float) – The 'pivot energy' or energy scale to use for the spectrum

**fn** ['fermiy.spectrum.SpectralFunction'] The functor

**initPars** ['~np.array'] Default set of initial parameter for this spectral type

**scale** [float] Energy scale (same as input)

**nE**
Return the number of energy bins. This is also the number of x-axis bins.

**specData**
Return a '~fermipy.castro.SpecData' with the spectral data

**spectrum_loglike**(*specType*, *params*, *scale=1000.0*)
return the log-likelihood for a particular spectrum

**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=[u'PowerLaw', u'LogParabola', u'PLExpCutoff']*)
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" : '~fermipy.spectrum.SpectralFunction' "Result" : tuple with the output of scipy.optimize.fmin "Spectrum" : ndarray with The 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 particul 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 ith energy bin

**derivative**(*x*, *der=1*)
Return the derivate of the log-like 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*) –
- **the best-fit normalization value** (*returns*) –

**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*)
Fit for the free parameters of a spectral function

**Parameters**

- **specFunc** (*The Spectral Function*) –
- **initPars** (*The initial values of the parameters*) –

**Returns**

- **result** (*tuple*) – The output of scipy.optimize.fmin
- **spec_out** (ndarray) – The best-fit spectral values
- **TS_spec** (*float*) – The TS of 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

**getLimits**(*alpha*, *upper=True*)
Evaluate the limits corresponding to a C.L. of (1-alpha)%.

**Parameters**

- **alpha** (*limit confidence level.*) –
- **upper** (*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_type**
Return the normalization type flag

**nx**
Return the number of profiles

**ny**
Return the number of profiles

**static stack_nll**(*shape*, *components*, *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**

- **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

**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 code specifying the quantity used for the flux

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. The implementation here can deal with the following options

NORM : Normalization w.r.t. to test source FLUX : Flux of the test source ( ph cm^-2 s^-1 ) EFLUX: Energy Flux of the test source ( MeV cm^-2 s^-1 ) NPRED: Number of predicted photons DFDE : Differential flux of the test source ( ph cm^-2 s^-1 MeV^-1 ) EDFDE: Differential energy flux of the test source ( MeV cm^-2 s^-1 MeV^-

**class** fermipy.castro.**SpecData**(*emin*, *emax*, *dfde*, *flux*, *eflux*, *npred*)
Bases: object

This class wraps spectral data, e.g., energy bin definitions, flux values and number of predicted photons

**bin_widths**
return the energy bin widths

**dfde**
return the differential flux values

**ebins**
return the energy bin edges

**eflux**
return the energy flux values

**emax**
return the lower energy bin edges

**emin**
return the lower energy bin edges

**evals**
return the energy centers

**log_ebins**
return the log10 of the energy bin edges

**nE**
return the number of energy bins

**npred**
return the number of predicted events

**class** fermipy.castro.**TSCube**(*tsmap*, *normmap*, *tscube*, *normcube*, *norm_vals*, *nll_vals*, *specData*, *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=u'FLUX'*)
Build a TSCube object from a fits file created by gttscube :param fitsfile: Path to the tscube FITS file. :type fitsfile: str :param norm_type: String specifying the quantity used for the normalization :type norm_type: str

**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*) –
- **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 '~fermipy.sourcefind.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

**specData**
Return the Spectral Data object

**test_spectra_of_peak** (*peak, spec_types=[u'PowerLaw', u'LogParabola', u'PLExpCutoff']*)
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** (*'~fermipy.castro.CastroData'*) – The castro data object for the pixel corresponding to the peak
- **test_dict** (*dict*) – The dictionary returned by ~fermipy.castro.CastroData.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

**tsmap**
return the Map of the TestStatistic value

fermipy.castro.**build_source_dict**(*src_name*, *peak_dict*, *spec_dict*, *spec_type*)

### fermipy.tsmap module

**class** fermipy.tsmap.**TSCubeGenerator**
Bases: object

**tscube**(*prefix=u''*, ***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

**class** fermipy.tsmap.**TSMapGenerator**
Bases: object

Mixin class for *GTAnalysis* that generates TS maps.

