

# ctools “user” Tutorial

Francesco Longo

Universita' di Trieste and INFN Trieste

# Summary

- Status of ctools SW
- GammaLib installation
- ctools installation
- ctools examples
- CTA response functions
- cscripts examples

# ctools introduction



CTA

Cherenkov Telescope Array Science Analysis Software

[Home](#)

[Get it](#)

## ctools

### About

ctools is a software package developed for the scientific analysis of Cherenkov Telescope Array (CTA) data. Analysis of data from existing Imaging Air Cherenkov Telescopes (such as H.E.S.S., MAGIC or VERITAS) is also supported, provided that the data and response functions are available in the format defined for CTA.

ctools comprises a set of ftools-like binary executables with a command-line interface allowing for interactive step-wise data analysis. ctools comprises also a Python module allowing to control all executables. Creation of shell or Python scripts and pipelines is supported. ctools comprises also cscripts, which are Python scripts that behave like binary ftools executables. Extensions of the ctools package by user defined binary executable or Python scripts is supported.

ctools are based on GammaLib, a versatile toolbox for the high-level analysis of astronomical gamma-ray data. Besides CTA, GammaLib supports also the analysis of Fermi/LAT and COMPTEL data, and extensions to support further gamma-ray instruments are planned. An interface to virtual observatory resources is also in preparation. By making use of the GammaLib multi-instrument capabilities, ctools supports the joint analysis of CTA (or any IACT providing data in the CTA format), Fermi/LAT and COMPTEL data.

ctools is free software distributed under the [GNU GPL license version 3](#)

<http://cta.irap.omp.eu/ctools/>

# Status of ctools



CTA

Cherenkov Telescope Array Science Analysis Software

[Home](#)

[Download](#)

[Home](#) | [Documentation](#) »

## Download

ctools can be obtained in form of releases or directly from the git development repository. Prefer a release if you intend using ctools for production (and publications). Clone the code from git if you need the most recent code that implements new features and corrects known bugs.

## Releases

The latest ctools release is `ctools-0.9.0` (22 May 2015).

Below a table of ctools releases. Please note that at this stage of the project there is a strict link between the ctools and gammalib versions. Please make sure that you have the corresponding gammalib version installed before installing ctools. The Mac OS X packages comprise both ctools and gammalib.

<u>ctools</u>	<u>gammalib</u>	<b>Mac OS X package</b>
<a href="#">0.9.0</a>	<a href="#">0.10.0</a>	<a href="#">ctools-0.9.0-macosx10.3.dmg</a>
<a href="#">0.8.1</a>	<a href="#">0.9.1</a>	<a href="#">ctools-00-08-01-macosx10.3.dmg</a>
<a href="#">0.8.0</a>	<a href="#">0.9.0</a>	<a href="#">ctools-00-08-00-macosx10.3.dmg</a>
<a href="#">0.7.1</a>	<a href="#">0.8.1</a>	<a href="#">ctools-00-07-01-macosx10.3.dmg</a>
<a href="#">0.7.0</a>	<a href="#">0.8.0</a>	<a href="#">ctools-00-07-00-macosx10.3.dmg</a>

# GammaLib installation

# Gammalib



GammaLib

A versatile toolbox for scientific analysis of astronomical gamma-ray data

[Home](#)

[Get it](#)

## GammaLib

### About

The GammaLib is a versatile toolbox for the high-level analysis of astronomical gamma-ray data. It is implemented as a C++ library that is fully scriptable in the Python scripting language. The library provides core functionalities such as data input and output, interfaces for parameter specifications, and a reporting and logging interface. It implements instruments specific functionalities such as instrument response functions and data formats. Instrument specific functionalities share a common interface to allow for extension of the GammaLib to include new gamma-ray instruments. The GammaLib provides an abstract data analysis framework that enables simultaneous multi-mission analysis.

GammaLib does not rely on any third-party software, except of HEASARC's cfitsio library that is used to implement the FITS interface. Large parts of the code treat gamma-ray observations in an abstract representation, and do neither depend on the characteristics of the employed instrument, nor on the particular formats in which data and instrument response functions are delivered. Instrument specific aspects are implemented as isolated and well defined modules that interact with the rest of the library through a common interface. This philosophy also enables the joint analysis of data from different instruments, providing a framework that allows for consistent broad-band spectral fitting or imaging. So far, GammaLib supports analysis of COMPTEL, Fermi/LAT, and Cherenkov telescope data (CTA, H.E.S.S., MAGIC, VERITAS).

GammaLib is free software distributed under the [GNU GPL license version 3](#)

<http://gammalib.sourceforge.net/>

# Gammalib installation

## Installation

### Note

These installation instructions apply to gammalib-00-04-10 and later. gammalib-00-04-10 has been built, installed and tested successfully on Debian, Ubuntu, Mandriva, OpenSUSE, Scientific Linux, CentOS, gentoo, Mac OS X, FreeBSD, and OpenSolaris (using gcc), so hopefully it also installs on your distribution. In case you encounter problems, please post a report on the bug tracker or send an e-mail to the [mailing list](#).

## Getting GammaLib

The latest version of the GammaLib source code, documentation, and example programs are available on the World-Wide Web from <https://sourceforge.net/projects/gammalib/>.

## Prerequisites

GammaLib should compile on every modern Unix system without any need to install other libraries. To enable support for FITS file handling, however, the cfitsio library from HEASARC needs to be installed. cfitsio exists on many distributions as a prebuilt library, so in general you can directly install it using your package manager. Make sure that you install the library and the development package, as the cfitsio header file (which usually comes in the development package) is needed to compiling GammaLib.

If cfitsio is not available as a prebuilt package, or if you encounter some problems with the prebuilt package, cfitsio can be downloaded from <http://heasarc.gsfc.nasa.gov/fitsio> and be installed from the source (for details, please refer to the installation instructions on the [HEASARC](#) site). We recommend installation of cfitsio in the default GammaLib install directory as a shared library by typing

```
$ ./configure --prefix=/usr/local/gamma
$ make shared
$ make install
```

GammaLib can also benefit from the presence of the readline library that provides line-editing and history capabilities for text input (GammaLib offers however also full functionality without having readline installed). readline (which depends on ncurses) is available on most system as a prebuilt library, so also here we recommend to use your package manager to install the libraries. Also here, the readline (and ncurses) development packages are required, so that the header files become available and the symbolic links are set correctly.

<http://gammalib.sourceforge.net/>

# gammalib installation

## Building GammaLib

Download the latest source tarball, save it in an appropriate location, and type

```
$ tar xvfz GammaLib-XX-XX-XX.tar.gz
```

where `XX-XX-XX` is the version number of the library. Step in the created directory using

```
$ cd gammalib-XX-XX-XX
```

and build GammaLib by typing

```
$ ./configure  
$ make
```

at the operating system prompt. The configuration command customizes the Makefiles for the particular system, the make command compiles the source files and builds the library, and the make install command installs the library in the install directory. Type `./configure` and not simply `configure` to ensure that the configuration script in the current directory is run and not some other system-wide configuration script. By default, the install directory is set to `/usr/local/gamma`. To change the install directory an optional `--prefix` argument should be given, for example

```
$ ./configure --prefix=/usr/local
```

If `cfitsio` and/or `readline` is not installed in a standard location for libraries (such as `/usr/lib` or `/usr/local/lib`), the appropriate location(s) can be specified using the `LDFLAGS` (for the library directory) and `CPPFLAGS` (for the include directory) options:

```
$ ./configure LDFLAGS='-L/opt/local/lib' CPPFLAGS='-I/opt/local/include'
```

A full list of configuration options can be found using

```
$ ./configure --help
```

<http://gammalib.sourceforge.net/>



# gammalib installation

## Installing GammaLib

Install GammaLib by typing

```
$ make install
```

at the operating system prompt.

## Setting up your environment

Before using GammaLib you have to setup some environment variables. This will be done automatically by an initialisation script that will be installed in the bin directory. Assuming that you have installed GammaLib in the default directory `/usr/local/gamma` you need to add the following to your `$HOME/.bashrc` or `$HOME/.profile` script on a Linux machine:

```
export GAMMALIB=/usr/local/gamma
source $GAMMALIB/bin/gammalib-init.sh
```

If you use C shell or a variant then add the following to your `$HOME/.cshrc` or `$HOME/.tcshrc` script:

```
setenv GAMMALIB /usr/local/gamma
source $GAMMALIB/bin/gammalib-init.csh
```

<http://gammalib.sourceforge.net/>

ctools installation

# ctools installation

## Building ctools

After downloading the ctools tarball (see [Download](#)), save it in an appropriate location (for example `$HOME/builds`), and type

```
$ tar xvfz ctools-0.9.0.tar.gz
```

(the `$` symbol indicates the console prompt and is not part of the command that you should type in).

Step in the directory and build the ctools by typing

```
$ cd ctools-0.9.0  
$ ./configure  
$ make
```

at the operating system prompt. The `./configure` command customizes the Makefiles for the particular system, the `make` command compiles the source files and builds the executables. Type `./configure` and not simply `configure` to ensure that the configuration script in the current directory is run and not some other system-wide configuration script.

