FirstSteps

November 18, 2019

1 First step with ctools

We start by importing the gammalib, ctools, and cscripts Python packages. We will also import the plotting library matplotlib to visualize the result. If you run ctools from the command line there is no need to go through this step.

```
In [1]: import gammalib
        import ctools
        import cscripts
        import matplotlib.pyplot as plt
        # add the following line to display matplotlib inline in a notebook
        %matplotlib inline
```

1.1 Simulating event data

CTA does not exist yet. We will simulate event data to learn how to use the tools. We will simulate a short observation of the Crab nebula pointing at a slight offset from the source. We will use a simple model dispatched with the tools that contains the Crab nebula as pointlike source with a power-law spectrum, and the instrumental background modeled based on the IRFs. We will enable energy dispersion, that, by default, is not taken into account for the sake of computing speed.

```
In [2]: sim = ctools.ctobssim()
        sim['inmodel'] = '${CTOOLS}/share/models/crab.xml'
        sim['outevents'] = 'events.fits'
        sim['caldb'] = 'prod3b-v2'
        sim['irf'] = 'South_z40_0.5h'
        sim['ra'] = 83.5
        sim['dec'] = 22.8
        sim['rad'] = 5.0
        sim['tmin'] = '2020-01-01T00:00:00'
        sim['tmax'] = '2020-01-01T01:00:00'
        sim['emin'] = 0.03 # energies as user parameters are always in TeV
        sim['emax'] = 150.0
        sim['edisp'] = True
        sim.execute()
```

The first line generates an instance of the ctobssim tool as a Python class. User parameters are then set using the [ ] operator. You can consult the manual of each tool to find out how it works and to discover all the input parameters that you can set.

```
In [3]: !ctobssim --help
```

```
ctobssim
========

Simulate event list(s).


Synopsis
--------

This tool simulates event list(s) using the instrument characteristics
specified by the instrument response function(s) and an input model. The
simulation includes photon events from astrophysical sources and background
events from an instrumental background model.

By default, ctobssim creates a single event list. ctobssim queries a pointing
direction, the radius of the simulation region, a time interval, an energy
interval, an instrumental response function, and an input model. ctobssim uses
a numerical random number generator for the simulations with a seed value
provided by the hidden seed parameter. Changing this parameter for
subsequent runs will lead to different event samples.

ctobssim performs a safety check on the maximum photon rate for all model
components to avoid that the tool locks up and requests huge memory
resources, which may happen if a mistake was made in setting up the input
model (for example if an error in the flux units is made). The maximum allowed
photon rate is controlled by the hidden maxrate parameter, which by default
is set to 1e6.

ctobssim can also generate multiple event lists if an observation definition
file is specified on input using the hidden inobs parameter. In that
case, simulation information will be gathered from the file, and for each
observation an event list will be created.

For each event file, the simulation parameters will be written as data
selection keywords to the FITS header. These keywords are mandatory for any
unbinned maximum likelihood analysis of the event data.


General parameters
------------------
```

```
inobs [file]
    Input event list or observation definition XML file. If provided (i.e. the
    parameter is not blank or NONE), the pointing definition and eventually the
    response information will be extracted from the input file for event
    simulation.

inmodel [file]
    Input model XML file.

caldb [string]
    Calibration database.

irf [string]
    Instrumental response function.

(edisp = no) [boolean]
    Apply energy dispersion?

outevents [file]
    Output event list or observation definition XML file.

(prefix = sim_events_) [string]
    Prefix for event list in observation definition XML file.

(startindex = 1) [integer]
    Start index of event list in observation definition XML file.

(seed = 1) [integer]
    Integer seed value to be used for Monte Carlo simulations. Keep this
    parameter at the same value for repeatable simulations, or increment
    this value for subsequent runs if non-repeatable simulations are
    required.

ra [real]
    Right Ascension of CTA pointing (J2000, in degrees).

dec [real]
    Declination of CTA pointing (J2000, in degrees).

rad [real]
    Radius of CTA field of view (simulation cone radius) (in degrees).

tmin [time]
    Start time (UTC string, JD, MJD or MET in seconds).

tmax [time]
    Stop time (UTC string, JD, MJD or MET in seconds).
```

```
(mjdref = 51544.5) [real]
    Reference Modified Julian Day (MJD) for simulated events. The times in
    seconds for each event are counted from this reference time on.

emin [real]
    Lower energy limit of simulated events (in TeV).

emax [real]
    Upper energy limit of simulated events (in TeV).

(deadc = 0.95) [real]
    Average deadtime correction factor.

(maxrate = 1.0e6) [real]
    Maximum photon rate for source models. Source models that exceed this
    maximum photon rate will lead to an exception as very likely the
    specified model normalisation is too large (probably due to the
    a misinterpretation of units). Note that ctools specifies intensity
    units per MeV.


Standard parameters
-------------------

(nthreads = 0) [integer]
    Number of parallel processes (0=use all available CPUs).

(publish = no) [boolean]
    Specifies whether the event list(s) should be published on VO Hub.

(chatter = 2) [integer]
    Verbosity of the executable:
     chatter = 0: no information will be logged

     chatter = 1: only errors will be logged

     chatter = 2: errors and actions will be logged

     chatter = 3: report about the task execution

     chatter = 4: detailed report about the task execution

(clobber = yes) [boolean]
    Specifies whether existing files should be overwritten.

(debug = no) [boolean]
    Enables debug mode. In debug mode the executable will dump any log file output to the conso
```

```
(mode = ql) [string]
    Mode of automatic parameters (default is ql, i.e. "query and learn").

(logfile = ctobssim.log) [string]
    Name of log file.


Related tools or scripts
------------------------

csobsdef
```

