# **Flarestack Documentation**

Release 2.0-beta.1

**Robert Stein** 

## **CONTENTS:**

1	Base		1
	1.1	Time PDFs	1
	1.2	Energy PDFs	5
	1.3	Spatial PDFs	6
2	Comp	posite PDF Objects	9
	2.1	Injector	9
	2.2	Log Likelihood	11
3	Utils		17
	3.1	IceCube Utils	17
4	Indic	dices and tables	
Ру	thon N	Module Index	21
In	dex		23

**CHAPTER** 

ONE

## **BASE PDFS**

## 1.1 Time PDFs

## class flarestack.core.time\_pdf.Box(t\_pdf\_dict, season)

The simplest time-dependent case for a Time PDF. Used for a source that is uniformly emitting for a fixed period of time. Requires arguments of Pre-Window and Post\_window, and gives a box from Pre-Window days before the reference time to Post-Window days after the reference time.

```
__init__(t_pdf_dict, season)
```

Initialize self. See help(type(self)) for accurate signature.

## effective\_injection\_time(source)

Calculates the effective injection time for the given PDF. The livetime is measured in days, but here is converted to seconds.

Parameters source - Source to be considered

**Returns** Effective Livetime in seconds

## flare\_time\_mask(source)

In this case, the interesting period for Flare Searches is the period of overlap of the flare and the box. Thus, for a given season, return the source and data

Returns Start time (MJD) and End Time (MJD) for flare search period

## raw\_injection\_time(source)

Calculates the 'raw injection time' which is the injection time assuming a detector with 100% uptime. Useful for calculating source emission times for source-frame energy estimation.

Parameters source - Source to be considered

Returns Time in seconds for 100% uptime

## sig\_t0 (source)

Calculates the starting time for the window, equal to the source reference time in MJD minus the length of the pre-reference-time window (in days).

Parameters source - Source to be considered

Returns Time of Window Start

## sig t1(source)

Calculates the starting time for the window, equal to the source reference time in MJD plus the length of the post-reference-time window (in days).

Parameters source - Source to be considered

Returns Time of Window End

### signal f(t, source)

In this case, the signal PDF is a uniform PDF for a fixed duration of time. It is normalised with the length of the box in LIVETIME rather than days, to give an integral of 1.

#### **Parameters**

- t. Time
- source Source to be considered

**Returns** Value of normalised box function at t

## signal\_integral (t, source)

In this case, the signal PDF is a uniform PDF for a fixed duration of time. Thus, the integral is simply a linear function increasing between t0 (box start) and t1 (box end). After t1, the integral is equal to 1, while it is equal to 0 for t < t0.

## **Parameters**

- **t** Time
- source Source to be considered

**Returns** Value of normalised box function at t

```
class flarestack.core.time_pdf.FixedEndBox(t_pdf_dict, season)
```

The simplest time-dependent case for a Time PDF. Used for a source that is uniformly emitting for a fixed period of time. In this case, the start and end time for the box is unique for each source. The sources must have a field "Start Time (MJD)" and another "End Time (MJD)", specifying the period of the Time PDF.

```
init (t pdf dict, season)
```

Initialize self. See help(type(self)) for accurate signature.

```
sig_t0 (source)
```

Calculates the starting time for the window, equal to the source reference time in MJD minus the length of the pre-reference-time window (in days).

Parameters source - Source to be considered

**Returns** Time of Window Start

```
sig_t1 (source)
```

Calculates the starting time for the window, equal to the source reference time in MJD plus the length of the post-reference-time window (in days).

Parameters source - Source to be considered

Returns Time of Window End

```
class flarestack.core.time pdf.FixedRefBox(t pdf dict, season)
```

The simplest time-dependent case for a Time PDF. Used for a source that is uniformly emitting for a fixed period of time. In this case, the start and end time for the box is unique for each source. The sources must have a field "Start Time (MJD)" and another "End Time (MJD)", specifying the period of the Time PDF.

```
___init__ (t_pdf_dict, season)
```

Initialize self. See help(type(self)) for accurate signature.

```
sig_t0 (source)
```

Calculates the starting time for the window, equal to the source reference time in MJD minus the length of the pre-reference-time window (in days).

Parameters source - Source to be considered

**Returns** Time of Window Start

## sig\_t1 (source)

Calculates the starting time for the window, equal to the source reference time in MJD plus the length of the post-reference-time window (in days).

Parameters source - Source to be considered

Returns Time of Window End

## class flarestack.core.time\_pdf.Steady(t\_pdf\_dict, season)

The time-independent case for a Time PDF. Requires no additional arguments in the dictionary for \_\_init\_\_. Used for a steady source that is continuously emitting.

## effective\_injection\_time(source)

Calculates the effective injection time for the given PDF. The livetime is measured in days, but here is converted to seconds.

Parameters source – Source to be considered

**Returns** Effective Livetime in seconds

## flare\_time\_mask(source)

In this case, the interesting period for Flare Searches is the entire season. Thus returns the start and end times for the season.

Returns Start time (MJD) and End Time (MJD) for flare search period

## raw\_injection\_time (source)

Calculates the 'raw injection time' which is the injection time assuming a detector with 100% uptime. Useful for calculating source emission times for source-frame energy estimation.

Parameters source - Source to be considered

Returns Time in seconds for 100% uptime

## sig\_t0 (source)

Calculates the starting time for the window, equal to the source reference time in MJD minus the length of the pre-reference-time window (in days).

Parameters source - Source to be considered

**Returns** Time of Window Start

## sig\_t1 (source)

Calculates the starting time for the window, equal to the source reference time in MJD plus the length of the post-reference-time window (in days).