You can get the full list of configuration options by typing

```
$ ./configure --help
```

<http://cta.irap.omp.eu/ctools/>

# ctools installation

## Testing ctools

Before installing the ctools you should execute the unit test suite to make sure that ctools have been built correctly. For this, type

```
$ make check
```

If you have automake version 1.13 or newer installed, you should see the following output at the end of the unit testing:

```
PASS: test_ctools.sh
PASS: test_cscripts.sh
PASS: test_python.py
make[4]: Nothing to be done for `all'.
=====
Testsuite summary for ctools 0.9.0
=====
# TOTAL: 3
# PASS: 3
# SKIP: 0
# XFAIL: 0
# FAIL: 0
# XPASS: 0
# ERROR: 0
=====
```

<http://cta.irap.omp.eu/ctools/>

# ctools installation

For older automake version, you should see

```
*****
* Test ctools *
*****
Test ctobssim: ... ok
Test ctskymap: .. ok
Test ctbin: ... ok
Test ctmodel: ..... ok
Test ctselect: .. ok
Test ctlike: ..... ok
PASS: test_ctools.sh

*****
* Test cscripts *
*****
Test cspull: .. ok
Test cstdist: .. ok
PASS: test_cscripts.sh

*****
* ctools Python interface testing *
*****
Test executable analysis: ... ok
Test in-memory analysis: ... ok
Test diffuse map cube with constant analysis: .... ok
Test diffuse map cube with power law analysis: ..... ok
PASS: test_python.py
=====
All 3 tests passed
=====
```

The same detailed information is also available for the newer automake versions, but there it is written in log files that you can find in the `test` directory of the ctools:

```
test_ctools.sh.log
test_cscripts.sh.log
test_python.py.log
```

<http://cta.irap.omp.eu/ctools/>

# ctools installation

## Installing ctools

The ctools are installed by typing

```
$ make install
```

If the destination directory is owned by `root` (which is normally the case when using the default), administrator privileges are needed for installation. In this case, type

```
$ sudo make install
```

By default, the install directory is set to `/usr/local/gamma`. To change the install directory (for example in case that you do not have system administrator privileges), an optional `--prefix` argument can be given, for example:

```
$ ./configure --prefix=$HOME/gamma
```

## Setting up the ctools environment

You have to configure ctools by setting up some environment variables. This will be done automatically by an initialisation script that is found in the `bin` directory of the directory into which ctools has been installed. Assuming that you have installed ctools into `/usr/local/ctools` you need to add the following to your `$HOME/.bashrc` or `$HOME/.profile` script on a Linux machine:

```
export CTOOLS=/usr/local/ctools
source $CTOOLS/bin/ctools-init.sh
```

If you use C shell or a variant then add the following to your `$HOME/.cshrc` or `$HOME/.tcshrc` script:

```
setenv CTOOLS /usr/local/ctools
source $CTOOLS/bin/ctools-init.csh
```

<http://cta.irap.omp.eu/ctools/>

# ctools installation

## Known problems

### GammaLib unit tests fail

Some users have reported failure of a large fraction of the GammaLib unit tests after after typing `make check`. In all cases, this was related to the absence of the directory where the shared `libcfitsio` library resides in the library load path. To solve the issue, locate the directory where the shared `libcfitsio` library resides and then type

```
export LD_LIBRARY_PATH=/directory/to/lib:$LD_LIBRARY_PATH
```

on Unix based systems or

```
export DYLD_LIBRARY_PATH=/directory/to/lib:$DYLD_LIBRARY_PATH
```

on Mac OS X (`/directory/to/lib` should be replaced by the correct library path on your system).

### Python support

ctools comes with Python wrappers so that all classes can be directly used from Python. To compile-in Python support, ctools needs the `Python.h` header file, which on many distributions is not installed by default. To make `Python.h` available, install the Python developer package in your distribution. Otherwise you will not be able to use ctools from Python.

<http://cta.irap.omp.eu/ctools/>

# Example

```
[longof@farmui02 gamma]$ more setup_new.sh
export GAMMALIB=/gpfs/glast/CTA_Software/release_FL_newR00T/ctools/gamma
source $GAMMALIB/bin/gammalib-init.sh
export CALDB=/gpfs/glast/CTA_Software/release_FL_newR00T/ctools/gamma/share/caldb
export CTTOOLS=/gpfs/glast/CTA_Software/release_FL_newR00T/ctools/gamma
source $CTTOOLS/bin/ctools-init.sh
export PATH=/gpfs/glast/Software/Utilities/saods9/bin/:${PATH}
export LD_LIBRARY_PATH=/gpfs/glast/CTA_Software/release_FL_newR00T/ctools/gamma/lib:${LD_LIBRARY_PATH}
```



ctools examples

# ctools

## Reference Manual

This manual provides reference information for all ctools and csripts. General information on ctools usage can be found [here](#).

Below you find links to the command line reference for the tools and scripts that are available.

### ctools

- [ctbin — Generates counts cube](#)
- [ctbkgcube — Generates background cube](#)
- [ctbutterfly — Compute butterfly](#)
- [ctcubemask — Filter counts cube](#)
- [ctexpcube — Generates exposure cube](#)
- [ctlike — Performs maximum likelihood fitting](#)
- [ctmodel — Computes model counts cube](#)
- [ctobssim — Simulate CTA observations](#)
- [ctpsfcube — Generates point spread function cube](#)
- [ctselect — Selects event data](#)
- [ctskymap — Generates sky map](#)
- [cttmap — Generates Test Statistics map](#)
- [ctulimit — Calculates upper limit](#)

# How to use ctobssim?

## Simulating CTA data

CTA data are simulated by the executable ctobssim. To start the executable, type ctobssim at the console prompt (which is denoted by \$). ctobssim will query for a number of parameters:

```
$ ctobssim
Model [$CTOOLS/share/models/crab.xml]
RA of pointing (degrees) (0-360) [83.63]
Dec of pointing (degrees) (-90-90) [22.01]
Calibration database [dummy]
Instrument response function [cta_dummy_irf]
Radius of FOV (degrees) (0-180) [5.0]
Start time (MET in s) (0) [0.0]
End time (MET in s) (0) [1800.0]
Lower energy limit (TeV) (0) [0.1]
Upper energy limit (TeV) (0) [100.0]
Output event data file or observation definition file [events.fits]
```

Each line represents a query for one parameter. The line starts with a short description of the parameter, followed by the default parameter value proposed by ctobssim in squared brackets [ ]. If no parameter is entered (which is the case for the majority of parameters shown here), the default parameter will be used. Otherwise, the specified parameter will overwrite the default parameter.

You may have recognised that the environment variable \$CTOOLS has been used in the path names of the first two parameters. ctools will automatically expand the environment variables.

The CTA instrument properties (effective area, PSF width) are taken for the moment from a dummy performance table that is located in \$CTOOLS/share/caldb/cta.

Events are simulated based on the instrument properties and based on a source and background model. Only events that fall within the specified region of interest (ROI), defined as a circle around a sky position in Right Ascension and Declination (in degrees), will be stored in the output event data file. The duration of the simulation is taken here to 30 minutes (or 1800 seconds). Events are simulated for energies between 0.1 and 100 TeV.

<http://cta.irap.omp.eu/ctools/>

# How to?

## Implementation

The general model is describe in ctools using a model definition XML file. Below is a simple example of such a file comprising one source and one background model. Each model is factorised in a spectral (tag `<spectrum>`) and a spatial component (tags `<spatialModel>` and `<radialModel>`):

$$M(x, y, E) = M_{\text{spectral}}(E) \times M_{\text{spatial}}(x, y)$$

In this specific example, the source component `Crab` describes a point source at the location of the Crab nebula with a power law spectral shape. The background component `Background` is modelled as a radial Gaussian function in offset angle squared (with the offset angle being defined as the angle between pointing and measured event direction) and a spectral function that is tabulated in an ASCII file.

```
<?xml version="1.0" standalone="no"?>
<source_library title="source library">
  <source name="Crab" type="PointSource">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" scale="1e-16" value="5.7" min="1e-07" max="1000.0" free="1"/>
      <parameter name="Index" scale="-1" value="2.48" min="0.0" max="+5.0" free="1"/>
      <parameter name="Scale" scale="1e6" value="0.3" min="0.01" max="1000.0" free="0"/>
    </spectrum>
    <spatialModel type="SkyDirFunction">
      <parameter name="RA" scale="1.0" value="83.6331" min="-360" max="360" free="1"/>
      <parameter name="DEC" scale="1.0" value="22.0145" min="-90" max="90" free="1"/>
    </spatialModel>
  </source>
  <source name="Background" type="RadialAcceptance" instrument="CTA">
    <spectrum type="FileFunction" file="$CTOOLS/share/models/bkg_dummy.txt">
      <parameter name="Normalization" scale="1.0" value="1.0" min="0.0" max="1000.0" free="1"/>
    </spectrum>
    <radialModel type="Gaussian">
      <parameter name="Sigma" scale="1.0" value="3.0" min="0.01" max="10.0" free="1"/>
    </radialModel>
  </source>
</source_library>
```

# How to use?

The source and background model is defined by the XML file `$CTOOLS/share/models/crab.xml`:

```
<?xml version="1.0" standalone="no"?>
<source_library title="source library">
  <source name="Crab" type="PointSource">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" scale="1e-16" value="5.7" min="1e-07" max="1000.0" free="1"/>
      <parameter name="Index" scale="-1" value="2.48" min="0.0" max="+5.0" free="1"/>
      <parameter name="Scale" scale="1e6" value="0.3" min="0.01" max="1000.0" free="0"/>
    </spectrum>
    <spatialModel type="SkyDirFunction">
      <parameter name="RA" scale="1.0" value="83.6331" min="-360" max="360" free="0"/>
      <parameter name="DEC" scale="1.0" value="22.0145" min="-90" max="90" free="0"/>
    </spatialModel>
  </source>
  <source name="Background" type="RadialAcceptance" instrument="CTA">
    <spectrum type="FileFunction" file="$CTOOLS/share/models/bkg_dummy.txt">
      <parameter scale="1.0" name="Normalization" min="0.0" max="1000.0" value="1.0" free="1"/>
    </spectrum>
    <radialModel type="Gaussian">
      <parameter name="Sigma" scale="1.0" value="3.0" min="0.01" max="10.0" free="1"/>
    </radialModel>
  </source>
</source_library>
```

The model consists of a source library that contains 2 “sources”: the Crab nebula and an instrumental background model.

The Crab nebula is modelled by a factorized sky model that has a spectral and a spatial component (tags `<spectrum>` and `<spatialModel>`, respectively). The spectrum is modelled by a power law, which is defined by 3 parameters: the `Prefactor`, the `Index` and the `Scale`. The spatial model has 2 parameters: Right Ascension in degrees (RA), and Declination in degrees (DEC). Each parameter has a value and a scale factor, the real value of the parameter being the product  $\text{value} * \text{scale}$ . Typically, scale is chosen so that value is of the order of 1 (this is relevant for model fitting later). In addition, value is bound by a minimum (min) and maximum (max) value, and a parameter may be free (`free="1"`) or fixed (`free="0"`). The min, max, and free attributes are not relevant here for the simulations, but they will be important for the model fitting later.

<http://cta.irap.omp.eu/ctools/>

# How to use?

The spectral intensity  $I(E)$  (in units of photons/cm<sup>2</sup>/s/MeV) of the power law is given by

$$\frac{dN}{dE} = N_0 \left( \frac{E}{E_0} \right)^\gamma$$

where the parameters in the XML definition have the following mappings:

- $N_0$  = Prefactor
- $\gamma$  = Index
- $E_0$  = Scale

Note that energies are given in MeV.