If you run a tool from the command line you will be asked directly for all the relevant parameters, and you will be offered a default choice. This is also a great way to discover how to use the tools. After setting all parameters the execute() method is called to execute the ctobssim tool. On output the events.fits FITS file is created.

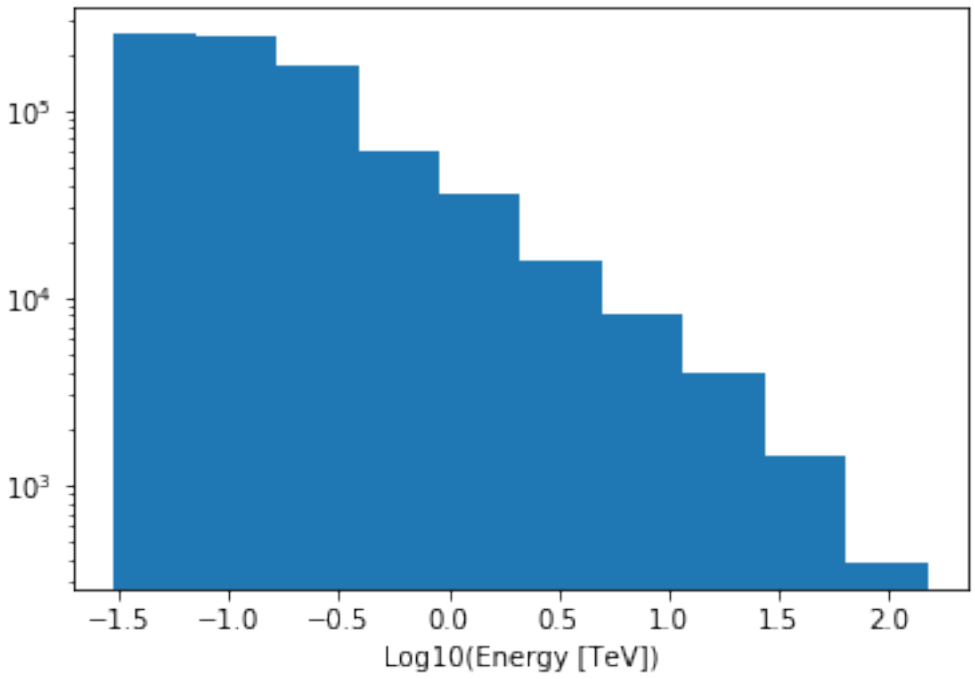
From Python alternatively you can use the run() method, that will run the task without writing any output to disk. The output will be accessible from Python, and can be used to build in memory-pipelines to implement complex analysis workflows. The key object in memory after the simulation is a container of simulated observations, that we will now inspect.

```
In [4]: print(sim.obs())
        print(sim.obs()[0])

=== GObservations ===
 Number of observations ...: 1
 Number of models ...: 2
 Number of observed events .: 801933
 Number of predicted events : 0
=== GCTAObservation ===
 Name ...:
 Identifier ...: 000001
 Instrument ...: CTA
 Event file ...: events.fits
 Event type ...: EventList
 Statistic ...: cstat
 Ontime ...: 3599.99999976158 s
 Livetime ...: 3527.99999976635 s
 Deadtime correction ...: 0.98
 User energy range ...: undefined
=== GCTAPointing ===
 Pointing direction ...: (RA,Dec)=(83.5,22.8)
=== GCTAResponseIrf ===
 Caldb mission ...: cta
 Caldb instrument ...: prod3b-v2
 Response name ...: South_z40_0.5h
 Energy dispersion ...: Not used
```

```
 Safe energy range ...: undefined
=== GCTAEventList ===
 Number of events ...: 801933 (disposed in "events.fits")
 Time interval ...: 58849.0008007407 - 58849.0424674074 days
 Energy interval ...: 0.03 - 150 TeV
 Region of interest ...: RA=83.5, DEC=22.8 [0,0] Radius=5 deg
=== GSkyRegions ===
 Number of regions ...: 0
```