**tsmap**(*prefix=u''*, ***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 (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **exclude** (*str or list of str*) – Source or sources that will be removed from the model when computing the TS map.
- **loge_bounds** (*list*) – 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.
- **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. The default value is 3 degrees.
- **make_plots** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write a FITS file.
- **write_npy** (*bool*) – Write a numpy file.

**Returns** **maps** – 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*)
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*, *background*, *model*)
Wrapper for cash statistics, that defines the model function.

**Parameters**

- **x** (*float*) – Model amplitude.
- **counts** (ndarray) – Count map slice, where model is defined.
- **background** (ndarray) – Background map slice, where model is defined.
- **model** (ndarray) – Source template (multiplied with exposure).

fermipy.tsmap.**f_cash_sum**(*x*, *counts*, *background*, *model*)

fermipy.tsmap.**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.tsmap.`**poisson_log_like**(*counts*, *model*)
Compute the Poisson log-likelihood function for the given counts and model arrays.

`fermipy.tsmap.`**sum_arrays**(*x*)

`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=u''*, ***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.
- **model** (*dict*) – Dictionary defining the properties of the convolution kernel.
- **exclude** (*str or list of str*) – Source or sources that will be removed from the model when computing the residual map.
- **loge_bounds** (*list*) – 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** (*bool*) – Write image files.
- **write_fits** (*bool*) – Write FITS files.

**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*)

#### Module contents

### 1.1.9 Changelog

This page is a changelog for releases of Fermipy.

#### 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 `loge_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 *tsmap*.

### 0.7.0 (04/19/2016)

- some features

CHAPTER 2

# Indices and tables

## f

fermipy, 81
fermipy.castro, 71
fermipy.config, 41
fermipy.defaults, 42
fermipy.logger, 56
fermipy.residmap, 80
fermipy.roi_model, 56
fermipy.sed, 66
fermipy.skymap, 69
fermipy.sourcefind, 66
fermipy.tsmap, 78
fermipy.utils, 61

## Symbols

__call__() (fermipy.castro.CastroData_Base method), 73
__call__() (fermipy.castro.Interpolator method), 75
__delattr__ (fermipy.gtanalysis.GTAnalysis attribute), 42
__format__() (fermipy.gtanalysis.GTAnalysis method), 42
__getattribute__ (fermipy.gtanalysis.GTAnalysis attribute), 42
__getitem__() (fermipy.castro.CastroData_Base method), 73
__hash__ (fermipy.gtanalysis.GTAnalysis attribute), 42
__reduce__() (fermipy.gtanalysis.GTAnalysis method), 42
__reduce_ex__() (fermipy.gtanalysis.GTAnalysis method), 42
__repr__ (fermipy.gtanalysis.GTAnalysis attribute), 42
__setattr__ (fermipy.gtanalysis.GTAnalysis attribute), 42
__sizeof__() (fermipy.gtanalysis.GTAnalysis method), 42
__str__ (fermipy.gtanalysis.GTAnalysis attribute), 42

## A

add_gauss_prior() (fermipy.gtanalysis.GTAnalysis method), 42
add_name() (fermipy.roi_model.Model method), 57
add_source() (fermipy.gtanalysis.GTAnalysis method), 42
add_sources_from_roi() (fermipy.gtanalysis.GTAnalysis method), 42
apply_minmax_selection() (in module fermipy.utils), 61
assoc (fermipy.roi_model.Model attribute), 57
associations (fermipy.roi_model.Source attribute), 60

## B

bin_widths (fermipy.castro.SpecData attribute), 76
bowtie() (fermipy.gtanalysis.GTAnalysis method), 43
build_source_dict() (in module fermipy.castro), 78