The instrumental background of CTA is modelled by a factorized data model that has a spectral and a radial component (tags `<spectrum>` and `<radialModel>`, respectively). The spectral component describes the on-axis background counting rate of CTA as function of energy in units of counts/s/sr/TeV. The radial component describes the variation of the background rate with offset angle squared, (i.e. square of the offset angle with respect to the pointing direction) which is modelled here by a Gaussian. The only parameter of the radial component is the width of the Gaussian Sigma, which is here set to 3 degrees squared.

<http://cta.irap.omp.eu/ctools/>

# Spectral Models

Power law

The `GModelSpectralPlaw` class implements the power law function

$$\frac{dN}{dE} = k_0 \left( \frac{E}{E_0} \right)^\gamma$$

where the parameters in the XML definition have the following mappings:

- $k_0$  = Prefactor
- $\gamma$  = Index
- $E_0$  = Scale

The XML format for specifying a power law is:

```
<spectrum type="PowerLaw">
  <parameter name="Prefactor" scale="1e-16" value="5.7" min="1e-07" max="1000.0" free="1"/>
  <parameter name="Index" scale="-1" value="2.48" min="0.0" max="+5.0" free="1"/>
  <parameter name="Scale" scale="1e6" value="0.3" min="0.01" max="1000.0" free="0"/>
</spectrum>
```

An alternative power law function is defined by the `GModelSpectralPlaw2` class that uses the integral flux as parameter rather than the Prefactor:

$$\frac{dN}{dE} = \frac{N(\gamma + 1)E^\gamma}{E_{\max}^{\gamma+1} - E_{\min}^{\gamma+1}}$$

where the parameters in the XML definition have the following mappings:

- $N$  = Integral
- $\gamma$  = Index
- $E_{\min}$  = LowerLimit
- $E_{\max}$  = UpperLimit

The XML format for specifying a power law defined by the integral flux is:

```
<spectrum type="PowerLaw2">
  <parameter scale="1e-07" name="Integral" min="1e-07" max="1000.0" value="1.0" free="1"/>
  <parameter scale="1.0" name="Index" min="-5.0" max="+5.0" value="-2.0" free="1"/>
  <parameter scale="1.0" name="LowerLimit" min="10.0" max="1000000.0" value="100.0" free="0"/>
  <parameter scale="1.0" name="UpperLimit" min="10.0" max="1000000.0" value="500000.0" free="0"/>
</spectrum>
```

**NOTE:** The UpperLimit and LowerLimit parameters are always treated as fixed and, as should be apparent from this definition, the flux given by the Integral parameter is over the range (LowerLimit, UpperLimit). Use of this model allows the errors on the integrated flux to be evaluated directly by likelihood, obviating the need to propagate the errors if one is using the PowerLaw form.

[http://gammalib.sourceforge.net/user\\_manual/modules/model.html#sec-model](http://gammalib.sourceforge.net/user_manual/modules/model.html#sec-model)

# Spectral Models

Exponentially cut-off power law

The `GModelSpectralExpPlaw` class implements the exponentially cut-off power law function

$$\frac{dN}{dE} = k_0 \left( \frac{E}{E_0} \right)^\gamma \exp \left( \frac{-E}{E_{\text{cut}}} \right)$$

where the parameters in the XML definition have the following mappings:

- $k_0$  = Prefactor
- $\gamma$  = Index
- $E_0$  = Scale
- $E_{\text{cut}}$  = Cutoff

The XML format for specifying an exponentially cut-off power law is:

```
<spectrum type="ExpCutoff">
  <parameter name="Prefactor" scale="1e-16" value="5.7" min="1e-07" max="1000.0" free="1"/>
  <parameter name="Index" scale="-1" value="2.48" min="0.0" max="+5.0" free="1"/>
  <parameter name="Cutoff" scale="1e6" value="1.0" min="0.01" max="1000.0" free="1"/>
  <parameter name="Scale" scale="1e6" value="0.3" min="0.01" max="1000.0" free="0"/>
</spectrum>
```

[http://gammalib.sourceforge.net/user\\_manual/modules/model.html#sec-model](http://gammalib.sourceforge.net/user_manual/modules/model.html#sec-model)



# Spectral Models

Broken power law

The [GModelSpectralBrokenPlaw](#) class implements the broken power law function

$$\frac{dN}{dE} = k_0 \times \begin{cases} \left(\frac{E}{E_b}\right)^{\gamma_1} & \text{if } E < E_b \\ \left(\frac{E}{E_b}\right)^{\gamma_2} & \text{otherwise} \end{cases}$$

where the parameters in the XML definition have the following mappings:

- $k_0$  = Prefactor
- $\gamma_1$  = Index1
- $\gamma_2$  = Index2
- $E_b$  = BreakValue

The XML format for specifying a broken power law is:

```
<spectrum type="BrokenPowerLaw">
  <parameter name="Prefactor" scale="1e-16" value="5.7" min="1e-07" max="1000.0" free="1"/>
  <parameter name="Index1" scale="-1" value="2.48" min="0.0" max="+5.0" free="1"/>
  <parameter name="BreakValue" scale="1e6" value="0.3" min="0.01" max="1000.0" free="1"/>
  <parameter name="Index2" scale="-1" value="2.70" min="0.01" max="1000.0" free="1"/>
</spectrum>
```

[http://gammalib.sourceforge.net/user\\_manual/modules/model.html#sec-model](http://gammalib.sourceforge.net/user_manual/modules/model.html#sec-model)

# Spectral Models

## Log parabola

The `GModelSpectralLogParabola` class implements the log parabola function

$$\frac{dN}{dE} = k_0 \left( \frac{E}{E_0} \right)^{\gamma + \eta \ln(E/E_0)}$$

where the parameters in the XML definition have the following mappings:

- $k_0$  = Prefactor
- $\gamma$  = Index
- $\eta$  = Curvature
- $E_0$  = Scale

The XML format for specifying a log parabola spectrum is:

```
<spectrum type="LogParabola">
  <parameter name="Prefactor" scale="1e-17" value="5.878" min="1e-07" max="1000.0" free="1"/>
  <parameter name="Index" scale="-1" value="2.32473" min="0.0" max="+5.0" free="1"/>
  <parameter name="Curvature" scale="-1" value="0.074" min="-5.0" max="+5.0" free="1"/>
  <parameter name="Scale" scale="1e6" value="1.0" min="0.01" max="1000.0" free="0"/>
</spectrum>
```

An alternative XML format is supported for compatibility with the Fermi/LAT XML format:

```
<spectrum type="LogParabola">
  <parameter name="Prefactor" scale="1e-17" value="5.878" min="1e-07" max="1000.0" free="1"/>
  <parameter name="alpha" scale="1" value="2.32473" min="0.0" max="+5.0" free="1"/>
  <parameter name="beta" scale="1" value="0.074" min="-5.0" max="+5.0" free="1"/>
  <parameter name="Scale" scale="1e6" value="1.0" min="0.01" max="1000.0" free="0"/>
</spectrum>
```

where

- $\alpha$  = -Index
- $\beta$  = -Curvature

[http://gammalib.sourceforge.net/user\\_manual/modules/model.html#sec-model](http://gammalib.sourceforge.net/user_manual/modules/model.html#sec-model)

# Spectral Models

## Log parabola

The `GModelSpectralLogParabola` class implements the log parabola function

$$\frac{dN}{dE} = k_0 \left( \frac{E}{E_0} \right)^{\gamma + \eta \ln(E/E_0)}$$

where the parameters in the XML definition have the following mappings:

- $k_0$  = Prefactor
- $\gamma$  = Index
- $\eta$  = Curvature
- $E_0$  = Scale

The XML format for specifying a log parabola spectrum is:

```
<spectrum type="LogParabola">
  <parameter name="Prefactor" scale="1e-17" value="5.878" min="1e-07" max="1000.0" free="1"/>
  <parameter name="Index" scale="-1" value="2.32473" min="0.0" max="+5.0" free="1"/>
  <parameter name="Curvature" scale="-1" value="0.074" min="-5.0" max="+5.0" free="1"/>
  <parameter name="Scale" scale="1e6" value="1.0" min="0.01" max="1000.0" free="0"/>
</spectrum>
```

An alternative XML format is supported for compatibility with the Fermi/LAT XML format:

```
<spectrum type="LogParabola">
  <parameter name="Prefactor" scale="1e-17" value="5.878" min="1e-07" max="1000.0" free="1"/>
  <parameter name="alpha" scale="1" value="2.32473" min="0.0" max="+5.0" free="1"/>
  <parameter name="beta" scale="1" value="0.074" min="-5.0" max="+5.0" free="1"/>
  <parameter name="Scale" scale="1e6" value="1.0" min="0.01" max="1000.0" free="0"/>
</spectrum>
```

where

- $\alpha$  = -Index
- $\beta$  = -Curvature

[http://gammalib.sourceforge.net/user\\_manual/modules/model.html#sec-model](http://gammalib.sourceforge.net/user_manual/modules/model.html#sec-model)

# Spectral Models

## File function

A function defined using an input ASCII file with columns of energy and differential flux values. The energy units are assumed to be MeV and the flux values are assumed to be  $\text{cm}^{-2}\text{s}^{-1}\text{MeV}^{-1}$  (the only exception being a model for which the spatial component is a constant diffuse model [GModelSpatialDiffuseConst](#); in this case, the units are  $\text{cm}^{-2}\text{s}^{-1}\text{MeV}^{-1}\text{sr}^{-1}$ ). The sole parameter is a multiplicative normalization:

$$\frac{dN}{dE} = N_0 \left. \frac{dN}{dE} \right|_{\text{file}}$$

where the parameters in the XML definition have the following mappings:

- $N_0 = \text{Normalization}$

The XML format for specifying a file function is:

```
<spectrum type="FileFunction" file="data/filefunction.txt">
  <parameter scale="1.0" name="Normalization" min="0.0" max="1000.0" value="1.0" free="1"/>
</spectrum>
```

If the file is given as relative path, the path is relative to the working directory of the executable. Alternatively, an absolute path may be specified. Any environment variable present in the path name will be expanded.

[http://gammalib.sourceforge.net/user\\_manual/modules/model.html#sec-model](http://gammalib.sourceforge.net/user_manual/modules/model.html#sec-model)

# How to use?

ctobssim will write 2 files in the working directory: `events.fits` and `ctobssim.log`. The first file contains the simulated events in FITS format and can be inspected using `fv` or `ds9`. The FITS file will contain 3 extensions: an empty primary image, a binary table named `EVENTS` that holds the events (one row per event), and a binary table named `GTI` holding the Good Time Intervals (for the moment a single row with 2 columns providing the start and the stop time of the simulated time interval).