Parameters source - Source to be considered

Returns Time of Window End

### signal\_f (t, source)

In the case of a steady source, the signal PDF is a uniform PDF in time. It is thus simply equal to the season\_f, normalised with the length of the season to give an integral of 1. It is thus equal to the background PDF.

#### **Parameters**

- t Time
- source Source to be considered

Returns Value of normalised box function at t

## signal integral(t, source)

In the case of a steady source, the signal PDF is a uniform PDF in time. Thus, the integral is simply a

1.1. Time PDFs 3

linear function increasing between t0 (box start) and t1 (box end). After t1, the integral is equal to 1, while it is equal to 0 for t < t0.

#### **Parameters**

- t Time
- source Source to be considered

**Returns** Value of normalised box function at t

class flarestack.core.time\_pdf.TimePDF (t\_pdf\_dict, season)

```
___init__(t_pdf_dict, season)
```

Initialize self. See help(type(self)) for accurate signature.

## background\_f (t, source)

In all cases, we assume that the background is uniform in time. Thus, the background PDF is just a normalised version of the season\_f box function.

#### **Parameters**

- t. Time
- source Source to be considered

**Returns** Value of normalised box function at t

classmethod create(t pdf dict, season)

### inverse interpolate(source)

Calculates the values for the integral of the signal PDF within the season. Then rescales these values, such that the start of the season yields 0, and then end of the season yields 1. Creates a function to interpolate between these values. Then, for a number between 0 and 1, the interpolated function will return the MJD time at which that fraction of the cumulative distribution was reached.

Parameters source - Source to be considered

**Returns** Interpolated function

## product\_integral (t, source)

Calculates the product of the given signal PDF with the season box function. Thus gives 0 everywhere outside the season, and otherwise the value of the normalised integral. The season function is offset by 1e-9, to ensure that f(t1) is equal to 1. (i.e the function is equal to 1 at the end of the box).

#### **Parameters**

- **t** Time
- source Source to be considered

**Returns** Product of signal integral and season

classmethod register\_subclass(time\_pdf\_name)

```
simulate_times (source, n_s)
```

Randomly draws times for n\_s events for a given source, all lying within the current season. The values are based on an interpolation of the integrated time PDF.

## **Parameters**

- source Source being considered
- n s Number of event times to be simulated

**Returns** Array of times in MJD for a given source

```
Flarestack Documentation, Release 2.0-beta.1
     subclasses = {'Box': <class 'flarestack.core.time_pdf.Box'>, 'FixedEndBox':
flarestack.core.time_pdf.box_func(t, t0, t1)
     Box function that is equal to 1 between t0 and t1, and 0 otherwise. Equal to 0.5 at to and t1.
          Parameters
               • t – Time to be evaluated
               • t0 - Start time of box
               • ±1 – End time of box
          Returns Value of Box function at t
flarestack.core.time_pdf.read_t_pdf_dict(t_pdf_dict)
     Ensures backwards compatibility for t_pdf_dict objects
1.2 Energy PDFs
This script contains the EnergyPDF classes, that are used for weighting events based on a given energy PDF.
class flarestack.core.energy_pdf.EnergyPDF (e_pdf_dict)
     ___init__(e_pdf_dict)
          Initialize self. See help(type(self)) for accurate signature.
     classmethod create(e_pdf_dict)
     static f(energy)
```

## flux\_integral()

fluence\_integral()

second that is radiated.

Integrates over energy PDF to give integrated flux (dN/dT)

```
integrate_over_E (f, lower=None, upper=None)
```

Uses Newton's method to integrate function f over the energy range. By default, uses 100GeV to 10PeV, unless otherwise specified. Uses 1000 logarithmically-spaced bins to calculate integral.

Performs an integral for fluence over a given energy range. This is gives the total energy per unit area per

**Parameters f** – Function to be integrated

**Returns** Integral of function

```
classmethod register_subclass(energy_pdf_name)
```

Adds a new subclass of EnergyPDF, with class name equal to "energy\_pdf\_name".

```
return_energy_parameters()
subclasses = {'PowerLaw': <class 'flarestack.core.energy_pdf.PowerLaw'>, 'Spline':
```

**class** flarestack.core.energy\_pdf.**PowerLaw** (*e\_pdf\_dict=None*)

A Power Law energy PDF. Takes an argument of gamma in the dictionary for the init function, where gamma is

```
__init___(e_pdf_dict=None)
```

the spectral index of the Power Law.

Creates a PowerLaw object, which is an energy PDF based on a power law. The power law is generated from e\_pdf\_dict, which can specify a spectral index (Gamma), as well as an optional minimum energy (E Min) and a maximum energy (E Max)

1.2. Energy PDFs 5

**Parameters** e\_pdf\_dict - Dictionary containing parameters

**f** (energy)

## fluence\_integral()

Performs an integral for fluence over a given energy range. This is gives the total energy per unit area per second that is radiated.

## flux\_integral()

Integrates over energy PDF to give integrated flux (dN/dT)

```
return_energy_parameters()
```

```
return_injected_parameters()
```

```
weight_mc (mc, gamma=None)
```

Returns an array containing the weights for each MC event, given that the spectral index gamma has been chosen. Weights each event as (E/GeV)^-gamma, and multiplies this by the pre-existing MC oneweight value, to give the overall oneweight.