## C

cash() (in module fermipy.tsmap), 79
cast_config() (in module fermipy.config), 41
CastroData (class in fermipy.castro), 71
CastroData_Base (class in fermipy.castro), 73
castroData_from_ipix() (fermipy.castro.TSCube method), 76
castroData_from_pix_xy() (fermipy.castro.TSCube method), 77
check_cuts() (fermipy.roi_model.Model method), 57
cl_to_dlnl() (in module fermipy.utils), 61
cleanup() (fermipy.gtanalysis.GTAnalysis method), 43
clear() (fermipy.roi_model.ROIModel method), 58
close() (fermipy.logger.StreamLogger method), 56
collect_dirs() (in module fermipy.utils), 61
components (fermipy.gtanalysis.GTAnalysis attribute), 43
config (fermipy.config.Configurable attribute), 41
config (fermipy.gtanalysis.GTAnalysis attribute), 43
configdir (fermipy.config.Configurable attribute), 41
configdir (fermipy.gtanalysis.GTAnalysis attribute), 43
ConfigManager (class in fermipy.config), 41
Configurable (class in fermipy.config), 41
configure() (fermipy.config.Configurable method), 41
configure() (fermipy.gtanalysis.GTAnalysis method), 43
constrain_norms() (fermipy.gtanalysis.GTAnalysis method), 43
convert_to_cached_wcs() (fermipy.skymap.HpxMap method), 69
convert_tscube() (in module fermipy.tsmap), 79
convolve2d_disk() (in module fermipy.utils), 61
convolve2d_gauss() (in module fermipy.utils), 62
convolve_map() (in module fermipy.residmap), 81
copy_source() (fermipy.roi_model.ROIModel method), 58
counts (fermipy.skymap.Map_Base attribute), 70
counts_map() (fermipy.gtanalysis.GTAnalysis method), 43
cov_to_correlation() (in module fermipy.utils), 62
create() (fermipy.config.ConfigManager static method), 41
create() (fermipy.gtanalysis.GTAnalysis static method), 43
create() (fermipy.roi_model.ROIModel static method), 58
create() (fermipy.skymap.Map static method), 69

create_default_config() (in module fermipy.config), 41
create_from_dict() (fermipy.roi_model.Model static method), 57
create_from_dict() (fermipy.roi_model.Source static method), 60
create_from_fits() (fermipy.castro.CastroData static method), 71
create_from_fits() (fermipy.castro.TSCube static method), 77
create_from_fits() (fermipy.skymap.Map static method), 69
create_from_hdu() (fermipy.skymap.HpxMap static method), 69
create_from_hdu() (fermipy.skymap.Map static method), 69
create_from_hdulist() (fermipy.skymap.HpxMap static method), 69
create_from_position() (fermipy.roi_model.ROIModel static method), 58
create_from_roi_data() (fermipy.roi_model.ROIModel static method), 58
create_from_sedfile() (fermipy.castro.CastroData static method), 71
create_from_source() (fermipy.roi_model.ROIModel static method), 58
create_from_stack() (fermipy.castro.CastroData static method), 71
create_from_tables() (fermipy.castro.CastroData static method), 72
create_from_xml() (fermipy.roi_model.Source static method), 60
create_functor() (fermipy.castro.CastroData method), 72
create_hpx_disk_region_string() (in module fermipy.utils), 62
create_image_hdu() (fermipy.skymap.Map method), 69
create_model_name() (in module fermipy.utils), 62
create_primary_hdu() (fermipy.skymap.Map method), 69
create_roi_from_ft1() (fermipy.roi_model.ROIModel static method), 58
create_source() (fermipy.roi_model.ROIModel method), 58
create_source_name() (in module fermipy.utils), 62
create_source_table() (in module fermipy.roi_model), 61
create_table() (fermipy.roi_model.ROIModel method), 58
create_xml_element() (in module fermipy.utils), 62