The second file produced by ctobssim is a human readable log file that contains information about the job execution. As example, the last lines from this file are shown here:

```
2014-10-30T22:35:06: +=====+
2014-10-30T22:35:06: | Simulate observation |
2014-10-30T22:35:06: +=====+
2014-10-30T22:35:06: === Observation ===
2014-10-30T22:35:06: Simulation area .....: 1.9635e+11 cm2
2014-10-30T22:35:06: Simulation cone .....: RA=83.63 deg, Dec=22.01 deg, r=5.5 deg
2014-10-30T22:35:06: Time interval .....: 0 - 1800 s
2014-10-30T22:35:06: Photon energy range .....: 100 GeV - 100 TeV
2014-10-30T22:35:06: Event energy range .....: 100 GeV - 100 TeV
2014-10-30T22:35:06: MC source photons .....: 207547 [Crab]
2014-10-30T22:35:06: MC source events .....: 995 [Crab]
2014-10-30T22:35:06: MC source events .....: 995 (all source models)
2014-10-30T22:35:06: MC background events .....: 5146
2014-10-30T22:35:06: MC events .....: 6141 (all models)
2014-10-30T22:35:06:
2014-10-30T22:35:06: +=====+
2014-10-30T22:35:06: | Save observation |
2014-10-30T22:35:06: +=====+
2014-10-30T22:35:06:
2014-10-30T22:35:06: Application "ctobssim" terminated after 10 wall clock seconds, consuming 0.3604 seconds o
```

Each line starts with the UTC time at which the line has been written. In this run, 207547 Crab photons have been thrown over an area of 19.6 square kilometres during a time interval of 1800 seconds. 995 of these photons have been registered by CTA as events. In the same time interval, 5146 background events have been registered by CTA.

<http://cta.irap.omp.eu/ctools/>

# Example

```
2015-06-21T16:02:41: *****
2015-06-21T16:02:41: *                               ctobssim                               *
2015-06-21T16:02:41: * ----- *
2015-06-21T16:02:41: * Version: 1.0.0 *
2015-06-21T16:02:41: *****
2015-06-21T16:03:12: +=====+
2015-06-21T16:03:12: | Parameters |
2015-06-21T16:03:12: +=====+
2015-06-21T16:03:12: inobs .....: NONE
2015-06-21T16:03:12: inmodel .....: grb_130427A_10000.xml
2015-06-21T16:03:12: outevents .....: events_10000.fits
2015-06-21T16:03:12: prefix .....: sim_events_
2015-06-21T16:03:12: caldb .....: prod2
2015-06-21T16:03:12: irf .....: North_5h
2015-06-21T16:03:12: seed .....: 1
2015-06-21T16:03:12: ra .....: 83.63
2015-06-21T16:03:12: dec .....: 22.01
2015-06-21T16:03:12: rad .....: 5.0
2015-06-21T16:03:12: tmin .....: 0.0
2015-06-21T16:03:12: tmax .....: 1800
2015-06-21T16:03:12: emin .....: 0.02
2015-06-21T16:03:12: emax .....: 0.1
2015-06-21T16:03:12: edisp .....: no
2015-06-21T16:03:12: deadc .....: 0.95
2015-06-21T16:03:12: maxrate .....: 1.0e6
2015-06-21T16:03:12: chatter .....: 2
2015-06-21T16:03:12: clobber .....: yes
2015-06-21T16:03:12: debug .....: no
2015-06-21T16:03:12: mode .....: ql
2015-06-21T16:03:12: logfile .....: ctobssim.log
2015-06-21T16:03:12:
```

# Example

```
2015-06-21T16:03:12: +=====+
2015-06-21T16:03:12: | Observation |
2015-06-21T16:03:12: +=====+
2015-06-21T16:03:12: === GObservations ===
2015-06-21T16:03:12:   Number of observations .....: 1
2015-06-21T16:03:12:   Number of predicted events : 0
2015-06-21T16:03:12: === GCTAObservation ===
2015-06-21T16:03:12:   Name .....:
2015-06-21T16:03:12:   Identifier .....:
2015-06-21T16:03:12:   Instrument .....: CTA
2015-06-21T16:03:12:   Event file .....:
2015-06-21T16:03:12:   Event type .....: EventList
2015-06-21T16:03:12:   Statistics .....: Poisson
2015-06-21T16:03:12:   Ontime .....: 1800 s
2015-06-21T16:03:12:   Livetime .....: 1710 s
2015-06-21T16:03:12:   Deadtime correction .....: 0.95
2015-06-21T16:03:12:   User energy range .....: undefined
2015-06-21T16:03:12: === GCTAPointing ===
2015-06-21T16:03:12:   Pointing direction .....: (RA,Dec)=(83.63,22.01)
2015-06-21T16:03:12: === GCTAResponseIrf ===
2015-06-21T16:03:12:   Caldb mission .....: cta
2015-06-21T16:03:12:   Caldb instrument .....: prod2
2015-06-21T16:03:12:   Response name .....: North_5h
2015-06-21T16:03:12:   Energy dispersion .....: Not used
2015-06-21T16:03:12:   Save energy range .....: undefined
2015-06-21T16:03:12: === GCTAEventList ===
2015-06-21T16:03:12:   Number of events .....: 0
2015-06-21T16:03:12:   Time interval .....: 51544.5 - 51544.5 days
2015-06-21T16:03:12:   Energy interval .....: 0.02 - 0.1 TeV
2015-06-21T16:03:12:   Region of interest .....: RA=83.63, DEC=22.01 [0,0] Radius=5 deg
2015-06-21T16:03:12: ~~~~ ~~~~ ~~~~ ~~~~ ~~~~
```

# Example

```
2015-06-21T16:03:12: +=====+
2015-06-21T16:03:12: | Simulate observation |
2015-06-21T16:03:12: +=====+
2015-06-21T16:03:12: === Observation ===
2015-06-21T16:03:12: Simulation area .....: 1.9635e+11 cm2
2015-06-21T16:03:12: Simulation cone .....: RA=83.63 deg, Dec=22.01 deg, r=5.5 deg
2015-06-21T16:03:12: Time interval .....: 0 - 1800 s
2015-06-21T16:03:12: Photon energy range .....: 20 GeV - 100 GeV
2015-06-21T16:03:12: Event energy range .....: 20 GeV - 100 GeV
2015-06-21T16:03:14: MC source photons .....: 1047448 [grb_130427A_1000]
2015-06-21T16:03:14: MC source events .....: 586 [grb_130427A_1000]
2015-06-21T16:03:14: MC source events .....: 586 (all source models)
2015-06-21T16:03:17: MC background events .....: 37139
2015-06-21T16:03:17: MC events .....: 37725 (all models)
2015-06-21T16:03:21:
2015-06-21T16:03:21: +=====+
2015-06-21T16:03:21: | Save observation |
2015-06-21T16:03:21: +=====+
2015-06-21T16:03:21:
2015-06-21T16:03:21: Application "ctobssim" terminated after 40 wall clock seconds, consuming 10.68 seconds of CPU time.
```



# How to use?

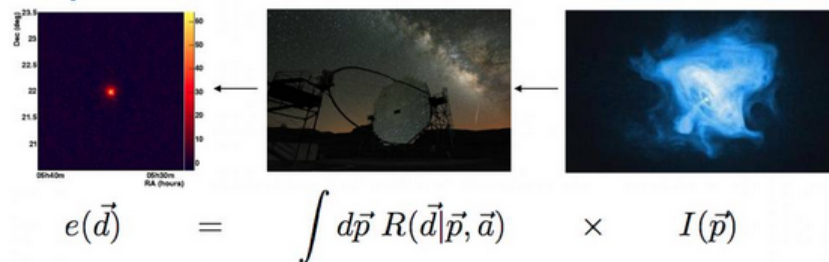
## CTA Instrument Response Functions

### Note

Instrument response functions are formally not part of ctools as they should be provided by the instrument teams. The GammaLib framework on which ctools are built comes with a set of basic response functions, but for getting the latest instrument response functions you should contact the relevant instrument teams. CTA consortium members should [download](#) the latest calibration database containing Prod1 and Prod2 instrument response functions from the consortium Sharepoint site (requires password).

### What are instrument response functions?

The instrument response functions provide a mathematical description that links the measured quantities  $\vec{d}$  of an event to the physical quantities  $\vec{p}$  of the incident photon. The following figure illustrates this relationship:



$I(\vec{p})$  is the gamma-ray intensity arriving at Earth as a function of photon properties  $\vec{p}$  (which usually are true photon energy, true photon incident direction, and true photon arrival time), while  $e(\vec{d})$  is the expected event rate as function of event properties  $\vec{d}$  (which usually are the measured photon energy, measured or reconstructed photon incident direction, and measured photon arrival time). The expected event rate is obtained by integrating the product of the instrumental response function  $R(\vec{d}|\vec{p}, \vec{a})$  and the emitted intensity  $I(\vec{p})$  over the photon properties  $\vec{p}$ . The argument  $\vec{a}$  in the response function comprises any auxiliary parameter on which the response function may depend on (e.g. pointing direction, triggered telescopes, optical efficiencies, atmospheric conditions, etc.). All these quantities and hence the instrument response function may depend on time.

[http://cta.irap.omp.eu/ctools/user\\_manual/getting\\_started/response.html](http://cta.irap.omp.eu/ctools/user_manual/getting_started/response.html)

# How to use?

## ARF, RMF and PSF files

Instrument response information for the first CTA Data Challenge (1DC) has been provided in a format that was heavily inspired by the [RMF and ARF file formats](#) that have been introduced by [HEASARC](#) for spectral analysis of X-ray data.

The Redistribution Matrix File (RMF) describes how an incoming photon with a given true energy is redistributed in measured energy. In other words, it describes the energy dispersion of the instrument. The RMF is organised as a two-dimensional matrix, with the first axis being in true energies while the second axis represents the measured energies.

The Ancillary Response File (ARF) describes the sensitivity of the instrument to photons of a given true energy. The ARF gives the effective area of the detector system after applying an event selection cut (theta cut). For computing the ARF, some knowledge of the Point Spread Function (PSF) is needed, so that the fraction of photons that falls into the event selection region for a given source can be properly estimated. Once this is done, no further PSF information is needed for the analysis.

ctools however relies on the full modelling of the instrument, including the Point Spread Function, and hence, PSF information has also been provided for 1DC. Two data formats have been used for this: a first that is based on a simple one-dimensional vector, providing the width of a 2-dimensional Gaussian as function of energy; and a second that is based on a three-component 2D Gaussian as function of energy and offset angle in the camera system. While the former was implemented by a simple FITS table with two columns, the latter was implemented by [Response tables](#).