## **Parameters**

- mc Monte Carlo
- gamma Spectral Index (default is value in e\_pdf\_dict)

**Returns** Weights Array

```
class flarestack.core.energy_pdf.Spline(e_pdf_dict={})
```

A Power Law energy PDF. Takes an argument of gamma in the dictionary for the init function, where gamma is the spectral index of the Power Law.

```
___init___(e_pdf_dict={})
```

Creates a PowerLaw object, which is an energy PDF based on a power law. The power law is generated from e\_pdf\_dict, which can specify a spectral index (Gamma), as well as an optional minimum energy (E Min) and a maximum energy (E Max)

Parameters e\_pdf\_dict - Dictionary containing parameters

```
weight mc(mc)
```

Returns an array containing the weights for each MC event, given that the spectral index gamma has been chosen. Weights each event using the energy spline, and multiplies this by the pre-existing MC oneweight value, to give the overall oneweight.

Parameters mc - Monte Carlo

**Returns** Weights Array

```
flarestack.core.energy pdf.read e pdf dict(e pdf dict)
```

Ensures backwards compatibility of e pdf dict objects.

```
Parameters e_pdf_dict - Energy PDF dictionary
```

Returns Updated Energy PDF dictionary compatible with new format

## 1.3 Spatial PDFs

This script contains the EnergyPDF classes, that are used for weighting events based on a given energy PDF.

```
class flarestack.core.energy_pdf.EnergyPDF (e_pdf_dict)
```

```
___init___(e_pdf_dict)
          Initialize self. See help(type(self)) for accurate signature.
     classmethod create(e_pdf_dict)
     static f(energy)
     fluence integral()
          Performs an integral for fluence over a given energy range. This is gives the total energy per unit area per
          second that is radiated.
     flux_integral()
          Integrates over energy PDF to give integrated flux (dN/dT)
     integrate_over_E (f, lower=None, upper=None)
          Uses Newton's method to integrate function f over the energy range. By default, uses 100GeV to 10PeV,
          unless otherwise specified. Uses 1000 logarithmically-spaced bins to calculate integral.
              Parameters f – Function to be integrated
              Returns Integral of function
     classmethod register_subclass(energy_pdf_name)
          Adds a new subclass of EnergyPDF, with class name equal to "energy pdf name".
     return_energy_parameters()
     subclasses = {'PowerLaw': <class 'flarestack.core.energy_pdf.PowerLaw'>, 'Spline':
class flarestack.core.energy_pdf.PowerLaw(e_pdf_dict=None)
     A Power Law energy PDF. Takes an argument of gamma in the dictionary for the init function, where gamma is
     the spectral index of the Power Law.
     ___init__(e_pdf_dict=None)
          Creates a PowerLaw object, which is an energy PDF based on a power law. The power law is generated
          from e_pdf_dict, which can specify a spectral index (Gamma), as well as an optional minimum energy (E
          Min) and a maximum energy (E Max)
              Parameters e_pdf_dict - Dictionary containing parameters
     f (energy)
     fluence integral()
          Performs an integral for fluence over a given energy range. This is gives the total energy per unit area per
          second that is radiated.
     flux_integral()
          Integrates over energy PDF to give integrated flux (dN/dT)
     return_energy_parameters()
     return_injected_parameters()
     weight_mc (mc, gamma=None)
          Returns an array containing the weights for each MC event, given that the spectral index gamma has been
          chosen. Weights each event as (E/GeV)^-gamma, and multiplies this by the pre-existing MC oneweight
          value, to give the overall oneweight.
              Parameters
                   • mc - Monte Carlo
                   • gamma – Spectral Index (default is value in e_pdf_dict)
```

1.3. Spatial PDFs 7

**Returns** Weights Array

```
class flarestack.core.energy_pdf.Spline(e_pdf_dict={})
```

A Power Law energy PDF. Takes an argument of gamma in the dictionary for the init function, where gamma is the spectral index of the Power Law.

```
___init___(e_pdf_dict={})
```

Creates a PowerLaw object, which is an energy PDF based on a power law. The power law is generated from e\_pdf\_dict, which can specify a spectral index (Gamma), as well as an optional minimum energy (E Min) and a maximum energy (E Max)

**Parameters e\_pdf\_dict** – Dictionary containing parameters

```
weight_mc(mc)
```

Returns an array containing the weights for each MC event, given that the spectral index gamma has been chosen. Weights each event using the energy spline, and multiplies this by the pre-existing MC oneweight value, to give the overall oneweight.

Parameters mc - Monte Carlo

**Returns** Weights Array

flarestack.core.energy\_pdf.read\_e\_pdf\_dict(e\_pdf\_dict)

Ensures backwards compatibility of e\_pdf\_dict objects.