## D

data (fermipy.roi_model.Model attribute), 57
data (fermipy.roi_model.Source attribute), 60
defaults (fermipy.gtanalysis.GTAnalysis attribute), 43
defaults (fermipy.roi_model.ROIModel attribute), 58
delete_source() (fermipy.gtanalysis.GTAnalysis method), 43
delete_sources() (fermipy.gtanalysis.GTAnalysis method), 44
delete_sources() (fermipy.roi_model.ROIModel method), 58
derivative() (fermipy.castro.CastroData_Base method), 73
derivative() (fermipy.castro.Interpolator method), 75
dfde (fermipy.castro.SpecData attribute), 76
diffuse (fermipy.roi_model.IsoSource attribute), 56
diffuse (fermipy.roi_model.MapCubeSource attribute), 56
diffuse (fermipy.roi_model.Source attribute), 60
diffuse_sources (fermipy.roi_model.ROIModel attribute), 58

## E

ebins (fermipy.castro.SpecData attribute), 76
edge_to_center() (in module fermipy.utils), 62
edge_to_width() (in module fermipy.utils), 62
eflux (fermipy.castro.SpecData attribute), 76
emax (fermipy.castro.SpecData attribute), 76
emin (fermipy.castro.SpecData attribute), 76
energies (fermipy.gtanalysis.GTAnalysis attribute), 44
enumbins (fermipy.gtanalysis.GTAnalysis attribute), 44
eq2gal() (in module fermipy.utils), 62
estimate_pos_and_err_parabolic() (in module fermipy.sourcefind), 68
evals (fermipy.castro.SpecData attribute), 76
extend_array() (in module fermipy.utils), 62
extended (fermipy.roi_model.Source attribute), 60
extension() (fermipy.gtanalysis.GTAnalysis method), 44
extract_array() (in module fermipy.tsmap), 79
extract_images_from_tscube() (in module fermipy.tsmap), 79
extract_large_array() (in module fermipy.tsmap), 79
extract_mapcube_region() (in module fermipy.utils), 62
extract_small_array() (in module fermipy.tsmap), 79

## F

f_cash() (in module fermipy.tsmap), 79
f_cash_sum() (in module fermipy.tsmap), 79
fermipy (module), 81
fermipy.castro (module), 71
fermipy.config (module), 41
fermipy.defaults (module), 42
fermipy.logger (module), 56
fermipy.residmap (module), 80
fermipy.roi_model (module), 56
fermipy.sed (module), 66
fermipy.skymap (module), 69
fermipy.sourcefind (module), 66
fermipy.tsmap (module), 78
fermipy.utils (module), 61
filefunction (fermipy.roi_model.IsoSource attribute), 56

find_and_refine_peaks() (fermipy.castro.TSCube method), 77
find_function_root() (in module fermipy.utils), 62
find_peaks() (in module fermipy.sourcefind), 68
find_sources() (fermipy.castro.TSCube method), 77
find_sources() (fermipy.gtanalysis.GTAnalysis method), 45
find_sources() (fermipy.sourcefind.SourceFinder method), 66
fit() (fermipy.gtanalysis.GTAnalysis method), 45
fit_correlation() (fermipy.gtanalysis.GTAnalysis method), 46
fit_error_ellipse() (in module fermipy.sourcefind), 68
fit_parabola() (in module fermipy.utils), 63
fit_spectrum() (fermipy.castro.CastroData_Base method), 74
fitNorm_v2() (fermipy.castro.CastroData_Base method), 73
fitNormalization() (fermipy.castro.CastroData_Base method), 73
fits_recarray_to_dict() (in module fermipy.utils), 63
flush() (fermipy.logger.StreamLogger method), 56
fn_mle() (fermipy.castro.LnLFn method), 75
fn_mles() (fermipy.castro.CastroData_Base method), 74
format_filename() (in module fermipy.utils), 63
free_index() (fermipy.gtanalysis.GTAnalysis method), 46
free_norm() (fermipy.gtanalysis.GTAnalysis method), 46
free_parameter() (fermipy.gtanalysis.GTAnalysis method), 46
free_shape() (fermipy.gtanalysis.GTAnalysis method), 46
free_source() (fermipy.gtanalysis.GTAnalysis method), 46
free_sources() (fermipy.gtanalysis.GTAnalysis method), 47