## Response tables

The CTA response table class `GCTAResponseTable` provides a generic handle for multi-dimensional response information. It is based on the response format used for storing response information for the *Fermi*/LAT telescope. In this format, all information is stored in a single row of a FITS binary table. Each element of the row contains a vector column, that describes the axes of the multi-dimensional response cube and the response information. Note that this class may in the future be promoted to the GammaLib core, as a similar class has been implemented in the *Fermi*/LAT interface.

[http://cta.irap.omp.eu/ctools/user\\_manual/getting\\_started/response.html](http://cta.irap.omp.eu/ctools/user_manual/getting_started/response.html)

# How to use?

## Performance tables

In the early days, the instrument performances derived from Monte-Carlo simulations have been summarised in what we call here Performance Tables, which are ASCII files that contain as function of energy the on-axis performance parameters of CTA, such as effective area, point spread function containment radius, energy resolution, background count rate and differential sensitivity.

Below an example of a CTA performance table:

log(E)	Area	r68	r80	ERes.	BG Rate	Diff Sens
-1.7	261.6	0.3621	0.4908	0.5134	0.0189924	6.88237e-11
-1.5	5458.2	0.2712	0.3685	0.4129	0.1009715	1.72717e-11
-1.3	15590.0	0.1662	0.2103	0.2721	0.0575623	6.16963e-12
-1.1	26554.1	0.1253	0.1567	0.2611	0.0213008	2.89932e-12
-0.9	52100.5	0.1048	0.1305	0.1987	0.0088729	1.39764e-12
-0.7	66132.1	0.0827	0.1024	0.1698	0.0010976	6.03531e-13
-0.5	108656.8	0.0703	0.0867	0.1506	0.0004843	3.98147e-13
-0.3	129833.0	0.0585	0.0722	0.1338	0.0001575	3.23090e-13
-0.1	284604.3	0.0531	0.0656	0.1008	0.0001367	2.20178e-13
0.1	263175.3	0.0410	0.0506	0.0831	0.0000210	1.87452e-13
0.3	778048.6	0.0470	0.0591	0.0842	0.0000692	1.53976e-13
0.5	929818.8	0.0391	0.0492	0.0650	0.0000146	1.18947e-13
0.7	1078450.0	0.0335	0.0415	0.0541	0.0000116	1.51927e-13
0.9	1448579.1	0.0317	0.0397	0.0516	0.0000047	1.42439e-13
1.1	1899905.0	0.0290	0.0372	0.0501	0.0000081	1.96670e-13
1.3	2476403.8	0.0285	0.0367	0.0538	0.0000059	2.20695e-13
1.5	2832570.6	0.0284	0.0372	0.0636	0.0000073	3.22523e-13
1.7	3534065.3	0.0290	0.0386	0.0731	0.0000135	4.84153e-13
1.9	3250103.4	0.0238	0.0308	0.0729	0.0000044	6.26265e-13
2.1	3916071.6	0.0260	0.0354	0.0908	0.0000023	7.69921e-13

-----

- 1) log(E) = log<sub>10</sub>(E/TeV) - bin centre
- 2) Eff Area - in square metres after background cut (no theta cut)
- 3) Ang. Res - 68% containment radius of gamma-ray PSF post cuts - in degrees
- 4) Ang. Res - 80% containment radius of gamma-ray PSF post cuts - in degrees
- 5) Fractional Energy Resolution (rms)
- 6) BG Rate - inside point-source selection region - post call cuts - in Hz
- 7) Diff Sens - differential sensitivity for this bin expressed as E<sup>2</sup> dN/dE  
- in erg cm<sup>-2</sup> s<sup>-1</sup> - for a 50 hours exposure - 5 sigma significance including systematics and statistics and at least 10 photons.

[http://cta.irap.omp.eu/ctools/user\\_manual/getting\\_started/response.html](http://cta.irap.omp.eu/ctools/user_manual/getting_started/response.html)

# Response Functions

## CTA response functions

The instrument response functions for CTA are factorised into the effective area  $A_{\text{eff}}(d, p, E, t)$  (units  $\text{cm}^2$ ), the point spread function  $PSF(p'|d, p, E, t)$ , and the energy dispersion  $E_{\text{disp}}(E'|d, p, E, t)$  following:

$$R(p', E', t|d, p, E, t) = A_{\text{eff}}(d, p, E, t) \times PSF(p'|d, p, E, t) \times E_{\text{disp}}(E'|d, p, E, t)$$

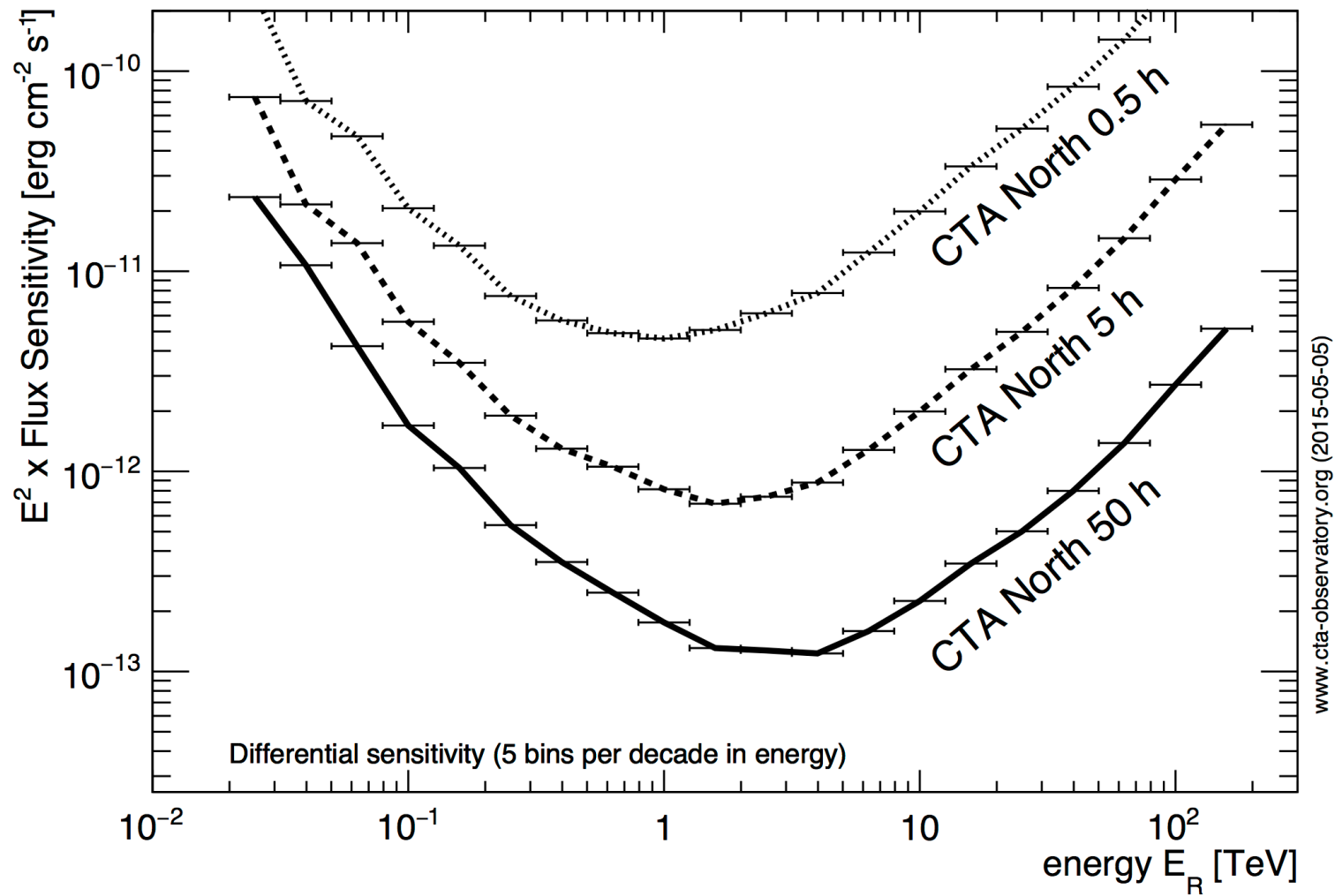
ctools are shipped with response functions for the northern and southern arrays, and variants are available that have been optimised for exposure times of 0.5 hours, 5 hours and 50 hours. In total, the following six instrument response functions are available: `North_0.5h`, `North_5h`, `North_50h`, `South_0.5h`, `South_5h`, and `South_50h`.

Each response is stored in a single FITS file, and each component of the response factorisation is stored in a binary table of that FITS file. In addition, the response files contain an additional table that describes the background rate as function of energy and position in the field of view. An example of a CTA response file is shown below:

Index	Extension	Type	Dimension	View				
<input type="checkbox"/> 0	Primary	Image	0	Header	Image	Table		
<input type="checkbox"/> 1	EFFECTIVE AREA	Binary	6 cols X 1 rows	Header	Hist	Plot	All	Select
<input type="checkbox"/> 2	POINT SPREAD FUNCTION	Binary	10 cols X 1 rows	Header	Hist	Plot	All	Select
<input type="checkbox"/> 3	ENERGY DISPERSION	Binary	7 cols X 1 rows	Header	Hist	Plot	All	Select
<input type="checkbox"/> 4	BACKGROUND	Binary	7 cols X 1 rows	Header	Hist	Plot	All	Select

Each table in the response file is in a standardised format that is the one that is also used for the Fermi/LAT telescope. As an example, the effective area component of the response file is shown below. Response information is stored in a n-dimensional cube, and each axis of this cube is described by the lower and upper edges of the axis bins. In this example the effective area is stored as a 2D matrix with the first axis being energy and the second axis being offaxis angle. Effective area information is stored for true (`EFFAREA`) and reconstructed (`EFFAREA_RECO`) energy. Vector columns are used to store all information.

# Response Function



# How to use?

## Binning CTA data

As next analysis step you will bin the data in a counts cube using the executable `ctbin`. A counts cube is a 3 dimensional data cube, spanned by Right Ascension (or Galactic longitude), Declination (or Galactic latitude), and the logarithm (base 10) of energy.