Parameters e\_pdf\_dict - Energy PDF dictionary

Returns Updated Energy PDF dictionary compatible with new format

## COMPOSITE PDF OBJECTS

## 2.1 Injector

calculate\_n\_exp\_single(source)

```
class flarestack.core.injector.BaseInjector(season, sources, **kwargs)
     Base Injector Class
     ___init___(season, sources, **kwargs)
          Initialize self. See help(type(self)) for accurate signature.
     calculate_n_exp()
     calculate_n_exp_single(source)
     classmethod create (season, sources, **kwargs)
     create_dataset (scale, pull_corrector)
          Create a dataset based on scrambled data for background, and Monte Carlo simulation for signal. Returns
          the composite dataset. The source flux can be scaled by the scale parameter.
              Parameters scale – Ratio of Injected Flux to source flux
              Returns Simulated dataset
     static get_dec_and_omega(source)
     get_expectation (source, scale)
     get_n_exp_single(source)
     inject_signal(scale)
     classmethod register_subclass(inj_name)
          Adds a new subclass of EnergyPDF, with class name equal to "energy_pdf_name".
     subclasses = {}
     update_sources (sources)
          Reuses an injector with new sources
              Parameters sources - Sources to be added
class flarestack.core.injector.EffectiveAreaInjector(season, sources, **kwargs)
     Class for injecting signal events by relying on effective areas rather than pre-existing Monte Carlo simulation.
     This Injector should be used for analysing public data, as no MC is provided.
     __init__ (season, sources, **kwargs)
          Initialize self. See help(type(self)) for accurate signature.
     calculate_energy_proxy(source)
```

```
calculate_single_source (source, scale)
     inject_signal(scale)
class flarestack.core.injector.LowMemoryInjector(season, sources, **kwargs)
     For large numbers of sources O(\sim 100), saving MC masks becomes increasingly burdensome. As a solution, the
     LowMemoryInjector should be used instead. It will be somewhat slower, but will have much more reasonable
     memory consumption.
     init (season, sources, **kwargs)
          Initialize self. See help(type(self)) for accurate signature.
     calculate_n_exp()
     get_band_mask (source, min_dec, max_dec)
     load_band_mask (index)
     make_injection_band_mask()
class flarestack.core.injector.MCInjector(season, sources, **kwargs)
     Core Injector Class, returns a dataset on which calculations can be performed. This base class is tailored for
     injection of MC into mock background. This can be either MC background, or scrambled real data.
     __init__ (season, sources, **kwargs)
          Initialize self. See help(type(self)) for accurate signature.
     calculate_fluence (source, scale, source_mc, band_mask, omega)
          Function to calculate the fluence for a given source, and multiply the oneweights by this. After this step,
          the oneweight sum is equal to the expected neutrino number.
              Parameters
                  • source - Source to be calculated
                  • scale - Flux scale
                  • source mc – MC that is close to source
                  • band_mask - Closeness mask for MC
                  • omega – Solid angle covered by MC mask
              Returns Modified source MC
     calculate_n_exp_single(source)
     calculate_single_source (source, scale)
          Calculate the weighted MC for a single source, given a flux scale and a distance scale.
              Parameters
                  • source -
                  • scale -
              Returns
     get_band_mask (source, min_dec, max_dec)
     inject_signal(scale)
          Randomly select simulated events from the Monte Carlo dataset to simulate a signal for each source. The
```

Parameters scale – Ratio of Injected Flux to source flux.

**Returns** Set of signal events for the given IC Season.

source flux can be scaled by the scale parameter.

```
select mc band(source)
          For a given source, selects MC events within a declination band of width +/- 5 degrees that contains the
          source. Then returns the MC data subset containing only those MC events.
              Parameters source - Source to be simulated
               Returns mc (cut): Simulated events which lie within the band
              Returns omega: Solid Angle of the chosen band
              Returns band mask: The mask which removes events outside band
     subclasses = {'low_memory_injector': <class 'flarestack.core.injector.LowMemoryInject</pre>
class flarestack.core.injector.MockUnblindedInjector(season,
                                                                                       sources=nan,
                                                                       **kwargs)
     If the data is not really to be unblinded, then MockUnblindedInjector should be called. In this case, the cre-
     ate_dataset function simply returns one background scramble.
     ___init___ (season, sources=nan, **kwargs)
          Initialize self. See help(type(self)) for accurate signature.
     create_dataset (scale, pull_corrector)
          Returns a background scramble
               Returns Scrambled data
class flarestack.core.injector.TrueUnblindedInjector(season, sources, **kwargs)
     If the data is unblinded, then UnblindedInjector should be called. In this case, the create dataset function simply
     returns the unblinded dataset.
     init (season, sources, **kwargs)
          Initialize self. See help(type(self)) for accurate signature.
     create_dataset (scale, pull_corrector)
flarestack.core.injector.read_injector_dict(inj_dict)
     Ensures that injection dictionaries remain backwards-compatible
          Parameters inj_dict – Injection Dictionary
          Returns Injection Dictionary compatible with new format
2.2 Log Likelihood
class flarestack.core.llh.FixedEnergyLLH(season, sources, llh_dict)
     ___init___(season, sources, llh_dict)
          Initialize self. See help(type(self)) for accurate signature.
     calculate_test_statistic (params, weights, **kwargs)
          Calculates the test statistic, given the parameters. Uses numexpr for faster calculations.
```

## **Parameters**

- params Parameters from Minimisation
- weights Normalised fraction of n\_s allocated to each source

**Returns** 2 \* 1lh value (Equal to Test Statistic)

create\_acceptance\_function(acc\_path)

2.2. Log Likelihood 11

### create energy functions()

Creates the acceptance function, which parameterises signal acceptance as a function of declination, and the energy weighting function, which gives the energy signal-over-background ratio

**Returns** Acceptance function, energy\_weighting\_function

```
create energy weighting function (SoB path)
```

```
create_kwargs (data, pull_corrector, weight_f=None)
```

Creates a likelihood function to minimise, based on the dataset.

Parameters data - Dataset

**Returns** LLH function that can be minimised

```
fit_energy = False
```

```
class flarestack.core.llh.LLH (season, sources, llh_dict)
    Base class LLH.
```

```
___init___ (season, sources, llh_dict)
```

Initialize self. See help(type(self)) for accurate signature.

```
static assume_background(n_s, n_coincident, n_all)
```

To save time with likelihood calculation, it can be assumed that all events defined as "non-coincident", because of distance in space and time to the source, are in fact background events. This is equivalent to setting S=0 for all non-coincident events. IN this case, the likelihood can be calculated as the product of the number of non-coincident events, and the likelihood of an event which has S=0.

#### **Parameters**

- n s Array of expected number of events
- n\_coincident Number of events that were not assumed to have S=0
- n\_all The total number of events

Returns Log Likelihood value for the given

## background\_pdf (source, cut\_data)

Calculates the value of the background spatial PDF for a given source for each event in the coincident data subsample. Thus is done by calling the self.bkg\_spline spline function, which was fitted to the Sin(Declination) distribution of the data.