## G

gal2eq() (in module fermipy.utils), 63
generate_model() (fermipy.gtanalysis.GTAnalysis method), 47
get() (fermipy.logger.Logger static method), 56
get_catalog_dict() (fermipy.roi_model.Model method), 57
get_config() (fermipy.config.Configurable class method), 41
get_config() (fermipy.gtanalysis.GTAnalysis method), 47
get_dist_to_edge() (in module fermipy.roi_model), 61
get_free_param_vector() (fermipy.gtanalysis.GTAnalysis method), 47
get_free_source_params() (fermipy.gtanalysis.GTAnalysis method), 47
get_linear_dist() (in module fermipy.roi_model), 61
get_map_values() (fermipy.skymap.Map method), 69
get_nearby_sources() (fermipy.roi_model.ROIModel method), 58
get_norm() (fermipy.gtanalysis.GTAnalysis method), 47
get_norm() (fermipy.roi_model.Model method), 57
get_parameter_limits() (in module fermipy.utils), 63
get_params() (fermipy.gtanalysis.GTAnalysis method), 47
get_params_dict() (in module fermipy.roi_model), 61
get_pixel_indices() (fermipy.skymap.Map method), 70
get_pixel_indices() (fermipy.skymap.Map_Base method), 70
get_pixel_skydirs() (fermipy.skymap.Map method), 70
get_skydir_distance_mask() (in module fermipy.roi_model), 61
get_source_by_name() (fermipy.roi_model.ROIModel method), 58
get_source_dfde() (fermipy.gtanalysis.GTAnalysis method), 47
get_source_kernel() (in module fermipy.residmap), 81
get_source_name() (fermipy.gtanalysis.GTAnalysis method), 48
get_sources() (fermipy.gtanalysis.GTAnalysis method), 48
get_sources() (fermipy.roi_model.ROIModel method), 59
get_sources_by_name() (fermipy.roi_model.ROIModel method), 59
get_sources_by_position() (fermipy.roi_model.ROIModel method), 59
get_sources_by_property() (fermipy.roi_model.ROIModel method), 59
get_src_model() (fermipy.gtanalysis.GTAnalysis method), 48
getInterval() (fermipy.castro.LnLFn method), 75
getLimit() (fermipy.castro.LnLFn method), 75
getLimits() (fermipy.castro.CastroData_Base method), 74
GTAnalysis (class in fermipy.gtanalysis), 42

## H

has_source() (fermipy.roi_model.ROIModel method), 59
hpx (fermipy.skymap.HpxMap attribute), 69
HpxMap (class in fermipy.skymap), 69

## I

interp (fermipy.castro.LnLFn attribute), 75
interpolate_function_min() (in module fermipy.utils), 63
Interpolator (class in fermipy.castro), 75
ipix_swap_axes() (fermipy.skymap.Map method), 70
ipix_to_xypix() (fermipy.skymap.Map method), 70
IsoSource (class in fermipy.roi_model), 56
isstr() (in module fermipy.utils), 63
items() (fermipy.roi_model.Model method), 57

## J

join_strings() (in module fermipy.utils), 63

## L

like (fermipy.gtanalysis.GTAnalysis attribute), 48
LnLFn (class in fermipy.castro), 75
load() (fermipy.config.ConfigManager static method), 41
load() (fermipy.roi_model.ROIModel method), 59
load_data() (in module fermipy.utils), 63
load_diffuse_srcs() (fermipy.roi_model.ROIModel method), 59
load_fits_catalog() (fermipy.roi_model.ROIModel method), 59
load_from_catalog() (fermipy.roi_model.Source method), 60
load_npy() (in module fermipy.utils), 63
load_roi() (fermipy.gtanalysis.GTAnalysis method), 48
load_source() (fermipy.roi_model.ROIModel method), 59
load_sources() (fermipy.roi_model.ROIModel method), 59
load_xml() (fermipy.gtanalysis.GTAnalysis method), 48
load_xml() (fermipy.roi_model.ROIModel method), 59
load_xml_elements() (in module fermipy.utils), 63
load_yaml() (in module fermipy.utils), 63
localize() (fermipy.gtanalysis.GTAnalysis method), 48
localize() (fermipy.sourcefind.SourceFinder method), 67
log_ebins (fermipy.castro.SpecData attribute), 76
log_energies (fermipy.gtanalysis.GTAnalysis attribute), 49
log_level() (in module fermipy.logger), 56
loge_bounds (fermipy.gtanalysis.GTAnalysis attribute), 49
Logger (class in fermipy.logger), 56
lonlat_to_xyz() (in module fermipy.utils), 63