`ctbin` is executed by typing:

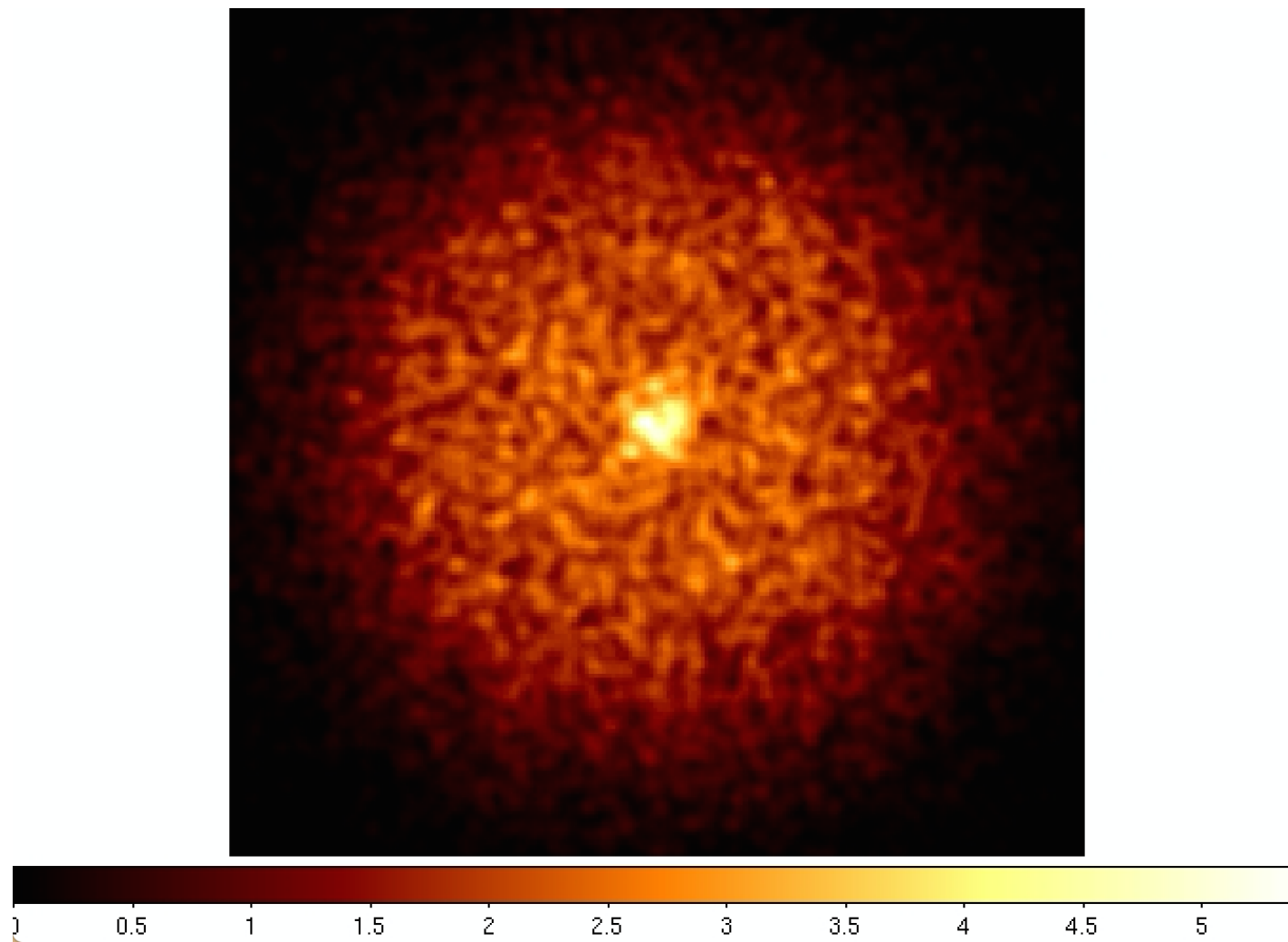
```
$ ctbin
Input event list or observation definition file [events.fits]
First coordinate of image center in degrees (RA or galactic l) [83.63]
Second coordinate of image center in degrees (DEC or galactic b) [22.01]
Projection method e.g. AIT|AZP|CAR|MER|STG|TAN (AIT|AZP|CAR|MER|STG|TAN) [CAR]
Coordinate system (CEL - celestial, GAL - galactic) (CEL|GAL) [CEL]
Image scale (in degrees/pixel) [0.02]
Size of the X axis in pixels [200]
Size of the Y axis in pixels [200]
Algorithm for defining energy bins (FILE|LIN|LOG) [LOG]
Start value for first energy bin in TeV [0.1]
Stop value for last energy bin in TeV [100.0]
Number of energy bins [20]
Output counts cube [cntmap.fits]
```

In this example we adjust the event data file name and accept all the remaining parameter defaults as they perfectly satisfy our needs. The counts cube will be centred on the location of the Crab (Right Ascension 83.63 degrees, Declination 22.01 degrees) and will be aligned in celestial coordinates. A cartesian projection has been selected. The counts cube has 200 x 200 spatial pixels of 0.02 x 0.02 degrees in size, hence it covers a total area of 4 x 4 degrees.

The counts cube will contain 20 maps, which are logarithmically spaced in energy, and which cover the energy range from 0.1 TeV to 100 TeV. In this example, the counts cube will be saved as `cntmap.fits` in the working directory. In addition to the counts cube, that is stored as the primary image extension, the FITS file also contains an extension named `EBOUNDS` that defines the energy boundaries that were used, and an extension `GTI` that defines the Good Time Intervals that have been used. The following image shows the resulting FITS file. The `EBOUNDS` table has 20 rows, one for each energy bin, while the `GTI` table has just a single row, indicating the start and stop time of the simulated data.

File Edit Tools				Help				
Index	Extension	Type	Dimension	View				
<input type="checkbox"/> 0	Primary	Image	200 X 200 X 20	Header	Image	Table		
<input type="checkbox"/> 1	EBOUNDS	Binary	2 cols X 20 rows	Header	Hist	Plot	All	Select
<input type="checkbox"/> 2	GTI	Binary	2 cols X 1 rows	Header	Hist	Plot	All	Select

ctbin





# How to use?

## Fitting CTA data

Now we are ready to fit the simulated data with a model. For simplicity we use in this example the same model that we used to simulate the data with `ctobssim`. Model fitting is done using the executable `ctlike`, and we do the fit by typing:

```
$ clike
Event list, counts cube or observation definition file [events.fits] cntmap.fits
Calibration database [dummy]
Instrument response function [cta_dummy_irf]
Source model [%CTOOLS/share/models/crab.xml]
Source model output file [crab_results.xml]
```

Fitting of the data is done in *binned* mode, which means that the events have been binned into a counts cube and the fit computes the log-likelihood function by summing over all 200 x 200 x 20 bins of the counts cube. There is an alternative method, the so called *unbinned* mode, where the events are not binned into a counts cube and the log-likelihood is computed directly by summing over all events. We will explore the *unbinned* mode later.

One of the parameters given to `ctlike` is a source model output file (we specified `crab_results.xml` in the example), and this file will be a copy of the model XML file where the parameter values have been replaced by the fit results. In addition, the statistical uncertainties are added for each fitted parameter using the attribute `error`. Below we show the XML result file that has been produced by the run:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<source_library title="source library">
  <source name="Crab" type="PointSource">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" value="6.07928" error="0.204582" scale="1e-16" min="1e-07" max="1000" free="1" />
      <parameter name="Index" value="2.5009" error="0.0252057" scale="-1" min="0" max="5" free="1" />
      <parameter name="Scale" value="0.3" scale="1e+06" min="0.01" max="1000" free="0" />
    </spectrum>
    <spatialModel type="SkyDirFunction">
      <parameter name="RA" value="83.6331" scale="1" min="-360" max="360" free="0" />
      <parameter name="DEC" value="22.0145" scale="1" min="-90" max="90" free="0" />
    </spatialModel>
  </source>
</source_library>
```



# Likelihood

## Doing an unbinned analysis

As gamma-ray events are rare, the counts cubes generated by `ctbin` will in general be sparse, having many empty pixels, in particular at high energies. An alternative analysis technique consists of working directly on the event list without binning the events in a counts cube. We will see the benefit of such an analysis later once you re-run `ctlike` in unbinned mode.

For unbinned analysis you first have to define the data space region over which the analysis is done. This is similar to the `ctbin` step in binned analysis where you defined the size of the counts cube, the energy range, and the time interval. For unbinned analysis you have no such thing as a counts cube, but you have to define over which region of the data space the selected events are spread (because the `ctools` have to integrate over this region to compute the total number of predicted events in the data space that you analyse). Furthermore, you have to define what energy range is covered, and what time interval is spanned by the data. All this is done by the executable `ctselect`, which replaces the `ctbin` step in an unbinned analysis.

`ctselect` performs an event selection by choosing only events within a given region-of-interest (ROI), within a given energy band, and within a given time interval from the input event list. The ROI is a circular region on the sky, for which you define the centre (in celestial coordinates) and the radius. Such a circular ROI is sometimes also called an acceptance cone. The following example shows how to run `ctselect`:

```
$ ctselect
Input event list or observation definition file [events.fits]
RA for ROI centre (degrees) (0-360) [83.63]
Dec for ROI centre (degrees) (-90-90) [22.01]
Radius of ROI (degrees) (0-180) [3.0]
Start time (CTA MET in seconds) (0) [0.0]
End time (CTA MET in seconds) (0) [0.0]
Lower energy limit (TeV) (0) [0.1]
Upper energy limit (TeV) (0) [100.0]
Output event list or observation definition file [selected_events.fits]
```

# ctlike

```
$ clike
Event list, counts cube or observation definition file [cntmap.fits] selected_events.fits
Calibration database [dummy]
Instrument response function [cta_dummy_irf]
Source model [$CTOOLS/share/models/crab.xml]
Source model output file [crab_results.xml]
```

You will recognise that clike runs much faster in unbinned mode compared to binned mode. This is understandable as the selected event list contains only 6127 events, while the binned counts cube we used before had  $200 \times 200 \times 20 = 800000$  pixels. As unbinned maximum likelihood fitting loops over the events (while binned maximum likelihood loops over the pixels), there are much less operations to perform in unbinned than in binned mode (there is some additional overhead in unbinned mode that comes from integrating the models over the region of interest, yet this is negligible compared to the operations needed when looping over the pixels). So as long as you work with short event lists, unbinned mode is faster. Unbinned clike should also be more precise as no binning is performed, hence there is no loss of information due to histogramming.

# example

