Fermipy Documentation

Release 0.17.3+dirty

Matthew Wood

Jul 12, 2018

Contents

CHAPTER 1

Introduction

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

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

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

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

For instructions on installing Fermipy see the *Installation* page. For a short introduction to using Fermipy see the *Quickstart Guide*.

1.1 Getting Help

If you have questions about using Fermipy please open a GitHub Issue or email the Fermipy developers.

1.2 Acknowledging Fermipy

To acknowledge Fermipy in a publication please cite Wood et al. 2017.

1.3 Documentation Contents

1.3.1 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. The *Installing with Docker* instructions can be used to install the STs on OSX and Linux machines that are new enough to support Docker. To install the development version of Fermipy follow the *Installing Development Versions* 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-20150518-x86_64-unknown-linux-gnu-libc2.19-10-without-rootA.tar.gz
$ tar xzf ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.19-10-
↪without-rootA.tar.gz
$ export FERMI_DIR=ScienceTools-v10r0p5-fssc-20150518-x86_64-unknown-linux-gnu-libc2.
↪19-10-without-rootA/x86_64-unknown-linux-gnu-libc2.19-10
$ 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 will install fermipy and its dependencies into the python distribution that comes with the Fermi Science Tools. First verify that you're running the python from the Science Tools

```
$ which python
```

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

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

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

Check if pip is correctly installed:

```
$ which pip
```

Once again, if this isn't the pip in the Science Tools, something went wrong. Now install fermipy by running

```
$ pip install fermipy
```

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

```
$ pip install jupyter
```

Finally, check that fermipy imports:

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

Installing with Anaconda Python

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

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

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

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

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

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

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

```
$ conda install fermipy
```

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

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

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

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

Installing with Docker

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

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

- macOS 10.10.3 Yosemite or later
- Ubuntu Precise 12.04 or later
- Debian 8.0 or later
- RHEL7 or later
- Windows 10 or later

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

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

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

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

Now switch to the directory where you plan to run your analysis and execute the following command to launch a docker container instance:

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

This will start an ipython notebook server that will be attached to port 8888. Once you start the server it will print a URL that you can use to connect to it with the web browser on your host machine. The `-v $PWD:/workdir` argument mounts the current directory to the working area of the container. Additional directories may be mounted by adding more volume arguments `-v` with host and container paths separated by a colon.

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

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

By default interactive graphics will not be enabled. The following commands can be used to enable X11 forwarding for interactive graphics on an OSX machine. This requires you to have installed XQuartz 2.7.10 or later. First enable remote connections by default and start the X server:

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

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

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

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

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

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

Installing Development Versions

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

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

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

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

Running at SLAC

This section provides specific installation instructions for running in the SLAC computing environment. We suggest to follow these instruction if you are running Fermipy at SLAC. You will create your own conda installation in this way you will not depend on old version of programs present in the SLAC machines. First grab the installation and setup scripts from the fermipy github repository:

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

Now choose an installation path. This should be a new directory (e.g. $HOME/anaconda) that has at least 2-4 GB available. We will assign this location to the `CONDABASE` environment variable which is used by the setup script to find the location of your python installation. To avoid setting this every time you log in it's recommended to set `CONDABASE` into your .bashrc file.

Now run the following commands to install anaconda and fermipy. This will take about 5-10 minutes.

```
$ export CONDABASE=<path to install directory>
$ bash condainstall.sh $CONDABASE
```

Once anaconda is installed you will initialize your python and ST environment by running the slacsetup function in `slacsetup.sh`. This function will set the appropriate environment variables needed to run the STs and python.

```
$ source slacsetup.sh
$ slacsetup
```

For convenience you can also copy this function into your .bashrc file so that it will automatically be available when you launch a new shell session. By default the function will setup your environment to point to a recent version of the STs and the installation of python in `CONDABASE`. If `CONDABASE` is not defined then it will use the installation of python that is packaged with a given release of the STs. The slacsetup function takes two optional arguments which can be used to override the ST version or python installation path.

```
# Use ST 10-00-05
$ slacsetup 10-00-05
# Use ST 11-01-01 and python distribution located at <PATH>
$ slacsetup 11-01-01 <PATH>
```

The installation script only installs packages that are required by fermipy and the STs. Once you've initialized your shell environment you are free to install additional python packages with the conda package manager tool with conda install <package name>. Packages that are not available on conda can also be installed with pip.

conda can also be used to upgrade packages. For instance you can upgrade fermipy to the newest version with the conda update command:

```
$ conda update fermipy
```

You can verify that the installation has succeeded by importing `GTAnalysis`:

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

Upgrading

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

```
$ pip show fermipy
```

or `conda info`:

```
$ conda info fermipy
```

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

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

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

```
$ conda update fermipy
```

Developer Installation

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

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

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

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

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

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

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

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

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

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

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

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

Issues

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

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

However note that this backend does not support interactive plotting.

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

```
dyld: Library not loaded
```

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

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

1.3.2 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']
```

(continues on next page)

(continued from previous page)

```
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, make_plots=True)
m1 = gta.tsmap('fit_model', model=model, make_plots=True)
```

More documentation on these methods is available in the *TS Map* and *Residual Map* pages.

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

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

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

Extracting Analysis Results

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

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

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

The output dictionary contains the following top-level elements:

Table 1: File Dictionary

Key	Description	
`roi`	`dict`	A dictionary containing information about the ROI as a whole.
`sources`	`dict`	A dictionary containing information about 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.

```
>>> list(c['sources'].keys())
['3FGL J1032.7+3735',
```

(continues on next page)

(continued from previous page)

```
'3FGL J1033.2+4116',
...
'3FGL J1145.8+4425',
'galdiff',
'isodiff']
>>> c['sources']['3FGL J1104.4+3812']['ts']
87455.9709683
>>> 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')
>>> 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:

```
>>> 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
```

(continues on next page)

(continued from previous page)

```
# 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
```

One can also run the notebooks from a docker container following the *Installing with Docker* instructions:

```
$ git clone https://github.com/fermiPy/fermipy-extra.git
$ cd fermipy-extra
$ docker pull fermipy/fermipy
$ docker run -it --rm -p 8888:8888 -v $PWD:/workdir -w /workdir fermipy/fermipy
```

After launching the notebook server, paste the URL that appears into your web browser and navigate to the *notebooks* directory.

1.3.3 Configuration

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

Class Configuration

Classes in the Fermipy package own a configuration state dictionary that is initialized when the class instance is created. Elements of the configuration dictionary can be scalars (str, int, float) or dictionaries containing groups of parameters. The settings in this dictionary are used to control the runtime behavior of the class.

When creating a class instance, the configuration is initialized by passing either a configuration dictionary or configuration file path to the class constructor. Keyword arguments can be passed to the constructor to override configuration parameters in the input dictionary. 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 *emax* 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 structure that groups parameters into dictionaries that are keyed to a section name (*data*, *binning*, etc.).

Listing 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'
```

(continues on next page)

(continued from previous page)

```
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 has the same structure as the configuration dictionary such that one can read/write configurations using the load/dump methods of the yaml module:

```
import yaml
# Load a configuration
config = yaml.load(open('config.yaml'))
# Update a parameter and write a new configuration
config['selection']['emin'] = 1000.
yaml.dump(config, open('new_config.yaml','w'))
```

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

binning

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

Listing 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
```

Table 2: *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 independent subselections of the data. Each subselection will have its own binned likelihood instance that is combined in a global likelihood function for the ROI (implemented with the SummedLikelihood class in pyLikelihood). The *components* section is optional and when set to null (the default) only a single likelihood component will be created with the parameters of the root analysis configuration.

The component section is defined as a list of dictionaries where each element sets analysis parameters for a different subcomponent of the analysis. The component configurations follow the same structure and accept the same 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 two components. Files associated to each component will be given a suffix according to their order in the list (e.g. file_00.fits, file_01.fits, etc.).

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

data

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

Listing 3: Sample *data* Configuration

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

Table 3: *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.

Table 4: *extension* Options

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

fileio

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

Listing 4: Sample *fileio* Configuration

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

Table 5: *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`	[‘.fits$’, ‘.fit$’, ‘.xml$’, ‘.npy$’]	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`	[‘.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`).

Table 6: *gtlike* Options

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

lightcurve

The options in *lightcurve* control the default behavior of the `lightcurve` method. For more information about using this method see the *Light Curves* page.

Table 7: *lightcurve* Options

Option	Default	Description
binsz	86400.0	Set the lightcurve bin size in seconds.
free_background	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
free_params	None	Set the parameters of the source of interest that will be re-fit in each time bin. If this list is empty then all parameters will be freed.
free_radius	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
free_sources	None	List of sources to be freed. These sources will be added to the list of sources satisfying the free_radius selection.
make_plots	False	Generate diagnostic plots.
max_free_sources	5	Maximum number of sources that will be fit simultaneously with the source of interest.
multithread	False	Split the calculation across number of processes set by nthread option.
nbins	None	Set the number of lightcurve bins. The total time range will be evenly split into this number of time bins.
nthread	None	Number of processes to create when multithread is True. If None then one process will be created for each available core.
outdir	None	Store all data in this directory (e.g. "30days"). If None then use current directory.
save_bin_data	True	Save analysis directories for individual time bins. If False then only the analysis results table will be saved.
shape_ts_threshold	16.0	Set the TS threshold at which shape parameters of sources will be freed. If a source is detected with TS less than this value then its shape parameters will be fixed to values derived from the analysis of the full time range.
systematic	0.02	Systematic correction factor for TS:subscript:`var`. See Sect. 3.6 in 2FGL for details.
time_bins	None	Set the lightcurve bin edge sequence in MET. This option takes precedence over binsz and nbins.
use_local_ltcube	True	Generate a fast LT cube.
use_scaled_srcmap	False	Generate approximate source maps for each time bin by scaling the current source maps by the exposure ratio with respect to that time bin.
write_fits	True	Write the output to a FITS file.
write_npy	True	Write the output dictionary to a numpy file.

model

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

Listing 5: 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.
```

(continues on next page)

(continued from previous page)

```
  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
```

Table 8: *model* Options

Option	Default	Description
assoc_xmatch_cols	['3FGL_Name']	Choose a set of association columns on which to cross-match catalogs.
catalogs	None	
diffuse	None	
diffuse_dirs	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 path to one or more galactic IEM mapcubes. A separate component will be generated for each item in this list.
isodiff	None	Set the path to one or more isotropic templates. A separate component will be generated for each item in this list.
limbdiff	None	
merge_sources	True	Merge properties of sources that appear in multiple source catalogs. If merge_sources=false then subsequent sources with the same name will be ignored.
sources	None	
src_radius	None	Radius of circular region in degrees centered on the ROI that selects sources for inclusion in the model. If this parameter is none then no selection is applied. This selection is ORed with the src_roiwidth selection.
src_radius_roi	None	Half-width of src_roiwidth selection. This parameter can be used in lieu of src_roiwidth.
src_roiwidth	None	Width of square region in degrees centered on the ROI that selects sources for inclusion in the model. If this parameter is none then no selection is applied. This selection will be ORed with the src_radius selection.

optimizer

Table 9: *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

Table 10: *plotting* Options

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

residmap

The options in *residmap* control the default behavior of the `residmap` method. For more information about using this method see the *Residual Map* page.

Table 11: *residmap* Options

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

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.

Table 12: *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.

Table 13: *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 that are free. Setting this to None disables the prior.
`free_background`	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
`free_params`	None	Set the parameters of the source of interest that will be freed when performing the global fit. By default all parameters will be freed.
`free_radius`	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
`make_plots`	False	Generate diagnostic plots.
`ul_confidence`	0.95	Confidence level for flux upper limit.
`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.
`write_fits`	True	Write the output to a FITS file.
`write_npy`	True	Write the output dictionary to a numpy file.

selection

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

```
selection:

  # gtselect parameters
  emin    : 100
  emax    : 100000
  zmax    : 90
```

(continues on next page)

(continued from previous page)

```
  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'
```

Table 14: *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))
phasemax	None	Maximum pulsar phase
phasemin	None	Minimum pulsar phase
ra	None	
radius	None	Radius of data selection. If none this will be automatically set from the ROI size.
roicut	no	
target	None	Choose an object on which to center the ROI. This option takes 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 Finding* page.

Table 15: *sourcefind* Options

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

tsmap

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

Table 16: *tsmap* Options

Option	Default	Description
exclude	None	List of sources that will be removed from the model when computing the TS map.
loge_bounds	None	Restrict the analysis to an energy range (emin,emax) in log10(E/MeV) that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either emin/emax are None then only an upper/lower bound on the energy range wil be applied.
make_plots	False	Generate diagnostic plots.
max_kernel_radius	3.0	Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
multithread	False	Split the calculation across number of processes set by nthread option.
nthread	None	Number of processes to create when multithread is True. If None then one process will be created for each available core.
write_fits	True	Write the output to a FITS file.
write_npy	True	Write the output dictionary to a numpy file.

tscube

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

Table 17: *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. A value of zero disables damping.
max_iter	30	Maximum number of iterations for the Newtons method fitter.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
nnorm	10	Number of points in the likelihood v. normalization scan
norm_sigma	5.0	Number of sigma to use for the scan range
remake_test_source	False	If true, recomputes the test source image (otherwise just shifts it)
st_scan_level	0	Level to which to do ST-based fitting (for testing)
tol	0.001	Critetia for fit convergence (estimated vertical distance to min < tol)
tol_type	0	Absoulte (0) or relative (1) criteria for convergence.

1.3.4 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:

Table 18: File Dictionary

Key	Type	Description
roi	dict	A dictionary containing information about the ROI as a whole.
sources	dict	A dictionary containing information about 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 19: 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 in the XML model.
SourceType	str	Source type string (PointSource or DiffuseSource).
SpectrumType	str	Spectrum type string. This corresponds to the type attribute of the spectrum component in the XML model (e.g. PowerLaw, LogParabola, etc.).
Spatial_Filename	str	Path to spatial template associated to this source.
SpectrumFilename	str	Path to file associated to the spectral model of this source.
correlation	dict	Dictionary of correlation coefficients.
model_counts	ndarray	Vector of predicted counts for this source in each analysis energy bin.
model_counts_wt	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.
ra	float	Right ascension of the source (deg).
dec	float	Declination of the source (deg).
glon	float	Galactic longitude of the source (deg).
glat	float	Galactic latitude of the source (deg).
ra_err	float	Std. deviation of positional uncertainty in right ascension (deg).
dec_err	float	Std. deviation of positional uncertainty in declination (deg).
glon_err	float	Std. deviation of positional uncertainty in galactic longitude (deg).
glat_err	float	Std. deviation of positional uncertainty in galactic latitude (deg).
pos_err	float	1-sigma positional uncertainty (deg).
pos_r68	float	68% positional uncertainty (deg).
pos_r95	float	95% positional uncertainty (deg).
pos_r99	float	99% positional uncertainty (deg).
pos_err_semimajor	float	1-sigma uncertainty (deg) along major axis of uncertainty ellipse.
pos_err_semiminor	float	1-sigma uncertainty (deg) along minor axis of uncertainty ellipse.
pos_angle	float	Position angle of uncertainty ellipse with respect to major axis.
pos_gal_cov	ndarray	Covariance matrix of positional uncertainties in local projection in galactic coordinates.
pos_gal_corr	ndarray	Correlation matrix of positional uncertainties in local projection in galactic coordinates.
pos_cel_cov	ndarray	Covariance matrix of positional uncertainties in local projection in celestial coordinates.
pos_cel_corr	ndarray	Correlation matrix of positional uncertainties in local projection in celestial coordinates.
offset_ra	float	Right ascension offset from ROI center in local celestial projection (deg).
offset_dec	float	Declination offset from ROI center in local celestial projection (deg).
offset_glon	float	Galactic longitude offset from ROI center in local galactic projection (deg).
offset_glat	float	Galactic latitude offset from ROI center in local galactic projection (deg).
offset_roi_edge	float	Distance from the edge of the ROI (deg). Negative (positive) values indicate locations inside (outside) the ROI.
offset	float	Angular offset from ROI center (deg).
param_names	ndarray	Names of spectral parameters.
param_values	ndarray	Spectral parameter values.
param_errors	ndarray	Spectral parameters errors.
ts	float	Source test statistic.
loglike	float	Log-likelihood of the model evaluated at the best-fit normalization of the source.
loglike_scan	ndarray	Log-likelihood values for scan of source normalization.
dloglike_scan	ndarray	Delta Log-likelihood values for scan of source normalization.
eflux_scan	ndarray	Energy flux values for scan of source normalization.

Continued on next page

Table 19 – continued from previous page

Key	Type	Description
flux_scan	ndarray	Flux values for scan of source normalization.
norm_scan	ndarray	Normalization parameters values for scan of source normalization.
npred	float	Number of predicted counts from this source integrated over the analysis energy range.
npred_wt	float	Number of predicted counts from this source integrated over the analysis energy range.
pivot_energy	float	Decorrelation energy in MeV.
flux	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
flux100	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV.
flux_err	float	Photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
flux100_err	float	Photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000_err	float	Photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000_err	float	Photon flux 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	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
eflux100	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV.
eflux_err	float	Energy flux uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
eflux100_err	float	Energy flux uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000_err	float	Energy flux uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000_err	float	Energy flux 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 range
eflux100_ul95	float	95% CL upper limit on the energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
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 GeV.
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 GeV.
dnde	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at the pivot energy.
dnde100	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 100 MeV.
dnde1000	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 1 GeV.
dnde10000	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 10 GeV.
dnde_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at the pivot energy.
dnde100_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 100 MeV.
dnde1000_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 1 GeV.
dnde10000_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 10 GeV.
dnde_index	float	Logarithmic slope of the differential photon spectrum evaluated at the pivot energy.
dnde100_index	float	Logarithmic slope of the differential photon spectrum evaluated at 100 MeV.
dnde1000_index	float	Logarithmic slope of the differential photon spectrum evaluated evaluated at 1 GeV.
dnde10000_index	float	Logarithmic slope of the differential photon spectrum evaluated at 10 GeV.

1.3.5 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:      501.128
>>> 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:

Table 20: *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.

Table 21: *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.

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

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

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

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

Editing the Model at Runtime

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

```
from fermipy.gtanalysis import GTAnalysis

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

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

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

gta.setup()

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

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

1.3.7 Developer Notes

Creating a New Release

The following are steps for creating a new release:

1. Update the Changelog page (in `docs/changelog.rst`) with notes for the release and commit those changes.

2. Update documentation tables by running `make_tables.py` inside the `docs` subdirectory and commit any resulting changes to the configuration table files under `docs/config`.
3. Checkout `master` and ensure that you have pulled all commits from origin.
4. Create the release tag and push it to GitHub.

```
$ git tag -a XX.YY.ZZ -m ""
$ git push --tags
```

5. Upload the release to pypi.

```
$ python setup.py sdist upload -r pypi
```

6. Create a new release on conda-forge by opening a PR on the fermipy-feedstock repo. There is a fork of `fermipy-feedstock` in the fermipy organization that you can use for this purpose. Edit `recipe/meta.yaml` by entering the new package version and updating the sha256 hash to the value copied from the pypi download page. Update the package dependencies as necessary in the `run` section of `requirements`. Verify that `entry_points` contains the desired set of command-line scripts. Generally this section should match the contents `entry_points` in `setup.py`. Before merging the PR confirm that all tests have successfully passed.

1.3.8 Advanced Analysis Methods

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

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

SED Analysis

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

The `free_background`, `free_radius`, and `cov_scale` parameters control how nuisance parameters are dealt with in the fit. By default the method will fix the parameters of background components ROI when fitting the source

normalization in each energy bin (`free_background=False`). Setting `free_background=True` will profile the normalizations of all background components that were free when the method was executed. In order to minimize overfitting, background normalization parameters are constrained with priors taken from the global fit. The strength of the priors is controlled with the `cov_scale` parameter. A larger (smaller) value of `cov_scale` applies a weaker (stronger) constraint on the background amplitude. Setting `cov_scale=None` performs an unconstrained fit without priors.

Examples

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

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

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

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

By default the method will use the energy bins of the underlying analysis. The `loge_bins` keyword argument can be used to override the default binning with the restriction that the SED energy bins must align with the analysis bins. The bins used in the analysis can be found with `gta.log_energies`. For example if in the analysis 8 energy bins per decade are considered and you want to make the SED in 4 bins per decade you can specify `loge_bins=gta.log_energies[::2]`.

The return value of *sed()* is a dictionary with the results of the analysis. The following example shows how to extract values from the output dictionary and load the SED data from the output FITS file:

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

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

# Reload the SED table from the output FITS file
from astropy.table import Table
sed_tab = Table.read('sed.fits')
```

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

SED FITS File

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

Table 22: *sed* Output Dictionary

HDU	Column Name	Description
SED	e_min	Lower edges of SED energy bins (MeV).
SED	e_ref	Upper edges of SED energy bins (MeV).
SED	e_max	Centers of SED energy bins (MeV).
SED	ref_dnde_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_dnde_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_dnde	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	dnde	Central value of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$).
MODEL_FLUX	dnde_lo	Lower 1-sigma bound of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$).
MODEL_FLUX	dnde_hi	Upper 1-sigma bound of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$).
MODEL_FLUX	dnde_err	Symmetric error of spectral band ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$).
MODEL_FLUX	dnde_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 23: *sed* Output Dictionary

Key	Type	Description
loge_min	ndarray	Lower edges of SED energy bins (log10(E/MeV)).
loge_max	ndarray	Upper edges of SED energy bins (log10(E/MeV)).
loge_ctr	ndarray	Centers of SED energy bins (log10(E/MeV)).
loge_ref	ndarray	Reference energies of SED energy bins (log10(E/MeV)).
e_min	ndarray	Lower edges of SED energy bins (MeV).
e_max	ndarray	Upper edges of SED energy bins (MeV).
e_ctr	ndarray	Centers of SED energy bins (MeV).
e_ref	ndarray	Reference energies of SED energy bins (MeV).

Continued on next page

Table 23 – continued from previous page

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_dnde	ndarray	Differential flux of the reference model evaluated at the bin center ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$)
ref_dnde_min	ndarray	Differential flux of the reference model evaluated at the lower bin edge ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$)
ref_dnde_max	ndarray	Differential flux of the reference model evaluated at the upper bin edge ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$)
ref_e2dnde	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 ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$).
dnde	ndarray	Differential flux in each bin ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$).
e2dnde	ndarray	E^2 x the differential flux in each bin ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$).
dnde_err	ndarray	1-sigma error on dnde evaluated from likelihood curvature.
dnde_err_lo	ndarray	Lower 1-sigma error on dnde evaluated from the profile likelihood (MINOS errors).
dnde_err_hi	ndarray	Upper 1-sigma error on dnde evaluated from the profile likelihood (MINOS errors).
dnde_ul95	ndarray	95% CL upper limit on dnde evaluated from the profile likelihood (MINOS errors).
dnde_ul	ndarray	Upper limit on dnde evaluated from the profile likelihood using a CL = `ul_confidence`.
e2dnde_err	ndarray	1-sigma error on e2dnde evaluated from likelihood curvature.
e2dnde_err_lo	ndarray	Lower 1-sigma error on e2dnde evaluated from the profile likelihood (MINOS errors).
e2dnde_err_hi	ndarray	Upper 1-sigma error on e2dnde evaluated from the profile likelihood (MINOS errors).
e2dnde_ul95	ndarray	95% CL upper limit on e2dnde evaluated from the profile likelihood (MINOS errors).
e2dnde_ul	ndarray	Upper limit on e2dnde evaluated from the profile likelihood using a 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 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).
fit_status	ndarray	Fit status parameter (0=ok).
index	ndarray	Spectral index of the power-law model used to fit this bin.
norm_scan	ndarray	Array of NxM normalization values for the profile likelihood scan in N energy bins and M scan points. A row-wise multiplication with any of `ref` columns can be used to convert this matrix to the respective unit.
dloglike_scan	ndarray	Array of NxM delta-loglikelihood values for the profile likelihood scan in N energy bins and M scan points.
loglike_scan	ndarray	Array of NxM loglikelihood values for the profile likelihood scan in N energy bins and M scan points.
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 parameterization for the source.
config	dict	Copy of input configuration to this method.

Configuration

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

Table 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 that are free. Setting this to None disables the prior.
free_background	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
free_pars	None	Set the parameters of the source of interest that will be freed when performing the global fit. By default all parameters will be freed.
free_radius	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
make_plots	False	Generate diagnostic plots.
ul_confidence	0.95	Confidence level for flux upper limit.
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.
write_fits	True	Write the output to a FITS file.
write_npy	True	Write the output dictionary to a numpy file.

Reference/API

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

Generate a spectral energy distribution (SED) for a source. This function will fit the normalization of the source in each energy bin. By default the SED will be generated with the analysis energy bins but a custom binning can be defined with the 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. (default : 2.0)
- **cov_scale** (*float*) – Scale factor that sets the strength of the prior on nuisance parameters that are free. Setting this to None disables the prior. (default : 3.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_pars** (*list*) – Set the parameters of the source of interest that will be freed when performing the global fit. By default all parameters will be freed. (default : None)

- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **ul_confidence** (*float*) – Confidence level for flux upper limit. (default : 0.95)
- **use_local_index** (*bool*) – Use a power-law approximation to the shape of the global spectrum in each bin. If this is false then a constant index set to `bin_index` will be used. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

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

Return type dict

Light Curves

`lightcurve()` fits the charateristics of a source (flux, TS, etc.) in a sequence of time bins. This method uses the data selection and model of a baseline analysis (e.g. the full mission) and is therefore restricted to analyzing time bins that are encompassed by the time selection of the baseline analysis. In general when using this method it is recommended to use a baseline time selection of at least several years or more to ensure the best characterization of background sources in the ROI.

When fitting a time bin the method will initialize the model to the current parameters of the baseline analysis. The parameters to be refit in each time bin may be controlled with `free_background`, `free_sources`, `free_radius`, `free_params`, and `shape_ts_threshold` options.

Examples

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

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

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

# Generate a lightcurve with arbitrary MET binning
lc = gta.lightcurve('sourceA', time_bins=[239557414,242187214,250076614],
                    free_radius=3.0)
```

Optimizing Computation Speed

By default the `lightcurve` method will run an end-to-end analysis in each time bin using the same processing steps as the baseline analysis. Depending on the data selection and ROI size each time bin may take 10-15 minutes to process. There are several options which can be used to reduce the lightcurve computation time. The `multithread` option splits the analysis of time bins across multiple cores:

```
# Split lightcurve across all available cores
lc = gta.lightcurve('sourceA', nbins=2, multithread=True)

# split lightcurve across 2 cores
lc = gta.lightcurve('sourceA', nbins=2, multithread=True, nthread=2)
```

Note that when using the `multithread` option in a computing cluster environment one should reserve the appropriate number of cores when submitting the job.

The `use_scaled_srcmap` option generates an approximate source map for each time bin by scaling the source map of the baseline analysis by the relative exposure.