There is only one CTA observation in the container. The observation contains a CTA event list. We can therefore inspect some of the event properties, for example look at their energy spectrum.

```
In [5]: events = sim.obs()[0].events()

        ax = plt.subplot()
        ax.set_yscale('log')
        ax.set_xlabel('Log10(Energy [TeV])')

        energies = []
        for event in events:
            energies.append(event.energy().log10TeV())

        n, bins, patches = plt.hist(energies)
```



1.2 Selecting event data

As next step you may want to select some events from the simulated event data. For example, let's say that you want to restrict the events to a region of interest (ROI) of 3 degrees around the pointing direction, you want to extract a 30 minutes time slice from the data, and you want to limit the energy range of the events to 0.1 - 100 TeV.

```
In [6]: sel = ctools.ctselect()
        sel['inobs'] = 'events.fits'
        sel['rad'] = 3.
        sel['tmin'] = '2020-01-01T00:00:00'
        sel['tmax'] = '2020-01-01T00:30:00'
        sel['emin'] = 0.1
        sel['emax'] = 100.
        sel.run()
```

If you want to skip selection in a given dimension you can pass 'INDEF' for one/several selection parameters. Otherwise, we can verify that the cuts reduced largely the number of events.

```
In [7]: print(sel.obs())
        print(sel.obs()[0])
```

```
=== GObservations ===
 Number of observations ...: 1
 Number of models ...: 0
 Number of observed events .: 143635
 Number of predicted events : 0
=== GCTAObservation ===
 Name ...:
 Identifier ...: 000001
 Instrument ...: CTA
 Event file ...: events.fits
 Event type ...: EventList
 Statistic ...: cstat
 Ontime ...: 1800.00000017881 s
 Livetime ...: 1764.00000017324 s
 Deadtime correction ...: 0.979999999998889
 User energy range ...: undefined
=== GCTAPointing ===
 Pointing direction ...: (RA,Dec)=(83.5,22.8)
 Response function ...: undefined
=== GCTAEventList ===
 Number of events ...: 143635 (loaded)
 Time interval ...: 58849.0008007407 - 58849.0216340741 days
 Energy interval ...: 0.1 - 100 TeV
 Region of interest ...: RA=83.5, DEC=22.8 [0,0] Radius=3 deg
=== GSkyRegions ===
 Number of regions ...: 0
```