If there is a signal Time PDF given, then the background time PDF is also calculated for each event. This is assumed to be a normalised uniform distribution for the season.

Returns either the background spatial PDF values, or the product of the background spatial and time PDFs.

## **Parameters**

- source Source to be considered
- cut\_data Subset of Dataset with coincident events

Returns Array of Background Spacetime PDF values

```
background_spatial(cut_data)
calculate_test_statistic(params, weights, **kwargs)
classmethod create(season, sources, llh_dict)
create_background_function()
```

```
create energy functions()
```

Creates the acceptance function, which parameterises signal acceptance as a function of declination, and the energy weighting function, which gives the energy signal-over-background ratio

**Returns** Acceptance function, energy\_weighting\_function

```
create_kwargs (data, pull_corrector, weight_f=None)
```

```
create_llh_function(data, pull_corrector, weight_f=None)
```

Creates a likelihood function to minimise, based on the dataset.

Parameters data - Dataset

Returns LLH function that can be minimised

```
classmethod get_injected_parameters(mh_dict)
```

```
classmethod get_parameters(llh_dict)
```

```
classmethod register_subclass(llh_name)
```

Adds a new subclass of EnergyPDF, with class name equal to "energy\_pdf\_name".

```
\verb|static return_injected_parameters| (mh\_dict)
```

```
static return_llh_parameters(llh_dict)
```

```
select_spatially_coincident_data(data, sources)
```

Checks each source, and only identifies events in data which are both spatially and time-coincident with the source. Spatial coincidence is defined as a +/- 5 degree box centered on the given source. Time coincidence is determined by the parameters of the LLH Time PDF. Produces a mask for the dataset, which removes all events which are not coincident with at least one source.

#### **Parameters**

- data Dataset to be tested
- sources Sources to be tested

**Returns** Mask to remove

```
signal_pdf (source, cut_data)
```

Calculates the value of the signal spatial PDF for a given source for each event in the coincident data subsample. If there is a Time PDF given, also calculates the value of the signal Time PDF for each event. Returns either the signal spatial PDF values, or the product of the signal spatial and time PDFs.

## **Parameters**

- source Source to be considered
- cut data Subset of Dataset with coincident events

**Returns** Array of Signal Spacetime PDF values

```
subclasses = {'fixed_energy': <class 'flarestack.core.llh.FixedEnergyLLH'>, 'spatial'
class flarestack.core.llh.SpatialLLH(season, sources, llh_dict)
```

Most basic LLH, in which only spatial, and optionally also temporal, information is included. No Energy PDF is used, and no energy weighting is applied.

```
___init___ (season, sources, llh_dict)
```

Initialize self. See help(type(self)) for accurate signature.

```
calculate_test_statistic(params, weights, **kwargs)
```

Calculates the test statistic, given the parameters. Uses numexpr for faster calculations.

#### **Parameters**

2.2. Log Likelihood 13

- params Parameters from Minimisation
- weights Normalised fraction of n s allocated to each source

**Returns** 2 \* 1lh value (Equal to Test Statistic)

## create\_energy\_function()

In the most simple case of spatial-only weighting, you would neglect the energy weighting of events. Then, you can simply assume that the detector acceptance is roughly proportional to the data rate, i.e assuming that the incident background atmospheric neutrino flux is uniform. Thus the acceptance of the detector is simply the background spatial PDF (which is a spline fitted to data as a function of declination). This method does, admittedly neglect the fact that background in the southern hemisphere is mainly composed of muon bundles, rather than atmospheric neutrinos. Still, it's slighty better than assuming a uniform detector acceptance

Returns 1D linear interpolation

## create\_llh\_function(data, pull\_corrector, weight\_f=None)

Creates a likelihood function to minimise, based on the dataset.

#### **Parameters**

- data Dataset
- pull\_corrector pull\_corrector

Returns LLH function that can be minimised

```
fit energy = False
```

class flarestack.core.llh.StandardLLH(season, sources, llh dict)

```
__init__ (season, sources, llh_dict)
```

Initialize self. See help(type(self)) for accurate signature.

## calculate\_test\_statistic(params, weights, \*\*kwargs)

Calculates the test statistic, given the parameters. Uses numexpr for faster calculations.

#### **Parameters**

- params Parameters from Minimisation
- weights Normalised fraction of n\_s allocated to each source

**Returns** 2 \* 1lh value (Equal to Test Statistic)

## create\_SoB\_energy\_cache (cut\_data)

Evaluates the Log(Signal/Background) values for all coincident data. For each value of gamma in self.gamma\_support\_points, calculates the Log(Signal/Background) values for the coincident data. Then saves each weight array to a dictionary.

Parameters cut\_data - Subset of the data containing only coincident events

Returns Dictionary containing SoB values for each event for each

gamma value.

## create\_acceptance\_function()

Creates a 2D linear interpolation of the acceptance of the detector for the given season, as a function of declination and gamma. Returns this interpolation function.

**Returns** 2D linear interpolation

## create\_energy\_functions()

Creates the acceptance function, which parameterises signal acceptance as a function of declination, and the energy weighting function, which gives the energy signal-over-background ratio

**Returns** Acceptance function, energy\_weighting\_function

```
create_kwargs (data, pull_corrector, weight_f=None)
```

Creates a likelihood function to minimise, based on the dataset.

Parameters data – Dataset

Returns LLH function that can be minimised

## estimate\_energy\_weights (gamma, energy\_SoB\_cache)

Quickly estimates the value of Signal/Background for Gamma. Uses pre-calculated values for first and second derivatives. Uses a Taylor series to estimate S(gamma), unless SoB has already been calculated for a given gamma.