```
# Enable scaled source map
lc = gta.lightcurve('sourceA', nbins=2, use_scaled_srcmap=True)
```

Enabling this option can speed up the lightcurve calculation by at least a factor of 2 or 3 at the cost of slightly reduced accuracy in the model evaluation. For point-source analysis on medium to long timescales (days to years) the additional systematic uncertainty incurred by using scaled source maps should be no more than 1-2%. For analysis of diffuse sources or short time scales (< day) one should verify the systematic uncertainty is less than the systematic uncertainty of the IRFs.

Output

The following tables describe the contents of the method output:

Table 25: *lightcurve* Output

Key	Type	Description
name	str	Name of Source,
tmin	ndarray	Lower edge of time bin in MET.
tmax	ndarray	Upper edge of time bin in MET.
fit_success	ndarray	Did the likelihood fit converge? True if yes.
config	dict	Copy of the input configuration to this method.
ts_var	float	TS of variability. Should be distributed as χ^2 with $n-1$ degrees of freedom, where n is the number of time bins.

Table 26: *lightcurve* Source Output

Key	Type	Description
param_names	ndarray	Names of spectral parameters.
param_values	ndarray	Spectral parameter values.
param_errors	ndarray	Spectral parameters errors.
ts	float	Source test statistic.
loglike	float	Log-likelihood of the model evaluated at the best-fit normalization of the source.
loglike_scan	ndarray	Log-likelihood values for scan of source normalization.
dloglike_scan	ndarray	Delta Log-likelihood values for scan of source normalization.
eflux_scan	ndarray	Energy flux values for scan of source normalization.
flux_scan	ndarray	Flux values for scan of source normalization.
norm_scan	ndarray	Normalization parameters values for scan of source normalization.
npred	float	Number of predicted counts from this source integrated over the analysis energy range.
npred_wt	float	Number of predicted counts from this source integrated over the analysis energy range.
pivot_energy	float	Decorrelation energy in MeV.

Continued on next page

Table 26 – continued from previous page

Key	Type	Description
flux	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
flux100	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000	float	Photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV.
flux_err	float	Photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
flux100_err	float	Photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
flux1000_err	float	Photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
flux10000_err	float	Photon flux 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	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
eflux100	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000	float	Energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 10 GeV to 316 GeV.
eflux_err	float	Energy flux uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated over analysis energy range
eflux100_err	float	Energy flux uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
eflux1000_err	float	Energy flux uncertainty ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 1 GeV to 316 GeV.
eflux10000_err	float	Energy flux 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 range
eflux100_ul95	float	95% CL upper limit on the energy flux ($\mathrm{MeV}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$) integrated from 100 MeV to 316 GeV.
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 GeV.
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 GeV.
dnde	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at the pivot energy.
dnde100	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 100 MeV.
dnde1000	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 1 GeV.
dnde10000	float	Differential photon flux ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 10 GeV.
dnde_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at the pivot energy.
dnde100_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 100 MeV.
dnde1000_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 1 GeV.
dnde10000_err	float	Differential photon flux uncertainty ($\mathrm{cm}^{-2}\ \mathrm{s}^{-1}\ \mathrm{MeV}^{-1}$) evaluated at 10 GeV.
dnde_index	float	Logarithmic slope of the differential photon spectrum evaluated at the pivot energy.
dnde100_index	float	Logarithmic slope of the differential photon spectrum evaluated at 100 MeV.
dnde1000_index	float	Logarithmic slope of the differential photon spectrum evaluated evaluated at 1 GeV.
dnde10000_index	float	Logarithmic slope of the differential photon spectrum evaluated at 10 GeV.

Reference/API

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

Generate a lightcurve for the named source. The function will complete the basic analysis steps for each bin and perform a likelihood fit for each bin. Extracted values (along with errors) are Integral Flux, spectral model, Spectral index, TS value, pred. # of photons. Note: successful calculation of TS:subscript:`var` requires at least one free background parameter and a previously optimized ROI model.

Parameters

- **name** (*str*) – source name
- **binsz** (*float*) – Set the lightcurve bin size in seconds. (default : 86400.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_params** (*list*) – Set the parameters of the source of interest that will be re-fit in each time bin. If this list is empty then all parameters will be freed. (default : None)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **free_sources** (*list*) – List of sources to be freed. These sources will be added to the list of sources satisfying the free_radius selection. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **max_free_sources** (*int*) – Maximum number of sources that will be fit simultaneously with the source of interest. (default : 5)
- **multithread** (*bool*) – Split the calculation across number of processes set by nthread option. (default : False)
- **nbins** (*int*) – Set the number of lightcurve bins. The total time range will be evenly split into this number of time bins. (default : None)
- **nthread** (*int*) – Number of processes to create when multithread is True. If None then one process will be created for each available core. (default : None)
- **outdir** (*str*) – Store all data in this directory (e.g. "30days"). If None then use current directory. (default : None)
- **save_bin_data** (*bool*) – Save analysis directories for individual time bins. If False then only the analysis results table will be saved. (default : True)
- **shape_ts_threshold** (*float*) – Set the TS threshold at which shape parameters of sources will be freed. If a source is detected with TS less than this value then its shape parameters will be fixed to values derived from the analysis of the full time range. (default : 16.0)
- **systematic** (*float*) – Systematic correction factor for TS:subscript:`var`. See Sect. 3.6 in 2FGL for details. (default : 0.02)
- **time_bins** (*list*) – Set the lightcurve bin edge sequence in MET. This option takes precedence over binsz and nbins. (default : None)
- **use_local_ltcube** (*bool*) – Generate a fast LT cube. (default : True)
- **use_scaled_srcmap** (*bool*) – Generate approximate source maps for each time bin by scaling the current source maps by the exposure ratio with respect to that time bin. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

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

Return type dict

Extension Fitting

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

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

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

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

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

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

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

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

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

Table 27: *extension* Output Dictionary

Key	Type	Description
name	str	Name of source.
file	str	Name of output FITS file.
config	dict	Copy of the input configuration to this method.
width	ndarray	Vector of width (intrinsic 68% containment radius) values (deg).
dloglike	ndarray	Delta-log-likelihood values for each point in the profile likelihood scan.
loglike	ndarray	Log-likelihood values for each point in the scan over the spatial extension.
loglike_ptsrc	float	Log-Likelihood value of the best-fit point-source model.
loglike_ext	float	Log-Likelihood of the best-fit extended source model.
loglike_init	float	Log-Likelihood of model before extension fit.
loglike_base	float	Log-Likelihood of model after initial spectral fit.
ext	float	Best-fit extension (68% containment radius) (deg).
ext_err_hi	float	Upper (1-sigma) error on the best-fit extension (deg).
ext_err_lo	float	Lower (1-sigma) error on the best-fit extension (deg).
ext_err	float	Symmetric (1-sigma) error on the best-fit extension (deg).

Continued on next page

Table 27 – continued from previous page

Key	Type	Description
ext_ul95	float	95% CL upper limit on the spatial extension (deg).
ts_ext	float	Test statistic for the extension hypothesis.
ebin_e_min	ndarray	
ebin_e_ctr	ndarray	
ebin_e_max	ndarray	
ebin_ext	ndarray	Best-fit extension as measured in each energy bin (intrinsic 68% containment radius) (deg).
ebin_ext_err	ndarray	Symmetric (1-sigma) error on best-fit extension in each energy bin (deg).
ebin_ext_err_hi	ndarray	Upper (1-sigma) error on best-fit extension in each energy bin (deg).
ebin_ext_err_lo	ndarray	Lower (1-sigma) error on best-fit extension in each energy bin (deg).
ebin_ext_ul95	ndarray	95% CL upper limit on best-fit extension in each energy bin (deg).
ebin_ts_ext	ndarray	Test statistic for extension hypothesis in each energy bin.
ebin_dloglike	ndarray	Delta-log-likelihood values for scan over the spatial extension in each energy bin.
ebin_loglike	ndarray	Log-likelihood values for scan over the spatial extension in each energy bin.
ebin_loglike_pts	ndarray	Log-Likelihood value of the best-fit point-source model in each energy bin.
ebin_loglike_ext	ndarray	Log-Likelihood value of the best-fit extended source model in each energy bin.
ra	float	Right ascension of best-fit position (deg).
dec	float	Declination of best-fit position (deg).
glon	float	Galactic Longitude of best-fit position (deg).
glat	float	Galactic Latitude of best-fit position (deg).
ra_err	float	Std. deviation of positional uncertainty in right ascension (deg).
dec_err	float	Std. deviation of positional uncertainty in declination (deg).
glon_err	float	Std. deviation of positional uncertainty in galactic longitude (deg).
glat_err	float	Std. deviation of positional uncertainty in galactic latitude (deg).
pos_offset	float	Angular offset (deg) between the old and new (localized) source positions.
pos_err	float	1-sigma positional uncertainty (deg).
pos_r68	float	68% positional uncertainty (deg).
pos_r95	float	95% positional uncertainty (deg).
pos_r99	float	99% positional uncertainty (deg).
pos_err_semimajor	float	1-sigma uncertainty (deg) along major axis of uncertainty ellipse.
pos_err_semiminor	float	1-sigma uncertainty (deg) along minor axis of uncertainty ellipse.
pos_angle	float	Position angle of uncertainty ellipse with respect to major axis.
tsmap	*Map*	
ptsrc_tot_map	*Map*	
ptsrc_src_map	*Map*	
ptsrc_bkg_map	*Map*	
ext_tot_map	*Map*	
ext_src_map	*Map*	
ext_bkg_map	*Map*	
source_fit	dict	Dictionary with parameters of the best-fit extended source model.

Configuration

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

Table 28: *extension* Options

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

Reference/API

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

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

Parameters

- **name** (*str*) – Source name.
- **fit_ebin** (*bool*) – Perform a fit for the angular extension in each analysis energy bin. (default : False)

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

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

Return type dict

TS Map

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

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

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

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

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

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

Examples

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

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

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

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

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

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

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

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

Key	Type	Description
amplitude	*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.

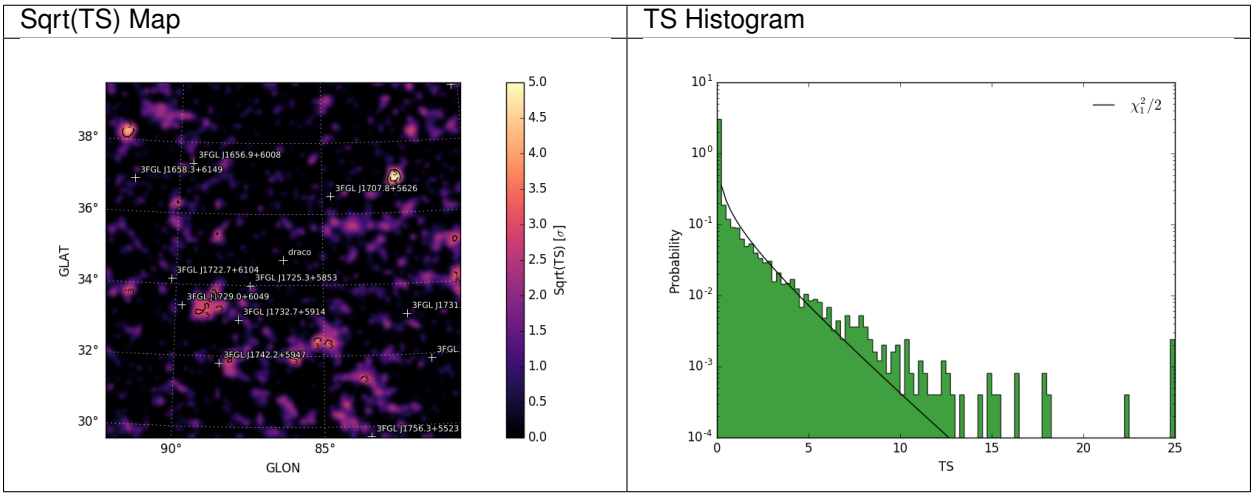
The `write_fits` and `write_npy` options can used to write the output to a FITS or numpy file. All output files are prepended with the `prefix` argument.

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

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

This will generate the following plots:

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



Configuration

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

Table 29: *tsmap* Options

Option	Default	Description
exclude	None	List of sources that will be removed from the model when computing the TS map.
loge_bounds	None	Restrict the analysis to an energy range (emin,emax) in log10(E/MeV) that is a subset of the analysis energy range. By default the full analysis energy range will be used. If either emin/emax are None then only an upper/lower bound on the energy range wil be applied.
make_plots	False	Generate diagnostic plots.
max_kernel_radius	3.0	Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
multithread	False	Split the calculation across number of processes set by nthread option.
nthread	None	Number of processes to create when multithread is True. If None then one process will be created for each available core.
write_fits	True	Write the output to a FITS file.
write_npy	True	Write the output dictionary to a numpy file.

Reference/API

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

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

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

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the TS map. (default : None)
- **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. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **max_kernel_radius** (*float*) – Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy. (default : 3.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across number of processes set by nthread option. (default : False)
- **nthread** (*int*) – Number of processes to create when multithread is True. If None then one process will be created for each available core. (default : None)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)

- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

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

Return type dict

TS Cube

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

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

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

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

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

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

Examples

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

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

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

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

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

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

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

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

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

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

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

Configuration

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

Table 30: *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. A value of zero disables damping.
max_iter	30	Maximum number of iterations for the Newtons method fitter.
model	None	Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum.
nnorm	10	Number of points in the likelihood v. normalization scan
norm_sigma	5.0	Number of sigma to use for the scan range
remake_test_source	False	If true, recomputes the test source image (otherwise just shifts it)
st_scan_level	0	Level to which to do ST-based fitting (for testing)
tol	0.001	Critetia for fit convergence (estimated vertical distance to min < tol)
tol_type	0	Absoulte (0) or relative (1) criteria for convergence.

Reference/API

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

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

Parameters

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

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

Return type dict

Residual Map

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

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

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

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

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

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

Examples

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

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

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

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

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

Key	Type	Description
sigma	*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.

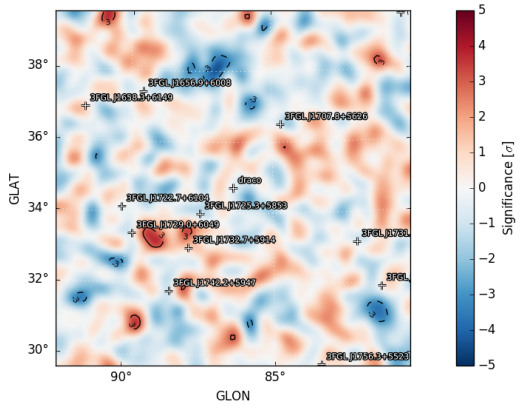
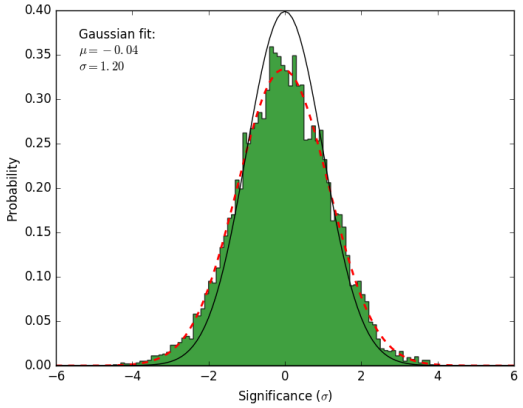
The `write_fits` and `write_npy` options can used to write the output to a FITS or numpy file. All output files are prepended with the `prefix` argument.

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

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

This will generate the following plots:

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

Residual Significance Map	Significance Histogram
	

Configuration

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

Table 31: *residmap* Options

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

Reference/API

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

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

Parameters

- **`prefix`** (*`str`*) – String that will be prefixed to the output residual map files.
- **`exclude`** (*`list`*) – List of sources that will be removed from the model when computing the residual map. (default : None)
- **`loge_bounds`** (*`list`*) – Restrict the analysis to an energy range (emin,emax) in 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. (default : None)

- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **use_weights** (*bool*) – Used weighted version of maps in making plots. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

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

Return type dict

Source Finding

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

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

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

Examples

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

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

Reference/API

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

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

Parameters

- **free_params** (*list*) – (default : None)

- **max_iter** (*int*) – Maximum number of source finding iterations. The source finder will continue adding sources until no additional peaks are found or the number of iterations exceeds this number. (default : 5)
- **min_separation** (*float*) – Minimum separation in degrees between sources detected in each iteration. The source finder will look for the maximum peak in the TS map within a circular region of this radius. (default : 1.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across number of processes set by nthread option. (default : False)
- **nthread** (*int*) – Number of processes to create when multithread is True. If None then one process will be created for each available core. (default : None)
- **sources_per_iter** (*int*) – Maximum number of sources that will be added in each iteration. If the number of detected peaks in a given iteration is larger than this number, only the N peaks with the largest TS will be used as seeds for the current iteration. (default : 4)
- **sqrt_ts_threshold** (*float*) – Source threshold in sqrt(TS). Only peaks with sqrt(TS) exceeding this threshold will be used as seeds for new sources. (default : 5.0)
- **tsmap_fitter** (*str*) – Set the method for generating the TS map. Valid options are tsmap or tscube. (default : tsmap)
- **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 likelihood surface in source position in a local patch around the nominal source position. The fit to the source position proceeds in two iterations:

- **TS Map Scan**: Obtain a first estimate of the source position by generating a likelihood map of the region using the *tsmap* method. In this step all background parameters are fixed to their nominal values. The size of the search region used for this step is set with the `dtheta_max` parameter.
- **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 in the first iteration. 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.

If a peak is found in the search region and the positional fit succeeds, the method will update the position of the source in the model to the new best-fit position.

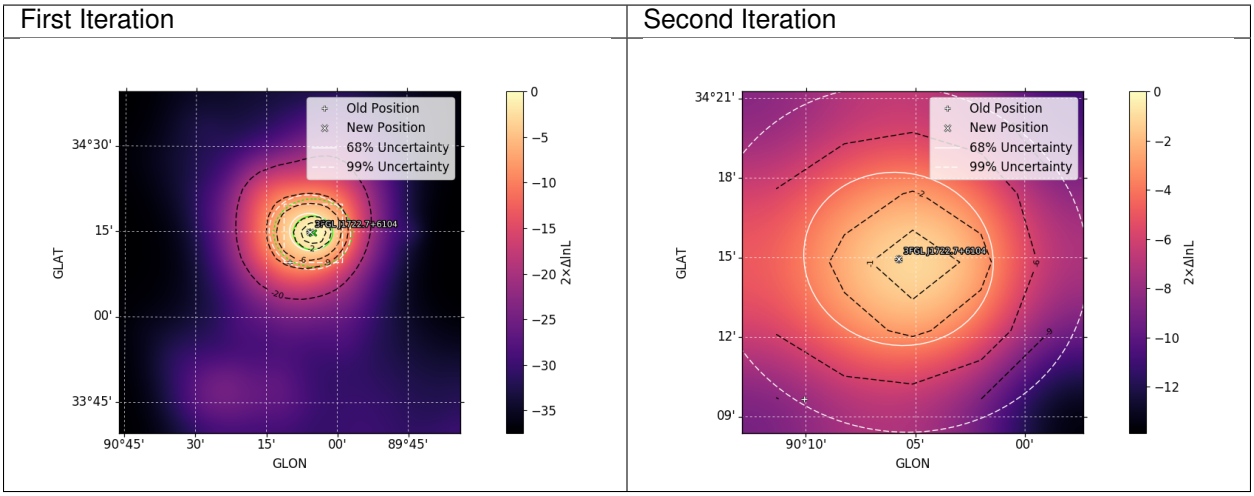
Examples

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 the same information to FITS and

numpy files.

```
>>> loc = gta.localize('3FGL J1722.7+6104', make_plots=True)
>>> print(loc['ra'],loc['dec'],loc['pos_r68'],loc['pos_r95'])
(260.53164555483784, 61.04493807148745, 0.14384100879403075, 0.23213050350030126)
```

When running with `make_plots=True` the method will save a diagnostic plot to the working directory with a visualization of the localization contours. The green and white contours show the uncertainty ellipse derived from the first and second iterations of the positional scan.



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('3FGL J1722.7+6104', update=True)
```

By default all background parameters will be fixed when the positional fit is performed. One can choose to free background parameters with the `free_background` and `free_radius` options:

```
# Free a nearby source that may be be partially degenerate with the
# source of interest
gta.free_norm('sourceB')
gta.localize('3FGL J1722.7+6104', free_background=True)

# Free normalizations of background sources within a certain
# distance of the source of interest
gta.localize('3FGL J1722.7+6104', free_radius=1.0)
```

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

Table 32: *localize* Output

Key	Type	Description
`name`	`str`	Name of source.
`file`	`str`	Name of output FITS file.
`config`	`dict`	Copy of the input configuration to this method.
`ra`	`float`	Right ascension of best-fit position (deg).
`dec`	`float`	Declination of best-fit position (deg).

Continued on next page

Table 32 – continued from previous page

Key	Type	Description
glon	float	Galactic Longitude of best-fit position (deg).
glat	float	Galactic Latitude of best-fit position (deg).
xpix	float	Longitude pixel coordinate of best-fit position.
ypix	float	Latitude pixel coordinate of best-fit position.
deltax	float	Longitude offset from old position (deg).
deltay	float	Latitude offset from old position (deg).
skydir	SkyCoord	
ra_preloc	float	Right ascension of pre-localization position (deg).
dec_preloc	float	Declination of pre-localization position (deg).
glon_preloc	float	Galactic Longitude of pre-localization position (deg).
glat_preloc	float	Galactic Latitude of pre-localization position (deg).
ra_err	float	Std. deviation of positional uncertainty in right ascension (deg).
dec_err	float	Std. deviation of positional uncertainty in declination (deg).
glon_err	float	Std. deviation of positional uncertainty in galactic longitude (deg).
glat_err	float	Std. deviation of positional uncertainty in galactic latitude (deg).
pos_offset	float	Angular offset (deg) between the old and new (localized) source positions.
pos_err	float	1-sigma positional uncertainty (deg).
pos_r68	float	68% positional uncertainty (deg).
pos_r95	float	95% positional uncertainty (deg).
pos_r99	float	99% positional uncertainty (deg).
pos_err_semimajor	float	1-sigma uncertainty (deg) along major axis of uncertainty ellipse.
pos_err_semiminor	float	1-sigma uncertainty (deg) along minor axis of uncertainty ellipse.
pos_angle	float	Position angle of uncertainty ellipse with respect to major axis.
pos_ecc	float	Eccentricity of uncertainty ellipse defined as sqrt(1-b**2/a**2).
pos_ecc2	float	Eccentricity of uncertainty ellipse defined as sqrt(a**2/b**2-1).
pos_gal_cov	ndarray	Covariance matrix of positional uncertainties in local projection in galactic coordinates.
pos_gal_corr	ndarray	Correlation matrix of positional uncertainties in local projection in galactic coordinates.
pos_cel_cov	ndarray	Covariance matrix of positional uncertainties in local projection in celestial coordinates.
pos_cel_corr	ndarray	Correlation matrix of positional uncertainties in local projection in celestial coordinates.
tsmap	*Map*	
tsmap_peak	*Map*	
loglike_init	float	Log-Likelihood of model before localization.
loglike_base	float	Log-Likelihood of model after initial spectral fit.
loglike_loc	float	Log-Likelihood of model after localization.
dloglike_loc	float	Difference in log-likelihood before and after localization.
fit_success	bool	
fit_inbounds	bool	
fit_init	dict	
fit_scan	dict	

Configuration

Table 33: *localize* Options

Option	Default	Description
dtheta_max	0.5	Half-width of the search region in degrees used for the first pass of the localization search.
fix_shape	False	Fix spectral shape parameters of the source of interest. If True then only the normalization parameter will be fit.
free_background	False	Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest.
free_radius	None	Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed.
make_plots	False	Generate diagnostic plots.
nstep	5	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.
update	True	Update the source model with the best-fit position.
write_fits	True	Write the output to a FITS file.
write_npy	True	Write the output dictionary to a numpy file.

Reference/API

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

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

Parameters

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

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

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

Return type dict

Phased Analysis

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

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

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

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

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

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

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

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

Sensitivity Tools

The `fermipy-flux-sensitivity` script calculates the LAT flux threshold for a gamma-ray source in bins of energy (differential sensitivity) and integrated over the full LAT energy range (integral sensitivity). The source flux threshold is the flux at which the median TS of a source (twice the likelihood ratio of the best-fit model with and without the source) equals a certain value. Primary inputs to this script are the livetime cube (output of `gtltcube`)

and the model cube for the galactic diffuse background. The `obs_time_yr` option can be used to rescale the livetime cube to a shorter or longer observation time.

```
$ fermipy-flux-sensitivity --glon=30 --glat=30 --output=lat_sensitivity.fits \
--ltcube=ltcube.fits --galdiff=gll_iem_v06.fits --event_class=P8R2_SOURCE_V6 \
--ts_thresh=25.0 --min_counts=10.0
```

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

A map of sensitivity with WCS or HEALPix pixelization can be generated by setting the `map_type` argument to either `wcs` or `hpx`:

```
# Generate a WCS sensitivity map of 50 x 50 deg centered at (glon,glat) = (30,30)
$ fermipy-flux-sensitivity  --glon=30 --glat=30 --output=lat_sensitivity_map.fits \
--ltcube=ltcube.fits --galdiff=gll_iem_v06.fits --event_class=P8R2_SOURCE_V6 \
--map_type=wcs --wcs_npix=100 --wcs_cdelt=0.5 --wcs_proj=AIT

# Generate a HPX sensitivity map of nside=16
$ fermipy-flux-sensitivity  --output=lat_sensitivity_map.fits \
--ltcube=ltcube.fits --galdiff=gll_iem_v06.fits --event_class=P8R2_SOURCE_V6 \
--map_type=hpx --hpx_nside=16
```

The integral and differential sensitivity maps will be written to the `MAP_INT_FLUX` and `MAP_DIFF_FLUX` extensions respectively.

By default the flux sensitivity will be computed for a point-source morphology. The assumed source morphology can be changed with the `spatial_model` and `spatial_size` parameters:

It is possible to choose among PowerLaw, LogParabola and PLSuperExpCutoff SED shapes using the option `sedshape`.

```
# Generate the sensitivity to a source with a 2D gaussian morphology
# and a 68% containment radius of 1 deg located at longitude 30deg and
# latitude 30 deg and with a PLSuperExpCutoff SED with index 2.0 and
# cutoff energy 10 GeV
$ fermipy-flux-sensitivity --output=lat_sensitivity_map.fits \
--ltcube=ltcube.fits --galdiff=gll_iem_v06.fits --event_class=P8R2_SOURCE_V6 \
--spatial_model=RadialGaussian --spatial_size=1.0 --glon=30 --glat=30
--sedshape=PLSuperExpCutoff --index=2.0 --cutoff=1e4

# Generate the sensitivity map in healpix with nside 128 of a point source with
# LogParabola SED and with spectral index 2.0 and curvature index beta=0.50
# between 1 and 10 GeV
$ fermipy-flux-sensitivity --output=lat_sensitivity_map.fits \
--ltcube=ltcube.fits --galdiff=gll_iem_v06.fits --event_class=P8R2_SOURCE_V6 \
--spatial_model=PointSource --sedshape=LogParabola --index=2.0 --beta=0.50 \
--hpx_nside=128 --map_type=hpx --emin=1000 --emax=10000
```

The output FITS file set with the `output` option contains the following tables. Note that `MAP` tables are only generated when the `map_type` argument is set.

- `DIFF_FLUX` : Differential flux sensitivity for a gamma-ray source at sky positition set by `glon` and `glat`.
- `INT_FLUX` : Integral flux sensitivity evaluated for PowerLaw sources with spectral indices between 1.0 and 5.0 at sky positition set by `glon` and `glat`. Columns starting with `ebin` contain the source amplitude vs. energy bin.
- `MAP_DIFF_FLUX` : Sky cube with differential flux threshold vs. sky position and energy.

- `MAP_DIFF_NPRED` : Sky cube with counts amplitude (NPred) of a source at the detection threshold vs. position and energy.
- `MAP_INT_FLUX` : Sky map with integral flux threshold vs. sky position. Integral sensitivity will be computed for a PowerLaw source with index equal to the `index` parameter.
- `MAP_INT_NPRED` : Sky map with counts amplitude (NPred) of a source at the detection threshold vs. sky position.

The output file can be read using the `Table` module:

```
from astropy.table import Table

tab = Table.read('lat_sensitivity.fits','DIFF_FLUX')
print(tab['e_min'], tab['e_max'], tab['flux'])
tab = Table.read('lat_sensitivity.fits','INT_FLUX')
print(tab['index'], tab['flux'])
```

1.3.9 Validation Tools

This page documents LAT validation tools.

Merit Skimmer

The `fermipy-merit-skimmer` script can be used to create skimmed Merit files (either MC or data) and serves as a replacement for the web-based merit skimmer tool. The script accepts as input a sequence of Merit file paths or lists of Merit file paths which can be either local (nfs) or xrootd.

```
$ fermipy-merit-skimmer merit_list.txt --output=merit.root --selection='FswGamFilter'␣
↪\
--aliases=aliases.yaml

$ fermipy-merit-skimmer merit_list.txt --output=merit-clean.root \
--selection='FswGamFilter && CLEAN' --aliases=EvtClassDefs_P8R2.xml
```

where `merit_list.txt` is a text file with one path per line. The `--selection` option sets the selection that will be applied when filtering events in each file. The `--output` option sets the path to the output merit file. The `--aliases` option can be used to load an alias file (set of key/value cut expression pairs). This option can accept either a YAML alias file or an XML event class definition file. The following illustrates the YAML alias file format:

```
FswGamFilter : FswGamState == 0
TracksCutFilter : FswGamState == 0 && TkrNumTracks > 0
CalEnergyFilter : FswGamState == 0 && CalEnergyRaw > 0
```

One can restrict the set of output branches with the `--branches` option which accepts an alias file or a YAML file containing a list of branch names. In the former case all branch names used in the given alias set will be extracted and added to the list of output branches.

```
$ fermipy-merit-skimmer merit_list.txt --output=merit.root --selection='FswGamFilter'␣
↪\
--aliases=EvtClassDefs_P8R2.xml --branches=EvtClassDefs_P8R2.xml
```

One can split the skimming task into separate batch jobs by running with the `--batch` option. This will subdivide task into N jobs when N is the number of files in the list divided by `--files_per_job`. The name of the output ROOT file of each job will be appended with the index of the job in the sequence (e.g. skim_000.root, skim_001.root, etc.). The `--time` and `--resources` options can be used to set the LSF wallclock time and resource flags.

```
$ fermipy-merit-skimmer merit_list.txt --output=merit.root --selection='SOURCE &&␣
↪FRONT' \
--branches=EvtClassDefs_P8R2.xml --files_per_job=1000 --batch --aliases=EvtClassDefs_
↪P8R2.xml
```

When skimming MC files it can be useful to extract the `jobinfo` for tracking the number of thrown events. The `--extra_trees` option can be used to copy one or more trees to the output file in addition to the Merit Tuple:

```
$ fermipy-merit-skimmer merit_list.txt --output=merit.root --extra_trees=jobinfo
```

1.3.10 fermipy package

fermipy.config module

class fermipy.config.**ConfigManager**
Bases: object

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

static load(*path*)

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

Class encapsulating a configuration schema.

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

add_section(*name*, *section*)

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

items()

class fermipy.config.**Configurable**(*config*, ***kwargs*)
Bases: object

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

config
Return the configuration dictionary of this class.

configdir

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

classmethod get_config()
Return a default configuration dictionary for this class.

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

schema
Return the configuration schema of this class.

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

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

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

fermipy.config.**update_from_schema**(*cfg*, *cfgin*, *schema*)
Update configuration dictionary `cfg` with the contents of `cfgin` using the `schema` dictionary to determine the valid input keys.

Parameters

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

Returns cfgout

Return type dict

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

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

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

fermipy.defaults module

fermipy.defaults.**make_attrs_class**(*typename*, *d*)

fermipy.defaults.**make_default_dict**(*d*)

fermipy.defaults.**make_default_tuple**(*d*)

fermipy.gtanalysis module

class fermipy.gtanalysis.**GTAnalysis**(*config*, *roi=None*, ***kwargs*)
Bases: *fermipy.config.Configurable*, *fermipy.sed.SEDGenerator*, *fermipy.residmap.ResidMapGenerator*, *fermipy.tsmap.TSMapGenerator*, *fermipy.tsmap.TSCubeGenerator*, *fermipy.sourcefind.SourceFind*, fermipy.extension.ExtensionFit, *fermipy.lightcurve.LightCurve*

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

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

add_source(*name*, *src_dict*, *free=None*, *init_source=True*, *save_source_maps=True*, *use_pylike=True*, *use_single_psf=False*, ***kwargs*)
Add a source to the ROI model. This function may be called either before or after *setup*.

Parameters

- **name** (*str*) – Source name.

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

add_sources_from_roi(*names*, *roi*, *free=False*, ***kwargs*)
Add multiple sources to the current ROI model copied from another ROI model.

Parameters

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

bowtie(*name*, *fd=None*, *loge=None*)
Generate a spectral uncertainty band (bowtie) for the given source. This will create an uncertainty band on the differential flux as a function of energy by propagating the errors on the global fit parameters. Note that this band only reflects the uncertainty for parameters that are currently free in the model.

Parameters

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

cleanup()

clone(*config*, ***kwargs*)
Make a clone of this analysis instance.

components
Return the list of analysis components.

config
Return the configuration dictionary of this class.

configdir

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

constrain_norms(*srcNames*, *cov_scale=1.0*)
Constrain the normalizations of one or more sources by adding gaussian priors with sigma equal to the parameter error times a scaling factor.

counts_map()
Return a *Map* representation of the counts map.

Returns map

Return type *Map*

classmethod 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.

create_roi_table()

curvature(*name*, ***kwargs*)

Test whether a source shows spectral curvature by comparing the likelihood ratio of PowerLaw and LogParabola spectral models.

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

defaults = {'binning': {'projtype': ('WCS', 'Projection mode (WCS or HPX).', <type '

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

Delete a source from the ROI model.

Parameters

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

Returns **src** – The deleted source object.

Return type *Model*

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

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

Parameters

- **cuts** (*dict*) – Dictionary of [min,max] selections on source properties.
- **distance** (*float*) – Cut on angular distance from `skydir`. If None then no selection will be applied.
- **skydir** (SkyCoord) – Reference sky coordinate for `distance` selection. If None then the distance selection will be applied with respect to the ROI center.
- **minmax_ts** (*list*) – Select 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*) – Select 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.
- **names** (*list*) – Select sources matching a name in this list.

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

Return type list

delete_workdir()

energies
Return the energy bin edges in MeV.

enumbins
Return the number of energy bins.

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

Parameters

- **name** (*str*) – Source name.
- **fit_ebin** (*bool*) – Perform a fit for the angular extension in each analysis energy bin. (default : False)
- **fit_position** (*bool*) – Perform a simultaneous fit to the source position and extension. (default : False)
- **fix_shape** (*bool*) – Fix spectral shape parameters of the source of interest. If True then only the normalization parameter will be fit. (default : False)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **make_tsmap** (*bool*) – Make a TS map for the source of interest. (default : True)
- **psf_scale_fn** (*tuple*) – Tuple of two vectors (logE,f) defining an energy-dependent PSF scaling function that will be applied when building spatial models for the source of interest. The tuple (logE,f) defines the fractional corrections f at the sequence of energies logE = log10(E/MeV) where f=0 corresponds to no correction. The correction function f(E) is evaluated by linearly interpolating the fractional correction factors f in log(E). The corrected PSF is given by P'(x;E) = P(x/(1+f(E));E) where x is the angular separation. (default : None)
- **save_model_map** (*bool*) – Save model counts cubes for the best-fit model of extension. (default : False)
- **spatial_model** (*str*) – Spatial model that will be used to test the sourceextension. The spatial scale parameter of the model will be set such that the 68% containment radius of the model is equal to the width parameter. (default : RadialGaussian)
- **sqrt_ts_threshold** (*float*) – Threshold on sqrt(TS_ext) that will be applied when `update` is True. If None then nothreshold is applied. (default : None)
- **update** (*bool*) – Update this source with the best-fit model for spatial extension if TS_ext > `tsext_threshold`. (default : False)

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

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

Return type dict

files

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

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

Parameters

- **free_params** (*list*) – (default : None)
- **max_iter** (*int*) – Maximum number of source finding iterations. The source finder will continue adding sources until no additional peaks are found or the number of iterations exceeds this number. (default : 5)
- **min_separation** (*float*) – Minimum separation in degrees between sources detected in each iteration. The source finder will look for the maximum peak in the TS map within a circular region of this radius. (default : 1.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across number of processes set by nthread option. (default : False)
- **nthread** (*int*) – Number of processes to create when multithread is True. If None then one process will be created for each available core. (default : None)
- **sources_per_iter** (*int*) – Maximum number of sources that will be added in each iteration. If the number of detected peaks in a given iteration is larger than this number, only the N peaks with the largest TS will be used as seeds for the current iteration. (default : 4)
- **sqrt_ts_threshold** (*float*) – Source threshold in sqrt(TS). Only peaks with sqrt(TS) exceeding this threshold will be used as seeds for new sources. (default : 5.0)

- **tsmap_fitter** (*str*) – Set the method for generating the TS map. Valid options are tsmap or tscube. (default : tsmap)
- **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/Fix a parameter of a source by name.