## M

make_cdisk_kernel() (in module fermipy.utils), 63
make_cgauss_kernel() (in module fermipy.utils), 63
make_coadd_map() (in module fermipy.skymap), 70
make_coadd_wcs() (in module fermipy.skymap), 70
make_default_dict() (in module fermipy.defaults), 42
make_disk_kernel() (in module fermipy.utils), 63
make_gaussian_kernel() (in module fermipy.utils), 64
make_pixel_offset() (in module fermipy.utils), 64
make_plots() (fermipy.gtanalysis.GTAnalysis method), 49
make_psf_kernel() (in module fermipy.utils), 64
make_wcs_from_hpx() (fermipy.skymap.HpxMap method), 69
Map (class in fermipy.skymap), 69
Map_Base (class in fermipy.skymap), 70
mapcube (fermipy.roi_model.MapCubeSource attribute), 56
MapCubeSource (class in fermipy.roi_model), 56
match_regex_list() (in module fermipy.utils), 64
match_source() (fermipy.roi_model.ROIModel method), 60
merge_dict() (in module fermipy.utils), 64
mkdir() (in module fermipy.utils), 64
mle() (fermipy.castro.LnLFn method), 76
mles() (fermipy.castro.CastroData_Base method), 74
Model (class in fermipy.roi_model), 56
model_counts_map() (fermipy.gtanalysis.GTAnalysis method), 49
model_counts_spectrum() (fermipy.gtanalysis.GTAnalysis method), 49

## N

name (fermipy.roi_model.Model attribute), 57
names (fermipy.roi_model.Model attribute), 57
nE (fermipy.castro.CastroData attribute), 72
nE (fermipy.castro.SpecData attribute), 76
nE (fermipy.castro.TSCube attribute), 77
nll_null (fermipy.castro.CastroData_Base attribute), 74
nN (fermipy.castro.TSCube attribute), 77
norm_type (fermipy.castro.CastroData_Base attribute), 74
norm_type (fermipy.castro.LnLFn attribute), 76
normcube (fermipy.castro.TSCube attribute), 77
normmap (fermipy.castro.TSCube attribute), 77
npix (fermipy.gtanalysis.GTAnalysis attribute), 49
npred (fermipy.castro.SpecData attribute), 76
nx (fermipy.castro.CastroData_Base attribute), 74
ny (fermipy.castro.CastroData_Base attribute), 74

## O

optimize() (fermipy.gtanalysis.GTAnalysis method), 49
outdir (fermipy.gtanalysis.GTAnalysis attribute), 50
overlap_slices() (in module fermipy.tsmap), 79

## P

parabola() (in module fermipy.utils), 64
params (fermipy.roi_model.Model attribute), 57
path_to_xmlpath() (in module fermipy.utils), 64
pix_center (fermipy.skymap.Map attribute), 70
pix_size (fermipy.skymap.Map attribute), 70
point_sources (fermipy.roi_model.ROIModel attribute), 60
poisson_lnl() (in module fermipy.residmap), 81
poisson_log_like() (in module fermipy.tsmap), 80
poly_to_parabola() (in module fermipy.utils), 64
prettify_xml() (in module fermipy.utils), 64
print_config() (fermipy.config.Configurable method), 41
print_config() (fermipy.gtanalysis.GTAnalysis method), 50
print_model() (fermipy.gtanalysis.GTAnalysis method), 50
print_params() (fermipy.gtanalysis.GTAnalysis method), 50

print_roi() (fermipy.gtanalysis.GTAnalysis method), 50
profile() (fermipy.gtanalysis.GTAnalysis method), 50
profile_norm() (fermipy.gtanalysis.GTAnalysis method), 50
project() (in module fermipy.utils), 64
projection (fermipy.roi_model.ROIModel attribute), 60
projtype (fermipy.gtanalysis.GTAnalysis attribute), 51