## **Parameters**

- gamma Spectral Index
- energy\_SoB\_cache Weight cache

**Returns** Estimated value for S(gamma)

```
fit_energy = True
```

```
new_acceptance (source, params=None)
```

Calculates the detector acceptance for a given source, using the 2D interpolation of the acceptance as a function of declination and gamma. If gamma IS NOT being fit, uses the default value of gamma for weighting (determined in \_\_init\_\_). If gamma IS being fit, it will be the last entry in the parameter array, and is the acceptance uses this value.

## **Parameters**

- source Source to be considered
- params Parameter array

Returns Value for the acceptance of the detector, in the given

season, for the source

```
static return_injected_parameters (mh_dict)
static return_llh_parameters (llh_dict)
```

class flarestack.core.llh.StandardMatrixLLH (season, sources, llh\_dict)

```
create_kwargs (data, pull_corrector, weight_f=None)
```

Creates a likelihood function to minimise, based on the dataset.

Parameters data - Dataset

Returns LLH function that can be minimised

 ${\tt class} \ \, {\tt flarestack.core.llh.StandardOverlappingLLH} \, (\textit{season}, \textit{sources}, \textit{llh\_dict}) \,$ 

```
calculate_test_statistic (params, weights, **kwargs)
```

Calculates the test statistic, given the parameters. Uses numexpr for faster calculations.

### **Parameters**

• params – Parameters from Minimisation

2.2. Log Likelihood 15

• weights – Normalised fraction of n\_s allocated to each source

**Returns** 2 \* 11h value (Equal to Test Statistic)

create\_kwargs (data, pull\_corrector, weight\_f=None)

Creates a likelihood function to minimise, based on the dataset.

Parameters data – Dataset

Returns LLH function that can be minimised

flarestack.core.llh.generate\_dynamic\_flare\_class (season, sources, llh\_dict)

flarestack.core.llh.read\_llh\_dict(llh\_dict)

Ensures that llh dictionaries remain backwards-compatible

Parameters 11h\_dict - LLH Dictionary

**Returns** LLH Dictionary compatible with new format

**CHAPTER** 

## **THREE**

## **UTILS**

## 3.1 IceCube Utils

Helper function to load data for a given season/set of season. Adds sinDec field if this is not available, and combines multiple years of data is appropriate (different sets of data from the same icecube configuration should be given as a list)

## **Parameters**

- data\_path Path to data or list of paths to data
- cut\_fields Boolean to remove unused fields from datasets on loading

**Returns** Loaded Dataset (experimental or MC)

```
flarestack.icecube_utils.dataset_loader.grl_loader(season)
flarestack.icecube_utils.dataset_loader.verify_grl_with_data(seasons)
```