Parameters

- **name** (*str*) – Source name.
- **par** (*str*) – Parameter name.

free_shape(*name*, *free=True*, ***kwargs*)
Free/Fix shape parameters of a source.

Parameters

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

free_source(*name*, *free=True*, *pars=None*, ***kwargs*)
Free/Fix parameters of a source.

Parameters

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

free_sources(*free=True*, *pars=None*, *cuts=None*, *distance=None*, *skydir=None*, *minmax_ts=None*, *minmax_npred=None*, *exclude=None*, *square=False*, ***kwargs*)
Free or fix sources in the ROI model satisfying the given selection. When multiple selections are defined, the selected sources will be those satisfying the logical AND of all selections (e.g. distance < X && minmax_ts[0] < ts < minmax_ts[1] && ...).

Parameters

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

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

Return type list

free_sources_by_name(*names*, *free=True*, *pars=None*, ***kwargs*)
Free all sources with names matching `names`.

Parameters

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

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

Return type list

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

geom
ROI geometry.

classmethod get_config()
Return a default configuration dictionary for this class.

get_free_param_vector()

get_free_source_params(*name*)

get_norm(*name*)

get_params(*freeonly=False*)

get_source_dnde(*name*)
Return differential flux distribution of a source. For sources with FileFunction spectral type this returns the internal differential flux array.

Returns

- **loge** (ndarray) – Array of energies at which the differential flux is evaluated (log10(E/MeV)).
- **dnde** (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_source_params(*name*)

get_sources(*cuts=None*, *distance=None*, *skydir=None*, *minmax_ts=None*, *minmax_npred=None*, *exclude=None*, *square=False*)
Retrieve list of sources in the ROI satisfying the given selections.

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

Return type list

get_src_model(*name*, *paramsonly=False*, *reoptimize=False*, *npts=None*, ***kwargs*)
Compose a dictionary for a source with the current best-fit parameters.

Parameters

- **name** (*str*) –

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

Returns src_dict

Return type dict

lightcurve(*name*, ***kwargs*)

Generate a lightcurve for the named source. The function will complete the basic analysis steps for each bin and perform a likelihood fit for each bin. Extracted values (along with errors) are Integral Flux, spectral model, Spectral index, TS value, pred. # of photons. Note: successful calculation of TS:subscript:`var` requires at least one free background parameter and a previously optimized ROI model.

Parameters

- **name** (*str*) – source name
- **binsz** (*float*) – Set the lightcurve bin size in seconds. (default : 86400.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_params** (*list*) – Set the parameters of the source of interest that will be re-fit in each time bin. If this list is empty then all parameters will be freed. (default : None)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **free_sources** (*list*) – List of sources to be freed. These sources will be added to the list of sources satisfying the free_radius selection. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **max_free_sources** (*int*) – Maximum number of sources that will be fit simultaneously with the source of interest. (default : 5)
- **multithread** (*bool*) – Split the calculation across number of processes set by nthread option. (default : False)
- **nbins** (*int*) – Set the number of lightcurve bins. The total time range will be evenly split into this number of time bins. (default : None)
- **nthread** (*int*) – Number of processes to create when multithread is True. If None then one process will be created for each available core. (default : None)
- **outdir** (*str*) – Store all data in this directory (e.g. “30days”). If None then use current directory. (default : None)
- **save_bin_data** (*bool*) – Save analysis directories for individual time bins. If False then only the analysis results table will be saved. (default : True)
- **shape_ts_threshold** (*float*) – Set the TS threshold at which shape parameters of sources will be freed. If a source is detected with TS less than this value then its shape parameters will be fixed to values derived from the analysis of the full time range. (default : 16.0)
- **systematic** (*float*) – Systematic correction factor for TS:subscript:`var`. See Sect. 3.6 in 2FGL for details. (default : 0.02)

- **time_bins** (*list*) – Set the lightcurve bin edge sequence in MET. This option takes precedence over binsz and nbins. (default : None)
- **use_local_ltcube** (*bool*) – Generate a fast LT cube. (default : True)
- **use_scaled_srcmap** (*bool*) – Generate approximate source maps for each time bin by scaling the current source maps by the exposure ratio with respect to that time bin. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

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

Return type dict

like
Return the global likelihood object.

load_parameters_from_yaml (*yamlfile*, *update_sources=False*)
Load model parameters from yaml

Parameters **yamlfile** (*str*) – Name of the input yaml file.

load_roi (*infile*, *reload_sources=False*)
This function reloads the analysis state from a previously saved instance generated with *write_roi*.

Parameters

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

load_xml (*xmlfile*)
Load model definition from XML.

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

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

Parameters

- **name** (*str*) – Source name.
- **dtheta_max** (*float*) – Half-width of the search region in degrees used for the first pass of the localization search. (default : 0.5)
- **fix_shape** (*bool*) – Fix spectral shape parameters of the source of interest. If True then only the normalization parameter will be fit. (default : False)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)

- **nstep** (*int*) – Number of steps in longitude/latitude that will be taken when refining the source position. The bounds of the scan range are set to the 99% positional uncertainty as determined from the TS map peak fit. The total number of sampling points will be nstep**2. (default : 5)
- **update** (*bool*) – Update the source model with the best-fit position. (default : True)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

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

Return type dict

lock_parameter(*name*, *par*, *lock=True*)
Set parameter to locked/unlocked state. A locked parameter will be ignored when running methods that free/fix sources or parameters.

Parameters

- **name** (*str*) – Source name.
- **par** (*str*) – Parameter name.
- **lock** (*bool*) – Set parameter to locked (True) or unlocked (False) state.

lock_source(*name*, *lock=True*)
Set all parameters of a source to a locked/unlocked state. Locked parameters will be ignored when running methods that free/fix sources or parameters.

Parameters

- **name** (*str*) – Source name.
- **lock** (*bool*) – Set source parameters to locked (True) or unlocked (False) state.

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

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

logger
Return the default loglevel.

loglevel
Return the default loglevel.

make_plots(*prefix*, *mcube_map=None*, ***kwargs*)
Make diagnostic plots using the current ROI model.

make_template(*src*)

model_counts_map(*name=None*, *exclude=None*, *use_mask=False*)
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.
- **use_mask** (*bool*) – Parameter that specifies in the model counts map should include mask pixels (i.e., ones whose weights are <= 0)

Returns map

Return type Map

model_counts_spectrum(*name*, *logemin=None*, *logemax=None*, *summed=False*, *weighted=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. If weighted=True return the weighted version of the counts spectrum

npix

Return the number of energy bins.

optimize(***kwargs*)

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

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

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

Parameters

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

outdir

Return the analysis output directory.

plotter

Return the plotter instance.

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

print_model(*loglevel=20*)

print_params (*allpars=False*, *loglevel=20*)
Print information about the model parameters (values, errors, bounds, scale).

print_roi (*loglevel=20*)
Print information about the spectral and spatial properties of the ROI (sources, diffuse components).

profile (*name*, *parName*, *logemin=None*, *logemax=None*, *reoptimize=False*, *xvals=None*, *npts=None*, *savestate=True*, ***kwargs*)
Profile the likelihood for the given source and parameter.

Parameters

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

Returns **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*, *init_source=True*)
Delete and reload a source in the model. This will update the spatial model of this source to the one defined in the XML model.

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

remove_prior (*name*, *parName*)

remove_priors ()
Clear all priors.

residmap (*prefix=''*, ***kwargs*)
Generate 2-D spatial residual maps using the current ROI model and the convolution kernel defined with the `model` argument.

Parameters

- **prefix** (*str*) – String that will be prefixed to the output residual map files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the residual map. (default : None)
- **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. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)

- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **use_weights** (*bool*) – Used weighted version of maps in making plots. (default : False)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

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

Return type dict

roi
Return the ROI object.

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

schema
Return the configuration schema of this class.

sed(*name*, ***kwargs*)
Generate a spectral energy distribution (SED) for a source. This function will fit the normalization of the source in each energy bin. By default the SED will be generated with the analysis energy bins but a custom binning can be defined with the `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. (default : 2.0)
- **cov_scale** (*float*) – Scale factor that sets the strength of the prior on nuisance parameters that are free. Setting this to None disables the prior. (default : 3.0)
- **free_background** (*bool*) – Leave background parameters free when performing the fit. If True then any parameters that are currently free in the model will be fit simultaneously with the source of interest. (default : False)
- **free_pars** (*list*) – Set the parameters of the source of interest that will be freed when performing the global fit. By default all parameters will be freed. (default : None)
- **free_radius** (*float*) – Free normalizations of background sources within this angular distance in degrees from the source of interest. If None then no sources will be freed. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **ul_confidence** (*float*) – Confidence level for flux upper limit. (default : 0.95)
- **use_local_index** (*bool*) – Use a power-law approximation to the shape of the global spectrum in each bin. If this is false then a constant index set to `bin_index` will be used. (default : False)

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

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

Return type dict

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*, *error=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.
- **error** (*float*) – Parameter error (optional). By default this argument should be the unscaled (True) parameter value.
- **update_source** (*bool*) – Update the source dictionary for the object.

set_parameter_bounds(*name*, *par*, *bounds*)
Set the bounds on the scaled value of a parameter.

Parameters

- **name** (*str*) – Source name.
- **par** (*str*) – Parameter name.

- **bounds** (*list*) – Upper and lower bound.

set_parameter_error(*name*, *par*, *error*)
Set the error on the value of a parameter.

Parameters

- **name** (*str*) – Source name.
- **par** (*str*) – Parameter name.
- **error** (*float*) – The value for the parameter error

set_parameter_scale(*name*, *par*, *scale*)
Update the scale of a parameter while keeping its value constant.

set_random_seed(*seed*)
Set the seed for the random number generator

set_source_dnde(*name*, *dnde*, *update_source=True*)
Set the differential flux distribution of a source with the FileFunction spectral type.

Parameters

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

set_source_morphology(*name*, ***kwargs*)
Set the spatial model of a source.

Parameters

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

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

Parameters

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

set_weights_map(*wmap*, *update_roi=True*)

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

Parameters

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

simulate_roi(*name=None*, *randomize=True*, *restore=False*)
Generate a simulation of the ROI using the current best-fit model and replace the data counts cube with this simulation. The simulation is created by generating an array of Poisson random numbers with expectation values drawn from the model cube of the binned analysis instance. This function will update the counts cube both in memory and in the source map file. The counts cube can be restored to its original state by calling this method with `restore` = True.

Parameters

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

simulate_source(*src_dict=None*)
Inject simulated source counts into the data.

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

stage_input()
Copy input files to working directory.

stage_output()
Copy data products to final output directory.

tmax
Return the MET time for the end of the observation.

tmin
Return the MET time for the start of the observation.

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

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **do_sed** (*bool*) – Compute the energy bin-by-bin fits.
- **nnorm** (*int*) – Number of points in the likelihood v. normalization scan.

- **norm_sigma** (*float*) – Number of sigma to use for the scan range.
- **tol** (*float*) – 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=''*, ***kwargs*)

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

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

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files.
- **exclude** (*list*) – List of sources that will be removed from the model when computing the TS map. (default : None)
- **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. (default : None)
- **make_plots** (*bool*) – Generate diagnostic plots. (default : False)
- **max_kernel_radius** (*float*) – Set the maximum radius of the test source kernel. Using a smaller value will speed up the TS calculation at the loss of accuracy. (default : 3.0)
- **model** (*dict*) – Dictionary defining the spatial/spectral properties of the test source. If model is None the test source will be a PointSource with an Index 2 power-law spectrum. (default : None)
- **multithread** (*bool*) – Split the calculation across number of processes set by nthread option. (default : False)
- **nthread** (*int*) – Number of processes to create when multithread is True. If None then one process will be created for each available core. (default : None)
- **write_fits** (*bool*) – Write the output to a FITS file. (default : True)
- **write_npy** (*bool*) – Write the output dictionary to a numpy file. (default : True)

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

Return type dict

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

update_source(*name*, *paramsonly=False*, *reoptimize=False*, ***kwargs*)
Update the dictionary for this source.

Parameters

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

weight_map()
Return a *Map* representation of the weights map.

Returns map

Return type *Map*

workdir
Return the analysis working directory.

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

write_fits(*fitsfile*)

write_model_map(*model_name*, *name=None*)
Save the counts model map to a FITS file.

Parameters

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

write_roi(*outfile=None*, *save_model_map=False*, ***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.

write_weight_map(*model_name*)
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.

write_xml(*xmlfile*)
Save current model definition as XML file.

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

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

fermipy.logger module

class fermipy.logger.**Logger**
Bases: object

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

static configure(*name*, *logfile*, *loglevel=10*)
Create a python logger instance and configure it.

Parameters

- **name** (*str*) – Logger name.
- **logfile** (*str*) – Path to the log file.
- **loglevel** (*int*) – Default log level for STDOUT.

static setup(*config=None*, *logfile=None*)
This method sets up the default configuration of the logger. Once this method is called all subsequent instances Logger instances will inherit this configuration.

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

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

close()

flush()

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

fermipy.logger.**log_level**(*level*)
This is a function that returns a python like level from a HEASOFT like level.

fermipy.roi_model module

class fermipy.roi_model.**CompositeSource**(*name*, *data*)
Bases: *fermipy.roi_model.Model*

diffuse

nested_sources

write_xml(*root*)

class fermipy.roi_model.**IsoSource**(*name*, *data*)
Bases: *fermipy.roi_model.Model*

diffuse

filefunction

write_xml(*root*, ***kwargs*)

class fermipy.roi_model.**MapCubeSource**(*name*, *data*)
Bases: *fermipy.roi_model.Model*

diffuse

mapcube

write_xml(*root*, ***kwargs*)

class fermipy.roi_model.**Model**(*name*, *data*)
Bases: object

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

add_name(*name*)

add_to_table(*tab*)

assoc

check_cuts(*cuts*)

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

data

get_catalog_dict()

get_norm()

is_free
returns True if any of the spectral model parameters is set to free, else False

items()

name

names

params

psf_scale_fn

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

set_psf_scale_fn(*fn*)

set_spectral_pars(*spectral_pars*)

spatial_pars

spectral_pars

update_data(*d*)

update_from_source(*src*)

update_spectral_pars(*spectral_pars*)

class fermipy.roi_model.**ROIModel**(*config=None*, ***kwargs*)
Bases: *fermipy.config.Configurable*

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

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

```
>>> skydir = astropy.coordinates.SkyCoord(0.0,0.0,unit='deg')
>>> roi = ROIModel({'catalogs' : ['3FGL'],'src_roiwidth' : 10.0},
↪skydir=skydir)
>>> print(roi)
name                SpatialModel   SpectrumType     offset        ts
↪ npred
--------------------------------------------------------------------------
↪------
```

(continues on next page)

(continued from previous page)

```
3FGL J2357.3-0150   PointSource    PowerLaw          1.956       nan
↪   0.0
3FGL J0006.2+0135   PointSource    PowerLaw          2.232       nan
↪   0.0
3FGL J0016.3-0013   PointSource    PowerLaw          4.084       nan
↪   0.0
3FGL J0014.3-0455   PointSource    PowerLaw          6.085       nan
↪   0.0
```

- Print a summary of an individual source

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

- Get the SkyCoord for a source

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

- Loop over all sources and print their names

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

clear()
Clear the contents of the ROI.

copy_source(*name*)

classmethod create(*selection*, *config*, ***kwargs*)
Create an ROIModel instance.

create_diffuse_srcs(*config*)

classmethod 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.

classmethod create_from_roi_data(*datafile*)
Create an ROI model.

classmethod create_from_source(*name*, *config*, ***kwargs*)
Create an ROI centered on the given source.

create_param_table()

classmethod create_roi_from_ft1(*ft1file*, *config*)
Create an ROI model by extracting the sources coordinates form an FT1 file.

create_source(*name*, *src_dict*, *build_index=True*, *merge_sources=True*, *rescale=True*)
Add a new source to the ROI model from a dictionary or an existing source object.

Parameters

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

Returns **src**

Return type *Source*

create_source_table()

create_table(*names=None*)
Create an astropy Table object with the contents of the ROI model.

defaults = {'assoc_xmatch_columns': (['3FGL_Name'], 'Choose a set of association colu

delete_sources(*srcs*)

diffuse_sources

extdir

geom

get_nearby_sources(*name*, *distance*, *min_dist=None*, *square=False*)

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

Parameters **name** (*str*) – Name string.

Returns **srcs** – A source object.

Return type *Model*

get_sources(*skydir=None*, *distance=None*, *cuts=None*, *minmax_ts=None*, *minmax_npred=None*, *exclude=None*, *square=False*, *coordsys='CEL'*, *names=None*)
Retrieve list of source objects satisfying the following selections:

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

Sources can be excluded from the selection by adding their name to the `exclude` list.

Returns **srcs** – List of source objects.

Return type list

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

Parameters **name** (*str*) –

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

Return type list

get_sources_by_position(*skydir*, *dist*, *min_dist=None*, *square=False*, *coordsys='CEL'*)
Retrieve sources within a certain angular distance of a sky coordinate. This function supports two types of geometric selections: circular (square=False) and square (square=True). The circular selection finds all sources with a given angular distance of the target position. The square selection finds sources within an ROI-like region of size R 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_existing_catalog(*cat*, ***kwargs*)
Load sources from an existing catalog object.

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

load_fits_catalog(*name*, ***kwargs*)
Load sources from a FITS catalog file.

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

load_source(*src*, *build_index=True*, *merge_sources=True*, ***kwargs*)
Load a single source.

Parameters

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

load_sources(*sources*)
Delete all sources in the ROI and load the input source list.

load_xml(*xmlfile*, ***kwargs*)
Load sources from an XML file.

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

point_sources

set_geom(*geom*)

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

sources

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

to_ds9(*free='box'*, *fixed='cross'*, *frame='fk5'*, *color='green'*, *header=True*)
Returns a list of ds9 region definitions :param free: one of the supported ds9 point symbols, used for free sources, see here: http://ds9.si.edu/doc/ref/region.html :type free: bool :param fixed: as free but for fixed sources :type fixed: bool :param frame: typically fk5, more to be implemented :type frame: str :param color: color used for symbols (only ds9 compatible colors) :type color: str :param header: if True, will prepend a global header line. :type header: bool

Returns lines – list of regions (and header if requested)

Return type list

write_ds9region(*region*, **args*, ***kwargs*)
Create a ds9 compatible region file from the ROI.

It calls the *to_ds9* method and write the result to the region file. Only the file name is required. All other parameters will be forwarded to the *to_ds9* method, see the documentation of that method for all accepted parameters and options. :param region: name of the region file (string) :type region: str

write_fits(*fitsfile*)
Write the ROI model to a FITS file.

write_xml(*xmlfile*, *config=None*)
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 >>> src['ts']

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

associations

classmethod create_from_dict(*src_dict*, *roi_skydir=None*, *rescale=False*)
Create a source object from a python dictionary.

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

static create_from_xml(*root*, *extdir=None*)
Create a Source object from an XML node.

Parameters

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

classmethod create_from_xmlfile(*xmlfile*, *extdir=None*)
Create a Source object from an XML file.

Parameters

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

data

diffuse

extended

radec

separation(*src*)

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

Parameters **skydir** (SkyCoord) –

set_radec(*ra*, *dec*)

set_roi_direction(*roidir*)

set_roi_geom(*geom*)

set_spatial_model(*spatial_model*, *spatial_pars*)

skydir
Return a SkyCoord representation of the source position.

Returns **skydir**

Return type SkyCoord

update_data(*d*)

write_xml(*root*)
Write this source to an XML node.

fermipy.roi_model.**create_source_table**(*scan_shape*)
Create an empty source table.

Returns **tab**

Return type Table

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

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

fermipy.roi_model.**get_skydir_distance_mask**(*src_skydir*, *skydir*, *dist*, *min_dist=None*, *square=False*, *coordsys='CEL'*)
Retrieve sources within a certain angular distance of an (ra,dec) coordinate. This function supports two types of geometric selections: circular (square=False) and square (square=True). The circular selection finds all sources with a given angular distance of the target position. The square selection finds sources within an ROI-like region of size R 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.**get_true_params_dict**(*pars_dict*)

fermipy.roi_model.**spatial_pars_from_catalog**(*cat*)

fermipy.roi_model.**spectral_pars_from_catalog**(*cat*)
Create spectral parameters from 3FGL catalog columns.

fermipy.utils module

fermipy.utils.**angle_to_cartesian**(*lon*, *lat*)
Convert spherical coordinates to cartesian unit vectors.

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

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

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

fermipy.utils.**collect_dirs**(*path*, *max_depth=1*, *followlinks=True*)
Recursively find directories under the given path.

fermipy.utils.**convolve2d_disk**(*fn*, *r*, *sig*, *nstep=200*)
Evaluate the convolution f'(r) = f(r) * g(r) where f(r) is azimuthally symmetric function in two dimensions and g is a step function given by:

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

Parameters

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

fermipy.utils.**convolve2d_gauss**(*fn*, *r*, *sig*, *nstep=200*)
Evaluate the convolution f'(r) = f(r) * g(r) where f(r) is azimuthally symmetric function in two dimensions and g is a 2D gaussian with standard deviation s given by:

g(r) = 1/(2*pi*s^2) 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*)
Compute the correlation matrix given the covariance matrix.

Parameters **cov** (ndarray) – N x N matrix of covariances among N parameters.

Returns **corr** – N x N matrix of correlations among N parameters.

Return type ndarray

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

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

fermipy.utils.**create_kernel_function_lookup**(*psf*, *fn*, *sigma*, *egy*, *dtheta*, *psf_scale_fn*)

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_radial_spline**(*psf*, *fn*, *sigma*, *egy*, *dtheta*, *psf_scale_fn*)

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

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

fermipy.utils.**dot_prod**(*xyz0*, *xyz1*)
Compute the dot product between two cartesian vectors where the second dimension contains the vector components.

fermipy.utils.**edge_to_center**(*edges*)

fermipy.utils.**edge_to_width**(*edges*)

fermipy.utils.**ellipse_to_cov**(*sigma_maj*, *sigma_min*, *theta*)
Compute the covariance matrix in two variables x and y given the std. deviation along the semi-major and semi-minor axes and the rotation angle of the error ellipse.

Parameters

- **sigma_maj** (*float*) – Std. deviation along major axis of error ellipse.
- **sigma_min** (*float*) – Std. deviation along minor axis of error ellipse.
- **theta** (*float*) – Rotation angle in radians from x-axis to ellipse major axis.

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

fermipy.utils.**eval_radial_kernel**(*psf*, *fn*, *sigma*, *idx*, *dtheta*, *psf_scale_fn*)

fermipy.utils.**extend_array**(*edges*, *binsz*, *lo*, *hi*)
Extend an array to encompass lo and hi values.

fermipy.utils.**find_function_root**(*fn*, *x0*, *xb*, *delta=0.0*, *bounds=None*)
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.**find_rows_by_string**(*tab, names, colnames=['assoc']*)
Find the rows in a table `tab` that match at least one of the strings in `names`. This method ignores whitespace and case when matching strings.

Parameters

- **tab** (`astropy.table.Table`) – Table that will be searched.
- **names** (*list*) – List of strings.

- **colname** (*str*) – Name of the table column that will be searched for matching string.

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

Return type ndarray

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

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

Parameters

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

fermipy.utils.**fits_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_bounded_slice**(*idx*, *dpix*, *shape*)

fermipy.utils.**get_parameter_limits**(*xval*, *loglike*, *cl_limit=0.95*, *cl_err=0.68269*, *tol=0.01*, *bounds=None*)

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 cubic 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.
- **cl_limit** (*float*) – Confidence level to use for limit calculation.
- **cl_err** (*float*) – Confidence level to use for two-sided confidence interval calculation.
- **tol** (*float*) – Absolute precision of likelihood values.

Returns

- **x0** (*float*) – Coordinate at maximum of likelihood function.
- **err_lo** (*float*) – Lower error for two-sided confidence interval with CL `cl_err`. Corresponds to point (x < x0) at which the log-likelihood falls by a given value with respect to the maximum (0.5 for 1 sigma). Set to nan if the change in the log-likelihood function at the lower bound of the `xval` input array is less than than the value for the given CL.
- **err_hi** (*float*) – Upper error for two-sided confidence interval with CL `cl_err`. Corresponds to point (x > x0) at which the log-likelihood falls by a given value with respect to the maximum (0.5 for 1 sigma). Set to nan if the change in the log-likelihood function at the upper bound of the `xval` input array is less than the value for the given CL.
- **err** (*float*) – Symmetric 1-sigma error. Average of `err_lo` and `err_hi` if both are defined.

- **ll** (*float*) – Lower limit evaluated at confidence level `cl_limit`.
- **ul** (*float*) – Upper limit evaluated at confidence level `cl_limit`.
- **lnlmax** (*float*) – Log-likelihood value at `x0`.

`fermipy.utils.`**`get_region_mask`**(*z*, *delta*, *xy=None*)
Get mask of connected region within delta of max(z).

`fermipy.utils.`**`init_matplotlib_backend`**(*backend=None*)
This function initializes the matplotlib backend. When no DISPLAY is available the backend is automatically set to 'Agg'.

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

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

`fermipy.utils.`**`is_fits_file`**(*path*)

`fermipy.utils.`**`isstr`**(*s*)
String instance testing method that works under both Python 2.X and 3.X. Returns true if the input is a string.

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

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

`fermipy.utils.`**`load_npy`**(*infile*)

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

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

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

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

Parameters

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

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

Parameters

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

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

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

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

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

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

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

Parameters

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

fermipy.utils.**make_radial_kernel**(*psf*, *fn*, *sigma*, *npix*, *cdelt*, *xpix*, *ypix*, *psf_scale_fn=None*, *normalize=False*, *klims=None*, *sparse=False*)
Make a kernel for a general radially symmetric 2D function.

Parameters

- **psf** (PSFModel) –
- **fn** (*callable*) – Function that evaluates the kernel at a radial coordinate r.
- **sigma** (*float*) – 68% containment radius 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.**memoize**(*obj*)

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.**merge_list_of_dicts**(*listofdicts*)

fermipy.utils.**met_to_mjd**(*time*)
"Convert mission elapsed time to mean julian date.

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

fermipy.utils.**onesided_cl_to_dlnl**(*cl*)
Compute the delta-loglikehood values that corresponds to an upper limit of the given confidence level.

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

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

Return type float

fermipy.utils.**onesided_dlnl_to_cl**(*dlnl*)
Compute the confidence level that corresponds to an upper limit with a given change in the loglikelihood value.

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

Returns **cl** – Confidence level.

Return type float

fermipy.utils.**overlap_slices**(*large_array_shape*, *small_array_shape*, *position*)
Modified version of overlap_slices.

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

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

Parameters

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

Returns

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

fermipy.utils.**parabola**(*xy*, *amplitude*, *x0*, *y0*, *sx*, *sy*, *theta*)
Evaluate a 2D parabola given by:

f(x,y) = f_0 - (1/2) * delta^T * R * Sigma * R^T * delta

where

delta = [(x - x_0), (y - y_0)]

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

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

Parameters

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

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

Return type ndarray

fermipy.utils.**path_to_xmlpath**(*path*)

fermipy.utils.**poly_to_parabola**(*coeff*)

fermipy.utils.**prettify_xml**(*elem*)
Return a pretty-printed XML string for the Element.

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

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

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

fermipy.utils.**resolve_file_path_list**(*pathlist*, *workdir*, *prefix=''*, *randomize=False*)
Resolve the path of each file name in the file `pathlist` and write the updated paths to a new file.

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

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

fermipy.utils.**separation_cos_angle**(*lon0*, *lat0*, *lon1*, *lat1*)
Evaluate the cosine of the angular separation between two direction vectors.

fermipy.utils.**split_bin_edges**(*edges*, *npts=2*)
Subdivide an array of bins by splitting each bin into `npts` subintervals.

Parameters

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

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

Return type ndarray

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

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

fermipy.utils.**tolist**(*x*)
convenience function that takes in a nested structure of lists and dictionaries and converts everything to its base objects. This is useful for 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.**twosided_cl_to_dlnl**(*cl*)
Compute the delta-loglikehood value that corresponds to a two-sided interval of the given confidence level.

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

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

Return type float

fermipy.utils.**twosided_dlnl_to_cl**(*dlnl*)
Compute the confidence level that corresponds to a two-sided interval with a given change in the loglikelihood value.

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

Returns **cl** – Confidence level.

Return type float

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

fermipy.utils.**unicode_to_str**(*args*)

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

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

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

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

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

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

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

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

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

fermipy.plotting module

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

defaults = {'catalogs': (None, '', <type 'list'>), 'cmap': ('magma', 'Set the colorma

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

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

make_residmap_plots(*maps*, *roi=None*, ***kwargs*)
Make plots from the output of *residmap*.

Parameters

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

make_roi_plots(*gta*, *mcube_tot*, ***kwargs*)
Make various diagnostic plots for the 1D and 2D counts/model distributions.

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

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

make_tsmap_plots(*maps*, *roi=None*, ***kwargs*)
Make plots from the output of *tsmap* or *tscube*. This method generates a 2D sky map for the best-fit test source in sqrt(TS) and Npred.

Parameters

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

run(*gta*, *mcube_map*, ***kwargs*)
Make all plots.

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

plot(*iaxis*)

class fermipy.plotting.**ImagePlotter**(*img*, *mapping=None*)
Bases: object

geom

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

projtype

class fermipy.plotting.**ROIPlotter**(*data_map*, *hpx2wcs=None*, ***kwargs*)
Bases: *fermipy.config.Configurable*

classmethod create_from_fits(*fitsfile*, *roi*, ***kwargs*)

data

defaults = {'catalogs': (None, '', <type 'list'>), 'cmap': ('ds9_b', '', <type 'str'

draw_circle(*radius*, ***kwargs*)

geom

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

map

plot(***kwargs*)

plot_catalog(*catalog*)

plot_projection(*iaxis*, ***kwargs*)

plot_roi(*roi*, ***kwargs*)

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

proj

projtype

static setup_projection_axis(*iaxis*, *loge_bounds=None*)

zoom(*zoom*)

class fermipy.plotting.**SEDPlotter**(*sed*)

Bases: object

static get_ylims(*sed*)

plot(*showlnl=False*, ***kwargs*)

static plot_flux_points(*sed*, ***kwargs*)

static plot_lnlscan(*sed*, ***kwargs*)

static plot_model(*model_flux*, ***kwargs*)

static plot_resid(*src*, *model_flux*, ***kwargs*)

static plot_sed(*sed*, *showlnl=False*, ***kwargs*)

Render a plot of a spectral energy distribution.

Parameters

- **showlnl** (*bool*) – Overlay a map of the delta-loglikelihood values vs. flux in each energy bin.
- **cmap** (*str*) – Colormap that will be used for the delta-loglikelihood map.
- **llhcut** (*float*) – Minimum delta-loglikelihood value.
- **ul_ts_threshold** (*float*) – TS threshold that determines whether the MLE or UL is plotted in each energy bin.

sed

fermipy.plotting.**annotate**(***kwargs*)

fermipy.plotting.**annotate_name**(*data*, *xy=(0.05*, *0.93)*, ***kwargs*)

fermipy.plotting.**get_xerr**(*sed*)

fermipy.plotting.**load_bluered_cmap**()

fermipy.plotting.**load_ds9_cmap**()

fermipy.plotting.**make_counts_spectrum_plot**(*o*, *roi*, *energies*, *imfile*, ***kwargs*)

fermipy.plotting.**make_cube_slice**(*map_in*, *loge_bounds*)

Extract a slice from a map cube object.

fermipy.plotting.**plot_error_ellipse**(*fit*, *xy*, *cdelt*, ***kwargs*)

fermipy.plotting.**plot_markers**(*lon*, *lat*, ***kwargs*)

fermipy.plotting.**truncate_colormap**(*cmap*, *minval=0.0*, *maxval=1.0*, *n=256*)

Function that extracts a subset of a colormap.

fermipy.sed module

Utilities for dealing with SEDs

Many parts of this code are taken from dsphs/like/lnlfn.py by Matthew Wood <mdwood@slac.stanford.edu>
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.
- **{options}** –
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

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

Return type dict

fermipy.sourcefind module

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

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

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

Parameters

- **{options}** –
- **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.
- **{options}** –
- **optimizer** (*dict*) – Dictionary that overrides the default optimizer settings.