```
2015-06-21T16:06:17: +=====+
2015-06-21T16:06:17: | Maximum likelihood optimization results |
2015-06-21T16:06:17: +=====+
2015-06-21T16:06:17: === GOptimizerLM ===
2015-06-21T16:06:17:   Optimized function value ...: 121106.914
2015-06-21T16:06:17:   Absolute precision .....: 0.005
2015-06-21T16:06:17:   Acceptable value decrease ..: 2
2015-06-21T16:06:17:   Optimization status .....: converged
2015-06-21T16:06:17:   Number of parameters .....: 10
2015-06-21T16:06:17:   Number of free parameters ..: 4
2015-06-21T16:06:17:   Number of iterations .....: 6
2015-06-21T16:06:17:   Lambda .....: 1e-07
2015-06-21T16:06:17:   Maximum log likelihood .....: -121106.914
2015-06-21T16:06:17:   Observed events (Nobs) ....: 37725.000
2015-06-21T16:06:17:   Predicted events (Npred) ...: 37724.997 (Nobs - Npred = 0.00259882)
2015-06-21T16:06:17: === GModels ===
2015-06-21T16:06:17:   Number of models .....: 2
2015-06-21T16:06:17:   Number of parameters .....: 10
2015-06-21T16:06:17: === GModelSky ===
2015-06-21T16:06:17:   Name .....: grb_130427A_1000
2015-06-21T16:06:17:   Instruments .....: all
2015-06-21T16:06:17:   Test Statistic .....: 410.209
2015-06-21T16:06:17:   Instrument scale factors ...: unity
2015-06-21T16:06:17:   Observation identifiers ....: all
2015-06-21T16:06:17:   Model type .....: PointSource
2015-06-21T16:06:17:   Model components .....: "SkyDirFunction" * "PowerLaw" * "Constant"
2015-06-21T16:06:17:   Number of parameters .....: 6
2015-06-21T16:06:17:   Number of spatial par's ....: 2
2015-06-21T16:06:17:     RA .....: 83.6331 [-360,360] deg (fixed,scale=1)
2015-06-21T16:06:17:     DEC .....: 22.0145 [-90,90] deg (fixed,scale=1)
2015-06-21T16:06:17:   Number of spectral par's ...: 2
```

# example

```
2015-06-21T16:06:17: === GModelSky ===
2015-06-21T16:06:17: Name .....: grb_130427A_1000
2015-06-21T16:06:17: Instruments .....: all
2015-06-21T16:06:17: Test Statistic .....: 410.209
2015-06-21T16:06:17: Instrument scale factors ...: unity
2015-06-21T16:06:17: Observation identifiers ....: all
2015-06-21T16:06:17: Model type .....: PointSource
2015-06-21T16:06:17: Model components .....: "SkyDirFunction" * "PowerLaw" * "Constant"
2015-06-21T16:06:17: Number of parameters .....: 6
2015-06-21T16:06:17: Number of spatial par's ....: 2
2015-06-21T16:06:17: RA .....: 83.6331 [-360,360] deg (fixed,scale=1)
2015-06-21T16:06:17: DEC .....: 22.0145 [-90,90] deg (fixed,scale=1)
2015-06-21T16:06:17: Number of spectral par's ...: 3
2015-06-21T16:06:17: Prefactor .....: 2.98596e-10 +/- 2.41992e-10 [1e-17,1e-07] ph/cm2/s/MeV (free,scale=1e-10,gradient)
2015-06-21T16:06:17: Index .....: -2.28532 +/- 0.185673 [-0,-5] (free,scale=-1,gradient)
2015-06-21T16:06:17: PivotEnergy .....: 826 [10,1e+06] MeV (fixed,scale=1000,gradient)
2015-06-21T16:06:17: Number of temporal par's ...: 1
2015-06-21T16:06:17: Normalization .....: 1 (relative value) (fixed,scale=1,gradient)
2015-06-21T16:06:17: === GCTAModelIrfBackground ===
2015-06-21T16:06:17: Name .....: CTABackgroundModel
2015-06-21T16:06:17: Instruments .....: CTA
2015-06-21T16:06:17: Instrument scale factors ...: unity
2015-06-21T16:06:17: Observation identifiers ....: all
2015-06-21T16:06:17: Model type .....: "PowerLaw" * "Constant"
2015-06-21T16:06:17: Number of parameters .....: 4
2015-06-21T16:06:17: Number of spectral par's ...: 3
2015-06-21T16:06:17: Prefactor .....: 0.995015 +/- 0.0443823 [0.001,1000] ph/cm2/s/MeV (free,scale=1,gradient)
2015-06-21T16:06:17: Index .....: -0.00286326 +/- 0.0135258 [-5,5] (free,scale=1,gradient)
2015-06-21T16:06:17: PivotEnergy .....: 1e+06 [10000,1e+09] MeV (fixed,scale=1e+06,gradient)
2015-06-21T16:06:17: Number of temporal par's ...: 1
2015-06-21T16:06:17: Normalization .....: 1 (relative value) (fixed,scale=1,gradient)
2015-06-21T16:06:17:
```

# TS calculation

## Note

The [\*ctlike\*](#) tool has the ability to estimate the detection significance for sources in the XML model. This is done by computing the Test Statistic value which is defined as twice the log-likelihood difference between fitting a source at a given position on top of a (background) model or fitting no source. Roughly spoken, the square root of the Test Statistic value gives the source detection significance in Gaussian sigmas, although the exact relation depends somewhat on the formulation of the statistical problem.

To instruct [\*ctlike\*](#) to compute the Test Statistic value for a given source you need to add the attribute `tscal="1"` to the XML file:

```
<source name="Crab" type="PointSource" tscal="1">
```

[\*ctlike\*](#) will then compute the Test Statistic value for that source and dump the result in the log file:

```
2015-05-22T19:58:43: === GModelSky ===
2015-05-22T19:58:43: Name .....: Crab
2015-05-22T19:58:43: Instruments .....: all
2015-05-22T19:58:43: Test Statistic .....: 18662.6
```

The Test Statistic value will also be added as new attribute `ts` to the XML result file:

```
<source name="Crab" type="PointSource" ts="18662.576" tscal="1">
```

# example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<source_library title="source library">
  <source name="grb_130427A_1000" type="PointSource" ts="410.209" tscal="1">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" value="2.98596" error="2.41992" scale="1e-10" min="1e-07" max="1000" free="1" />
      <parameter name="Index" value="2.28532" error="0.185673" scale="-1" min="0" max="5" free="1" />
      <parameter name="Scale" value="0.826" scale="1000" min="0.01" max="1000" free="0" />
    </spectrum>
    <spatialModel type="SkyDirFunction">
      <parameter name="RA" value="83.6331" scale="1" min="-360" max="360" free="0" />
      <parameter name="DEC" value="22.0145" scale="1" min="-90" max="90" free="0" />
    </spatialModel>
  </source>
  <source name="CTABackgroundModel" type="CTAIrfBackground" instrument="CTA">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" value="0.995015" error="0.0443823" scale="1" min="0.001" max="1000" free="1" />
      <parameter name="Index" value="-0.00286326" error="0.0135258" scale="1" min="-5" max="5" free="1" />
      <parameter name="Scale" value="1" scale="1e+06" min="0.01" max="1000" free="0" />
    </spectrum>
  </source>
</source_library>
```

# Butterfly

## Calculate and visualise butterfly

To visualise the analysis results retrieved above, one can calculate the confidence band of the spectral fit. The tool `ctbutterfly` takes the optimised source model as input. It takes the covariance matrix from the fit to conduct a Gaussian error propagation for each energy value. It will write the butterfly information into an ASCII file. The following example shows how to compute such a butterfly from the command line.

```
$ ctbutterfly
Input event list, cube or observation definition file [events.fits]
Calibration database [dummy]
Instrument response function [cta_dummy_irf]
Source model [%CTOOLS/share/models/crab.xml] crab_results.xml
Source of interest [Crab]
Start value for first energy bin in TeV [0.1]
Stop value for last energy bin in TeV [100.0]
Output ascii file [butterfly.txt]
```

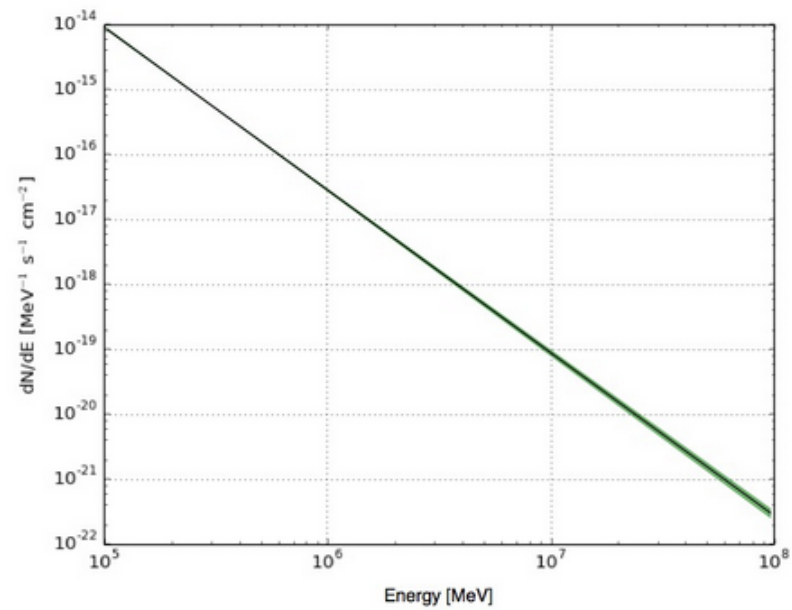
Below some lines of the `ctbutterfly.log`:

```
2014-10-30T17:51:39: +=====+
2014-10-30T17:51:39: | Compute covariance matrix |
2014-10-30T17:51:39: +=====+
2014-10-30T17:51:39:
2014-10-30T17:51:39: +=====+
2014-10-30T17:51:39: | Generate butterfly |
2014-10-30T17:51:39: +=====+
2014-10-30T17:51:39:
2014-10-30T17:51:39: +=====+
2014-10-30T17:51:39: | Save Butterfly to file |
2014-10-30T17:51:39: +=====+
2014-10-30T17:51:39:
2014-10-30T17:51:39: Application "ctbutterfly" terminated after 15 wall clock seconds, consuming 0.051253
```

# Butterfly

```
python $CTOOLS/share/examples/python/show_butterfly.py butterfly.txt
```

This will result in a canvas which should look like the following:



*Confidence band of the fit*




# cscripts

## cscripts

- cscaldb — Lists available instrument response functions
- csobsdef — Generates observation definition file
- cslightcrv — Computes lightcurve
- cspull — Generates pull distribution
- cssens — Computes CTA sensitivity
- csspec — Computes spectral points
- csresmap — Generates residual map
- cstdist — Generates TS distribution

# Next ...



- Overview
- Activity
- Roadmap
- Issues
- News
- Documents
- Wiki**
- Code status
- Links
- Forums
- Files
- Repository

[Wiki » Coding sprints »](#)

## Fourth coding sprint

The fourth coding sprint will take place at IRAP, Toulouse in the week 29 June – 3 July. We will start around noon on the first day and stop around noon on the last day so that you can travel on the same day to Toulouse (we may adapt the exact time of start on stop to your travel schedule).