## R

radec (fermipy.roi_model.Source attribute), 60
read_map_from_fits() (in module fermipy.skymap), 70
rebin_map() (in module fermipy.utils), 64
refine_peak() (in module fermipy.sourcefind), 68
reload_source() (fermipy.gtanalysis.GTAnalysis method), 51
remove_prior() (fermipy.gtanalysis.GTAnalysis method), 51
remove_priors() (fermipy.gtanalysis.GTAnalysis method), 51
residmap() (fermipy.gtanalysis.GTAnalysis method), 51
residmap() (fermipy.residmap.ResidMapGenerator method), 80
ResidMapGenerator (class in fermipy.residmap), 80
resolve_file_path() (in module fermipy.roi_model), 61
resolve_path() (in module fermipy.utils), 64
roi (fermipy.gtanalysis.GTAnalysis attribute), 51
ROIModel (class in fermipy.roi_model), 57

## S

scale_parameter() (fermipy.gtanalysis.GTAnalysis method), 51
scale_parameter() (in module fermipy.utils), 64
sed() (fermipy.gtanalysis.GTAnalysis method), 51
sed() (fermipy.sed.SEDGenerator method), 66
SEDGenerator (class in fermipy.sed), 66
separation() (fermipy.roi_model.Source method), 60
set_edisp_flag() (fermipy.gtanalysis.GTAnalysis method), 52
set_energy_range() (fermipy.gtanalysis.GTAnalysis method), 52
set_free_param_vector() (fermipy.gtanalysis.GTAnalysis method), 52
set_log_level() (fermipy.gtanalysis.GTAnalysis method), 52
set_name() (fermipy.roi_model.Model method), 57
set_norm() (fermipy.gtanalysis.GTAnalysis method), 52
set_norm_bounds() (fermipy.gtanalysis.GTAnalysis method), 52
set_norm_scale() (fermipy.gtanalysis.GTAnalysis method), 52
set_parameter() (fermipy.gtanalysis.GTAnalysis method), 52
set_parameter_bounds() (fermipy.gtanalysis.GTAnalysis method), 53
set_parameter_scale() (fermipy.gtanalysis.GTAnalysis method), 53
set_position() (fermipy.roi_model.Source method), 60
set_projection() (fermipy.roi_model.ROIModel method), 60
set_roi_direction() (fermipy.roi_model.Source method), 60
set_roi_projection() (fermipy.roi_model.Source method), 60
set_source_dfde() (fermipy.gtanalysis.GTAnalysis method), 53
set_source_spectrum() (fermipy.gtanalysis.GTAnalysis method), 53
set_spatial_model() (fermipy.roi_model.Source method), 61
set_spectral_pars() (fermipy.roi_model.Model method), 57
setup() (fermipy.gtanalysis.GTAnalysis method), 53
setup() (fermipy.logger.Logger static method), 56
simulate_roi() (fermipy.gtanalysis.GTAnalysis method), 53
simulate_source() (fermipy.gtanalysis.GTAnalysis method), 54
skydir (fermipy.roi_model.ROIModel attribute), 60
skydir (fermipy.roi_model.Source attribute), 61
skydir (fermipy.skymap.Map attribute), 70
Source (class in fermipy.roi_model), 60
SourceFinder (class in fermipy.sourcefind), 66
sources (fermipy.roi_model.ROIModel attribute), 60
spatial_pars (fermipy.roi_model.Model attribute), 57
SpecData (class in fermipy.castro), 76
specData (fermipy.castro.CastroData attribute), 72
specData (fermipy.castro.TSCube attribute), 77
spectral_pars (fermipy.roi_model.Model attribute), 57
spectrum_loglike() (fermipy.castro.CastroData method), 72
src_name_cols (fermipy.roi_model.ROIModel attribute), 60
stack_nll() (fermipy.castro.CastroData_Base static method), 74
stage_input() (fermipy.gtanalysis.GTAnalysis method), 54
stage_output() (fermipy.gtanalysis.GTAnalysis method), 54
StreamLogger (class in fermipy.logger), 56
strip_suffix() (in module fermipy.utils), 64
sum_arrays() (in module fermipy.tsmap), 80
sum_over_energy() (fermipy.skymap.Map method), 70