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

Return type dict

fermipy.spectrum module

class fermipy.spectrum.**DMFitFunction**(*params*, *chan='bb'*, *jfactor=1e+19*, *tablepath=None*, *dfactor=1e+17*)

Bases: *fermipy.spectrum.SpectralFunction*

Class that evaluates the spectrum for a DM particle of a given mass, channel, cross section, and J-factor. The parameterization is given by:

F(x) = 1 / (8 * pi) * (1/mass^2) * sigmav * J * dN/dE(E,mass,i)

where the `params` array should be defined with:

- params[0] : sigmav (or tau for decay)
- params[1] : mass

Note that this class assumes that mass and J-factor are provided in units of GeV and GeV^2 cm^-5 while energies are defined in MeV.

For decay the D-factor is in uits of GeV cm^-2 s

ann_channel_names

chan

Return the channel string.

chan_code

Return the channel code.

channel_index_mapping = {1: 8, 2: 6, 3: 3, 4: 1, 5: 2, 6: 7, 7: 4, 8: 5, 9: 0

channel_name_mapping = {1: ['e+e-', 'ee'], 2: ['mu+mu-', 'mumu', 'musrc'], 3: ['tau

channel_rev_map = {'W+W-': 7, 'W+W-_decay': 107, 'ZZ': 8, 'ZZ_decay': 108, 'bb': 4

channel_shortname_mapping = {1: 'ee', 2: 'mumu', 3: 'tautau', 4: 'bb', 5: 'tt', 6

static channels()

Return all available DMFit channel strings

decay

Return True if this is a decay spectrum

decay_channel_names

static nparam()

set_channel(*chan*)

class fermipy.spectrum.**LogParabola**(*params=None*, *scale=1.0*, *extra_params=None*)
Bases: *fermipy.spectrum.SpectralFunction*

Class that evaluates a function with the parameterization:

F(x) = p_0 * (x/x_s)^(p_1 - p_2*log(x/x_s))

where x_s is a scale parameter. The `params` array should be defined with:

- params[0] : Prefactor (p_0)
- params[1] : Index (p_1)
- params[2] : Curvature (p_2)

static nparam()

class fermipy.spectrum.**PLExpCutoff**(*params=None*, *scale=1.0*, *extra_params=None*)
Bases: *fermipy.spectrum.SpectralFunction*

Class that evaluates a function with the parameterization:

F(x) = p_0 * (x/x_s)^(p_1) * exp(- x/p_2)

where x_s is the scale parameter. The `params` array should be defined with:

- params[0] : Prefactor (p_0)
- params[1] : Index (p_1)
- params[2] : Cutoff (p_2)

static log_to_params(*params*)

static nparam()

static params_to_log(*params*)

class fermipy.spectrum.**PLSuperExpCutoff**(*params=None*, *scale=1.0*, *extra_params=None*)
Bases: *fermipy.spectrum.SpectralFunction*

Class that evaluates a function with the parameterization:

F(x) = p_0 * (x/x_s)^(p_1) * exp(-(x/p2)**p3)

where x_s is the scale parameter. The `params` array should be defined with:

- params[0] : Prefactor (p_0)
- params[1] : Index1 (p_1)
- params[2] : Curvature (p_2)
- params[3] : Index2 (p3)

static log_to_params(*params*)

static nparam()

static params_to_log(*params*)

class fermipy.spectrum.**PowerLaw**(*params=None*, *scale=1.0*, *extra_params=None*)
Bases: *fermipy.spectrum.SpectralFunction*

Class that evaluates a power-law function with the parameterization:

F(x) = p_0 * (x/x_s)^p_1

where x_s is the scale parameter. The `params` array should be defined with:

- params[0] : Prefactor (p_0)
- params[1] : Index (p_1)

classmethod eval_eflux(*emin*, *emax*, *params*, *scale=1.0*, *extra_params=None*)

static eval_flux(*emin*, *emax*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_norm(*scale*, *index*, *emin*, *emax*, *flux*)

static nparam()

class fermipy.spectrum.**SEDEFluxFunctor**(*sfn*, *emin*, *emax*)
Bases: *fermipy.spectrum.SEDFunctor*

Functor that computes the energy flux of a source in a pre-defined sequence of energy bins.

class fermipy.spectrum.**SEDFluxFunctor**(*sfn*, *emin*, *emax*)
Bases: *fermipy.spectrum.SEDFunctor*

Functor that computes the flux of a source in a pre-defined sequence of energy bins.

class fermipy.spectrum.**SEDFunctor**(*sfn*, *emin*, *emax*)
Bases: object

Functor object that wraps a *SpectralFunction* and computes the normalization of the model in a sequence of SED energy bins. The evaluation method of this class accepts a single vector for the parameters of the model. This class serves as an object that can be passed to likelihood optimizers.

emax

emin

params

scale

spectral_fn

class fermipy.spectrum.**SpectralFunction**(*params*, *scale=1.0*, *extra_params=None*)
Bases: object

Base class for spectral models. Spectral models inheriting from this class should implement at a minimum an `_eval_dnde` method which evaluates the differential flux at a given energy.

classmethod create_eflux_functor(*emin*, *emax*, *params=None*, *scale=1.0*, *extra_params=None*)

classmethod create_flux_functor(*emin*, *emax*, *params=None*, *scale=1.0*, *extra_params=None*)

classmethod create_from_eflux(*params*, *emin*, *emax*, *eflux*, *scale=1.0*)
Create a spectral function instance given its energy flux.

classmethod create_from_flux(*params*, *emin*, *emax*, *flux*, *scale=1.0*)
Create a spectral function instance given its flux.

classmethod create_functor(*spec_type*, *func_type*, *emin*, *emax*, *params=None*, *scale=1.0*, *extra_params=None*)

dnde(*x*, *params=None*)
Evaluate differential flux.

dnde_deriv(*x*, *params=None*)
Evaluate derivative of the differential flux with respect to E.

e2dnde(*x*, *params=None*)
Evaluate E^2 times differential flux.

e2dnde_deriv(*x*, *params=None*)
Evaluate derivative of E^2 times differential flux with respect to E.

ednde(*x*, *params=None*)
Evaluate E times differential flux.

ednde_deriv(*x*, *params=None*)
Evaluate derivative of E times differential flux with respect to E.

eflux(*emin*, *emax*, *params=None*)
Evaluate the integral energy flux.

classmethod eval_dnde(*x*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_dnde_deriv(*x*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_e2dnde(*x*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_e2dnde_deriv(*x*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_ednde(*x*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_ednde_deriv(*x*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_eflux(*emin*, *emax*, *params*, *scale=1.0*, *extra_params=None*)

classmethod eval_flux(*emin*, *emax*, *params*, *scale=1.0*, *extra_params=None*)

extra_params
Dictionary containing additional parameters needed for evaluation of the function.

flux(*emin*, *emax*, *params=None*)
Evaluate the integral flux.

log_params
Return transformed parameter vector in which norm and scale parameters are converted to log10.

params
Return parameter vector of the function.

scale

fermipy.spectrum.**cast_args**(*x*)

fermipy.spectrum.**cast_params**(*params*)

fermipy.skymap module

class fermipy.skymap.**HpxMap**(*counts*, *hpx*)
Bases: *fermipy.skymap.Map_Base*

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

convert_to_cached_wcs(*hpx_in*, *sum_ebins=False*, *normalize=True*)
Make a WCS object and convert HEALPix data into WCS projection

Parameters

- **hpx_in** (ndarray) – HEALPix input data

- **sum_ebins** (*bool*) – sum energy bins over energy bins before reprojecting
- **normalize** (*bool*) – True -> perserve integral by splitting HEALPix values between bins
- **(WCS object, np.ndarray() with reprojected data)** (*returns*) –

classmethod create_from_fits(*fitsfile*, ***kwargs*)

classmethod create_from_hdu(*hdu*, *ebins*)
Creates and returns an HpxMap object from a FITS HDU.

hdu : The FITS ebins : Energy bin edges [optional]

classmethod create_from_hdulist(*hdulist*, ***kwargs*)
Creates and returns an HpxMap object from a FITS HDUList

extname : The name of the HDU with the map data ebounds : The name of the HDU with the energy bin data

create_image_hdu(*name=None*, ***kwargs*)

expanded_counts_map()
return the full counts map

explicit_counts_map(*pixels=None*)
return a counts map with explicit index scheme

Parameters **pixels** (np.ndarray or None) – If set, grab only those pixels. If none, grab only non-zero pixels

get_map_values(*lons*, *lats*, *ibin=None*)
Return the indices in the flat array corresponding to a set of coordinates

Parameters

- **lons** (*array-like*) – 'Longitudes' (RA or GLON)
- **lats** (*array-like*) – 'Latitidues' (DEC or GLAT)
- **ibin** (*int or array-like*) – Extract data only for a given energy bin. None -> extract data for all bins

Returns **vals** – Values of pixels in the flattened map, np.nan used to flag coords outside of map

Return type numpy.ndarray((n))

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

get_pixel_skydirs()
Get a list of sky coordinates for the centers of every pixel.

hpx

interpolate(*lon*, *lat*, *egy=None*, *interp_log=True*)
Interpolate map values.

Parameters **interp_log** (*bool*) – Interpolate the z-coordinate in logspace.

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

NOTE: this re-calculates the mapping, if you have already calculated the mapping it is much faster to use convert_to_cached_wcs() instead

Parameters

- **sum_ebins** (*bool*) – sum energy bins over energy bins before reprojecting
- **proj** (*str*) – WCS-projection
- **oversample** (*int*) – Oversampling factor for WCS map
- **normalize** (*bool*) – True -> perserve integral by splitting HEALPix values between bins
- **(WCS object, np.ndarray() with reprojected data)** (*returns*) –

sparse_counts_map()
return a counts map with sparse index scheme

sum_over_energy()
Reduce a counts cube to a counts map

swap_scheme()

ud_grade(*order*, *preserve_counts=False*)

class fermipy.skymap.**Map**(*counts*, *wcs*, *ebins=None*)
Bases: *fermipy.skymap.Map_Base*

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

classmethod create(*skydir*, *cdelt*, *npix*, *coordsys='CEL'*, *projection='AIT'*, *ebins=None*, *differential=False*)

classmethod create_from_fits(*fitsfile*, ***kwargs*)

classmethod create_from_hdu(*hdu*, *wcs*)

create_image_hdu(*name=None*, ***kwargs*)

create_primary_hdu()

get_map_values(*lons*, *lats*, *ibin=None*)
Return the map values corresponding to a set of coordinates.

Parameters

- **lons** (*array-like*) – 'Longitudes' (RA or GLON)
- **lats** (*array-like*) – 'Latitidues' (DEC or GLAT)
- **ibin** (*int or array-like*) – Extract data only for a given energy bin. None -> extract data for all bins

Returns **vals** – Values of pixels in the flattened map, np.nan used to flag coords outside of map

Return type numpy.ndarray((n))

get_pixel_indices(*lons*, *lats*, *ibin=None*)
Return the indices in the flat array corresponding to a set of coordinates

Parameters

- **lons** (*array-like*) – 'Longitudes' (RA or GLON)
- **lats** (*array-like*) – 'Latitidues' (DEC or GLAT)
- **ibin** (*int or array-like*) – Extract data only for a given energy bin. None -> extract data for all energy bins.

Returns **pixcrd** – Pixel indices along each dimension of the map.

Return type list

get_pixel_skydirs()
Get a list of sky coordinates for the centers of every pixel.

interpolate(*lon*, *lat*, *egy=None*)
Return the interpolated map values corresponding to a set of coordinates.

interpolate_at_skydir(*skydir*)

ipix_swap_axes(*ipix*, *colwise=False*)
Return the transposed pixel index from the pixel xy coordinates

if colwise is True (False) this assumes the original index was in column wise scheme

ipix_to_xypix(*ipix*, *colwise=False*)
Return array multi-dimensional pixel indices from flattened index.

Parameters **colwise** (*bool*) – Use column-wise pixel indexing.

npix

pix_center
Return the ROI center in pixel coordinates.

pix_size
Return the pixel size along the two image dimensions.

skydir
Return the sky coordinate of the image center.

sum_over_energy()
Reduce a 3D counts cube to a 2D counts map

wcs

width
Return the dimensions of the image.

xypix_to_ipix(*xypix*, *colwise=False*)
Return the flattened pixel indices from an array multi-dimensional pixel indices.

Parameters

- **xypix** (*list*) – List of pixel indices in the order (LON,LAT,ENERGY).
- **colwise** (*bool*) – Use column-wise pixel indexing.

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

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

counts

data

get_map_values(*lons*, *lats*, *ibin=None*)
Return the map values corresponding to a set of coordinates.

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

get_pixel_skydirs()
Get a list of sky coordinates for the centers of every pixel.

interpolate(*lon*, *lat*, *egy=None*)
Return the interpolated map values corresponding to a set of coordinates.

sum_over_energy()
Reduce a counts cube to a counts map by summing over the energy planes

fermipy.skymap.**coadd_maps**(*geom*, *maps*, *preserve_counts=True*)
Coadd a sequence of Map objects.

fermipy.skymap.**make_coadd_hpx**(*maps*, *hpx*, *shape*, *preserve_counts=True*)

fermipy.skymap.**make_coadd_map**(*maps*, *proj*, *shape*, *preserve_counts=True*)

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*, *refSpec*, *norm_type*)
Bases: *fermipy.castro.CastroData_Base*

This class wraps the data needed to make a "Castro" plot, namely the log-likelihood as a function of normalization for a series of energy bins.

classmethod create_from_fits(*fitsfile*, *norm_type='eflux'*, *hdu_scan='SCANDATA'*, *hdu_energies='EBOUNDS'*, *irow=None*)
Create a CastroData object from a tscube FITS file.

Parameters

- **fitsfile** (*str*) – Name of the fits file
- **norm_type** (*str*) – Type of normalization to use. Valid options are:
  - norm : Normalization w.r.t. to test source
  - flux : Flux of the test source (ph cm^-2 s^-1)
  - eflux: Energy Flux of the test source (MeV cm^-2 s^-1)
  - npred: Number of predicted photons (Not implemented)
  - dnde : Differential flux of the test source (ph cm^-2 s^-1 MeV^-1)
- **hdu_scan** (*str*) – Name of the FITS HDU with the scan data
- **hdu_energies** (*str*) – Name of the FITS HDU with the energy binning and normalization data
- **irow** (*int or None*) – If none, then this assumes that there is a single row in the scan data table Otherwise, this specifies which row of the table to use

Returns castro

Return type *CastroData*

classmethod create_from_flux_points(*txtfile*)
Create a Castro data object from a text file containing a sequence of differential flux points.

classmethod create_from_sedfile(*fitsfile*, *norm_type='eflux'*)
Create a CastroData object from an SED fits file

Parameters

- **fitsfile** (*str*) – Name of the fits file
- **norm_type** (*str*) – Type of normalization to use, options are:
  - norm : Normalization w.r.t. to test source
  - flux : Flux of the test source (ph cm^-2 s^-1)
  - eflux: Energy Flux of the test source (MeV cm^-2 s^-1)
  - npred: Number of predicted photons (Not implemented)
  - dnde : Differential flux of the test source (ph cm^-2 s^-1 MeV^-1)

Returns castro

Return type *CastroData*

classmethod create_from_stack(*shape*, *components*, *ylims*, *weights=None*)
Combine the log-likelihoods from a number of components.

Parameters

- **shape** (*tuple*) – The shape of the return array
- **components** (*[CastroData_Base]*) – The components to be stacked
- **weights** (*array-like*) –

Returns castro

Return type *CastroData*

classmethod create_from_tables(*norm_type='eflux'*, *tab_s='SCANDATA'*, *tab_e='EBOUNDS'*)
Create a CastroData object from two tables

Parameters

- **norm_type** (*str*) – Type of normalization to use. Valid options are:
  - norm : Normalization w.r.t. to test source
  - flux : Flux of the test source (ph cm^-2 s^-1)
  - eflux: Energy Flux of the test source (MeV cm^-2 s^-1)
  - npred: Number of predicted photons (Not implemented)
  - dnde : Differential flux of the test source (ph cm^-2 s^-1 MeV^-1)
- **tab_s** (*str*) – table scan data
- **tab_e** (*str*) – table energy binning and normalization data

Returns castro

Return type *CastroData*

classmethod create_from_yamlfile(*yamlfile*)
Create a Castro data object from a yaml file contains the likelihood data.

create_functor(*specType*, *initPars=None*, *scale=1000.0*)
Create a functor object that computes normalizations in a sequence of energy bins for a given spectral model.

Parameters

- **specType** (*str*) – The type of spectrum to use. This can be a string corresponding to the spectral model class name or a *SpectralFunction* object.
- **initPars** (ndarray) – Arrays of parameter values with which the spectral function will be initialized.
- **scale** (*float*) – The 'pivot energy' or energy scale to use for the spectrum

Returns **fn** – A functor object.

Return type *SEDFunctor*

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

refSpec
Return a *ReferenceSpec* with the spectral data

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

Parameters

- **specTypes** (*str*) – The type of spectrum to try
- **params** (*array-like*) – The spectral parameters
- **scale** (*float*) – The energy scale or 'pivot' energy

test_spectra(*spec_types=None*)
Test different spectral types against the SED represented by this CastroData.

Parameters **spec_types** (*[str,..]*) – List of spectral types to try

Returns

retDict – A dictionary of dictionaries. The top level dictionary is keyed by spec_type. The sub-dictionaries each contain:

- "Function" : *SpectralFunction*
- "Result" : tuple with the output of scipy.optimize.fmin
- "Spectrum" : ndarray with best-fit spectral values
- "ScaleEnergy" : float, the 'pivot energy' value
- "TS" : float, the TS for the best-fit spectrum

Return type dict

x_edges()

class fermipy.castro.**CastroData_Base**(*norm_vals*, *nll_vals*, *nll_offsets*, *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.

build_lnl_fn(*normv*, *nllv*)

build_scandata_table()
Build an astropy.table.Table object from these data.

chi2_vals(*x*)
Compute the difference in the log-likelihood between the MLE in each energy bin and the normalization predicted by a global best-fit model. This array can be summed to get a goodness-of-fit chi2 for the model.

Parameters **x** (ndarray) – An array of normalizations derived from a global fit to all energy bins.

Returns **chi2_vals** – An array of chi2 values for each energy bin.

Return type ndarray

derivative(*x*, *der=1*)
Return the 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*) –

Returns **norm** – Best-fit normalization value

Return type float

fitNormalization(*specVals*, *xlims*)
Fit the normalization given a set of spectral values that define a spectral shape

This version is faster, and solves for the root of the derivatvie

Parameters

- **specVals** (*an array of (nebin values that define a spectral shape*) –
- **xlims** (*fit limits*) –
- **the best-fit normalization value** (*returns*) –

fit_spectrum(*specFunc*, *initPars*, *freePars=None*)
Fit for the free parameters of a spectral function

Parameters

- **specFunc** (*SpectralFunction*) – The Spectral Function
- **initPars** (ndarray) – The initial values of the parameters
- **freePars** (ndarray) – Boolean array indicating which parameters should be free in the fit.

Returns

- **params** (ndarray) – Best-fit parameters.
- **spec_vals** (ndarray) – The values of the best-fit spectral model in each energy bin.
- **ts_spec** (*float*) – The TS of the best-fit spectrum
- **chi2_vals** (ndarray) – Array of chi-squared values for each energy bin.
- **chi2_spec** (*float*) – Global chi-squared value for the sum of all energy bins.
- **pval_spec** (*float*) – p-value of chi-squared for the best-fit spectrum.

fn_mles()
returns the summed likelihood at the maximum likelihood estimate

Note that simply sums the maximum likelihood values at each bin, and does not impose any sort of constrain between bins

getIntervals(*alpha*)
Evaluate the two-sided intervals corresponding to a C.L. of (1-alpha)%.

Parameters **alpha** (*float*) – limit confidence level.

Returns

- **limit_vals_hi** (ndarray) – An array of lower limit values.
- **limit_vals_lo** (ndarray) – An array of upper limit values.

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

Parameters

- **alpha** (*float*) – limit confidence level.
- **upper** (*bool*) – upper or lower limits.
- **an array of values, one for each energy bin** (*returns*) –

mles()
return the maximum likelihood estimates for each of the energy bins

nll_null
Return the negative log-likelihood for the null-hypothesis

nll_offsets
Return the offsets in the negative log-likelihoods for each bin

norm_derivative(*spec*, *norm*)

norm_type
Return the normalization type flag

nx
Return the number of profiles

ny
Return the number of profiles

static stack_nll(*shape*, *components*, *ylims*, *weights=None*)
Combine the log-likelihoods from a number of components.

Parameters

- **shape** (*tuple*) – The shape of the return array

- **components** (*CastroData_Base*) – The components to be stacked
- **weights** (*array-like*) –

Returns

- **norm_vals** (numpy.ndarray) – N X M array of Normalization values
- **nll_vals** (numpy.ndarray) – N X M array of log-likelihood values
- **nll_offsets** (numpy.ndarray) – N array of maximum log-likelihood values in each bin

ts_vals()
returns test statistic values for each energy bin

x_edges()

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

derivative(*x*, *der=1*)
return the derivative a an array of input values

x : the inputs der : the order of derivative

x
return the x values used to construct the split

xmax
return the maximum value over which the spline is defined

xmin
return the minimum value over which the spline is defined

y
return the y values used to construct the split

class fermipy.castro.**LnLFn**(*x*, *y*, *norm_type=0*)
Bases: object

Helper class for interpolating a 1-D log-likelihood function from a set of tabulated values.

TS()
return the Test Statistic

fn_mle()
return the function value at the maximum likelihood estimate

getDeltaLogLike(*dlnl*, *upper=True*)
Find the point at which the log-likelihood changes by a given value with respect to its value at the MLE.

getInterval(*alpha*)
Evaluate the interval corresponding to a C.L. of (1-alpha)%.

Parameters **alpha** (*limit confidence level.*) –

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

Parameters

- **alpha** (*limit confidence level.*) –

- **upper** (*upper or lower limits.*) –

interp
return the underlying Interpolator object

mle()
return the maximum likelihood estimate

This will return the cached value, if it exists

norm_type
Return a string specifying the quantity used for the normalization. This isn't actually used in this class, but it is carried so that the class is self-describing. The possible values are open-ended.

class fermipy.castro.**ReferenceSpec**(*emin*, *emax*, *ref_dnde*, *ref_flux*, *ref_eflux*, *ref_npred*, *eref=None*)

Bases: object

This class encapsulates data for a reference spectrum.

Parameters

- **ne** (int) – Number of energy bins
- **ebins** (ndarray) – Array of bin edges.
- **emin** (ndarray) – Array of lower bin edges.
- **emax** (ndarray) – Array of upper bin edges.
- **bin_widths** (ndarray) – Array of energy bin widths.
- **eref** (ndarray) – Array of reference energies. Typically these are the geometric mean of the energy bins
- **ref_dnde** (ndarray) – Array of differential photon flux values.
- **ref_flux** (ndarray) – Array of integral photon flux values.
- **ref_eflux** (ndarray) – Array of integral energy flux values.
- **ref_npred** (ndarray) – Array of predicted number of photons in each energy bin.

bin_widths

build_ebound_table()
Build and return an EBOUNDS table with the encapsulated data.

classmethod create_from_table(*tab_e*)

Parameters **tab_e** (Table) – EBOUNDS table.

ebins

emax

emin

eref

log_ebins

nE

ref_dnde

ref_eflux
return the energy flux values

ref_flux
return the flux values

ref_npred
return the number of predicted events

class fermipy.castro.**SpecData**(*ref_spec*, *norm*, *norm_err*)
Bases: *fermipy.castro.ReferenceSpec*

This class encapsulates spectral analysis results (best-fit normalizations, errors, etc.), energy binning, and reference spectrum definition.

Parameters

- **norm** (ndarray) –
- **norm_err** (ndarray) –
- **flux** (ndarray) – Array of integral photon flux values.
- **eflux** (ndarray) – Array of integral energy flux values.
- **dnde** (ndarray) – Differential flux values
- **dnde_err** (ndarray) – Uncertainties on differential flux values
- **e2dnde** (ndarray) – Differential flux values scaled by E^2
- **e2dnde_err** (ndarray) – Uncertainties on differential flux values scaled by E^2

build_spec_table()

classmethod create_from_table(*tab*)

dnde

dnde_err

e2dnde

e2dnde_err

eflux

flux

norm

norm_err

class fermipy.castro.**TSCube**(*tsmap*, *normmap*, *tscube*, *normcube*, *norm_vals*, *nll_vals*, *refSpec*, *norm_type*)
Bases: object

A class wrapping a TSCube, which is a collection of CastroData objects for a set of directions.

This class wraps a combination of:

- Pixel data,
- Pixel x Energy bin data,
- Pixel x Energy Bin x Normalization scan point data

castroData_from_ipix(*ipix*, *colwise=False*)
Build a CastroData object for a particular pixel

castroData_from_pix_xy(*xy*, *colwise=False*)
Build a CastroData object for a particular pixel

classmethod create_from_fits(*fitsfile*, *norm_type='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*) – Peak threshold in TS.
- **min_separation** (*float*) – Radius of region size in degrees. Sets the minimum allowable separation between peaks.
- **use_cumul** (*bool*) – If true, used the cumulative TS map (i.e., the TS summed over the energy bins) instead of the TS Map from the fit to and index=2 powerlaw.

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

Return type list

find_sources(*threshold*, *min_separation=1.0*, *use_cumul=False*, *output_peaks=False*, *output_castro=False*, *output_specInfo=False*, *output_src_dicts=False*, *output_srcs=False*)

nE
return the number of energy bins

nN
return the number of sample points in each energy bin

normcube
return the Cube of the normalization value per pixel / energy bin

normmap
return the Map of the Best-fit normalization value

nvals
Return the number of values in the tscube

refSpec
Return the Spectral Data object

test_spectra_of_peak(*peak*, *spec_types=None*)
Test different spectral types against the SED represented by the CastroData corresponding to a single pixel in this TSCube

Parameters **spec_types** (*[str,..]*) – List of spectral types to try

Returns

- **castro** (*CastroData*) – The castro data object for the pixel corresponding to the peak
- **test_dict** (*dict*) – The dictionary returned by *test_spectra*

ts_cumul
return the Map of the cumulative TestStatistic value per pixel (summed over energy bin)

tscube
return the Cube of the TestStatistic value per pixel / energy bin

tsmap
return the Map of the TestStatistic value

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

fermipy.castro.**convert_sed_cols**(*tab*)
Cast SED column names to lowercase.

fermipy.tsmap module

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

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

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files (FITS and rendered images).
- **model** (*dict*) – Dictionary defining the properties of the test source.
- **do_sed** (*bool*) – Compute the energy bin-by-bin fits.
- **nnorm** (*int*) – Number of points in the likelihood v. normalization scan.
- **norm_sigma** (*float*) – Number of sigma to use for the scan range.
- **tol** (*float*) – 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=''*, ***kwargs*)
Generate a spatial TS map for a source component with properties defined by the model argument. The TS map will have the same geometry as the ROI. The output of this method is a dictionary containing *Map* objects with the TS and amplitude of the best-fit test source. By default this method will also save maps to FITS files and render them as image files.

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

Parameters

- **prefix** (*str*) – Optional string that will be prepended to all output files.
- **{options}** –

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

Return type dict

fermipy.tsmap.**cash**(*counts*, *model*)
Compute the Poisson log-likelihood function.

fermipy.tsmap.**convert_tscube**(*infile*, *outfile*)

fermipy.tsmap.**convert_tscube_old**(*infile*, *outfile*)
Convert between old and new TSCube formats.

fermipy.tsmap.**extract_array**(*array_large*, *array_small*, *position*)

fermipy.tsmap.**extract_images_from_tscube**(*infile*, *outfile*)
Extract data from table HDUs in TSCube file and convert them to FITS images

fermipy.tsmap.**extract_large_array**(*array_large*, *array_small*, *position*)

fermipy.tsmap.**extract_small_array**(*array_small*, *array_large*, *position*)

fermipy.tsmap.**f_cash**(*x*, *counts*, *bkg*, *model*)
Wrapper for cash statistics, that defines the model function.

Parameters

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

fermipy.tsmap.**f_cash_sum**(*x*, *counts*, *bkg*, *model*, *bkg_sum=0*, *model_sum=0*)

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

fermipy.tsmap.**truncate_array**(*array1*, *array2*, *position*)
Truncate array1 by finding the overlap with array2 when the array1 center is located at the given position in array2.

fermipy.residmap module

class fermipy.residmap.**ResidMapGenerator**
Bases: object

Mixin class for *GTAnalysis* that generates spatial residual maps from the difference of data and model maps smoothed with a user-defined spatial/spectral template. The map of residual significance can be interpreted in the same way as a TS map (the likelihood of a source at the given location).

residmap(*prefix=''*, ***kwargs*)
Generate 2-D spatial residual maps using the current ROI model and the convolution kernel defined with the model argument.

Parameters

- **prefix** (*str*) – String that will be prefixed to the output residual map files.
- **{options}** –

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

Return type dict

fermipy.residmap.**convolve_map**(*m*, *k*, *cpix*, *threshold=0.001*, *imin=0*, *imax=None*, *wmap=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.
- **wmap** (ndarray) – 3-D map containing a sequence of 2-D spatial maps of weights. First dimension should be energy. This map should have the same dimension as m.

fermipy.residmap.**convolve_map_hpx**(*m*, *k*, *cpix*, *threshold=0.001*, *imin=0*, *imax=None*, *wmap=None*)

Perform an energy-dependent convolution on a sequence of 2-D spatial maps.

Parameters

- **m** (ndarray) – 2-D map containing a sequence of 1-D HEALPix maps. First dimension should be energy.
- **k** (ndarray) – 2-D map containing a sequence of convolution kernels (PSF) for each slice in m. This map should have the same dimension as m.
- **threshold** (*float*) – Kernel amplitude
- **imin** (*int*) – Minimum index in energy dimension.
- **imax** (*int*) – Maximum index in energy dimension.
- **wmap** (ndarray) – 2-D map containing a sequence of 1-D HEALPix maps of weights. First dimension should be energy. This map should have the same dimension as m.

fermipy.residmap.**convolve_map_hpx_gauss**(*m*, *sigmas*, *imin=0*, *imax=None*, *wmap=None*)

Perform an energy-dependent convolution on a sequence of 2-D spatial maps.

Parameters

- **m** (HpxMap) – 2-D map containing a sequence of 1-D HEALPix maps. First dimension should be energy.
- **sigmas** (ndarray) – 1-D map containing a sequence gaussian widths for smoothing
- **imin** (*int*) – Minimum index in energy dimension.
- **imax** (*int*) – Maximum index in energy dimension.

- **wmap** (ndarray) – 2-D map containing a sequence of 1-D HEALPix maps of weights. First dimension should be energy. This map should have the same dimension as m.

fermipy.residmap.**get_source_kernel**(*gta*, *name*, *kernel=None*)
Get the PDF for the given source.

fermipy.residmap.**poisson_lnl**(*nc*, *mu*)

fermipy.lightcurve module

class fermipy.lightcurve.**LightCurve**
Bases: object

lightcurve(*name*, ***kwargs*)
Generate a lightcurve for the named source. The function will complete the basic analysis steps for each bin and perform a likelihood fit for each bin. Extracted values (along with errors) are Integral Flux, spectral model, Spectral index, TS value, pred. # of photons. Note: successful calculation of TS:subscript:`var` requires at least one free background parameter and a previously optimized ROI model.