18 Chapter 3. Utils

## **CHAPTER**

## **FOUR**

## **INDICES AND TABLES**

- genindex
- modindex
- search

## **PYTHON MODULE INDEX**

22 Python Module Index

## **INDEX**

Symbols	background_pdf() (flarestack.core.llh.LLH		
init() (flarestack.core.energy_pdf.EnergyPDF method), 5, 6	<pre>method), 12 background_spatial() (flarestack.core.llh.LLH</pre>		
init() (flarestack.core.energy_pdf.PowerLaw method), 5, 7init() (flarestack.core.energy_pdf.Spline	method), 12 BaseInjector (class in flarestack.core.injector), 9 Box (class in flarestack.core.time_pdf), 1 box_func() (in module flarestack.core.time_pdf), 5		
method), 6, 8			
init() (flarestack.core.injector.BaseInjector method), 9	C		
init() (flarestack.core.injector.EffectiveAreaInject			
method), 9	(flarestack.core.injector.EffectiveAreaInjector method), 9		
init() (flarestack.core.injector.LowMemoryInjector method), 10	calculate_fluence()		
init() (flarestack.core.injector.MCInjector method), 10	(flarestack.core.injector.MCInjector method), 10		
init() (flarestack.core.injector.MockUnblindedInj			
method), 11	(flarestack.core.injector.BaseInjector method),		
init() (flarestack.core.injector.TrueUnblindedInje			
method), 11init()	<pre>calculate_n_exp()      (flarestack.core.injector.LowMemoryInjector</pre>		
method), 11	method), 10		
init() (flarestack.core.llh.LLH method), 12	<pre>calculate_n_exp_single()</pre>		
init() (flarestack.core.llh.SpatialLLH method),  13	(flarestack.core.injector.BaseInjector method), 9		
init() (flarestack.core.llh.StandardLLH	<pre>calculate_n_exp_single()</pre>		
method), 14	(flarestack.core.injector.EffectiveAreaInjector		
init() (flarestack.core.time_pdf.Box method), 1	<pre>method), 9 calculate_n_exp_single()</pre>		
init() (flarestack.core.time_pdf.FixedEndBox method), 2	(flarestack.core.injector.MCInjector method),		
init() (flarestack.core.time_pdf.FixedRefBox	10		
method), 2	<pre>calculate_single_source()</pre>		
init() (flarestack.core.time_pdf.TimePDF method), 4	(flarestack.core.injector. $\it Effective Area Injectormethod$ ), $10$		
Α	<pre>calculate_single_source()     (flarestack.core.injector.MCInjector method),</pre>		
assume_background() (flarestack.core.llh.LLH static method), 12	10 calculate_test_statistic()		
В	(flarestack.core.llh.FixedEnergyLLH method),		
<pre>background_f() (flarestack.core.time_pdf.TimePDF     method), 4</pre>	calculate_test_statistic() (flarestack.core.llh.LLH method), 12		
	<pre>calculate_test_statistic()</pre>		

```
(flarestack.core.llh.SpatialLLH
                                                                      method),
                                                                                    create_kwargs() (flarestack.core.llh.StandardOverlappingLLH
              13
                                                                                                    method), 16
calculate test statistic()
                                                                                      create_llh_function() (flarestack.core.llh.LLH
              (flarestack.core.llh.StandardLLH
                                                                                                    method), 13
                                                                      method),
                                                                                      create_llh_function()
                                                                                                    (flarestack.core.llh.SpatialLLH
calculate test statistic()
                                                                                                                                                            method),
              (flarestack.core.llh.StandardOverlappingLLH
              method), 15
                                                                                      create_SoB_energy_cache()
convert_grl()
                                                (in
                                                                        module
                                                                                                     (flarestack.core.llh.StandardLLH
                                                                                                                                                            method),
              flarestack.icecube_utils.dataset_loader),
                                                                                                     14
                         (flarestack.core.energy_pdf.EnergyPDF
create()
              class method), 5, 7
                                                                                      data_loader()
                                                                                                                                                              module
                                                                                                                                      (in
create() (flarestack.core.injector.BaseInjector class
                                                                                                    flarestack.icecube_utils.dataset_loader),
              method), 9
                                                                                                     17
create() (flarestack.core.llh.LLH class method), 12
                                                                                      Ε
                    (flarestack.core.time_pdf.TimePDF class
create()
              method), 4
                                                                                      effective_injection_time()
create_acceptance_function()
                                                                                                     (flarestack.core.time_pdf.Box method), 1
              (flarestack.core.llh.FixedEnergyLLH method),
                                                                                      effective_injection_time()
                                                                                                     (flarestack.core.time_pdf.Steady
                                                                                                                                                            method),
create_acceptance_function()
              (flarestack.core.llh.StandardLLH
                                                                      method),
                                                                                      EffectiveAreaInjector
                                                                                                                                                 (class
                                                                                                                                                                      in
                                                                                                    flarestack.core.injector), 9
create_background_function()
                                                                                      EnergyPDF (class in flarestack.core.energy_pdf), 5, 6
              (flarestack.core.llh.LLH method), 12
                                                                                      estimate_energy_weights()
create_dataset() (flarestack.core.injector.BaseInjector
                                                                                                    (flarestack.core.llh.StandardLLH
                                                                                                                                                            method),
              method), 9
                                                                                                     15
create_dataset() (flarestack.core.injector.MockUnblindedInjector
              method), 11
\verb|create_dataset|| (flarestack.core.injector.TrueUnblindedInjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlinjectAlin
              method), 11
                                                                                                    method), 5, 7
create_energy_function()
                                                                                      f () (flarestack.core.energy_pdf.PowerLaw method), 6, 7
              (flarestack.core.llh.SpatialLLH
                                                                      method),
                                                                                      fit_energy (flarestack.core.llh.FixedEnergyLLH at-
                                                                                                    tribute), 12
create_energy_functions()
                                                                                      fit_energy (flarestack.core.llh.SpatialLLH attribute),
              (flarestack.core.llh.FixedEnergyLLH method),
                                                                                                     14
                                                                                                                (flarestack.core.llh.StandardLLH
                                                                                      fit_energy
create_energy_functions()
                                                                                                    tribute), 15
              (flarestack.core.llh.LLH method), 12
                                                                                      FixedEndBox (class in flarestack.core.time_pdf), 2
create energy functions()
                                                                                      FixedEnergyLLH (class in flarestack.core.llh), 11
              (flarestack.core.llh.StandardLLH
                                                                      method),
                                                                                      FixedRefBox (class in flarestack.core.time_pdf), 2
                                                                                      flare_time_mask()
                                                                                                                              (flarestack.core.time_pdf.Box
create_energy_weighting_function()
                                                                                                    method), 1
              (flarestack.core.llh.FixedEnergyLLH method),
                                                                                      flare_time_mask()
                                                                                                    (flarestack.core.time_pdf.Steady
                                                                                                                                                            method),
create_kwargs() (flarestack.core.llh.FixedEnergyLLH
              method), 12
                                                                                      flarestack.core.energy_pdf (module), 5, 6
create_kwargs() (flarestack.core.llh.LLH method),
                                                                                      flarestack.core.injector(module), 9
                                                                                      flarestack.core.llh (module), 11
                                   (flarestack.core.llh.StandardLLH
create_kwargs()
                                                                                      flarestack.core.time_pdf (module), 1
              method), 15
                                                                                       flarestack.icecube_utils.dataset_loader
create_kwargs() (flarestack.core.llh.StandardMatrixLLH
                                                                                                     (module), 17
              method), 15
                                                                                      flarestack.utils (module), 17
```