Note that we have loaded the events from disk. From Python you can also get the events by passing to the tool constructor the observation container from a previous tool.

```
In [8]: sel = ctools.ctselect(sim.obs())
        sel['rad'] = 3.
        sel['tmin'] = '2020-01-01T00:00:00'
        sel['tmax'] = '2020-01-01T00:30:00'
        sel['emin'] = 0.1
        sel['emax'] = 100.
        sel.run()
```

Note that observation containers can have models attached. In this case we carry the information of the model used for the simulation.

```
In [9]: print(sel.obs().models())
```

```
=== GModels ===
 Number of models ...: 2
 Number of parameters ...: 10
=== GModelSky ===
 Name ...: Crab
 Instruments ...: all
 Observation identifiers ...: all
 Model type ...: PointSource
 Model components ...: "PointSource" * "PowerLaw" * "Constant"
 Number of parameters ...: 6
 Number of spatial par's ...: 2
  RA ...: 83.6331 [-360,360] deg (fixed,scale=1)
  DEC ...: 22.0145 [-90,90] deg (fixed,scale=1)
 Number of spectral par's ..: 3
  Prefactor ...: 5.7e-16 +/- 0 [1e-23,1e-13] ph/cm2/s/MeV (free,scale=1e-16,gradient)
  Index ...: -2.48 +/- 0 [-0,-5]  (free,scale=-1,gradient)
  PivotEnergy ...: 300000 [10000,1000000000] MeV (fixed,scale=1000000,gradient)
 Number of temporal par's ..: 1
  Normalization ...: 1 (relative value) (fixed,scale=1,gradient)
 Number of scale par's ...: 0
=== GCTAModelIrfBackground ===
 Name ...: CTABackgroundModel
 Instruments ...: CTA
 Observation identifiers ...: all
 Model type ...: "PowerLaw" * "Constant"
 Number of parameters ...: 4
 Number of spectral par's ..: 3
  Prefactor ...: 1 +/- 0 [0.001,1000] ph/cm2/s/MeV (free,scale=1,gradient)
  Index ...: 0 +/- 0 [-5,5]  (free,scale=1,gradient)
  PivotEnergy ...: 1000000 [10000,1000000000] MeV (fixed,scale=1000000,gradient)
 Number of temporal par's ..: 1
  Normalization ...: 1 (relative value) (fixed,scale=1,gradient)
```