### Participants

- Michael Mayer (HU Berlin)
- Rolf Buehler (DESY)
- Christoph Deil (MPIK)
- Jürgen Knödlseider (IRAP)
- Tarek Hassan (IFAE)
- Nathan Kelley-Hoskins (DESY)
- Johan Bregeon (LUPM)
- Lili Yang (UNG)

### Practical information

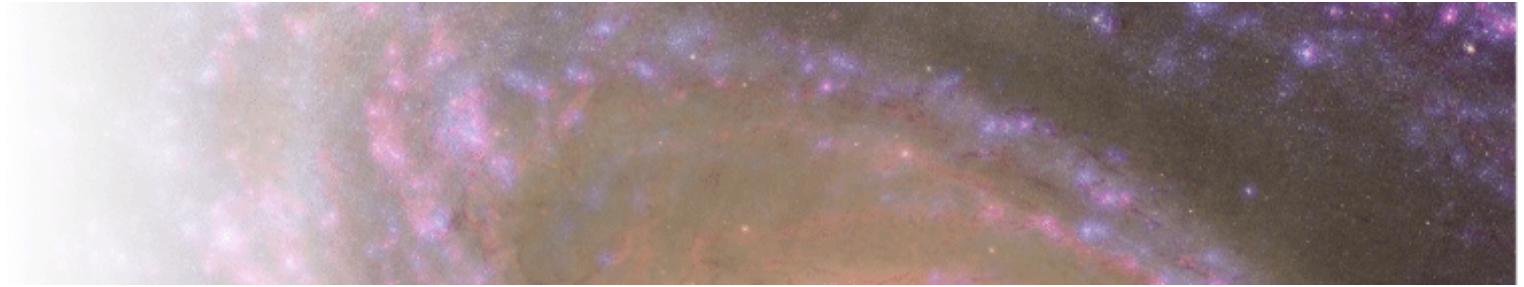
The meeting will take place at IRAP–Roche, information of how to reach the location can be found at <http://cta.irap.omp.eu/toulouse2011/practical.html>. A suggestion of possible hotels is on <http://cta.irap.omp.eu/toulouse2011/accomodation.html>. Note that since recently we have a tramway connection to the airport. You may take the tram to go to the terminus (Palais de Justice) and from there take the metro to Faculté de Pharmacie (direction Ramonville) to go to the lab.

### Collection of issues to be addressed during the sprint

I propose that the main goal of the 4th coding sprint would be to get the gammalib and ctools release 1.0 out. There are all couple of outstanding issues (see [Gammalib](#) and [ctools](#) roadmaps), probably non of these issues should be blocking. We can then focus on fixing remaining issues, doing science verification and writing pending documentation. We should also start to work on the paper.

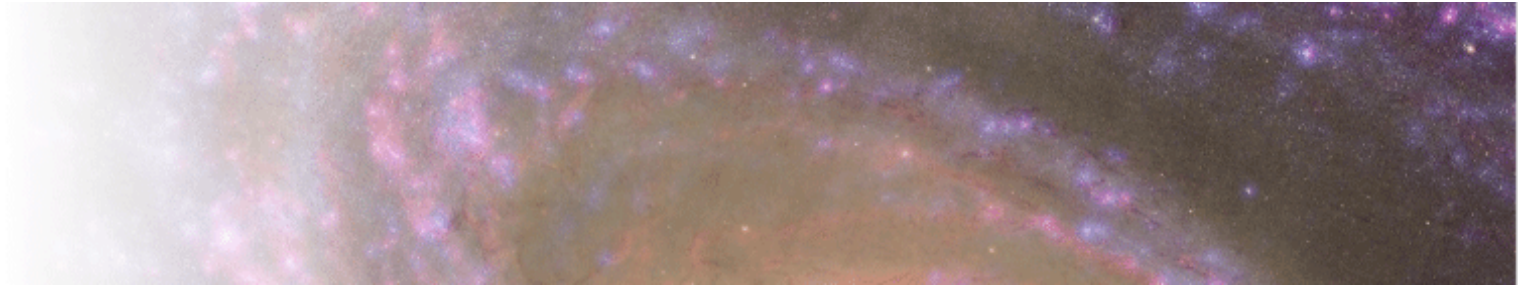
Besides this, here a list of issues that can be addressed:

- Improve binned analysis at the energy threshold ([#1362](#))
- Implement containment\_radius method for GCTAPsf classes ([#1459](#))
- Optimise and test usage of energy resolution
- IRFs: status, limitations, roadmap



# Towards the Cherenkov Telescope Array and Future Gamma-ray Experiments

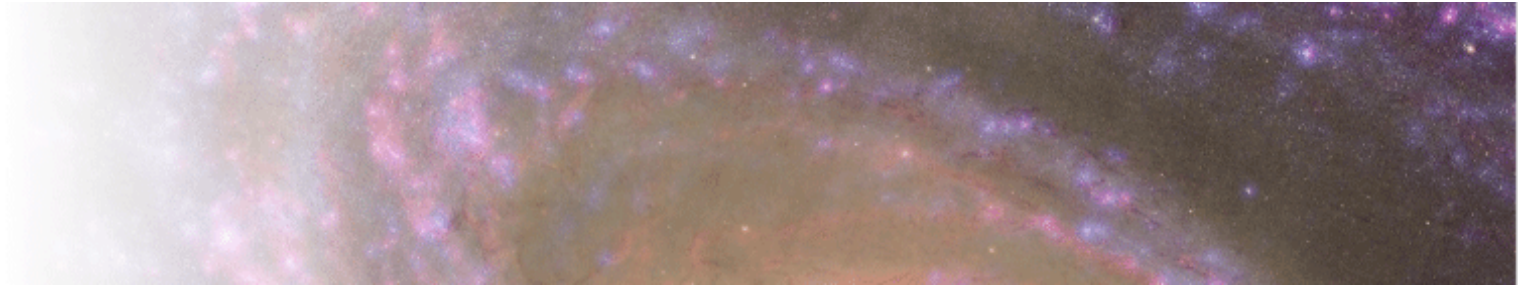
Summer School  
July, 27 – 31, 2015  
Sexten – Dolomites Italy



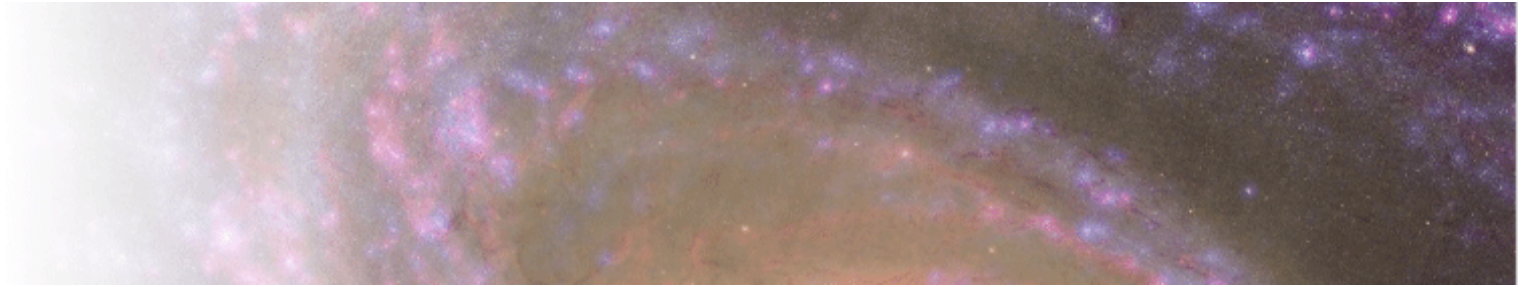
- A beautiful location !





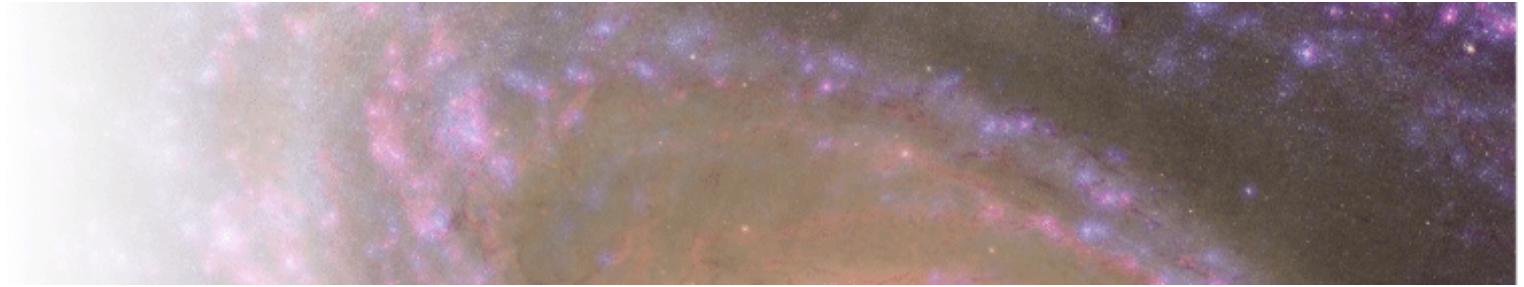


- A nice program!
- 1. The legacy of current cosmic-ray and gamma-ray space telescopes**
- 2. The legacy of the current VHE telescopes**
- 3. The development of CTA and telescope prototypes**
- 4. Science with CTA**
- 5. Future Gamma-ray experiments**



- For more information follow the link

<http://www.sexten-cfa.eu/en/conferences/2015/details/60-Towards-the-Cherenkov-Telescope-Array.html>



- See you soon!!