Parameters

- **name** (*str*) – source name
- **{options}** –

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

Return type dict

fermipy.lightcurve.**calcTS_var**(*loglike*, *loglike_const*, *flux_err*, *flux_const*, *systematic*, *fit_success*)

Module contents

fermipy.**get_st_version**()
Get the version string of the ST release.

fermipy.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)
Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

Parameters

- **package** (*str, optional*) – The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.
- **test_path** (*str, optional*) – Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.
- **args** (*str, optional*) – Additional arguments to be passed to pytest.main in the `args` keyword argument.
- **plugins** (*list, optional*) – Plugins to be passed to pytest.main in the `plugins` keyword argument.
- **verbose** (*bool, optional*) – Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

- **pastebin** (*{'failed','all',None}, optional*) – Convenience option for turning on py.test pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.
- **remote_data** (*bool, optional*) – Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.
- **pep8** (*bool, optional*) – Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.
- **pdb** (*bool, optional*) – Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.
- **coverage** (*bool, optional*) – Generate a test coverage report. The result will be placed in the directory htmlcov.
- **open_files** (*bool, optional*) – Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Works only on platforms with a working `lsof` command.
- **parallel** (*int, optional*) – When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.
- **kwargs** – Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

1.3.11 fermipy.jobs subpackage

The fermipy.jobs sub-package is a light-weight, largely standalone, package to manage data analysis pipelines. It allows the user to build up increasingly complex analysis pipelines from single applications that are callable either from inside python or from the unix command line.

Link objects

The basic building block of an analysis pipeline is a `Link` object. In general a `Link` is a single application that can be called from the command line.

The fermipy.jobs package imlements five types of `Link` objects, and the idea is that users can make sub-classes to perform the steps of their analysis.

Every link sub-class has a small required header block, for example:

```
class AnalyzeROI(Link):
    """Small class that wraps an analysis script.

    This particular script does baseline fitting of an ROI.
    """
    appname = 'fermipy-analyze-roi'
    linkname_default = 'analyze-roi'
    usage = '%s [options]' % (appname)
    description = "Run analysis of a single ROI"

    default_options = dict(config=defaults.common['config'],
                           roi_baseline=defaults.common['roi_baseline'],
```

(continues on next page)

(continued from previous page)

```
                        make_plots=defaults.common['make_plots'])

__doc__ += Link.construct_docstring(default_options)
```

The various pieces of the header are:

- appname This is the unix command that will invoke this link.
- linkname_default This is the default name that links of this type will be given when then are put into analysis pipeline.
- usage, description These are passed to the argument parser and used to build the help string.
- default_options This is the set of options and default values for this link
- The __doc__ += Link.construct_docstring(default_options) line ensures that the default options will be included in the class's docstring.

Link sub-classes

There are five types of `Link` sub-classes implemented here.

- `Link`

  This is the sub-class to use for a user-defined function. In this case in addition to providing the header material above, the sub-class will need to implement the run_analysis() to perform that function.

```
def run_analysis(self, argv):
    """Run this analysis"""
    args = self._parser.parse_args(argv)

    do stuff
```

- `Gtlink`

  This is the sub-class to use to invoke a Fermi ScienceTools gt-tool, such as gtsrcmaps or gtexcube2. In this case the user only needs to provide the header content to make the options they want availble to the interface.

- `AppLink`

  This is the sub-class to use to invoke a pre-existing unix command. In this case the user only needs to provide the header content to make the options they want availble to the interface.

- `ScatterGather`

  This is the sub-class to use to send a set of similar jobs to a computing batch farm. In this case the user needs to provide the standard header content and a couple of addtional things. Here is an example:

```
 class AnalyzeROI_SG(ScatterGather):
     """Small class to generate configurations for the `AnalyzeROI`␣
↪class.

     This loops over all the targets defined in the target list.
     """
     appname = 'fermipy-analyze-roi-sg'
     usage = "%s [options]" % (appname)
     description = "Run analyses on a series of ROIs"
     clientclass = AnalyzeROI
```

(continues on next page)

(continued from previous page)

```
    job_time = 1500

    default_options = dict(ttype=defaults.common['ttype'],
                           targetlist=defaults.common['targetlist'],
                           config=defaults.common['config'],
                           roi_baseline=defaults.common['roi_baseline'],
                           make_plots=defaults.common['make_plots'])

    __doc__ += Link.construct_docstring(default_options)

    def build_job_configs(self, args):
        """Hook to build job configurations
        """
        job_configs = {}

        ttype = args['ttype']

        do stuff

        return job_configs


The job_time class parameter should be an estimate of the time the average
job managed by this class will take.  That is used to decided which batch
farm resources to use to run the job, and how often to check from job␣
↪completion.

The used defined build_job_configs() function should build a dictionary of
dictionaries that contains the parameters to use for each instance of the␣
↪command that
will run.
```

- `Chain`

  This is the sub-class to use to run multiple `Link` objects in sequence.

  For `Chain` sub-classes, in addtion to the standard header material, the user should profile a map_arguments() method that builds up the chain and sets the options of the component `Link` objects using the _set_link() method. Here is an example:

```
def _map_arguments(self, input_dict):
    """Map from the top-level arguments to the arguments provided to
    the indiviudal links """

    config_yaml = input_dict['config']
    config_dict = load_yaml(config_yaml)

    data = config_dict.get('data')
    comp = config_dict.get('comp')
    sourcekeys = config_dict.get('sourcekeys')

    mktimefilter = config_dict.get('mktimefilter')

    self._set_link('expcube2', Gtexpcube2wcs_SG,
                   comp=comp, data=data,
```

(continues on next page)

(continued from previous page)

```
                  mktimefilter=mktimefilter)

    self._set_link('exphpsun', Gtexphpsun_SG,
                   comp=comp, data=data,
                   mktimefilter=mktimefilter)

    self._set_link('suntemp', Gtsuntemp_SG,
                   comp=comp, data=data,
                   mktimefilter=mktimefilter,
                   sourcekeys=sourcekeys)
```

Using Links and sub-classes

The main aspect of the `Link` interface are:

- Running the `Link`:

```
link.run()
```

- Seeing the status of the `Link`:

```
link.check_job_status()
```

- Seeing the jobs associated to this `Link`:

```
link.jobs
```

- Setting the arguments used to run this `Link`:

```
link.update_args(dict=(option_name=option_value,
                       option_name2=option_value2,
                       ...))
```

Module contents

Link class and trivial sub-classes

class fermipy.jobs.link.**Link**(***kwargs*)

Bases: object

A wrapper for a command line application.

This class keeps track for the arguments to pass to the application as well as input and output files.

This can be used either with other *Link* objects to build a `Chain`, or as standalone wrapper to pass configuration to the application.

Derived classes will need to override the appname and linkname-default class parameters.

Parameters

- **appname** (*str*) – Name of the application run by this *Link*
- **linkname_default** (*str*) – Default name for *Link* of this type
- **default_options** (*dict*) – Dictionry with options, defaults, helpstring and types for the parameters associated with the *Link*

- **default_file_args** (*dict*) – Dictionary specifying if particular parameters are associated with input or output files.
- **linkname** (*str*) – Name of this *Link*, used as a key to find it in a Chain.
- **link_prefix** (*str*) – Optional prefix for this *Link*, used to distinguish between similar *Link* objects on different Chain objects.
- **args** (*dict*) – Up-to-date dictionary with the arguments that will be passed to the application
- **_options** (*dict*) – Dictionary with the options that we are allowed to set and default values
- **files** (FileDict) – Object that keeps track of input and output files
- **jobs** (OrderedDict) – Dictionary mapping keys to JobDetails. This contains information about all the batch jobs associated to this *Link*

appname = 'dummy'

arg_names
Return the list of arg names

check_input_files(*return_found=True*, *return_missing=True*)
Check if input files exist.

Parameters

- **return_found** (*list*) – A list with the paths of the files that were found.
- **return_missing** (*list*) – A list with the paths of the files that were missing.

Returns

- **found** (*list*) – List of the found files, if requested, otherwise None
- **missing** (*list*) – List of the missing files, if requested, otherwise None

check_job_status(*key='__top__'*, *fail_running=False*, *fail_pending=False*, *force_check=False*)
Check the status of a particular job

By default this checks the status of the top-level job, but can by made to drill into the sub-jobs.

Parameters

- **key** (*str*) – Key associated to the job in question
- **fail_running** (bool) – If True, consider running jobs as failed
- **fail_pending** (bool) – If True, consider pending jobs as failed
- **force_check** (bool) – Drill into status of individual jobs‘ instead of using top level job only

Returns **status** – Job status flag

Return type JobStatus

check_jobs_status(*fail_running=False*, *fail_pending=False*)
Check the status of all the jobs run from this link and return a status flag that summarizes that.

Parameters

- **fail_running** (bool) – If True, consider running jobs as failed
- **fail_pending** (bool) – If True, consider pending jobs as failed

Returns **status** – Job status flag that summarizes the status of all the jobs,

Return type JobStatus

check_output_files(*return_found=True*, *return_missing=True*)
Check if output files exist.

Parameters

- **return_found** (*list*) – A list with the paths of the files that were found.
- **return_missing** (*list*) – A list with the paths of the files that were missing.

Returns

- **found** (*list*) – List of the found files, if requested, otherwise None
- **missing** (*list*) – List of the missing files, if requested, otherwise None

clean_jobs(*recursive=False*)
Clean out all of the jobs associated to this link.

For sub-classes, if recursive is True this also clean jobs from any internal *Link*

clear_jobs(*recursive=True*)
Clear the self.jobs dictionary that contains information about jobs associated with this *Link*.

For sub-classes, if recursive is True this also clean jobs from any internal *Link*

command_template()
Build and return a string that can be used as a template invoking this chain from the command line.

The actual command can be obtainted by using `self.command_template().format(**self.args)`

static construct_docstring(*options*)
Construct a docstring for a set of options

classmethod create(***kwargs*)
Build and return a *Link*

default_file_args = {}

default_options = {}

description = 'Link to run dummy'

formatted_command()
Build and return the formatted command for this *Link*.

This is exactly the command as called from the Unix command line.

full_linkname
Return the linkname with the prefix attached This is useful to distinguish between links on different Chain objects.

get_failed_jobs(*fail_running=False*, *fail_pending=False*)
Return a dictionary with the subset of jobs that are marked as failed

Parameters

- **fail_running** (bool) – If True, consider running jobs as failed
- **fail_pending** (bool) – If True, consider pending jobs as failed

Returns **failed_jobs** – Dictionary mapping from job key to JobDetails for the failed jobs.

Return type dict

get_jobs(*recursive=True*)
Return a dictionary with all the jobs

For sub-classes, if recursive is True this will include jobs from any internal *Link*

linkname_default = 'dummy'

classmethod main()
Hook to run this *Link* from the command line

missing_input_files()
Make and return a dictionary of the missing input files.

This returns a dictionary mapping filepath to list of *Link* that use the file as input.

missing_output_files()
Make and return a dictionary of the missing output files.

This returns a dictionary mapping filepath to list of links that produce the file as output.

print_summary(*stream=<open file '<stdout>'*, *mode 'w'>*, *indent=''*, *recurse_level=2*)
Print a summary of the activity done by this *Link*.

Parameters

- **stream** (file) – Stream to print to, must have 'write' method.
- **indent** (*str*) – Indentation at start of line
- **recurse_level** (*int*) – Number of recursion levels to print

classmethod register_class()
Regsiter this class in the LinkFactory

run(*stream=<open file '<stdout>'*, *mode 'w'>*, *dry_run=False*, *stage_files=True*, *resubmit_failed=False*)
Runs this *Link*.

This version is intended to be overwritten by sub-classes so as to provide a single function that behaves the same for all version of *Link*

Parameters

- **stream** (file) – Stream that this *Link* will print to, Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it.
- **stage_files** (*bool*) – Copy files to and from scratch staging area.
- **resubmit_failed** (*bool*) – Flag for sub-classes to resubmit failed jobs.

run_analysis(*argv*)
Implemented by sub-classes to run a particular analysis

run_command(*stream=<open file '<stdout>'*, *mode 'w'>*, *dry_run=False*)
Runs the command for this link. This method can be overridden by sub-classes to invoke a different command

Parameters

- **stream** (file) – Stream that this *Link* will print to, Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it

Returns **code** – Return code from sub-process

Return type int

run_with_log(*dry_run=False*, *stage_files=True*, *resubmit_failed=False*)
Runs this link with output sent to a pre-defined logfile

Parameters

- **dry_run** (*bool*) – Print command but do not run it.
- **stage_files** (*bool*) – Copy files to and from scratch staging area.
- **resubmit_failed** (*bool*) – Flag for sub-classes to resubmit failed jobs.

topkey = '__top__'

update_args(*override_args*)
Update the argument used to invoke the application

Note that this will also update the dictionary of input and output files.

Parameters **override_args** (*dict*) – Dictionary of arguments to override the current values

usage = 'dummy [options]'

Utilities to chain together a series of ScienceTools apps

class fermipy.jobs.gtlink.**Gtlink**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

A wrapper for a single ScienceTools application

This class keeps track for the arguments to pass to the application as well as input and output files.

This can be used either with other `Link` to build a `Chain`, or as as standalone wrapper to pass conifguration to the application.

See help for `chain.Link` for additional details

appname = 'dummy'

command_template()
Build and return a string that can be used as a template invoking this chain from the command line.

The actual command can be obtainted by using `self.command_template().format(**self.args)`

description = 'Link to run dummy'

get_gtapp()
Returns a `GTApp` object that will run this `Link`

linkname_default = 'dummy'

run_analysis(*argv*)
Implemented by sub-classes to run a particular analysis

run_command(*stream=<open file '<stdout>', mode 'w'>*, *dry_run=False*)
Runs the command for this link. This method can be overridden by sub-classes to invoke a different command

Parameters

- **stream** (`file`) – Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it

update_args(*override_args*)
Update the argument used to invoke the application

See help for `chain.Link` for details

This calls the base class function then fills the parameters of the GtApp object

usage = 'dummy [options]'

`fermipy.jobs.gtlink.`**build_gtapp**(*appname*, *dry_run*, ***kwargs*)
Build an object that can run ScienceTools application

Parameters

- **appname** (*str*) – Name of the application (e.g., gtbin)
- **dry_run** (*bool*) – Print command but do not run it
- **kwargs** (*arguments used to invoke the application*) –
- **GtApp.GtApp object that will run the application in question** (*Returns*) –

`fermipy.jobs.gtlink.`**extract_parameters**(*pil*, *keys=None*)
Extract and return parameter names and values from a pil object

Parameters

- **pil** (`Pil` object) –
- **keys** (*list*) – List of parameter names, if None, extact all parameters

Returns **out_dict** – Dictionary with parameter name, value pairs

Return type dict

`fermipy.jobs.gtlink.`**run_gtapp**(*gtapp*, *stream*, *dry_run*, ***kwargs*)
Runs one on the ScienceTools apps

Taken from fermipy.gtanalysis.run_gtapp by Matt Wood

Parameters

- **gtapp** (`GtApp.GtApp` object) – The application (e.g., gtbin)
- **stream** (*stream object*) – Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it
- **kwargs** (*arguments used to invoke the application*) –

`fermipy.jobs.gtlink.`**update_gtapp**(*gtapp*, ***kwargs*)
Update the parameters of the object that can run ScienceTools applications

Parameters

- **gtapp** (`GtApp.GtApp`) – Object that will run the application in question
- **kwargs** (*arguments used to invoke the application*) –

class `fermipy.jobs.app_link.`**AppLink**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

A wrapper for a single fermipy application

This class keeps track for the arguments to pass to the application as well as input and output files.

This can be used either with other `Link` to build a `Chain`, or as as standalone wrapper to pass conifguration to the application.

See help for Link for additional details

appname = 'dummy'

description = 'Link to run dummy'

linkname_default = 'dummy'

run_analysis(*argv*)
Implemented by sub-classes to run a particular analysis

usage = 'dummy [options]'

ScatterGather class

class fermipy.jobs.scatter_gather.**ScatterGather**(*link*, ***kwargs*)
Bases: *fermipy.jobs.link.Link*

Class to dispatch several jobs in parallel and collect and merge the results.

Sub-classes will need to generatare configuration for the jobs that they launch.

Parameters

- **clientclass** (*type*) – Type of Link object managed by this class.
- **job_time** (*int*) – Estimated maximum time it takes to run a job This is used to manage batch farm scheduling and checking for completion.

appname = 'dummy-sg'

build_job_configs(*args*)
Hook to build job configurations

Sub-class implementation should return:

job_configs [dict] Dictionary of dictionaries passed to parallel jobs

check_status(*stream=<open file '<stdout>', mode 'w'>*, *check_once=False*, *fail_pending=False*, *fail_running=False*, *no_wait=False*, *do_print=True*, *write_status=False*)
Loop to check on the status of all the jobs in job dict.

Parameters

- **stream** (file) – Stream that this function will print to, Must have 'write' function.
- **check_once** (*bool*) – Check status once and exit loop.
- **fail_pending** (bool) – If True, consider pending jobs as failed
- **fail_running** (bool) – If True, consider running jobs as failed
- **no_wait** (*bool*) – Do not sleep before checking jobs.
- **do_print** (*bool*) – Print summary stats.
- **write_status** (*bool*) – Write the status the to log file.

Returns **status_vect** – Vector that summarize the number of jobs in various states.

Return type JobStatusVector

clean_jobs(*recursive=False*)
Clean up all the jobs associated with this object.

If recursive is True this also clean jobs dispatch by this object.

clear_jobs(*recursive=True*)
Clear the self.jobs dictionary that contains information about jobs associated with this *ScatterGather*

If recursive is True this will include jobs from all internal `Link`

clientclass = None

classmethod create(***kwargs*)
Build and return a *ScatterGather* object

default_options = {}

default_options_base = {'action': ('run', 'Action to perform', <type 'str'>), 'check_

default_prefix_logfile = 'scatter'

description = 'Run multiple analyses'

get_jobs(*recursive=True*)
Return a dictionary with all the jobs

If recursive is True this will include jobs from all internal `Link`

job_time = 1500

classmethod main()
Hook for command line interface to sub-classes

print_failed(*stream=<open file '<stderr>', mode 'w'>*)
Print list of the failed jobs

print_summary(*stream=<open file '<stdout>', mode 'w'>, indent='', recurse_level=2*)
Print a summary of the activity done by this `Link`.

Parameters

- **stream** (`file`) – Stream to print to
- **indent** (*str*) – Indentation at start of line
- **recurse_level** (*int*) – Number of recursion levels to print

print_update(*stream=<open file '<stdout>', mode 'w'>, job_stats=None*)
Print an update about the current number of jobs running

resubmit(*stream=<open file '<stdout>', mode 'w'>, fail_running=False*)
Function to resubmit failed jobs and collect results

Parameters

- **stream** (`file`) – Stream that this function will print to, Must have 'write' function.
- **fail_running** (`bool`) – If True, consider running jobs as failed

Returns **status_vect** – Vector that summarize the number of jobs in various states.

Return type `JobStatusVector`

run(*stream=<open file '<stdout>', mode 'w'>, dry_run=False, stage_files=True, resubmit_failed=True*)
Runs this `Link`.

This version is intended to be overwritten by sub-classes so as to provide a single function that behaves the same for all version of `Link`

Parameters

- **stream** (`file`) – Stream that this `Link` will print to, Must have 'write' function

- **dry_run** (*bool*) – Print command but do not run it.
- **stage_files** (*bool*) – Copy files to and from scratch staging area.
- **resubmit_failed** (*bool*) – Flag for sub-classes to resubmit failed jobs.

run_analysis(*argv*)
Implemented by sub-classes to run a particular analysis

run_jobs(*stream=<open file '<stdout>', mode 'w'>*)
Function to dipatch jobs and collect results

Parameters **stream** (file) – Stream that this function will print to, Must have 'write' function.

Returns **status_vect** – Vector that summarize the number of jobs in various states.

Return type JobStatusVector

scatter_link
Return the Link object used the scatter phase of processing

update_args(*override_args*)
Update the arguments used to invoke the application

Note that this will also update the dictionary of input and output files

Parameters **override_args** (*dict*) – dictionary of arguments to override the current values

usage = 'dummy-sg [options]'

Chain class

class fermipy.jobs.chain.**Chain**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

An object tying together a series of applications into a single application.

This class keep track of the arguments to pass to the applications as well as input and output files.

Note that this class is itself a Link. This allows you to write a python module that implements a chain and also has a __main__ function to allow it to be called from the shell.

check_links_status(*fail_running=False*, *fail_pending=False*)
"Check the status of all the jobs run from the Link objects in this *Chain* and return a status flag that summarizes that.

Parameters

- **fail_running** (bool) – If True, consider running jobs as failed
- **fail_pending** (bool) – If True, consider pending jobs as failed

Returns **status** – Job status flag that summarizes the status of all the jobs,

Return type JobStatus

clear_jobs(*recursive=True*)
Clear a dictionary with all the jobs

If recursive is True this will include jobs from all internal Link

get_jobs(*recursive=True*)
Return a dictionary with all the jobs

If recursive is True this will include jobs from all internal Link

linknames
Return the name of the Link objects owned by this *Chain*

links
Return the OrderedDict of Link objects owned by this *Chain*

classmethod main()
Hook to run this *Chain* from the command line

missing_input_files()
Make and return a dictionary of the missing input files.

This returns a dictionary mapping filepath to list of Link that use the file as input.

missing_output_files()
Make and return a dictionary of the missing output files.

This returns a dictionary mapping filepath to list of links that produce the file as output.

print_status(*indent=''*, *recurse=False*)
Print a summary of the job status for each Link in this *Chain*

print_summary(*stream=<open file '<stdout>'*, *mode 'w'>*, *indent=''*, *recurse_level=2*)
Print a summary of the activity done by this *Chain*.

Parameters

- **stream** (file) – Stream to print to, must have 'write' method.
- **indent** (*str*) – Indentation at start of line
- **recurse_level** (*int*) – Number of recursion levels to print

run(*stream=<open file '<stdout>'*, *mode 'w'>*, *dry_run=False*, *stage_files=True*, *resubmit_failed=False*)
Runs this *Chain*.

Parameters

- **stream** (file) – Stream that this Link will print to, Must have 'write' function
- **dry_run** (*bool*) – Print command but do not run it.
- **stage_files** (*bool*) – Copy files to and from scratch staging area.
- **resubmit_failed** (*bool*) – Flag for sub-classes to resubmit failed jobs.

run_analysis(*argv*)
Implemented by sub-classes to run a particular analysis

update_args(*override_args*)
Update the argument used to invoke the application

Note that this will also update the dictionary of input and output files.

Parameters **override_args** (*dict*) – dictionary passed to the links

High-level analysis classes

These are Link sub-classes that implement *fermipy* analyses, or perform tasks related to *fermipy* analyses, such as plotting or collecting results for a set of simulations.

class fermipy.jobs.target_analysis.**AnalyzeROI**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class that wraps an analysis script.

This particular script does baseline fitting of an ROI.

Parameters

- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **make_plots** (*<type 'bool'>*) – Make plots [False]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]

appname = 'fermipy-analyze-roi'

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'ma

description = 'Run analysis of a single ROI'

linkname_default = 'analyze-roi'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-analyze-roi [options]'

class fermipy.jobs.target_analysis.**AnalyzeSED**(**kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to wrap an analysis script.

This particular script fits an SED for a target source with respect to the baseline ROI model.

Parameters

- **skydirs** (*<type 'str'>*) – Yaml file with blank sky directions. [None]
- **astro_bkgs** (*<type 'list'>*) – Astrophysical background sources [None]
- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **make_plots** (*<type 'bool'>*) – Make plots [False]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **profiles** (*<type 'list'>*) – List of profiles to analyze [[]]

appname = 'fermipy-analyze-sed'

default_options = {'astro_bkgs': (None, 'Astrophysical background sources', <type 'lis

description = 'Extract the SED for a single target'

linkname_default = 'analyze-sed'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-analyze-sed [options]'

class fermipy.jobs.target_collect.**CollectSED**(**kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to collect SED results from a series of simulations.

Parameters

- **dry_run** (*<type 'bool'>*) – Print commands but do not run them. [False]

- **sed_file** (*<type 'str'>*) – Path to SED file. [None]
- **nsims** (*<type 'int'>*) – Number of simulations to run. [20]
- **outfile** (*<type 'str'>*) – Path to output file. [None]
- **seed** (*<type 'int'>*) – Seed number for first simulation. [0]
- **summaryfile** (*<type 'str'>*) – Path to file with results summaries. [None]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **enumbins** (*<type 'int'>*) – Number of energy bins [12]

appname = 'fermipy-collect-sed'

collist = [{'name': 'e_min', 'unit': 'MeV'}, {'name': 'e_ref', 'unit': 'MeV'}, {'na

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'd

description = 'Collect SED results from simulations'

linkname_default = 'collect-sed'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-collect-sed [options]'

class fermipy.jobs.target_sim.**CopyBaseROI**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to copy a baseline ROI to a simulation area

This is useful for parallelizing analysis using the fermipy.jobs module.

Parameters

- **extracopy** (*<type 'list'>*) – Extra files to copy [[]]
- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **target** (*<type 'str'>*) – Name of analysis target. [None]
- **sim** (*<type 'str'>*) – Name of the simulation scenario. [None]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]

appname = 'fermipy-copy-base-roi'

classmethod copy_analysis_files(*orig_dir*, *dest_dir*, *copyfiles*)
Copy a list of files from orig_dir to dest_dir

classmethod copy_target_dir(*orig_dir*, *dest_dir*, *roi_baseline*, *extracopy*)
Create and populate directoris for target analysis

copyfiles = ['srcmap_*.fits', 'ccube.fits', 'ccube_*.fits']

default_options = {'extracopy': ([], 'Extra files to copy', <type 'list'>), 'roi_base

description = 'Copy a baseline ROI to a simulation area'

linkname_default = 'copy-base-roi'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-copy-base-roi [options]'

class fermipy.jobs.target_sim.**RandomDirGen**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to generate random sky directions inside an ROI

This is useful for parallelizing analysis using the fermipy.jobs module.

Parameters

- **outfile** (*<type 'str'>*) – Path to output file. [None]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **rand_config** (*<type 'str'>*) – Path to config file for genaration random sky dirs [None]

appname = 'fermipy-random-dir-gen'

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'o

description = 'Generate random sky directions in an ROI'

linkname_default = 'random-dir-gen'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-random-dir-gen [options]'

class fermipy.jobs.target_sim.**SimulateROI**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class wrap an analysis script.

This is useful for parallelizing analysis using the fermipy.jobs module.

Parameters

- **seed** (*<type 'int'>*) – Seed number for first simulation. [0]
- **sim_profile** (*<type 'str'>*) – Name of the profile to use for simulation. [default]
- **sim** (*<type 'str'>*) – Name of the simulation scenario. [None]
- **nsims** (*<type 'int'>*) – Number of simulations to run. [20]
- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **profiles** (*<type 'list'>*) – List of profiles to analyze [[]]

appname = 'fermipy-simulate-roi'

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'n

description = 'Run simulated analysis of a single ROI'

linkname_default = 'simulate-roi'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-simulate-roi [options]'

class fermipy.jobs.target_plotting.**PlotCastro**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to plot an SED as a 'Castro' plot.

Parameters

- **outfile** (*<type 'str'>*) – Path to output file. [None]
- **infile** (*<type 'str'>*) – Path to input file. [None]

appname = 'fermipy-plot-castro'

default_options = {'infile': (None, 'Path to input file.', <type 'str'>), 'outfile':

description = 'Plot likelihood v. flux normalization and energy'

linkname_default = 'plot-castro'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-plot-castro [options]'

High-level analysis job dispatch

These are `ScatterGather` sub-classes that invoke the `Link` sub-classes listed above.

class fermipy.jobs.target_analysis.**AnalyzeROI_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for the *AnalyzeROI* class.

This loops over all the targets defined in the target list.

Parameters

- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **make_plots** (*<type 'bool'>*) – Make plots [False]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **targetlist** (*<type 'str'>*) – Path to the target list. [None]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]

appname = 'fermipy-analyze-roi-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *AnalyzeROI*

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'ma

description = 'Run analyses on a series of ROIs'

job_time = 1500

usage = 'fermipy-analyze-roi-sg [options]'

class fermipy.jobs.target_analysis.**AnalyzeSED_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

This loops over all the targets defined in the target list, and over all the profiles defined for each target.

Parameters

- **skydirs** (*<type 'str'>*) – Yaml file with blank sky directions. [None]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]
- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **make_plots** (*<type 'bool'>*) – Make plots [False]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **targetlist** (*<type 'str'>*) – Path to the target list. [None]

appname = 'fermipy-analyze-sed-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *AnalyzeSED*

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'ma

description = 'Run analyses on a series of ROIs'

job_time = 1500

usage = 'fermipy-analyze-sed-sg [options]'

class fermipy.jobs.target_collect.**CollectSED_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *CollectSED*

This loops over all the targets defined in the target list

Parameters

- **write_summary** (*<type 'bool'>*) – Write file with summary of collected results [False]
- **write_full** (*<type 'bool'>*) – Write file with full collected results [False]
- **seed** (*<type 'int'>*) – Seed number for first simulation. [0]
- **nsims** (*<type 'int'>*) – Number of simulations to run. [20]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **targetlist** (*<type 'str'>*) – Path to the target list. [None]
- **sim** (*<type 'str'>*) – Name of the simulation scenario. [None]

appname = 'fermipy-collect-sed-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *CollectSED*

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'n

description = 'Collect SED data from a set of simulations for a series of ROIs'

job_time = 120

usage = 'fermipy-collect-sed-sg [options]'

class fermipy.jobs.target_sim.**CopyBaseROI_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

This adds the following arguments:

Parameters

- **extracopy** (*<type 'list'>*) – Extra files to copy [[]]
- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **targetlist** (*<type 'str'>*) – Path to the target list. [None]
- **sim** (*<type 'str'>*) – Name of the simulation scenario. [None]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]

appname = 'fermipy-copy-base-roi-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *CopyBaseROI*

default_options = {'extracopy': ([], 'Extra files to copy', <type 'list'>), 'roi_base

description = 'Run analyses on a series of ROIs'

job_time = 60

usage = 'fermipy-copy-base-roi-sg [options]'

class fermipy.jobs.target_sim.**RandomDirGen_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

This adds the following arguments:

Parameters

- **rand_config** (*<type 'str'>*) – Path to config file for genaration random sky dirs [None]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **targetlist** (*<type 'str'>*) – Path to the target list. [None]
- **sim** (*<type 'str'>*) – Name of the simulation scenario. [None]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]

appname = 'fermipy-random-dir-gen-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *RandomDirGen*

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'r

description = 'Run analyses on a series of ROIs'

job_time = 60

usage = 'fermipy-random-dir-gen-sg [options]'

class fermipy.jobs.target_sim.**SimulateROI_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

This adds the following arguments:

Parameters

- **sim_profile** (*<type 'str'>*) – Name of the profile to use for simulation. [default]
- **seed** (*<type 'int'>*) – Seed number for first simulation. [0]
- **nsims** (*<type 'int'>*) – Number of simulations to run. [20]
- **roi_baseline** (*<type 'str'>*) – Key for roi baseline file. [fit_baseline]
- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]
- **config** (*<type 'str'>*) – Path to fermipy config file. [None]
- **targetlist** (*<type 'str'>*) – Path to the target list. [None]
- **sim** (*<type 'str'>*) – Name of the simulation scenario. [None]
- **nsims_job** (*<type 'int'>*) – Number of simulations to run per job. [0]

appname = 'fermipy-simulate-roi-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *SimulateROI*

default_options = {'config': (None, 'Path to fermipy config file.', <type 'str'>), 'n

description = 'Run analyses on a series of ROIs'

job_time = 1500

usage = 'fermipy-simulate-roi-sg [options]'

class fermipy.jobs.target_plotting.**PlotCastro_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for the *PlotCastro* class.

This loops over all the targets defined in the target list.

Parameters

- **ttype** (*<type 'str'>*) – Type of target being analyzed. [None]

- **targetlist** (*<type 'str'>*) – Path to the target list. [None]

appname = 'fermipy-plot-castro-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *PlotCastro*

default_options = {'targetlist': (None, 'Path to the target list.', <type 'str'>), 't

description = 'Make castro plots for set of targets'

job_time = 60

usage = 'fermipy-plot-castro-sg [options]'

Batch and System Interfaces

Abstract interface for interactions with system for launching jobs.

class fermipy.jobs.sys_interface.**SysInterface**(***kwargs*)
Bases: object

Base class to handle job dispatching interface

classmethod check_job(*job_details*)
Check the status of a specfic job

clean_jobs(*link*, *job_dict=None*, *clean_all=False*)
Clean up all the jobs associated with this link.