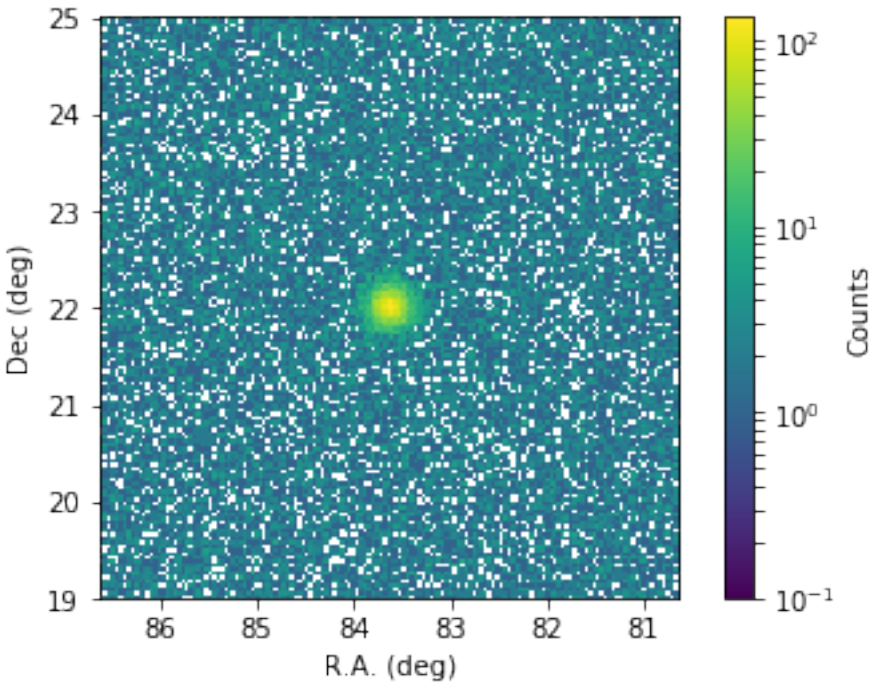
1.3 Generating a skymap

It's ofter useful to look at your data as sky map. The skymap is generated via the ctskymap tool.

```
In [10]: skymap = ctools.ctskymap(sel.obs())
         skymap['coordsys'] = 'CEL'
         skymap['proj'] = 'CAR'
         skymap['xref'] = 83.63
         skymap['yref'] = 22.01
         skymap['binsz'] = 0.02
         skymap['nxpix'] = 150
         skymap['nypix'] = 150
         skymap['emin'] = 0.1
         skymap['emax'] = 100.
         skymap['bkgsubtract'] = 'NONE'
         skymap.run()
```

We can now inspect the event map.

```
In [11]: # Slightly smooth the map for display to suppress statistical fluctuations
         #skymap.skymap().smooth('GAUSSIAN',0.02)

         from matplotlib.colors import LogNorm

         ax = plt.subplot()
         plt.imshow(skymap.skymap().array(),origin='lower',
                    extent=[83.63+0.02*150,83.63-0.02*150,22.01-0.02*150,22.01+0.02*150],
                    # boundaries of the coord grid
                    norm=LogNorm(vmin=0.1))
         ax.set_xlabel('R.A. (deg)')
         ax.set_ylabel('Dec (deg)')
         cbar = plt.colorbar()
         cbar.set_label('Counts')
```



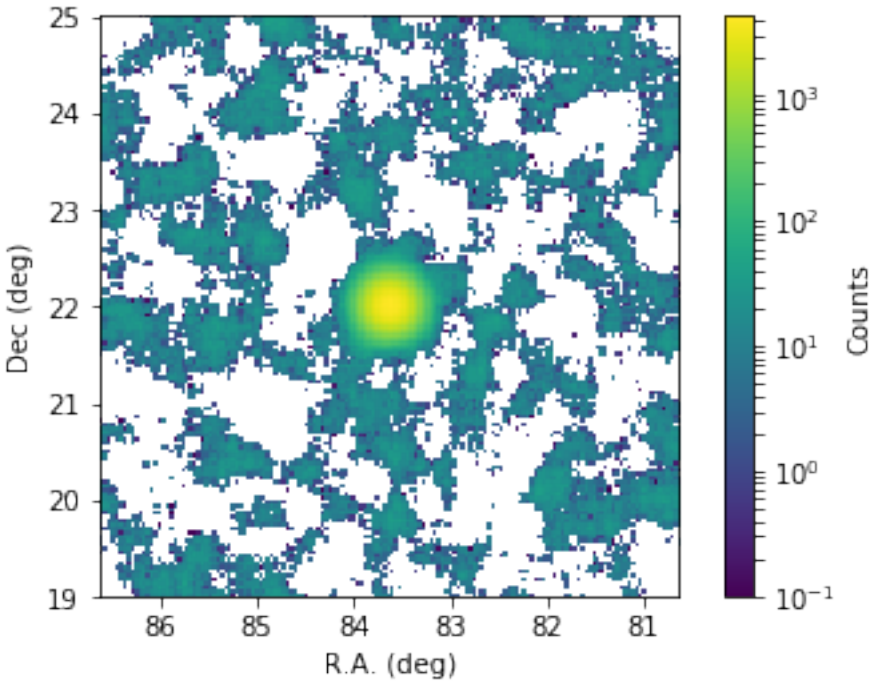
You can see that there is a background that is appreciable even compared to a bright source as the Crab nebula. We can produce a background-subtracted skymap. For this there are several options. We may use the background model in the IRF. We will test here the RING method, which is a classical method in which the Off/background region has the shape of a ring.

```
In [13]: skymap = ctools.ctskymap(sel.obs())
         skymap['coordsys'] = 'CEL'
         skymap['proj'] = 'CAR'
         skymap['xref'] = 83.63
         skymap['yref'] = 22.01
         skymap['binsz'] = 0.02
         skymap['nxpix'] = 150
         skymap['nypix'] = 150
         skymap['emin'] = 0.1
         skymap['emax'] = 100.
         skymap['bkgsubtract'] = 'RING'
         skymap['roiradius'] = 0.1 #On/source region radius
         skymap['inradius'] = 0.3 #Off/background ring inner radius
         skymap['outradius'] = 0.5 #Off/background ring outer radius
         skymap['iterations'] = 3
         skymap.run()
```