## T

test_spectra() (fermipy.castro.CastroData method), 72
test_spectra_of_peak() (fermipy.castro.TSCube method), 77
tolist() (in module fermipy.utils), 64

truncate_array() (in module fermipy.tsmap), 80
TS() (fermipy.castro.LnLFn method), 75
ts_cumul (fermipy.castro.TSCube attribute), 77
TS_spectrum() (fermipy.castro.CastroData_Base method), 73
ts_vals() (fermipy.castro.CastroData_Base method), 75
TSCube (class in fermipy.castro), 76
tscube (fermipy.castro.TSCube attribute), 77
tscube() (fermipy.gtanalysis.GTAnalysis method), 54
tscube() (fermipy.tsmap.TSCubeGenerator method), 78
TSCubeGenerator (class in fermipy.tsmap), 78
tsmap (fermipy.castro.TSCube attribute), 78
tsmap() (fermipy.gtanalysis.GTAnalysis method), 54
tsmap() (fermipy.tsmap.TSMapGenerator method), 78
TSMapGenerator (class in fermipy.tsmap), 78

## U

unicode_representer() (in module fermipy.utils), 65
unicode_to_str() (in module fermipy.utils), 65
unzero_source() (fermipy.gtanalysis.GTAnalysis method), 55
update_bounds() (in module fermipy.utils), 65
update_data() (fermipy.roi_model.Model method), 57
update_data() (fermipy.roi_model.Source method), 61
update_from_source() (fermipy.roi_model.Model method), 57
update_keys() (in module fermipy.utils), 65
update_source() (fermipy.gtanalysis.GTAnalysis method), 55
update_spectral_pars() (fermipy.roi_model.Model method), 57

## V

val_to_bin() (in module fermipy.utils), 65
val_to_bin_bounded() (in module fermipy.utils), 65
val_to_edge() (in module fermipy.utils), 65
validate_config() (in module fermipy.config), 41

## W

wcs (fermipy.skymap.Map attribute), 70
width (fermipy.skymap.Map attribute), 70
workdir (fermipy.gtanalysis.GTAnalysis attribute), 55
write() (fermipy.logger.StreamLogger method), 56
write_config() (fermipy.config.Configurable method), 41
write_config() (fermipy.gtanalysis.GTAnalysis method), 55
write_fits() (fermipy.roi_model.ROIModel method), 60
write_fits_image() (in module fermipy.utils), 65
write_hpx_image() (in module fermipy.utils), 65
write_model_map() (fermipy.gtanalysis.GTAnalysis method), 55
write_roi() (fermipy.gtanalysis.GTAnalysis method), 55
write_xml() (fermipy.gtanalysis.GTAnalysis method), 56
write_xml() (fermipy.roi_model.IsoSource method), 56
write_xml() (fermipy.roi_model.MapCubeSource method), 56
write_xml() (fermipy.roi_model.ROIModel method), 60
write_xml() (fermipy.roi_model.Source method), 61
write_yaml() (in module fermipy.utils), 65

## X

x (fermipy.castro.Interpolator attribute), 75
xmax (fermipy.castro.Interpolator attribute), 75
xmin (fermipy.castro.Interpolator attribute), 75
xmlpath_to_path() (in module fermipy.utils), 65
xy_pix_to_ipix() (fermipy.skymap.Map method), 70
xyz_to_lonlat() (in module fermipy.utils), 65

## Y

y (fermipy.castro.Interpolator attribute), 75

## Z

zero_source() (fermipy.gtanalysis.GTAnalysis method), 56