Returns a JobStatus enum

dispatch_job(*link*, *key*, *job_archive*, *stream=<open file '<stdout>', mode 'w'>*)
Function to dispatch a single job

Parameters

- **link** (Link) – Link object that sendes the job
- **key** (*str*) – Key used to identify this particular job
- **job_archive** (JobArchive) – Archive used to keep track of jobs
- **JobDetails object** (*Returns*) –

dispatch_job_hook(*link*, *key*, *job_config*, *logfile*, *stream=<open file '<stdout>', mode 'w'>*)
Hook to dispatch a single job

string_exited = 'Exited with exit code'

string_successful = 'Successfully completed'
C'tor

submit_jobs(*link*, *job_dict=None*, *job_archive=None*, *stream=<open file '<stdout>', mode 'w'>*)
Run the Link with all of the items job_dict as input.

If job_dict is None, the job_dict will be take from link.jobs

Returns a JobStatus enum

fermipy.jobs.sys_interface.**check_log**(*logfile*, *exited='Exited with exit code'*, *successful='Successfully completed'*)

Check a log file to determine status of LSF job

Often logfile doesn't exist because the job hasn't begun to run. It is unclear what you want to do in that case...

Parameters

- **logfile** (*str*) – String with path to logfile
- **exited** (*str*) – Value to check for in existing logfile for exit with failure
- **successful** (*str*) – Value to check for in existing logfile for success
- **str, one of 'Pending', 'Running', 'Done', 'Failed'** (*Returns*) –

fermipy.jobs.sys_interface.**clean_job**(*logfile*, *outfiles*, *dry_run=False*)

Removes log file and files created by failed jobs.

If dry_run is True, print name of files to be removed, but do not remove them.

fermipy.jobs.sys_interface.**remove_file**(*filepath*, *dry_run=False*)

Remove the file at filepath

Catches exception if the file does not exist.

If dry_run is True, print name of file to be removed, but do not remove it.

Implementation of ScatterGather class for dealing with LSF batch jobs

class fermipy.jobs.native_impl.**NativeInterface**(***kwargs*)

Bases: *fermipy.jobs.sys_interface.SysInterface*

Implmentation of ScatterGather that uses the native system

dispatch_job_hook(*link*, *key*, *job_config*, *logfile*, *stream=<open file '<stdout>', mode 'w'>*)

Send a single job to be executed

Parameters

- **link** (fermipy.jobs.chain.Link) – The link used to invoke the command we are running
- **key** (*str*) – A string that identifies this particular instance of the job
- **job_config** (*dict*) – A dictionrary with the arguments for the job. Used with the self._command_template job template
- **logfile** (*str*) – The logfile for this job, may be used to check for success/ failure

string_exited = 'Exited with exit code'

string_successful = 'Successfully completed'

submit_jobs(*link*, *job_dict=None*, *job_archive=None*, *stream=<open file '<stdout>', mode 'w'>*)

Submit all the jobs in job_dict

fermipy.jobs.native_impl.**get_native_default_args**()

Get the correct set of batch jobs arguments.

Implementation of ScatterGather interface class for dealing with LSF batch jobs at SLAC

class fermipy.jobs.slac_impl.**SlacInterface**(***kwargs*)

Bases: *fermipy.jobs.sys_interface.SysInterface*

Implmentation of ScatterGather that uses LSF

dispatch_job_hook(*link*, *key*, *job_config*, *logfile*, *stream=<open file '<stdout>', mode 'w'>*)
Send a single job to the LSF batch

Parameters

- **link** (fermipy.jobs.chain.Link) – The link used to invoke the command we are running
- **key** (*str*) – A string that identifies this particular instance of the job
- **job_config** (*dict*) – A dictionrary with the arguments for the job. Used with the self._command_template job template
- **logfile** (*str*) – The logfile for this job, may be used to check for success/ failure

string_exited = 'Exited with exit code'

string_successful = 'Successfully completed'

submit_jobs(*link*, *job_dict=None*, *job_archive=None*, *stream=<open file '<stdout>', mode 'w'>*)
Submit all the jobs in job_dict

fermipy.jobs.slac_impl.**build_bsub_command**(*command_template*, *lsf_args*)
Build and return a lsf batch command template

The structure will be 'bsub -s <key> <value> <command_template>' where <key> and <value> refer to items in lsf_args

fermipy.jobs.slac_impl.**get_lsf_status**()
Count and print the number of jobs in various LSF states

fermipy.jobs.slac_impl.**get_slac_default_args**(*job_time=1500*)
Create a batch job interface object.

Parameters **job_time** (*int*) – Expected max length of the job, in seconds. This is used to select the batch queue and set the job_check_sleep parameter that sets how often we check for job completion.

fermipy.jobs.slac_impl.**make_gpfs_path**(*path*)
Make a gpfs version of a file path. This just puts /gpfs at the beginning instead of /nfs

fermipy.jobs.slac_impl.**make_nfs_path**(*path*)
Make a nfs version of a file path. This just puts /nfs at the beginning instead of /gpfs

Factory module to return the default interace to the batch farm

fermipy.jobs.batch.**get_batch_job_args**(*job_time=1500*)
Get the correct set of batch jobs arguments.

Parameters **job_time** (*int*) – Expected max length of the job, in seconds. This is used to select the batch queue and set the job_check_sleep parameter that sets how often we check for job completion.

Returns **job_args** – Dictionary of arguments used to submit a batch job

Return type dict

fermipy.jobs.batch.**get_batch_job_interface**(*job_time=1500*)
Create a batch job interface object.

Parameters **job_time** (*int*) – Expected max length of the job, in seconds. This is used to select the batch queue and set the job_check_sleep parameter that sets how often we check for job completion.

Returns **job_interfact** – Object that manages interactions with batch farm

Return type SysInterface

File Archive module

Classes and utilites to keep track of files associated to an analysis.

The main class is *FileArchive*, which keep track of all the files associated to an analysis.

The *FileHandle* helper class encapsulates information on a particular file.

class fermipy.jobs.file_archive.**FileArchive**(***kwargs*)
Bases: object

Class that keeps track of the status of files used in an analysis

Parameters

- **table_file** (*str*) – Path to the file used to persist this *FileArchive*
- **table** (astropy.table.Table) – Persistent representation of this *FileArchive*
- **cache** (OrderedDict) – Transient representation of this *FileArchive*
- **base_path** (*str*) – Base file path for all files in this *FileArchive*

base_path
Return the base file path for all files in this *FileArchive*

classmethod build_archive(***kwargs*)
Return the singleton *FileArchive* instance, building it if needed

cache
Return the transiet representation of this *FileArchive*

classmethod get_archive()
Return the singleton *FileArchive* instance

get_file_ids(*file_list*, *creator=None*, *status=0*, *file_dict=None*)
Get or create a list of file ids based on file names

Parameters

- **file_list** (*list*) – The paths to the file
- **creatror** (*int*) – A unique key for the job that created these files
- **status** (*FileStatus*) – Enumeration giving current status of files
- **file_dict** (*FileDict*) – Mask giving flags set on this file
- **list of integers** (*Returns*) –

get_file_paths(*id_list*)
Get a list of file paths based of a set of ids

Parameters

- **id_list** (*list*) – List of integer file keys
- **list of file paths** (*Returns*) –

get_handle(*filepath*)
Get the *FileHandle* object associated to a particular file

register_file (*filepath*, *creator*, *status=0*, *flags=0*)
Register a file in the archive.

If the file already exists, this raises a KeyError

Parameters

- **filepath** (*str*) – The path to the file
- **creatror** (*int*) – A unique key for the job that created this file
- **status** (*FileStatus*) – Enumeration giving current status of file
- **flags** (*FileFlags*) – Enumeration giving flags set on this file
- **FileHandle** (*Returns*) –

table
Return the persistent representation of this *FileArchive*

table_file
Return the path to the file used to persist this *FileArchive*

update_file (*filepath*, *creator*, *status*)
Update a file in the archive

If the file does not exists, this raises a KeyError

Parameters

- **filepath** (*str*) – The path to the file
- **creatror** (*int*) – A unique key for the job that created this file
- **status** (*FileStatus*) – Enumeration giving current status of file
- **FileHandle** (*Returns*) –

update_file_status ()
Update the status of all the files in the archive

write_table_file (*table_file=None*)
Write the table to self._table_file

class fermipy.jobs.file_archive.**FileDict** (***kwargs*)
Bases: object

Small class to keep track of files used & createed by a link.

Parameters

- **file_args** (*dict*) – Dictionary mapping argument to *FileFlags* enum
- **file_dict** (*dict*) – Dictionary mapping file path to *FileFlags* enum

chain_input_files
Return a list of the input files needed by this chain.

For Link sub-classes this will return only those files that were not created by any internal Link

chain_output_files
Return a list of the all the output files produced by this link.

For Link sub-classes this will return only those files that were not marked as internal files or marked for removal.

gzip_files
Return a list of the files compressed by this link.

This returns all files that were explicitly marked for compression.

input_files
Return a list of the input files needed by this link.

For `Link` sub-classes this will return the union of all the input files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

input_files_to_stage
Return a list of the input files needed by this link.

For `Link` sub-classes this will return the union of all the input files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

internal_files
Return a list of the intermediate files produced by this link.

This returns all files that were explicitly marked as internal files.

items()
Return iterator over self.file_dict

latch_file_info(*args*)
Extract the file paths from a set of arguments

output_files
Return a list of the output files produced by this link.

For `Link` sub-classes this will return the union of all the output files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

output_files_to_stage
Return a list of the input files needed by this link.

For `Link` sub-classes this will return the union of all the input files of each internal `Link`.

That is to say this will include files produced by one `Link` in a `Chain` and used as input to another `Link` in the `Chain`

print_chain_summary(*stream=<open file '<stdout>', mode 'w'>*, *indent=''*)
Print a summary of the files in this file dict.

This version uses chain_input_files and chain_output_files to count the input and output files.

print_summary(*stream=<open file '<stdout>', mode 'w'>*, *indent=''*)
Print a summary of the files in this file dict.

This version explictly counts the union of all input and output files.

temp_files
Return a list of the temporary files produced by this link.

This returns all files that were explicitly marked for removal.

update(*file_dict*)
Update self with values from a dictionary mapping file path [str] to *FileFlags* enum

class fermipy.jobs.file_archive.**FileFlags**
Bases: object

Bit masks to indicate file types

gz_mask = 8

in_ch_mask = 23

in_stage_mask = 33

input_mask = 1

internal_mask = 16

no_flags = 0

out_ch_mask = 22

out_stage_mask = 34

output_mask = 2

rm_mask = 4

rmint_mask = 20

stageable = 32

class fermipy.jobs.file_archive.**FileHandle**(***kwargs*)
Bases: object

Class to keep track of infomration about a file file.

Parameters

- **key** (*int*) – Unique id for this particular file
- **creator** (*int*) – Unique id for the job that created this file
- **timestamp** (*int*) – File creation time cast as an int
- **status** (*FileStatus*) – Enum giving current status of file
- **flags** (*FileFlags*) – Mask giving flags set on this file
- **path** (*str*) – Path to file

append_to_table(*table*)
Add this instance as a row on a astropy.table.Table

check_status(*basepath=None*)
Check on the status of this particular file

classmethod create_from_row(*table_row*)
Build and return a *FileHandle* from an astropy.table.row.Row

classmethod make_dict(*table*)
Build and return a dict of *FileHandle* from an astropy.table.Table

The dictionary is keyed by FileHandle.key, which is a unique integer for each file

static make_table(*file_dict*)
Build and return an astropy.table.Table to store *FileHandle*

update_table_row(*table*, *row_idx*)
Update the values in an astropy.table.Table for this instances

class fermipy.jobs.file_archive.**FileStageManager**(*scratchdir*, *workdir*)
Bases: object

Small class to deal with staging files to and from a scratch area

construct_scratch_path(*dirname*, *basename*)
Construct and return a path in the scratch area.

This will be <self.scratchdir>/<dirname>/<basename>

static copy_from_scratch(*file_mapping*, *dry_run=True*)
Copy output files from scratch area

static copy_to_scratch(*file_mapping*, *dry_run=True*)
Copy input files to scratch area

get_scratch_path(*local_file*)
Construct and return a path in the scratch area from a local file.

static make_scratch_dirs(*file_mapping*, *dry_run=True*)
Make any directories need in the scratch area

map_files(*local_files*)
Build a dictionary mapping local paths to scratch paths.

Parameters

- **local_files** (*list*) – List of filenames to be mapped to scratch area
- **dict** (*Returns*) – Mapping local_file : fullpath of scratch file

split_local_path(*local_file*)
Split the local path into a directory name and a file name

If local_file is in self.workdir or a subdirectory of it, the directory will consist of the relative path from workdir.

If local_file is not in self.workdir, directory will be empty.

Returns (dirname, basename)

class fermipy.jobs.file_archive.**FileStatus**
Bases: object

Enumeration of file status types

exists = 2

expected = 1

missing = 3

no_file = 0

superseded = 4

temp_removed = 5

fermipy.jobs.file_archive.**get_timestamp**()
Get the current time as an integer

fermipy.jobs.file_archive.**get_unique_match**(*table*, *colname*, *value*)
Get the row matching value for a particular column. If exactly one row matchs, return index of that row, Otherwise raise KeyError.

fermipy.jobs.file_archive.**main_browse**()
Entry point for command line use for browsing a FileArchive

Job Archive module

Classes and utilites to keep track the various jobs that are running in an analysis pipeline.

The main class is *JobArchive*, which keep track of all the jobs associated to an analysis.

The *JobDetails* helper class encapsulates information on a instance of running a job.

class fermipy.jobs.job_archive.**JobArchive**(***kwargs*)
Bases: object

Class that keeps of all the jobs associated to an analysis.

Parameters

- **table_file** (*str*) – Path to the file used to persist this *JobArchive*
- **table** (astropy.table.Table) – Persistent representation of this *JobArchive*
- **table_ids** (astropy.table.Table) – Ancillary table with information about file ids
- **file_archive** (FileArchive) – Archive with infomation about all this files used and produced by this analysis

classmethod build_archive(***kwargs*)
Return the singleton *JobArchive* instance, building it if needed

classmethod build_temp_job_archive()
Build and return a *JobArchive* using defualt locations of persistent files.

cache
Return the transiet representation of this *JobArchive*

file_archive
Return the FileArchive with infomation about all the files used and produced by this analysis

classmethod get_archive()
Return the singleton *JobArchive* instance

get_details(*jobname*, *jobkey*)
Get the *JobDetails* associated to a particular job instance

make_job_details(*row_idx*)
Create a *JobDetails* from an astropy.table.row.Row

register_job(*job_details*)
Register a job in this *JobArchive*

register_job_from_link(*link*, *key*, ***kwargs*)
Register a job in the *JobArchive* from a Link object

register_jobs(*job_dict*)
Register a bunch of jobs in this archive

remove_jobs(*mask*)
Mark all jobs that match a mask as 'removed'

table
Return the persistent representation of this *JobArchive*

table_file
Return the path to the file used to persist this *JobArchive*

table_ids
Return the rpersistent epresentation of the ancillary info of this *JobArchive*

update_job(*job_details*)
Update a job in the *JobArchive*

update_job_status(*checker_func*)
Update the status of all the jobs in the archive

write_table_file(*job_table_file=None*, *file_table_file=None*)
Write the table to self._table_file

class fermipy.jobs.job_archive.**JobDetails**(***kwargs*)
Bases: object

A simple structure to keep track of the details of each of the sub-proccess jobs.

Parameters

- **dbkey** (*int*) – A unique key to identify this job
- **jobname** (*str*) – A name used to idenfity this job
- **jobkey** (*str*) – A string to identify this instance of the job
- **appname** (*str*) – The executable inovked to run the job
- **logfile** (*str*) – The logfile for this job, may be used to check for success/ failure
- **job_config** (*dict*) – A dictionrary with the arguments for the job
- **parent_id** (*int*) – Unique key identifying the parent job
- **infile_ids** (*list of int*) – Keys to identify input files to this job
- **outfile_ids** (*list of int*) – Keys to identify output files from this job
- **rmfile_ids** (*list of int*) – Keys to identify temporary files removed by this job
- **intfile_ids** (*list of int*) – Keys to identify internal files
- **status** (*int*) – Current job status, one of the enums above

append_to_tables(*table*, *table_ids*)
Add this instance as a row on a astropy.table.Table

check_status_logfile(*checker_func*)
Check on the status of this particular job using the logfile

classmethod create_from_row(*table_row*)
Create a *JobDetails* from an astropy.table.row.Row

fullkey
Return the fullkey for this job fullkey = <jobkey>@<jobname>

get_file_ids(*file_archive*, *creator=None*, *status=0*)
Fill the file id arrays from the file lists

Parameters

- **file_archive** (FileArchive) – Used to look up file ids
- **creator** (*int*) – A unique key for the job that created these file
- **status** (FileStatus) – Enumeration giving current status thse files

get_file_paths(*file_archive*, *file_id_array*)
Get the full paths of the files used by this object from the the id arrays

Parameters

- **file_archive** (FileArchive) – Used to look up file ids
- **file_id_array** (numpy.array) – Array that remaps the file indexes

classmethod make_dict(*table*)
Build a dictionary map int to *JobDetails* from an astropy.table.Table

static make_fullkey(*jobname*, *jobkey='__top__'*)
Combine jobname and jobkey to make a unique key fullkey = <jobkey>@<jobname>

static make_tables(*job_dict*)
Build and return an astropy.table.Table' to store `JobDetails

static split_fullkey(*fullkey*)
Split fullkey to make extract jobname, jobkey fullkey = <jobkey>@<jobname>

topkey = '__top__'

update_table_row(*table*, *row_idx*)
Add this instance as a row on a astropy.table.Table

class fermipy.jobs.job_archive.**JobStatus**
Bases: object

Enumeration of job status types

done = 5

failed = 6

no_job = -1

not_ready = 1

partial_failed = 7

pending = 3

ready = 2

removed = 8

running = 4

unknown = 0

class fermipy.jobs.job_archive.**JobStatusVector**
Bases: object

Vector that counts the status of jobs and returns an overall status flag based on those

get_status()
Return an overall status based on the number of jobs in various states.

n_done
Return the number of successfully completed jobs

n_failed
Return the number of failed jobs

n_pending
Return the number jobs submitted to batch, but not yet running

n_running
Return the number of running jobs

n_total
Return the total number of jobs

n_waiting
Return the number of jobs in various waiting states

reset()
Reset the counters

fermipy.jobs.job_archive.**main_browse**()
Entry point for command line use for browsing a JobArchive

1.3.12 fermipy.diffuse subpackage

Module contents

Configuration, binning, default options, etc...

Small helper class to represent the binning used for a single component of a summed likelihood in diffuse analysis

class fermipy.diffuse.binning.**Component**(**kwargs)
Bases: object

Small helper class to represent the binning used for a single component of a summed likelihood in diffuse analysis

Parameters

- **log_emin** (*float*) – Log base 10 of minimum energy for this component
- **log_emax** (*float*) – Log base 10 of maximum energy for this component
- **enumbins** (*int*) – Number of energy bins for this component
- **zmax** (*float*) – Maximum zenith angle cube for this component in degrees
- **mktimefilters** (*list*) – Filters for gtmktime.
- **hpx_order** (*int*) – HEALPix order to use for this component
- **coordsys** (*str*) – Coodinate system, 'CEL' or 'GAL'

classmethod build_from_energy_dict(*ebin_name*, *input_dict*)
Build a list of components from a dictionary for a single energy range

classmethod build_from_yamlfile(*yamlfile*)
Build a list of components from a yaml file

classmethod build_from_yamlstr(*yamlstr*)
Build a list of components from a yaml string

emax
Maximum energy for this component

emin
Minimum energy for this component

evtype
Event type bit mask for this component

make_key(*format_str*)
Make a key to identify this compoment

format_str is formatted using object __dict__

Analysis framework for all-sky diffuse emission fitting

Handle the naming conventions for composite likelihood analysis

class fermipy.diffuse.name_policy.**NameFactory**(***kwargs*)

Bases: object

Helper class to define file names and keys consistently.

angprofile(***kwargs*)

return the file name for sun or moon angular profiles

angprofile_format = 'templates/profile_{sourcekey}.fits'

bexpcube(***kwargs*)

return the name of a binned exposure cube file

bexpcube_format = 'bexp_cubes/bexcube_{dataset}_{mktime}_{component}_{coordsys}_{irf_v

bexpcube_moon(***kwargs*)

return the name of a binned exposure cube file

bexpcube_sun(***kwargs*)

return the name of a binned exposure cube file

bexpcubemoon_format = 'bexp_cubes/bexcube_{dataset}_{mktime}_{component}_{irf_ver}_moo

bexpcubesun_format = 'bexp_cubes/bexcube_{dataset}_{mktime}_{component}_{irf_ver}_sun.

catalog_split_yaml(***kwargs*)

return the name of a catalog split yaml file

catalog_split_yaml_format = 'models/catalog_{sourcekey}.yaml'

ccube(***kwargs*)

return the name of a counts cube file

ccube_format = 'counts_cubes/ccube_{dataset}_{mktime}_{component}_{coordsys}.fits'

comp_srcmdl_xml(***kwargs*)

return the name of a source model file

comp_srcmdl_xml_format = 'analysis/model_{modelkey}/srcmdl_{modelkey}_{component}.xml'

component(***kwargs*)

Return a key that specifies data the sub-selection

component_format = '{zcut}_{ebin}_{psftype}'

dataset(***kwargs*)

Return a key that specifies the data selection

dataset_format = '{data_pass}_{data_ver}_{data_time}_{evclass}'

diffuse_template(***kwargs*)

return the file name for other diffuse map templates

diffuse_template_format = 'templates/template_{sourcekey}.fits'

evclassmask(*evclass_str*)

Get the bitmask for a particular event class

ft1file(***kwargs*)

return the name of the input ft1 file list

ft1file_format = '{dataset}_{zcut}.lst'

ft2file(***kwargs*)
return the name of the input ft2 file list

ft2file_format = 'ft2_files/ft2_{data_time}.lst'

fullpath(***kwargs*)
Return a full path name for a given file

fullpath_format = '{basedir}/{localpath}'

galprop_gasmap(***kwargs*)
return the file name for Galprop input gasmaps

galprop_gasmap_format = 'gasmap/{sourcekey}_{projtype}_{galprop_run}.gz'

galprop_ringkey(***kwargs*)
return the sourcekey for galprop input maps : specifies the component and ring

galprop_ringkey_format = '{source_name}_{ringkey}'

galprop_rings_yaml(***kwargs*)
return the name of a galprop rings merging yaml file

galprop_rings_yaml_format = 'models/galprop_rings_{galkey}.yaml'

galprop_sourcekey(***kwargs*)
return the sourcekey for merged galprop maps : specifies the merged component and merging scheme

galprop_sourcekey_format = '{source_name}_{galpropkey}'

generic(*input_string*, ***kwargs*)
return a generic filename for a given dataset and component

irf_ver(***kwargs*)
Get the name of the IRF version

irfs(***kwargs*)
Get the name of IFRs associted with a particular dataset

ltcube(***kwargs*)
return the name of a livetime cube file

ltcube_format = 'lt_cubes/ltcube_{data_time}_{mktime}_{zcut}.fits'

ltcube_moon(***kwargs*)
return the name of a livetime cube file

ltcube_sun(***kwargs*)
return the name of a livetime cube file

ltcubemoon_format = 'sunmoon/ltcube_{data_time}_{mktime}_{zcut}_moon.fits'

ltcubesun_format = 'sunmoon/ltcube_{data_time}_{mktime}_{zcut}_sun.fits'

make_filenames(***kwargs*)
Make a dictionary of filenames for various types

master_srcmdl_xml(***kwargs*)
return the name of a source model file

master_srcmdl_xml_format = 'analysis/model_{modelkey}/srcmdl_{modelkey}_master.xml'

mcube(***kwargs*)
return the name of a model cube file

mcube_format = 'model_cubes/mcube_{sourcekey}_{dataset}_{mktime}_{component}_{coordsys

merged_gasmap (***kwargs*)
return the file name for Galprop merged gasmaps

merged_gasmap_format = 'merged_gasmaps/{sourcekey}_{projtype}.fits'

merged_sourcekey (***kwargs*)
return the sourcekey for merged sets of point sources : specifies the catalog and merging rule

merged_sourcekey_format = '{catalog}_{rulekey}'

merged_srcmaps (***kwargs*)
return the name of a source map file

merged_srcmaps_format = 'analysis/model_{modelkey}/srcmaps_{dataset}_{mktime}_{compone

mktime (***kwargs*)
return the name of a selected events ft1file

mktime_format = 'counts_cubes/mktime_{dataset}_{mktime}_{component}.fits'

model_yaml (***kwargs*)
return the name of a model yaml file

model_yaml_format = 'models/model_{modelkey}.yaml'

nested_srcmdl_xml (***kwargs*)
return the file name for source model xml files of nested sources

nested_srcmdl_xml_format = 'srcmdls/{sourcekey}_sources.xml'

residual_cr (***kwargs*)
Return the name of the residual CR analysis output files

residual_cr_format = 'residual_cr/residual_cr_{dataset}_{mktime}_{component}_{coordsys

select (***kwargs*)
return the name of a selected events ft1file

select_format = 'counts_cubes/select_{dataset}_{component}.fits'

sourcekey (***kwargs*)
Return a key that specifies the name and version of a source or component

sourcekey_format = '{source_name}_{source_ver}'

spectral_template (***kwargs*)
return the file name for spectral templates

spectral_template_format = 'templates/spectral_{sourcekey}.txt'

srcmaps (***kwargs*)
return the name of a source map file

srcmaps_format = 'srcmaps/srcmaps_{sourcekey}_{dataset}_{mktime}_{component}_{coordsys

srcmdl_xml (***kwargs*)
return the file name for source model xml files

srcmdl_xml_format = 'srcmdls/{sourcekey}.xml'

stamp (***kwargs*)
Return the path for a stamp file for a scatter gather job

stamp_format = 'stamps/{linkname}.stamp'

template_sunmoon(***kwargs*)
return the file name for sun or moon template files

templatesunmoon_format = 'templates/template_{sourcekey}_{zcut}.fits'

update_base_dict(*yamlfile*)
Update the values in baseline dictionary used to resolve names

Utilities and tools

Classes and utilities that manage spectral model specific to diffuse analyses

class fermipy.diffuse.spectral.**SpectralLibrary**(*spectral_dict*)
Bases: object

A small helper class that serves as an alias dictionary for spectral models

classmethod create_from_yaml(*yamlfile*)
Create the dictionary for a yaml file

classmethod create_from_yamlstr(*yamlstr*)
Create the dictionary for a yaml file

update(*spectral_dict*)
Update the dictionary

Small helper class to represent the selection of mktime filters used in the analysis

class fermipy.diffuse.timefilter.**MktimeFilterDict**(*aliases*, *selections*)
Bases: object

Small helper class toselection of mktime filters used in the analysis

static build_from_yamlfile(*yamlfile*)
Build a list of components from a yaml file

items()
Return the itratetor over key, value pairs

keys()
Return the iterator over keys

values()
Return the itratetor over values

Classes and utilities that create fermipy source objects

class fermipy.diffuse.source_factory.**SourceFactory**
Bases: object

Small helper class to build and keep track of sources

add_sources(*source_info_dict*)
Add all of the sources in source_info_dict to this factory

static build_catalog(***kwargs*)
Build a fermipy.catalog.Catalog object

Parameters

- **catalog_type** (*str*) – Specifies catalog type, options include 2FHL | 3FGL | 4FGLP
- **catalog_file** (*str*) – FITS file with catalog tables
- **catalog_extdir** (*str*) – Path to directory with extended source templates

classmethod copy_selected_sources(*roi*, *source_names*)
Build and return a *fermipy.roi_model.ROIModel* object by copying selected sources from another such object

static make_fermipy_roi_model_from_catalogs(*cataloglist*)
Build and return a `fermipy.roi_model.ROIModel object from a list of fermipy.catalog.Catalog` objects

classmethod make_roi(*sources=None*)
Build and return a *fermipy.roi_model.ROIModel* object from a dict with information about the sources

source_info_dict
Return the dictionary of source_info objects used to build sources

sources
Return the dictionary of sources

fermipy.diffuse.source_factory.**make_catalog_sources**(*catalog_roi_model*, *source_names*)
Construct and return dictionary of sources that are a subset of sources in catalog_roi_model.

Parameters

- **catalog_roi_model** (dict or *fermipy.roi_model.ROIModel*) – Input set of sources
- **source_names** (*list*) – Names of sourcs to extract
- **dict mapping source_name to fermipy.roi_model.Source object** (*Returns*) –

fermipy.diffuse.source_factory.**make_composite_source**(*name*, *spectrum*)
Construct and return a *fermipy.roi_model.CompositeSource* object

fermipy.diffuse.source_factory.**make_isotropic_source**(*name*, *Spectrum_Filename*, *spectrum*)
Construct and return a *fermipy.roi_model.IsoSource* object

fermipy.diffuse.source_factory.**make_mapcube_source**(*name*, *Spatial_Filename*, *spectrum*)
Construct and return a *fermipy.roi_model.MapCubeSource* object

fermipy.diffuse.source_factory.**make_point_source**(*name*, *src_dict*)
Construct and return a *fermipy.roi_model.Source* object

fermipy.diffuse.source_factory.**make_sources**(*comp_key*, *comp_dict*)
Make dictionary mapping component keys to a source or set of sources

Parameters

- **comp_key** (*str*) – Key used to access sources
- **comp_dict** (*dict*) – Information used to build sources
- **OrderedDict maping comp_key to fermipy.roi_model.Source** (*return*) –

Prepare data for diffuse all-sky analysis

fermipy.diffuse.utils.**create_inputlist**(*arglist*)
Read lines from a file and makes a list of file names.

Removes whitespace and lines that start with '#' Recursively read all files with the extension '.lst'

fermipy.diffuse.utils.**readlines**(*arg*)
Read lines from a file into a list.