24 Index

fluence_integral()	M
(flarestack.core.energy_pdf.EnergyPDF method), 5, 7	<pre>make_injection_band_mask()      (flarestack.core.injector.LowMemoryInjector</pre>
fluence_integral()	method), 10
(flarestack.core.energy_pdf.PowerLaw	MCInjector (class in flarestack.core.injector), 10
method), 6, 7	MockUnblindedInjector (class in
<pre>flux_integral() (flarestack.core.energy_pdf.EnergyP</pre>	flarestack.core.injector), 11
flux_integral() (flarestack.core.energy_pdf.PowerLa	n <b>N</b>
method), 6, 7	new_acceptance() (flarestack.core.llh.StandardLLH
G	method), 15
generate_dynamic_flare_class() (in module	P
flarestack.core.llh), 16	PowerLaw (class in flarestack.core.energy_pdf), 5, 7
<pre>get_band_mask() (flarestack.core.injector.LowMemory</pre>	VInjector product_integral()  (flarestack.core.time_pdf.TimePDF method), 4
<pre>get_band_mask() (flarestack.core.injector.MCInjector</pre>	
method), 10	R
get_dec_and_omega() (flarestack.core.injector.BaseInjector static	<pre>raw_injection_time()</pre>
method), 9	$(flarestack.core.time\_pdf.Box\ method),\ 1$
get_expectation()	raw_injection_time()
(flarestack.core.injector.BaseInjector method),	(flarestack.core.time_pdf.Steady method),
9	read_e_pdf_dict() (in module
get_injected_parameters()	flarestack.core.energy_pdf), 6, 8
(flarestack.core.llh.LLH class method), 13 get_n_exp_single()	<pre>read_injector_dict() (in module</pre>
(flarestack.core.injector.BaseInjector method),	flarestack.core.injector), 11
9	read_llh_dict() (in module flarestack.core.llh), 16
get_parameters() (flarestack.core.llh.LLH class	<pre>read_t_pdf_dict() (in module     flarestack.core.time_pdf), 5</pre>
method), 13	register_subclass()
grl_loader() (in module	(flarestack.core.energy_pdf.EnergyPDF class
flarestack.icecube_utils.dataset_loader), 17	method), 5, 7
	register_subclass()
	(flarestack.core.injector.BaseInjector class
inject_signal()( <i>flarestack.core.injector.BaseInjecto</i>	method), 9 register_subclass() (flarestack.core.llh.LLH
method), 9	
<pre>inject_signal() (flarestack.core.injector.EffectiveAre</pre>	aInjector register_subclass()
<pre>meinoa), 10 inject_signal() (flarestack.core.injector.MCInjector</pre>	(flarestack.core.time_pdf.TimePDF class
method), 10	method), 4
integrate_over_E()	return_energy_parameters()
(flarestack.core.energy_pdf.EnergyPDF	(flarestack.core.energy_pdf.EnergyPDF method), 5, 7
method), 5, 7	return_energy_parameters()
inverse_interpolate()	(flarestack.core.energy_pdf.PowerLaw
(flarestack.core.time_pdf.TimePDF method), 4	method), 6, 7
L	return_injected_parameters()
LLH (class in flarestack.core.llh), 12	(flarestack.core.energy_pdf.PowerLaw
load_band_mask() (flarestack.core.injector.LowMemo	method), 6, 7 ryInjector in increase and paper are ()
method), 10	(flarestack.core.llh.LLH static method), 13
LowMemoryInjector (class in	Junesiuckeone.un.ELH suute memoa), 13
flarestack core injector) 10	

Index 25

```
Τ
return_injected_parameters()
         (flarestack.core.llh.StandardLLH
                                               static
                                                      TimePDF (class in flarestack.core.time_pdf), 4
        method), 15
                                                      TrueUnblindedInjector
                                                                                           (class
                                                                                                        in
return_llh_parameters()
                                                               flarestack.core.injector), 11
         (flarestack.core.llh.LLH static method), 13
                                                      U
return llh parameters()
         (flarestack.core.llh.StandardLLH
                                               static
                                                      update sources()(flarestack.core.injector.BaseInjector
         method), 15
                                                               method), 9
S
                                                      V
select_mc_band() (flarestack.core.injector.MCInjector_verify_grl_with_data()
                                                                                                    module
        method), 10
                                                               flarestack.icecube_utils.dataset_loader),
select_spatially_coincident_data()
                                                               17
         (flarestack.core.llh.LLH method), 13
sig_t0() (flarestack.core.time_pdf.Box method), 1
                 (flarestack.core.time_pdf.FixedEndBox
                                                      weight mc()
                                                                       (flarestack.core.energy pdf.PowerLaw
         method), 2
                                                               method), 6, 7
sig_t0()
                 (flarestack.core.time_pdf.FixedRefBox
                                                      weight mc()
                                                                           (flarestack.core.energy_pdf.Spline
        method), 2
                                                               method), 6, 8
sig_t0() (flarestack.core.time_pdf.Steady method), 3
sig_t1() (flarestack.core.time_pdf.Box method), 1
sig t1()
                 (flarestack.core.time_pdf.FixedEndBox
         method), 2
                 (flarestack.core.time_pdf.FixedRefBox
sig_t1()
        method), 2
sig_t1() (flarestack.core.time_pdf.Steady method), 3
signal_f() (flarestack.core.time_pdf.Box method), 1
signal_f() (flarestack.core.time_pdf.Steady method),
                         (flarestack.core.time_pdf.Box
signal_integral()
        method), 2
signal_integral()
         (flarestack.core.time_pdf.Steady
                                            method),
signal_pdf() (flarestack.core.llh.LLH method), 13
simulate_times() (flarestack.core.time_pdf.TimePDF
         method), 4
SpatialLLH (class in flarestack.core.llh), 13
Spline (class in flarestack.core.energy pdf), 6, 7
StandardLLH (class in flarestack.core.llh), 14
StandardMatrixLLH (class in flarestack.core.llh), 15
StandardOverlappingLLH
                                     (class
                                                  in
        flarestack.core.llh), 15
Steady (class in flarestack.core.time_pdf), 3
subclasses (flarestack.core.energy_pdf.EnergyPDF
         attribute), 5, 7
subclasses (flarestack.core.injector.BaseInjector at-
         tribute), 9
subclasses (flarestack.core.injector.MCInjector at-
        tribute), 11
subclasses (flarestack.core.llh.LLH attribute), 13
subclasses
                    (flarestack.core.time_pdf.TimePDF
         attribute), 4
```

26 Index