Removes whitespace and lines that start with '#'

Helper classes to manage model building

class fermipy.diffuse.model_component.**ModelComponentInfo**(***kwargs*)
Bases: object

Information about a model component

Parameters

- **source_name** (*str*) – The name given to the component, e.g., loop_I or moon
- **source_ver** (*str*) – Key to indentify the model version of the source, e.g., v00
- **sourcekey** (*str*) – Key that identifies this component, e.g., loop_I_v00 or moon_v00
- **model_type** (*str*) – Type of model, 'MapCubeSource' | 'IsoSource' | 'CompositeSource' | 'Catalog' | 'PointSource'
- **srcmdl_name** (*str*) – Name of the xml file with the xml just for this component
- **moving** (*bool*) – Flag for moving sources (i.e., the sun and moon)
- **selection_dependent** (*bool*) – Flag for selection dependent sources (i.e., the residual cosmic ray model)
- **no_psf** (*bool*) – Flag to indicate that we do not smear this component with the PSF
- **components** (*dict*) – Sub-dictionary of *ModelComponentInfo* objects for moving and selection_dependent sources
- **comp_key** (*str*) – Component key for this component of moving and selection_dependent sources

add_component_info(*compinfo*)
Add sub-component specific information to a particular data selection

Parameters **compinfo** (*ModelComponentInfo* object) – Sub-component being added

clone_and_merge_sub(*key*)
Clones self and merges clone with sub-component specific information

Parameters

- **key** (*str*) – Key specifying which sub-component
- **ModelComponentInfo object** (*Returns*) –

get_component_info(*comp*)
Return the information about sub-component specific to a particular data selection

Parameters

- **comp** (binning.Component object) – Specifies the sub-component
- **ModelComponentInfo object** (*Returns*) –

update(***kwargs*)
Update data members from keyword arguments

class fermipy.diffuse.model_component.**CatalogInfo**(***kwargs*)
Bases: object

Information about a source catalog

Parameters

- **catalog_name** (*str*) – The name given to the merged component, e.g., merged_CO or merged_HI
- **catalog_file** (*str*) – Fits file with catalog data
- **catalog_extdir** (*str*) – Directory with extended source templates
- **catalog_type** (*str*) – Identifies the format of the catalog fits file: e.g., '3FGL' or '4FGLP'
- **catalog** (fermipy.catalog.Catalog) – Catalog object
- **roi_model** (*fermipy.roi_model.ROIModel*) – Fermipy object describing all the catalog sources
- **srcmdl_name** (*str*) – Name of xml file with the catalog source model

update(***kwargs*)
Update data members from keyword arguments

class fermipy.diffuse.model_component.**GalpropMergedRingInfo**(***kwargs*)
Bases: object

Information about a set of Merged Galprop Rings

Parameters

- **source_name** (*str*) – The name given to the merged component, e.g., merged_CO or merged_HI
- **ring** (*int*) – The index of the merged ring
- **sourcekey** (*str*) – Key that identifies this component, e.g., merged_CO_1, or merged_HI_3
- **galkey** (*str*) – Key that identifies how to merge the galprop rings, e.g., 'ref'
- **galprop_run** (*str*) – Key that idenfifies the galprop run used to make the input rings
- **files** (*str*) – List of files of the input gasmap files
- **merged_gasmap** (*str*) – Filename for the merged gasmap

update(***kwargs*)
Update data members from keyword arguments

class fermipy.diffuse.model_component.**ModelComponentInfo**(***kwargs*)
Bases: object

Information about a model component

Parameters

- **source_name** (*str*) – The name given to the component, e.g., loop_I or moon
- **source_ver** (*str*) – Key to indentify the model version of the source, e.g., v00
- **sourcekey** (*str*) – Key that identifies this component, e.g., loop_I_v00 or moon_v00
- **model_type** (*str*) – Type of model, 'MapCubeSource' | 'IsoSource' | 'CompositeSource' | 'Catalog' | 'PointSource'

- **srcmdl_name** (*str*) – Name of the xml file with the xml just for this component
- **moving** (*bool*) – Flag for moving sources (i.e., the sun and moon)
- **selection_dependent** (*bool*) – Flag for selection dependent sources (i.e., the residual cosmic ray model)
- **no_psf** (*bool*) – Flag to indicate that we do not smear this component with the PSF
- **components** (*dict*) – Sub-dictionary of *ModelComponentInfo* objects for moving and selection_dependent sources
- **comp_key** (*str*) – Component key for this component of moving and selection_dependent sources

add_component_info(*compinfo*)
Add sub-component specific information to a particular data selection

Parameters **compinfo** (*ModelComponentInfo* object) – Sub-component being added

clone_and_merge_sub(*key*)
Clones self and merges clone with sub-component specific information

Parameters

- **key** (*str*) – Key specifying which sub-component
- **ModelComponentInfo object** (*Returns*) –

get_component_info(*comp*)
Return the information about sub-component specific to a particular data selection

Parameters

- **comp** (binning.Component object) – Specifies the sub-component
- **ModelComponentInfo object** (*Returns*) –

update(***kwargs*)
Update data members from keyword arguments

class fermipy.diffuse.model_component.**IsoComponentInfo**(***kwargs*)
Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component represented by a IsoSource

Parameters **Spectral_Filename** (*str*) – Name of the template file for the spatial model

class fermipy.diffuse.model_component.**PointSourceInfo**(***kwargs*)
Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component represented by a PointSource

class fermipy.diffuse.model_component.**CompositeSourceInfo**(***kwargs*)
Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component represented by a CompositeSource

Parameters

- **source_names** (*list*) – The names of the nested sources
- **catalog_info** (model_component.CatalogInfo or None) – Information about the catalog containing the nested sources
- **roi_model** (*fermipy.roi_model.ROIModel*) – Fermipy object describing the nested sources

class fermipy.diffuse.model_component.**CatalogSourcesInfo**(***kwargs*)
Bases: *fermipy.diffuse.model_component.ModelComponentInfo*

Information about a model component consisting of sources from a catalog

Parameters

- **source_names** (*list*) – The names of the nested sources
- **catalog_info** (model_component.CatalogInfo or None) – Information about the catalog containing the nested sources
- **roi_model** (*fermipy.roi_model.ROIModel*) – Fermipy object describing the nested sources

class fermipy.diffuse.diffuse_src_manager.**GalpropMapManager**(***kwargs*)
Bases: object

Small helper class to keep track of Galprop gasmaps

This keeps track of two types of dictionaries. Both are keyed by: key = {source_name}_{ring}_{galkey}

Where: {source_name} is something like 'merged_C0' {ring} is the ring index {galkey} is a key specifying which version of galprop rings to use.

The two dictionaries are: ring_dict[key] = model_component.GalpropMergedRingInfo diffuse_comp_info_dict[key]] model_component.ModelComponentInfo

The dictionaries are defined in files called. models/galprop_rings_{galkey}.yaml

diffuse_comp_info_dicts(*galkey*)
Return the components info dictionary for a particular galprop key

galkeys()
Return the list of galprop keys used

make_diffuse_comp_info(*merged_name*, *galkey*)
Make the information about a single merged component

Parameters

- **merged_name** (*str*) – The name of the merged component
- **galkey** (*str*) – A short key identifying the galprop parameters
- **Model_component.ModelComponentInfo** (*Returns*) –

make_diffuse_comp_info_dict(*galkey*)
Make a dictionary maping from merged component to information about that component

Parameters **galkey** (*str*) – A short key identifying the galprop parameters

make_merged_name(*source_name*, *galkey*, *fullpath*)
Make the name of a gasmap file for a set of merged rings

Parameters

- **source_name** (*str*) – The galprop component, used to define path to gasmap files
- **galkey** (*str*) – A short key identifying the galprop parameters
- **fullpath** (*bool*) – Return the full path name

make_ring_dict(*galkey*)
Make a dictionary mapping the merged component names to list of template files

Parameters

- **galkey** (*str*) – Unique key for this ring dictionary
- **model_component.GalpropMergedRingInfo** (*Returns*) –

make_ring_filelist(*sourcekeys*, *rings*, *galprop_run*)
Make a list of all the template files for a merged component

Parameters

- **sourcekeys** (*list-like of str*) – The names of the componenents to merge
- **rings** (*list-like of int*) – The indices of the rings to merge
- **galprop_run** (*str*) – String identifying the galprop parameters

make_ring_filename(*source_name*, *ring*, *galprop_run*)
Make the name of a gasmap file for a single ring

Parameters

- **source_name** (*str*) – The galprop component, used to define path to gasmap files
- **ring** (*int*) – The ring index
- **galprop_run** (*str*) – String identifying the galprop parameters

make_xml_name(*source_name*, *galkey*, *fullpath*)
Make the name of an xml file for a model definition for a set of merged rings

Parameters

- **source_name** (*str*) – The galprop component, used to define path to gasmap files
- **galkey** (*str*) – A short key identifying the galprop parameters
- **fullpath** (*bool*) – Return the full path name

merged_components(*galkey*)
Return the set of merged components for a particular galprop key

read_galprop_rings_yaml(*galkey*)
Read the yaml file for a partiuclar galprop key

ring_dict(*galkey*)
Return the ring dictionary for a particular galprop key

class fermipy.diffuse.diffuse_src_manager.**DiffuseModelManager**(***kwargs*)
Bases: object

Small helper class to keep track of diffuse component templates

This keeps track of the 'diffuse component infomation' dictionary

This keyed by: key = {source_name}_{source_ver} Where: {source_name} is something like 'loopI' {source_ver} is somthinng like v00

The dictioary is diffuse_comp_info_dict[key] - > model_component.ModelComponentInfo

Note that some components (those that represent moving sources or are selection depedent) will have a sub-dictionary of diffuse_comp_info_dict object for each sub-component

The compoents are defined in a file called config/diffuse_components.yaml

diffuse_comp_info(*sourcekey*)
Return the Component info associated to a particular key

make_diffuse_comp_info(*source_name*, *source_ver*, *diffuse_dict*, *components=None*, *comp_key=None*)
Make a dictionary mapping the merged component names to list of template files

Parameters

- **source_name** (*str*) – Name of the source
- **source_ver** (*str*) – Key identifying the version of the source
- **diffuse_dict** (*dict*) – Information about this component
- **comp_key** (*str*) – Used when we need to keep track of sub-components, i.e., for moving and selection dependent sources.
- **model_component.ModelComponentInfo or** (*Returns*) –
- **model_component.IsoComponentInfo** –

make_diffuse_comp_info_dict(*diffuse_sources*, *components*)
Make a dictionary maping from diffuse component to information about that component

Parameters

- **diffuse_sources** (*dict*) – Dictionary with diffuse source defintions
- **components** (*dict*) – Dictionary with event selection defintions, needed for selection depenedent diffuse components

Returns ret_dict – Dictionary mapping sourcekey to `model_component.ModelComponentInfo`

Return type dict

make_template_name(*model_type*, *sourcekey*)
Make the name of a template file for particular component

Parameters

- **model_type** (*str*) – Type of model to use for this component
- **sourcekey** (*str*) – Key to identify this component
- **filename or None if component does not require a template file** (*Returns*) –

make_xml_name(*sourcekey*)
Make the name of an xml file for a model definition of a single component

Parameters **sourcekey** (*str*) – Key to identify this component

static read_diffuse_component_yaml(*yamlfile*)
Read the yaml file for the diffuse components

sourcekeys()
Return the list of source keys

class fermipy.diffuse.catalog_src_manager.**CatalogSourceManager**(***kwargs*)
Bases: object

Small helper class to keep track of how we deal with catalog sources

This keeps track of two dictionaries

One of the dictionaries is keyed by catalog name, and contains information about complete catalogs catalog_comp_info_dicts[catalog_name] : `model_component.CatalogInfo`

The other dictionary is keyed by [{catalog_name}_{split_ver}][{split_key}] Where: {catalog_name} is something like '3FGL' {split_ver} is somthing like 'v00' and specifes how to divide sources in the catalog {split_key} refers to a specific sub-selection of sources

split_comp_info_dicts[splitkey] : `model_component.ModelComponentInfo`

build_catalog_info(*catalog_info*)
Build a CatalogInfo object

catalog_comp_info_dict(*catkey*)
Return the roi_model for an entire catalog

catalog_components(*catalog_name*, *split_ver*)
Return the set of merged components for a particular split key

catalogs()
Return the list of full catalogs used

make_catalog_comp_info(*full_cat_info*, *split_key*, *rule_key*, *rule_val*, *sources*)
Make the information about a single merged component

Parameters

- **full_cat_info** (_model_component.CatalogInfo) – Information about the full catalog
- **split_key** (*str*) – Key identifying the version of the spliting used
- **rule_key** (*str*) – Key identifying the specific rule for this component
- **rule_val** (*list*) – List of the cuts used to define this component
- **sources** (*list*) – List of the names of the sources in this component
- **CompositeSourceInfo or CatalogSourcesInfo** (*Returns*) –

make_catalog_comp_info_dict(*catalog_sources*)
Make the information about the catalog components

Parameters **catalog_sources** (*dict*) – Dictionary with catalog source defintions

Returns

- **catalog_ret_dict** (*dict*) – Dictionary mapping catalog_name to `model_component.CatalogInfo`
- **split_ret_dict** (*dict*) – Dictionary mapping sourcekey to `model_component.ModelComponentInfo`

read_catalog_info_yaml(*splitkey*)
Read the yaml file for a particular split key

split_comp_info(*catalog_name*, *split_ver*, *split_key*)
Return the info for a particular split key

split_comp_info_dict(*catalog_name*, *split_ver*)
Return the information about a particular scheme for how to handle catalog sources

splitkeys()
Return the list of catalog split keys used

class fermipy.diffuse.model_manager.**ModelComponent**(***kwargs*)
Bases: `object`

Small helper class to tie a ModelComponentInfo to a spectrum

class fermipy.diffuse.model_manager.**ModelInfo**(***kwargs*)
Bases: object

Small helper class to keep track of a single fitting model

component_names
Return the list of name of the components

edisp_disable_list()
Return the list of source for which energy dispersion should be turned off

items()
Return the key, value pairs of model components

make_model_rois(*components*, *name_factory*)
Make the fermipy roi_model objects for each of a set of binning components

make_srcmap_manifest(*components*, *name_factory*)
Build a yaml file that specfies how to make the srcmap files for a particular model

Parameters

- **components** (*list*) – The binning components used in this analysis
- **name_factory** (NameFactory) – Object that handles naming conventions
- **a dictionary that contains information about where to find the** (*Returns*) –
- **maps for each component of the model** (*source*) –

class fermipy.diffuse.model_manager.**ModelManager**(***kwargs*)
Bases: object

Small helper class to create fitting models and manager XML files for fermipy

This class contains a 'library', which is a dictionary of all the source components:

specifically it maps:

sourcekey : model_component.ModelComponentInfo

csm
Return the CatalogSourceManager

dmm
Return the DiffuseModelManager

static get_sub_comp_info(*source_info*, *comp*)
Build and return information about a sub-component for a particular selection

gmm
Return the GalpropMapManager

make_fermipy_config_yaml(*modelkey*, *components*, *data*, ***kwargs*)
Build a fermipy top-level yaml configuration file

Parameters

- **modelkey** (*str*) – Key used to identify this particular model
- **components** (*list*) – The binning components used in this analysis
- **data** (*str*) – Path to file containing dataset definition

make_library(*diffuse_yaml*, *catalog_yaml*, *binning_yaml*)
Build up the library of all the components

Parameters

- **diffuse_yaml** (*str*) – Name of the yaml file with the library of diffuse component definitions
- **catalog_yaml** (*str*) – Name of the yaml file width the library of catalog split definitions
- **binning_yaml** (*str*) – Name of the yaml file with the binning definitions

make_model_info(*modelkey*)
Build a dictionary with the information for a particular model.

Parameters

- **modelkey** (*str*) – Key used to identify this particular model
- **ModelInfo** (*Return*) –

make_srcmap_manifest(*modelkey*, *components*, *data*)
Build a yaml file that specfies how to make the srcmap files for a particular model

Parameters

- **modelkey** (*str*) – Key used to identify this particular model
- **components** (*list*) – The binning components used in this analysis
- **data** (*str*) – Path to file containing dataset definition

read_model_yaml(*modelkey*)
Read the yaml file for the diffuse components

Trivial Link Sub-classes

class fermipy.diffuse.job_library.**Gtlink_select**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtselect :param zmax: Maximum zenith angle [degrees] [100.0] :type zmax: <type 'float'> :param emin: Minimum energy [MeV] [100.0] :type emin: <type 'float'> :param emax: Maximum energy [MeV] [100000.0] :type emax: <type 'float'> :param outfile: Output file [None] :type outfile: <type 'str'> :param pfiles: PFILES directory [None] :type pfiles: <type 'str'> :param evtype: Event type selections [None] :type evtype: <type 'int'> :param evclass: Event Class [None] :type evclass: <type 'int'> :param infile: Input file [None] :type infile: <type 'str'>

appname = 'gtselect'

default_file_args = {'infile': 1, 'outfile': 2}

default_options = {'emax': (100000.0, 'Maximum energy [MeV]', <type 'float'>), 'emin'

description = 'Link to run gtselect'

linkname_default = 'gtselect'

usage = 'gtselect [options]'

class fermipy.diffuse.job_library.**Gtlink_bin**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtbin :param algorithm: Binning alogrithm [HEALPIX] :type algorithm: <type 'str'> :param emin: Minimum energy [MeV] [100.0] :type emin: <type 'float'> :param emax: Maximum energy [MeV] [100000.0] :type emax: <type 'float'> :param hpx_order: HEALPIX order parameter [6] :type hpx_order: <type 'int'> :param evfile: Input FT1 eventfile [None] :type evfile: <type 'str'> :param outfile:

Output file [None] :type outfile: <type 'str'> :param pfiles: PFILES directory [None] :type pfiles: <type 'str'> :param coordsys: Coordinate system [GAL] :type coordsys: <type 'str'> :param enumbins: Number of energy bins [16] :type enumbins: <type 'int'>

```
appname = 'gtbin'

default_file_args = {'evfile':  33, 'outfile':  34}

default_options = {'algorithm':  ('HEALPIX', 'Binning alogrithm', <type 'str'>), 'coor

description = 'Link to run gtbin'

linkname_default = 'gtbin'

usage = 'gtbin [options]'
```

class fermipy.diffuse.job_library.**Gtlink_expcube2**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtexpcube2 :param irfs: Instrument response functions [CALDB] :type irfs: <type 'str'> :param outfile: Output file [None] :type outfile: <type 'str'> :param cmap: Input counts cube file [None] :type cmap: <type 'str'> :param coordsys: Coordinate system [GAL] :type coordsys: <type 'str'> :param evtype: Event type selections [None] :type evtype: <type 'int'> :param hpx_order: HEALPIX order parameter [6] :type hpx_order: <type 'int'> :param infile: Input livetime cube file [None] :type infile: <type 'str'>

```
appname = 'gtexpcube2'

default_file_args = {'cmap':  1, 'infile':  1, 'outfile':  2}

default_options = {'cmap':  (None, 'Input counts cube file', <type 'str'>), 'coordsys'

description = 'Link to run gtexpcube2'

linkname_default = 'gtexpcube2'

usage = 'gtexpcube2 [options]'
```

class fermipy.diffuse.job_library.**Gtlink_scrmaps**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtscrmaps :param irfs: Instrument response functions [CALDB] :type irfs: <type 'str'> :param expcube: Input Livetime cube file [None] :type expcube: <type 'str'> :param cmap: Input counts cube file [None] :type cmap: <type 'str'> :param srcmdl: Input source model xml file [None] :type srcmdl: <type 'str'> :param outfile: Output file [None] :type outfile: <type 'str'> :param bexpmap: Input binned exposure map file [None] :type bexpmap: <type 'str'>

```
appname = 'gtscrmaps'

default_file_args = {'bexpmap':  1, 'cmap':  1, 'expcube':  1, 'outfile':  2, 'srcmdl'

default_options = {'bexpmap':  (None, 'Input binned exposure map file', <type 'str'>),

description = 'Link to run gtscrmaps'

linkname_default = 'gtscrmaps'

usage = 'gtscrmaps [options]'
```

class fermipy.diffuse.job_library.**Gtlink_ltsum**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtltsum :param outfile: Output file [None] :type outfile: <type 'str'> :param infile2: Livetime cube 2 [none] :type infile2: <type 'str'> :param infile1: Livetime cube 1 or list of files [None] :type infile1: <type 'str'>

```
appname = 'gtltsum'
```

default_file_args = {'infile1': 1, 'outfile': 2}

default_options = {'infile1': (None, 'Livetime cube 1 or list of files', <type 'str'>

description = 'Link to run gtltsum'

linkname_default = 'gtltsum'

usage = 'gtltsum [options]'

class fermipy.diffuse.job_library.**Gtlink_mktime**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtmktime :param filter: Filter expression [None] :type filter: <type 'str'> :param evfile: Input FT1 File [None] :type evfile: <type 'str'> :param pfiles: PFILES directory [None] :type pfiles: <type 'str'> :param outfile: Output FT1 File [None] :type outfile: <type 'str'> :param roicut: Apply ROI-based zenith angle cut [False] :type roicut: <type 'bool'> :param scfile: Input FT2 file [None] :type scfile: <type 'str'>

appname = 'gtmktime'

default_file_args = {'evfile': 33, 'outfile': 34, 'scfile': 33}

default_options = {'evfile': (None, 'Input FT1 File', <type 'str'>), 'filter': (None

description = 'Link to run gtmktime'

linkname_default = 'gtmktime'

usage = 'gtmktime [options]'

class fermipy.diffuse.job_library.**Gtlink_ltcube**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtltcube :param dcostheta: Step size in cos(theta) [0.025] :type dcostheta: <type 'float'> :param phibins: Number of phi bins [0] :type phibins: <type 'int'> :param evfile: Input FT1 File [None] :type evfile: <type 'str'> :param zmin: Minimum zenith angle [0] :type zmin: <type 'float'> :param outfile: Output Livetime cube File [None] :type outfile: <type 'str'> :param pfiles: PFILES directory [None] :type pfiles: <type 'str'> :param binsz: Pixel size (degrees) [1.0] :type binsz: <type 'float'> :param zmax: Maximum zenith angle [105] :type zmax: <type 'float'> :param scfile: Input FT2 file [None] :type scfile: <type 'str'>

appname = 'gtltcube'

default_file_args = {'evfile': 33, 'outfile': 34, 'scfile': 33}

default_options = {'binsz': (1.0, 'Pixel size (degrees)', <type 'float'>), 'dcostheta

description = 'Link to run gtltcube'

linkname_default = 'gtltcube'

usage = 'gtltcube [options]'

class fermipy.diffuse.solar.**Gtlink_expcube2_wcs**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtexpcube2 :param irfs: Instrument response functions [CALDB] :type irfs: <type 'str'> :param emax: Stop energy (MeV) of last bin [1000000.0] :type emax: <type 'float'> :param nypix: Size of the Y axis in pixels [720] :type nypix: <type 'int'> :param binsz: Image scale (in degrees/pixel) [0.25] :type binsz: <type 'float'> :param evtype: Event type selections [None] :type evtype: <type 'int'> :param enumbins: Number of logarithmically-spaced energy bins [12] :type enumbins: <type 'int'> :param axisrot: Rotation angle of image axis, in degrees [0.0] :type axisrot: <type 'float'> :param xref: First coordinate of image center in degrees (RA or GLON) [0.0] :type xref: <type 'float'> :param outfile: Output file [None] :type outfile: <type 'str'> :param yref: Second coordinate of image center in degrees (DEC or GLAT) [0.0] :type yref: <type 'float'> :param emin: Start energy (MeV) of first bin [100.0] :type emin: <type 'float'> :param nxpix: Size of the X axis in pixels [1440] :type nxpix: <type 'int'> :param cmap: Input counts cube template [none] :type cmap:

<type 'str'> :param proj: Projection method e.g. AIT|ARC|CAR|GLS|MER|NCP|SIN|STG|TAN [CAR] :type proj: <type 'str'> :param coordsys: Coordinate system [GAL] :type coordsys: <type 'str'> :param infile: Input livetime cube file [None] :type infile: <type 'str'>

```
appname = 'gtexpcube2'

default_file_args = {'cmap':  1, 'infile':  1, 'outfile':  2}

default_options = {'axisrot':  (0.0, 'Rotation angle of image axis, in degrees', <type

description = 'Link to run gtexpcube2'

linkname_default = 'gtexpcube2'

usage = 'gtexpcube2 [options]'
```

class fermipy.diffuse.solar.**Gtlink_exphpsun**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtexphpsun :param irfs: Instrument response functions [CALDB] :type irfs: <type 'str'> :param emin: Start energy (MeV) of first bin [100.0] :type emin: <type 'float'> :param emax: Stop energy (MeV) of last bin [1000000.0] :type emax: <type 'float'> :param outfile: Output file [None] :type outfile: <type 'str'> :param enumbins: Number of logarithmically-spaced energy bins [12] :type enumbins: <type 'int'> :param binsz: Image scale (in degrees/pixel) [1.0] :type binsz: <type 'float'> :param evtype: Event type selection [3] :type evtype: <type 'int'> :param infile: Input livetime cube file [None] :type infile: <type 'str'>

```
appname = 'gtexphpsun'

default_file_args = {'infile':  1, 'outfile':  2}

default_options = {'binsz':  (1.0, 'Image scale (in degrees/pixel)', <type 'float'>),

description = 'Link to run gtexphpsun'

linkname_default = 'gtexphpsun'

usage = 'gtexphpsun [options]'
```

class fermipy.diffuse.solar.**Gtlink_suntemp**(***kwargs*)
Bases: *fermipy.jobs.gtlink.Gtlink*

Small wrapper to run gtsuntemp :param irfs: Instrument response functions [CALDB] :type irfs: <type 'str'> :param avgexp: Binned exposure [None] :type avgexp: <type 'str'> :param emax: Stop energy (MeV) of last bin [1000000.0] :type emax: <type 'float'> :param nypix: Size of the Y axis in pixels [720] :type nypix: <type 'int'> :param binsz: Image scale (in degrees/pixel) [0.25] :type binsz: <type 'float'> :param sunprof: Fits file containing solar intensity profile [None] :type sunprof: <type 'str'> :param evtype: Event type selection [3] :type evtype: <type 'int'> :param enumbins: Number of logarithmically-spaced energy bins [12] :type enumbins: <type 'int'> :param axisrot: Rotation angle of image axis, in degrees [0.0] :type axisrot: <type 'float'> :param xref: First coordinate of image center in degrees (RA or GLON) [0.0] :type xref: <type 'float'> :param outfile: Output file [None] :type outfile: <type 'str'> :param yref: Second coordinate of image center in degrees (DEC or GLAT) [0.0] :type yref: <type 'float'> :param emin: Start energy (MeV) of first bin [100.0] :type emin: <type 'float'> :param nxpix: Size of the X axis in pixels [1440] :type nxpix: <type 'int'> :param cmap: Counts map file [none] :type cmap: <type 'str'> :param proj: Projection method e.g. AIT|ARC|CAR|GLS|MER|NCP|SIN|STG|TAN [CAR] :type proj: <type 'str'> :param coordsys: Coordinate system (CEL - celestial, GAL -galactic) [GAL] :type coordsys: <type 'str'> :param expsun: Exposure binned in healpix and solar angles [None] :type expsun: <type 'str'>

```
appname = 'gtsuntemp'

default_file_args = {'avgexp':  1, 'expsun':  1, 'outfile':  2, 'sunprof':  1}

default_options = {'avgexp':  (None, 'Binned exposure', <type 'str'>), 'axisrot':  (0.

description = 'Link to run gtsuntemp'
```

linkname_default = 'gtsuntemp'

usage = 'gtsuntemp [options]'

class fermipy.diffuse.job_library.**Link_FermipyCoadd**(***kwargs*)
Bases: *fermipy.jobs.app_link.AppLink*

Small wrapper to run fermipy-coadd :param output: Output file [None] :type output: <type 'str'> :param args: List of input files [[]] :type args: <type 'list'>

appname = 'fermipy-coadd'

default_file_args = {'args': 1, 'output': 2}

default_options = {'args': ([], 'List of input files', <type 'list'>), 'output': (No

description = 'Link to run fermipy-coadd'

linkname_default = 'coadd'

usage = 'fermipy-coadd [options]'

class fermipy.diffuse.job_library.**Link_FermipyGatherSrcmaps**(***kwargs*)
Bases: *fermipy.jobs.app_link.AppLink*

Small wrapper to run fermipy-gather-srcmaps :param gzip: Compress output [False] :type gzip: <type 'bool'> :param rm: Remove input files [False] :type rm: <type 'bool'> :param args: List of input files [[]] :type args: <type 'list'> :param clobber: Overwrite output [False] :type clobber: <type 'bool'> :param output: Output file name [None] :type output: <type 'str'>

appname = 'fermipy-gather-srcmaps'

default_file_args = {'args': 1, 'output': 2}

default_options = {'args': ([], 'List of input files', <type 'list'>), 'clobber': (F

description = 'Link to run fermipy-gather-srcmaps'

linkname_default = 'gather-srcmaps'

usage = 'fermipy-gather-srcmaps [options]'

class fermipy.diffuse.job_library.**Link_FermipyVstack**(***kwargs*)
Bases: *fermipy.jobs.app_link.AppLink*

Small wrapper to run fermipy-vstack :param hdu: Name of HDU to stack [None] :type hdu: <type 'str'> :param rm: Remove input files [False] :type rm: <type 'bool'> :param gzip: Compress output [False] :type gzip: <type 'bool'> :param output: Output file name [None] :type output: <type 'str'> :param args: List of input files [[]] :type args: <type 'list'> :param clobber: Overwrite output [False] :type clobber: <type 'bool'>

appname = 'fermipy-vstack'

default_file_args = {'args': 1, 'output': 2}

default_options = {'args': ([], 'List of input files', <type 'list'>), 'clobber': (F

description = 'Link to run fermipy-vstack'

linkname_default = 'vstack'

usage = 'fermipy-vstack [options]'

class fermipy.diffuse.job_library.**Link_FermipyHealview**(***kwargs*)
Bases: *fermipy.jobs.app_link.AppLink*

Small wrapper to run fermipy-healview :param output: Output file name [None] :type output: <type 'str'> :param zscale: Scaling for color scale [log] :type zscale: <type 'str'> :param extension: FITS HDU with HEALPix map [None] :type extension: <type 'str'> :param input: Input file [None] :type input: <type 'str'>

appname = 'fermipy-healview'

default_file_args = {'args': 1, 'output': 2}

default_options = {'extension': (None, 'FITS HDU with HEALPix map', <type 'str'>), 'i

description = 'Link to run fermipy-healview'

linkname_default = 'fermipy-healview'

usage = 'fermipy-healview [options]'

Standalone Analysis Links

class fermipy.diffuse.gt_merge_srcmaps.**GtMergeSrcmaps**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to merge source maps for composite sources.

This is useful for parallelizing source map creation.

Parameters

- **irfs** (*<type 'str'>*) – Instrument response functions [CALDB]
- **expcube** (*<type 'str'>*) – Input Livetime cube file [None]
- **srcmaps** (*<type 'str'>*) – Input source maps file [None]
- **srcmdl** (*<type 'str'>*) – Input source model xml file [None]
- **outxml** (*<type 'str'>*) – Output source model xml file [None]
- **bexpmap** (*<type 'str'>*) – Input binned exposure map file [None]
- **outfile** (*<type 'str'>*) – Output file [None]
- **gzip** (*<type 'bool'>*) – Compress output file [False]
- **merged** (*<type 'str'>*) – Name of merged source [None]

NULL_MODEL = 'srcmdls/null.xml'

appname = 'fermipy-merge-srcmaps'

default_file_args = {'bexpmap': 1, 'cmap': 1, 'expcube': 1, 'outfile': 2, 'outxml'

default_options = {'bexpmap': (None, 'Input binned exposure map file', <type 'str'>),

description = 'Mrege source maps from a set of sources'

linkname_default = 'merge-srcmaps'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-merge-srcmaps [options]'

class fermipy.diffuse.gt_srcmap_partial.**GtSrcmapsDiffuse**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to create srcmaps for only once source in a model, and optionally for only some of the energy layers.

This is useful for parallelizing source map creation.

Parameters

- **irfs** (*<type 'str'>*) – Instrument response functions [CALDB]
- **expcube** (*<type 'str'>*) – Input Livetime cube file [None]
- **srcmdl** (*<type 'str'>*) – Input source model xml file [None]
- **outfile** (*<type 'str'>*) – Output file [None]
- **kmin** (*<type 'int'>*) – Minimum Energy Bin [0]
- **bexpmap** (*<type 'str'>*) – Input binned exposure map file [None]
- **source** (*<type 'str'>*) – Input source [None]
- **kmax** (*<type 'int'>*) – Maximum Energy Bin [-1]
- **cmap** (*<type 'str'>*) – Input counts cube file [None]
- **gzip** (*<type 'bool'>*) – Compress output file [False]
- **no_psf** (*<type 'bool'>*) – Do not apply PSF smearing [False]

NULL_MODEL = 'srcmdls/null.xml'

appname = 'fermipy-srcmaps-diffuse'

default_file_args = {'bexpmap': 1, 'cmap': 1, 'expcube': 1, 'outfile': 2, 'srcmdl'

default_options = {'bexpmap': (None, 'Input binned exposure map file', <type 'str'>),

description = 'Run gtsrcmaps for one or more energy planes for a single source'

linkname_default = 'srcmaps-diffuse'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-srcmaps-diffuse [options]'

class fermipy.diffuse.gt_srcmaps_catalog.**GtSrcmapsCatalog**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to create and write srcmaps for all the catalog sources, once source at a time.

This is useful for creating source maps for all the sources in a catalog

Parameters

- **irfs** (*<type 'str'>*) – Instrument response functions [CALDB]
- **expcube** (*<type 'str'>*) – Input Livetime cube file [None]
- **srcmdl** (*<type 'str'>*) – Input source model xml file [None]
- **srcmin** (*<type 'int'>*) – Index of first source [0]
- **bexpmap** (*<type 'str'>*) – Input binned exposure map file [None]
- **outfile** (*<type 'str'>*) – Output file [None]
- **cmap** (*<type 'str'>*) – Input counts cube file [None]

- **gzip** (*<type 'bool'>*) – Compress output file [False]
- **srcmax** (*<type 'int'>*) – Index of last source [-1]

NULL_MODEL = 'srcmdls/null.xml'

appname = 'fermipy-srcmaps-catalog'

default_file_args = {'bexpmap': 1, 'cmap': 1, 'expcube': 1, 'outfile': 2, 'srcmdl'

default_options = {'bexpmap': (None, 'Input binned exposure map file', <type 'str'>),

description = 'Run gtsrcmaps for for all the sources in a catalog'

linkname_default = 'srcmaps-catalog'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-srcmaps-catalog [options]'

class fermipy.diffuse.gt_assemble_model.**InitModel**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to preprate files fermipy analysis.

Specifically this create the srcmap_manifest and fermipy_config_yaml files

appname = 'fermipy-init-model'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Initialize model fitting directory'

linkname_default = 'init-model'

run_analysis(*argv*)
Build the manifest for all the models

usage = 'fermipy-init-model [options]'

class fermipy.diffuse.gt_assemble_model.**AssembleModel**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to assemple source map files for fermipy analysis.

This is useful for re-merging after parallelizing source map creation.

static append_hdus(*hdulist*, *srcmap_file*, *source_names*, *hpx_order*)
Append HEALPix maps to a list

Parameters

- **hdulist** (*list*) – The list being appended to
- **srcmap_file** (*str*) – Path to the file containing the HDUs
- **source_names** (*list of str*) – Names of the sources to extract from srcmap_file
- **hpx_order** (*int*) – Maximum order for maps

appname = 'fermipy-assemble-model'

static assemble_component(*compname*, *compinfo*, *hpx_order*)
Assemble the source map file for one binning component

Parameters

- **compname** (*str*) – The key for this component (e.g., E0_PSF3)

- **compinfo** (*dict*) – Information about this component
- **hpx_order** (*int*) – Maximum order for maps

static copy_ccube(*ccube*, *outsrcmap*, *hpx_order*)
Copy a counts cube into outsrcmap file reducing the HEALPix order to hpx_order if needed.

default_options = {'compname': (None, 'Component name.', <type 'str'>), 'hpx_order':

description = 'Assemble sourcemaps for model fitting'

linkname_default = 'assemble-model'

static open_outsrcmap(*outsrcmap*)
Open and return the outsrcmap file in append mode

run_analysis(*argv*)
Assemble the source map file for one binning component FIXME

usage = 'fermipy-assemble-model [options]'

class fermipy.diffuse.residual_cr.**ResidualCR**(***kwargs*)
Bases: *fermipy.jobs.link.Link*

Small class to analyze the residual cosmic-ray contaimination.

Parameters

- **bexpcube_clean** (*<type 'str'>*) – Input exposure cube for clean event class. [None]
- **ccube_clean** (*<type 'str'>*) – Input counts cube for clean event class. [None]
- **full_output** (*<type 'bool'>*) – Include diagnostic output [False]
- **mask_factor** (*<type 'float'>*) – Pixel selection factor for output mask [2.0]
- **hpx_order** (*<type 'int'>*) – HEALPIX order parameter [6]
- **clobber** (*<type 'bool'>*) – Overwrite files [False]
- **outfile** (*<type 'str'>*) – Output file [None]
- **bexpcube_dirty** (*<type 'str'>*) – Input exposure cube for dirty event class. [None]
- **sigma** (*<type 'float'>*) – Width of gaussian to smooth output maps [degrees] [3.0]
- **ccube_dirty** (*<type 'str'>*) – Input counts cube for dirty event class. [None]
- **select_factor** (*<type 'float'>*) – Pixel selection factor for Aeff Correction [5.0]

appname = 'fermipy-residual-cr'

default_file_args = {'bexpcube_clean': 1, 'bexpcube_dirty': 1, 'ccube_clean': 1, 'c

default_options = {'bexpcube_clean': (None, 'Input exposure cube for clean event class

description = 'Compute the residual cosmic-ray contamination'

linkname_default = 'residual-cr'

run_analysis(*argv*)
Run this analysis

usage = 'fermipy-residual-cr [options]'

Batch job dispatch classes

class fermipy.diffuse.job_library.**Gtexpcube2_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Gtlink_expcube2*

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **hpx_order_max** (*<type 'int'>*) – Maximum HEALPIX order for exposure cubes. [6]

appname = 'fermipy-gtexcube2-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Gtlink_expcube2*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit gtexpube2 jobs in parallel'

job_time = 300

usage = 'fermipy-gtexcube2-sg [options]'

class fermipy.diffuse.job_library.**Gtltsum_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Gtlink_ltsum*

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **ft1file** (*<type 'str'>*) – Input FT1 file [None]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-gtltsum-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Gtlink_ltsum*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit gtltsum jobs in parallel'

job_time = 300

usage = 'fermipy-gtltsum-sg [options]'

class fermipy.diffuse.solar.**Gtexpcube2wcs_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for gtexphpsun

Parameters

- **mktimefilter** (*<type 'str'>*) – Key for gtmktime selection [None]
- **binsz** (*<type 'float'>*) – Image scale (in degrees/pixel) [1.0]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **nypix** (*<type 'int'>*) – Size of the Y axis in pixels [180]
- **nxpix** (*<type 'int'>*) – Size of the X axis in pixels [360]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-gtexpcube2wcs-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Gtlink_expcube2_wcs*

default_options = {'binsz': (1.0, 'Image scale (in degrees/pixel)', <type 'float'>),

description = 'Submit gtexpcube2 jobs in parallel'

job_time = 300

usage = 'fermipy-gtexpcube2wcs-sg [options]'

class fermipy.diffuse.solar.**Gtexphpsun_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for gtexphpsun

Parameters

- **mktimefilter** (*<type 'str'>*) – Key for gtmktime selection [None]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]

appname = 'fermipy-gtexphpsun-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Gtlink_exphpsun*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit gtexphpsun jobs in parallel'

job_time = 300

usage = 'fermipy-gtexphpsun-sg [options]'

class fermipy.diffuse.solar.**Gtsuntemp_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for gtsuntemp

Parameters

- **mktimefilter** (*<type 'str'>*) – Key for gtmktime selection [None]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **sourcekeys** (*<type 'list'>*) – Keys for sources to make template for [None]

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]

appname = 'fermipy-gtsuntemp-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Gtlink_suntemp*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit gtsuntemp jobs in parallel'

job_time = 300

usage = 'fermipy-gtsuntemp-sg [options]'

class fermipy.diffuse.gt_coadd_split.**CoaddSplit_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for fermipy-coadd

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **ft1file** (*<type 'str'>*) – Input FT1 file [None]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-coadd-split-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *fermipy.diffuse.job_library.Link_FermipyCoadd*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit fermipy-coadd-split- jobs in parallel'

job_time = 300

usage = 'fermipy-coadd-split-sg [options]'

class fermipy.diffuse.job_library.**GatherSrcmaps_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Link_FermipyGatherSrcmaps*

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]

appname = 'fermipy-gather-srcmaps-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Link_FermipyGatherSrcmaps*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit fermipy-gather-srcmaps jobs in parallel'

job_time = 300

usage = 'fermipy-gather-srcmaps-sg [options]'

class fermipy.diffuse.job_library.**Vstack_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Link_FermipyVstack* to merge source maps

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]

appname = 'fermipy-vstack-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Link_FermipyVstack*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit fermipy-vstack jobs in parallel'

job_time = 300

usage = 'fermipy-vstack-sg [options]'

class fermipy.diffuse.job_library.**Healview_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Link_FermipyHealview*

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]

appname = 'fermipy-healviw-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Link_FermipyHealview*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Submit fermipy-healviw jobs in parallel'

job_time = 60

usage = 'fermipy-healviw-sg [options]'

class fermipy.diffuse.job_library.**SumRings_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Link_FermipyCoadd* to sum galprop ring gasmaps

Parameters

- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]
- **outdir** (*<type 'str'>*) – Output directory [None]

appname = 'fermipy-sum-rings-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Link_FermipyCoadd*

default_options = {'library': ('models/library.yaml', 'Path to yaml file defining mod

description = 'Submit fermipy-coadd jobs in parallel to sum GALProp rings'

job_time = 300

usage = 'fermipy-sum-rings-sg [options]'

class fermipy.diffuse.job_library.**SumRings_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *Link_FermipyCoadd* to sum galprop ring gasmaps

Parameters

- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]
- **outdir** (*<type 'str'>*) – Output directory [None]

appname = 'fermipy-sum-rings-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *Link_FermipyCoadd*

default_options = {'library': ('models/library.yaml', 'Path to yaml file defining mod

description = 'Submit fermipy-coadd jobs in parallel to sum GALProp rings'

job_time = 300

usage = 'fermipy-sum-rings-sg [options]'

class fermipy.diffuse.residual_cr.**ResidualCR_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

Parameters

- **mktimefilter** (*<type 'str'>*) – Key for gtmktime selection [None]
- **full_output** (*<type 'bool'>*) – Include diagnostic output [False]

- **mask_factor** (*<type 'float'>*) – Pixel selection factor for output mask [2.0]
- **hpx_order** (*<type 'int'>*) – HEALPIX order parameter [6]
- **sigma** (*<type 'float'>*) – Width of gaussian to smooth output maps [degrees] [3.0]
- **dirty** (*<type 'str'>*) – Dirty event class [source]
- **clean** (*<type 'str'>*) – Clean event class [ultracleanveto]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **select_factor** (*<type 'float'>*) – Pixel selection factor for Aeff Correction [5.0]

appname = 'fermipy-residual-cr-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *ResidualCR*

default_options = {'clean': ('ultracleanveto', 'Clean event class', <type 'str'>), 'c

description = 'Compute the residual cosmic-ray contamination'

job_time = 300

usage = 'fermipy-residual-cr-sg [options]'

class fermipy.diffuse.gt_merge_srcmaps.**MergeSrcmaps_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *GtMergeSrcmaps*

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]

appname = 'fermipy-merge-srcmaps-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *GtMergeSrcmaps*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Merge diffuse maps for all-sky analysis'

job_time = 300

usage = 'fermipy-merge-srcmaps-sg [options]'

class fermipy.diffuse.gt_srcmap_partial.**SrcmapsDiffuse_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for *GtSrcmapsDiffuse*

Parameters

- **make_xml** (*<type 'bool'>*) – Write xml files needed to make source maps [True]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]

appname = 'fermipy-srcmaps-diffuse-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *GtSrcmapsDiffuse*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run gtsrcmaps for diffuse sources'

job_time = 1500

usage = 'fermipy-srcmaps-diffuse-sg [options]'

class fermipy.diffuse.gt_srcmaps_catalog.**SrcmapsCatalog_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for gtsrcmaps for catalog sources

Parameters

- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **nsrc** (*<type 'int'>*) – Number of sources per job [500]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]
- **make_xml** (*<type 'bool'>*) – Make XML files. [True]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-srcmaps-catalog-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *GtSrcmapsCatalog*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run gtsrcmaps for catalog sources'

job_time = 1500

usage = 'fermipy-srcmaps-catalog-sg [options]'

class fermipy.diffuse.gt_assemble_model.**AssembleModel_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

Parameters

- **--compname** (*binning component definition yaml file*) –
- **--data** (*datset definition yaml file*) –

- **--models** (*model definitino yaml file*) –
- **args** (*Names of models to assemble source maps for*) –

appname = 'fermipy-assemble-model-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *AssembleModel*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Copy source maps from the library to a analysis directory'

job_time = 300

usage = 'fermipy-assemble-model-sg [options]'

class fermipy.diffuse.residual_cr.**ResidualCR_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for this script

Parameters

- **mktimefilter** (*<type 'str'>*) – Key for gtmktime selection [None]
- **full_output** (*<type 'bool'>*) – Include diagnostic output [False]
- **mask_factor** (*<type 'float'>*) – Pixel selection factor for output mask [2.0]
- **hpx_order** (*<type 'int'>*) – HEALPIX order parameter [6]
- **sigma** (*<type 'float'>*) – Width of gaussian to smooth output maps [degrees] [3.0]
- **dirty** (*<type 'str'>*) – Dirty event class [source]
- **clean** (*<type 'str'>*) – Clean event class [ultracleanveto]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **select_factor** (*<type 'float'>*) – Pixel selection factor for Aeff Correction [5.0]

appname = 'fermipy-residual-cr-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *ResidualCR*

default_options = {'clean': ('ultracleanveto', 'Clean event class', <type 'str'>), 'c

description = 'Compute the residual cosmic-ray contamination'

job_time = 300

usage = 'fermipy-residual-cr-sg [options]'

class fermipy.diffuse.gt_split_and_bin.**SplitAndBin_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for SplitAndBin

Parameters

- **scratch** (*<type 'str'>*) – Path to scratch area [None]
- **ft1file** (*<type 'str'>*) – Input FT1 file [None]
- **hpx_order_max** (*<type 'int'>*) – Maximum HEALPIX order for binning counts data. [9]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]

appname = 'fermipy-split-and-bin-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *SplitAndBin*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Prepare data for diffuse all-sky analysis'

job_time = 1500

usage = 'fermipy-split-and-bin-sg [options]'

class fermipy.diffuse.gt_split_and_mktime.**SplitAndMktime_SG**(*link*, ***kwargs*)
Bases: *fermipy.jobs.scatter_gather.ScatterGather*

Small class to generate configurations for SplitAndMktime

Parameters

- **ft1file** (*<type 'str'>*) – Path to list of input FT1 files [P8_P305_8years_source_zmax105.lst]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **scratch** (*<type 'str'>*) – Path to scratch area. [None]
- **do_ltsum** (*<type 'bool'>*) – Run gtltsum on inputs [False]
- **ft2file** (*<type 'str'>*) – Path to list of input FT2 files [ft2_8years.lst]
- **hpx_order_max** (*<type 'int'>*) – Maximum HEALPIX order for binning counts data. [9]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-split-and-mktime-sg'

build_job_configs(*args*)
Hook to build job configurations

clientclass
alias of *SplitAndMktime*

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Prepare data for diffuse all-sky analysis'

job_time = 1500

usage = 'fermipy-split-and-mktime-sg [options]'

Analysis chain classes

class fermipy.diffuse.gt_coadd_split.**CoaddSplit**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Small class to merge counts cubes for a series of binning components

This chain consists multiple `Link` objects:

coadd-EBIN-ZCUT-FILTER-EVTYPE [`_Link_FermipyCoadd`] Link to coadd data of a particular type.

Parameters

- **nfiles** (*<type 'int'>*) – Number of input files [96]
- **do_ltsum** (*<type 'bool'>*) – Sum livetime cube files [False]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]

appname = 'fermipy-coadd-split'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Merge a set of counts cube files'

linkname_default = 'coadd-split'

usage = 'fermipy-coadd-split [options]'

class fermipy.diffuse.gt_split_and_bin.**SplitAndBin**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Small class to split and bin data according to some user-provided specification

This chain consists multiple `Link` objects:

select-energy-EBIN-ZCUT [`Gtlink_select`] Initial splitting by energy bin and zenith angle cut

select-type-EBIN-ZCUT-FILTER-TYPE [`Gtlink_select`] Refinement of selection from event types

bin-EBIN-ZCUT-FILTER-TYPE [`Gtlink_bin`] Final binning of the data for each event type

Parameters

- **ft1file** (*<type 'str'>*) – Input FT1 file [None]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **scratch** (*<type 'str'>*) – Scratch area [None]
- **outkey** (*<type 'str'>*) – Key for this particular output file [None]
- **pfiles** (*<type 'str'>*) – Directory for .par files [None]
- **hpx_order_max** (*<type 'int'>*) – Maximum HEALPIX order for binning counts data. [9]
- **evclass** (*<type 'int'>*) – Event class bit mask [128]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]

- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **outdir** (*<type 'str'>*) – Base name for output files [counts_cubes_cr]

appname = 'fermipy-split-and-bin'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run gtselect and gtbin together'

linkname_default = 'split-and-bin'

usage = 'fermipy-split-and-bin [options]'

class fermipy.diffuse.gt_split_and_bin.**SplitAndBinChain**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Chain to run split and bin and then make exposure cubes

This chain consists of:

split-and-bin [*SplitAndBin_SG*] Chain to make the binned counts maps for each input file

coadd-split [CoaddSplit_SG] Link to co-add the binnec counts maps files

expcube2 [Gtexpcube2_SG] Link to make the corresponding binned exposure maps

Parameters

- **ft1file** (*<type 'str'>*) – Path to list of input FT1 files [P8_P305_8years_source_zmax105.lst]
- **hpx_order_ccube** (*<type 'int'>*) – Maximum HEALPIX order for binning counts data. [9]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **hpx_order_expcube** (*<type 'int'>*) – Maximum HEALPIX order for exposure cubes. [6]
- **scratch** (*<type 'str'>*) – Path to scratch area. [None]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-split-and-bin-chain'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run split-and-bin, coadd-split and exposure'

linkname_default = 'split-and-bin-chain'

usage = 'fermipy-split-and-bin-chain [options]'

class fermipy.diffuse.gt_split_and_mktime.**SplitAndMktime**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Small class to split, apply mktime and bin data according to some user-provided specification

This chain consists multiple Link objects:

select-energy-EBIN-ZCUT [Gtlink_select] Initial splitting by energy bin and zenith angle cut

mktime-EBIN-ZCUT-FILTER [Gtlink_mktime] Application of gtmktime filter for zenith angle cut

ltcube-EBIN-ZCUT-FILTER [Gtlink_ltcube] Computation of livetime cube for zenith angle cut

select-type-EBIN-ZCUT-FILTER-TYPE [Gtlink_select] Refinement of selection from event types

bin-EBIN-ZCUT-FILTER-TYPE [Gtlink_bin] Final binning of the data for each event type

Parameters

- **ft1file** (*<type 'str'>*) – Path to list of input FT1 files [P8_P305_8years_source_zmax105.lst]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **scratch** (*<type 'str'>*) – Scratch area [None]
- **outkey** (*<type 'str'>*) – Key for this particular output file [None]
- **do_ltsum** (*<type 'bool'>*) – Sum livetime cube files [False]
- **pfiles** (*<type 'str'>*) – Directory for .par files [None]
- **ft2file** (*<type 'str'>*) – Path to list of input FT2 files [ft2_8years.lst]
- **hpx_order_max** (*<type 'int'>*) – Maximum HEALPIX order for binning counts data. [9]
- **evclass** (*<type 'int'>*) – Event class bit mask [128]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]
- **outdir** (*<type 'str'>*) – Output directory [counts_cubes]

appname = 'fermipy-split-and-mktime'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run gtselect and gtbin together'

linkname_default = 'split-and-mktime'

usage = 'fermipy-split-and-mktime [options]'

class fermipy.diffuse.gt_split_and_mktime.**SplitAndMktimeChain**(***kwargs*)

Bases: *fermipy.jobs.chain.Chain*

Chain to run split and mktime and then make livetime and exposure cubes

This chain consists of:

split-and-mktime [SplitAndMkTime_SG] Chain to make the binned counts maps for each input file

coadd-split [CoaddSplit_SG] Link to co-add the binnec counts maps files

ltsum [Gtltsum_SG] Link to co-add the livetime cube files

expcube2 [Gtexpcube2_SG] Link to make the corresponding binned exposure maps

Parameters

- **ft1file** (*<type 'str'>*) – Path to list of input FT1 files [P8_P305_8years_source_zmax105.lst]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **hpx_order_expcube** (*<type 'int'>*) – Maximum HEALPIX order for exposure cubes. [6]
- **scratch** (*<type 'str'>*) – Path to scratch area. [None]
- **do_ltsum** (*<type 'bool'>*) – Run gtltsum on inputs [False]
- **hpx_order_ccube** (*<type 'int'>*) – Maximum HEALPIX order for binning counts data. [9]
- **ft2file** (*<type 'str'>*) – Path to list of input FT2 files [ft2_8years.lst]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

```
appname = 'fermipy-split-and-mktime-chain'
default_options = {'comp':  ('config/binning.yaml', 'Path to yaml file defining binnin
description = 'Run split-and-mktime, coadd-split and exposure'
linkname_default = 'split-and-mktime-chain'
usage = 'fermipy-split-and-mktime-chain [options]'
```

class fermipy.diffuse.diffuse_analysis.**DiffuseCompChain**(***kwargs*)

Bases: *fermipy.jobs.chain.Chain*

Chain to build srcmaps for diffuse components

This chain consists of:

sum-rings [SumRings_SG] Merge GALProp gas maps by type and ring

srcmaps-diffuse [SrcmapsDiffuse_SG] Compute diffuse component source maps in parallel

vstack-diffuse [Vstack_SG] Combine diffuse component source maps

Parameters

- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **outdir** (*<type 'str'>*) – Output directory [None]
- **make_xml** (*<type 'bool'>*) – Make XML files. [True]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

```
appname = 'fermipy-diffuse-comp-chain'
default_options = {'comp':  ('config/binning.yaml', 'Path to yaml file defining binnin
description = 'Run diffuse component analysis'
linkname_default = 'diffuse-comp'
usage = 'fermipy-diffuse-comp-chain [options]'
```

class fermipy.diffuse.diffuse_analysis.**CatalogCompChain**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Small class to build srcmaps for catalog components

This chain consists of:

srcmaps-catalog [SrcmapsCatalog_SG] Build source maps for all catalog sources in parallel

gather-srcmaps [GatherSrcmaps_SG] Gather source maps into

merge-srcmaps [MergeSrcmaps_SG] Compute source maps for merged sources

Parameters

- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]
- **make_xml** (*<type 'bool'>*) – Make XML files for diffuse components [False]
- **comp** (*<type 'str'>*) – Path to yaml file defining binning. [config/binning.yaml]
- **nsrc** (*<type 'int'>*) – Number of sources per job [500]
- **library** (*<type 'str'>*) – Path to yaml file defining model components. [models/library.yaml]
- **data** (*<type 'str'>*) – Path to yaml file defining dataset. [config/dataset_source.yaml]

appname = 'fermipy-catalog-comp-chain'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run catalog component analysis'

linkname_default = 'catalog-comp'

usage = 'fermipy-catalog-comp-chain [options]'

class fermipy.diffuse.gt_assemble_model.**AssembleModelChain**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Small class to split, apply mktime and bin data according to some user-provided specification

appname = 'fermipy-assemble-model-chain'

default_options = {'comp': ('config/binning.yaml', 'Path to yaml file defining binnin

description = 'Run init-model and assemble-model'

linkname_default = 'assemble-model-chain'

usage = 'fermipy-assemble-model-chain [options]'

class fermipy.diffuse.diffuse_analysis.**DiffuseAnalysisChain**(***kwargs*)
Bases: *fermipy.jobs.chain.Chain*

Chain to define diffuse all-sky analysis

This chain consists of:

prepare [SplitAndBinChain] Bin the data and make the exposure maps

diffuse-comp [*DiffuseCompChain*] Make source maps for diffuse components

catalog-comp [*CatalogCompChain*] Make source maps for catalog components

assemble-model [AssembleModelChain] Assemble the models for fitting

Parameters

- **config** (*<type 'str'>*) – Config yaml file [None]
- **dry_run** (*<type 'bool'>*) – Print commands but do not run them [False]

```
appname = 'fermipy-diffuse-analysis'
default_options = {'config':  (None, 'Config yaml file', <type 'str'>), 'dry_run':  (F
description = 'Run diffuse analysis chain'
linkname_default = 'diffuse'
usage = 'fermipy-diffuse-analysis [options]'
```

class fermipy.diffuse.residual_cr.**ResidualCRChain**(***kwargs*)

Bases: *fermipy.jobs.chain.Chain*

Chain to preform analysis of residual cosmic-ray contamination

This chain consists of:

split-and-mktime [SplitAndMktimeChain] Chain to bin up the data and make exposure cubes

residual-cr [*ResidualCR*] Residual CR analysis

Parameters **config** (*<type 'str'>*) – Config yaml file [None]

```
appname = 'fermipy-residual-cr-chain'
default_options = {'config':  (None, 'Config yaml file', <type 'str'>)}
description = 'Run residual cosmic ray analysis'
linkname_default = 'residual-cr-chain'
usage = 'fermipy-residual-cr-chain [options]'
```

class fermipy.diffuse.solar.**SunMoonChain**(***kwargs*)

Bases: *fermipy.jobs.chain.Chain*

Chain to construct sun and moon templates

This chain consists of:

exphpsun [*Gtexphpsun_SG*] Build the sun-centered exposure cubes

suntemp [*Gtsuntemp_SG*] Build the templates

```
appname = 'fermipy-sunmoon-chain'
default_options = {'config':  (None, 'Config yaml file', <type 'str'>)}
description = 'Run sun and moon template construction'
linkname_default = 'summoon'
usage = 'fermipy-sunmoon-chain [options]'
```

1.3.13 Changelog

This page is a changelog for releases of Fermipy. You can also browse releases on Github.

0.17.3 (7/12/2018)

- Added fitting code to fermipy.diffuse module
- Improved fermipy.jobs to deal with analyses with multiple components
- Added capability to plot global minimum in castro plots
- Added spectra for dark matter decay to DMFitFunction
- Added code to split simulations into smaller batch jobs
- Added fixed shape lightcurve to correct TS_var computation

0.17.2 (5/30/2018)

- Added lots of documentation for the fermipy.jobs module.
- Minor changes to the fermipy.jobs module to work with the dark matter analysis pipeline (dmpipe).

0.17.1 (5/23/2018)

- Patch release to get versioning working with GitHub release system.

0.17.0 (5/22/2018)

- The LogParabola, PowerLawSuperExponetial and Dark Matter SEDs have been added to the sensitivity.py script.
- There are a lot of additions to perform a stacking analysis. This can be applied for instance for the search of dark matter with a stacking analysis of Milky Way dSphs, Galaxy Clusters or other galaxies.
- It contains scripts to send jobs to SLAC Batch Farm and collect the results.
- It includes scripts and functions to perform all sky fits.
- It also fixes a few issues with glon and glat in the localization (#225), and the wrong orientation of residual and TS maps (#216)

0.16.0 (12/27/2017)

- Improvements and refactoring in the internals of the `lightcurve` method (see #156, #157, #160, #161, #162). Resolve fit stability issues that were arising when the source of interest was not significantly detected in a given time bin. Added options to speed up source map calculation by rescaling source maps (enabled with `use_scaled_srcmap=True`) and split the lightcurve calculation across N cores (enabled with `multithread=True` and `nthread=N`). Add calculation of `TS_var` to test for variability using method from the 2FGL.
- Updates to validation tools. Added MeritSkimmer script (`fermipy-merit-skimmer`) for skimming ROOT merit tuples either locally or on xrootd.

0.15.0 (09/05/2017)

- Bug fix related to restoring analysis state for phased analysis (scaled exposure).
- Many improvements and feature additions to senstivity tools (see e.g. #148, #149, and #152).
- Various updates to support DM pipeline package (#146).
- Improve robustness of algorithms for extracting peak and uncertainty ellipse from 2D likelihood surface.
- Added `curvature` method for testing a source for spectral curvature.
- Added `fix_shape` option to `extension` and `localize` to fix spectral shape parameters. Spectral shape parameters of the source of interest are now free by default when localizing or fitting extension.

0.14.0 (03/28/2017)

- Refactoring and improvements in `localize` and `extension` (see #124). Cleanup of columns in `localize`. Add new columns for 1-sigma errors projected in CEL and GAL coordinates as well as associated covariance and correlation matrices. Add positional errors when running `extension` with `fit_position=True`.
- Add `free_radius` option to `localize`, `extension`, and `sed`. This can be used to free background sources within a certain distance of the analyzed source.
- Relocalize point-source hypothesis when testing extension of extended sources.
- Improve speed and accuracy of source map calculation (see #123). Exposures are now extracted directly from the exposure map.
- Write analysis configuration to `CONFIG` header keyword of all FITS output files.
- Add `jobs` and `diffuse` submodules (see #120 and #122). These contain functionality for peforming all-sky diffuse analysis and setting up automated analysis pipelines. More detailed documentation on these features to be provided in a future release.

0.13.0 (01/16/2017)

- Rewrite LTCube class to add support for fast LT cube generation. The `gtlike.use_local_ltcube` option can be used to enable the python-based LT cube calculation in lieu of `gtltcube`.
- Bug fixes and improvements to lightcurve method (see #102). Python-based LT cube generation is now enabled by default resulting in much faster execution time when generating light curves over long time spans.
- Add `fit_position` option to `extension` that can be used to enable a joint fit of extension and position.
- New scheme for auto-generating parameter docstrings.
- Add new `set_source_morphology` method to update the spatial model of a source at runtime.
- Major refactoring of `extension` and `localize` (see #106 and #110).
- Pulled in many new modules and scripts for diffuse all-sky analysis (see #105).

0.12.0 (11/20/2016)

- Add support for phased analysis (#87). `gtlike.expscale` and `gtlike.src_expscale` can be used to apply a constant exposure correction to a whole component or individual sources within a component. See *Phased Analysis* for examples.

- Add script and tools for calculating flux sensitivity (#88 and #95). The `fermipy-flux-sensitivity` script evaluates both the differential and integral flux sensitivity for a given TS threshold and minimum number of detected counts. See *Sensitivity Tools* for examples.
- Add `fermipy-healview` script for generating images of healpix maps and cubes.
- Improvements to HPX-related classes and utilities.
- Refactoring in `irfs` module to support development of new validation tools.
- Improvements to configuration handling to allow parameter validation when updating configuration at runtime.
- Add lightcurve method (#80). See *Light Curves* for documentation.
- Change convention for flux arrays in source object. Values and uncertainties are now stored in separate arrays (e.g. `flux` and `flux_err`).
- Add *Docker-based installation* instructions. This can be used to run the RHEL6 SLAC ST builds on any machine that supports Docker (e.g. OSX Yosemite or later).
- Adopt changes to column name conventions in SED format. All column names are now lowercase.

0.11.0 (08/24/2016)

- Add support for weighted likelihood fits (supported in ST 11-03-00 or later). Weights maps can be specified with the `wmap` parameter in *gtlike*.
- Implemented performance improvements in *`tsmap`* including switching to newton's method for step-size calculation and masking of empty pixels (see #79).
- Ongoing development and refactoring of classes for dealing with CastroData (binned likelihood profiles).
- Added *`reload_sources`* method for faster recomputation of source maps.
- Fixed sign error in localization plotting method that gave wrong orientation for error ellipse..
- Refactored classes in *`spectrum`* and simplified interface for doing spectral fits (see #69).
- Added DMFitFunction spectral model class in *`spectrum`* (see #66). This uses the same lookup tables as the ST DMFitFunction class but provides a pure python implementation which can be used independently of the STs.

0.10.0 (07/03/2016)

- Implement support for more spectral models (DMFitFunction, EblAtten, FileFunction, Gaussian).
- New options (`outdir_regex` and `workdir regex`) for fine-grained control over input/output file staging.
- Add `offset_roi_edge` to source dictionary. Defined as the distance from the source position to the edge of the ROI (< 0 = inside the ROI, > 0 = outside the ROI).
- Add new variables in *`fit`* output (`edm`, `fit_status`).
- Add new package scripts (`fermipy-collect-sources`, `fermipy-cluster-sources`).
- Various refactoring and improvements in code for dealing with castro data.
- Add `MODEL_FLUX` and `PARAMS` HDUs to SED FITS file. Many new elements added SED output dictionary.
- Support NEWTON fitter with the same interface as MINUIT and NEWMINUIT. Running *`fit`* with `optimizer` = NEWTON will use the NEWTON fitter where applicable (only free norms) and MINUIT otherwise. The `optimizer` argument to *`sed`*, *`extension`*, and *`localize`* can be used to override the default optimizer at runtime. Note that the NEWTON fitter is only supported by ST releases *after* 11-01-01.

0.9.0 (05/25/2016)

- Bug fixes and various refactoring in TSCube and CastroData. Classes for reading and manipulating bin-by-bin likelihoods are now moved to the *castro* module.
- Rationalized naming conventions for energy-related variables. Properties and method arguments with units of the logarithm of the energy now consistently contain `log` in the name.
  - *energies* now returns bin energies in MeV (previously it returned logarithmic energies). *log_energies* can be used to access logarithmic bin energies.
  - Changed `erange` parameter to `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*.

CHAPTER 2

Indices and tables

Python Module Index

f

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z