Note that to be sure to exclude regions with significant gamma-ray emission from the Off/background estimation regions we have used an iterative procedure with 3 iterations.

```
In [14]: ax = plt.subplot()
         plt.imshow(skymap.skymap().array(),origin='lower',
```

```
                extent=[83.63+0.02*150,83.63-0.02*150,22.01-0.02*150,22.01+0.02*150],
                # boundaries of the coord grid
                norm=LogNorm(vmin=0.1))
ax.set_xlabel('R.A. (deg)')
ax.set_ylabel('Dec (deg)')
cbar = plt.colorbar()
cbar.set_label('Counts')
```



As expected the background has been suppressed. Note that in this case the image in intrinsically smoothed to the scale of the On region chosen.

1.4 Unbinned analysis

An unbinned 3D analysis is the simplest option, since we just need to pass the simulated/selected event list to the tool ctlike. Note that our event list has already a sky model attached that we can therefore use for the fit. You can reset the model or change it before passing the observations to ctlike if you need to. If you run ctlike from the command line you will need to pass a model independently. Note that we ignore energy dispersion in the fit for the sake of computing speed.

```
In [15]: like = ctools.ctlike(sel.obs())
         like.run()
```

We can check the status of the likelihood optimisation.

```
In [16]: print(like.opt())
```

```
=== GOptimizerLM ===
 Optimized function value ..: 719960.296
 Absolute precision ...: 0.005
 Acceptable value decrease .: 2
 Optimization status ...: converged
 Number of parameters ...: 10
 Number of free parameters .: 4
 Number of iterations ...: 2
 Lambda ...: 1e-05
```

Let's look at the fitted model.

```
In [17]: print(like.obs().models())
```

```
=== GModels ===
 Number of models ...: 2
 Number of parameters ...: 10
=== GModelSky ===
 Name ...: Crab
 Instruments ...: all
 Observation identifiers ...: all
 Model type ...: PointSource
 Model components ...: "PointSource" * "PowerLaw" * "Constant"
 Number of parameters ...: 6
 Number of spatial par's ...: 2
  RA ...: 83.6331 [-360,360] deg (fixed,scale=1)
  DEC ...: 22.0145 [-90,90] deg (fixed,scale=1)
 Number of spectral par's ..: 3
  Prefactor ...: 5.78700392009128e-16 +/- 8.19496046111996e-18 [1e-23,1e-13] ph/cm2/s/MeV (free
  Index ...: -2.47440817790876 +/- 0.0122563594809813 [-0,-5]  (free,scale=-1,gradient)
  PivotEnergy ...: 300000 [10000,1000000000] MeV (fixed,scale=1000000,gradient)
 Number of temporal par's ..: 1
  Normalization ...: 1 (relative value) (fixed,scale=1,gradient)
 Number of scale par's ...: 0
=== GCTAModelIrfBackground ===
 Name ...: CTABackgroundModel
 Instruments ...: CTA
 Observation identifiers ...: all
 Model type ...: "PowerLaw" * "Constant"
 Number of parameters ...: 4
 Number of spectral par's ..: 3
  Prefactor ...: 1.00010974533329 +/- 0.00517346502054174 [0.001,1000] ph/cm2/s/MeV (free,scale
  Index ...: 0.000736147285176287 +/- 0.00307607601825231 [-5,5]  (free,scale=1,gradient)
  PivotEnergy ...: 1000000 [10000,1000000000] MeV (fixed,scale=1000000,gradient)
 Number of temporal par's ..: 1
  Normalization ...: 1 (relative value) (fixed,scale=1,gradient)
```

As you can see al the free parameters are compatible with the true Mote Carlo values. Note that we have adjusted also the background model by means of a power-law correction.

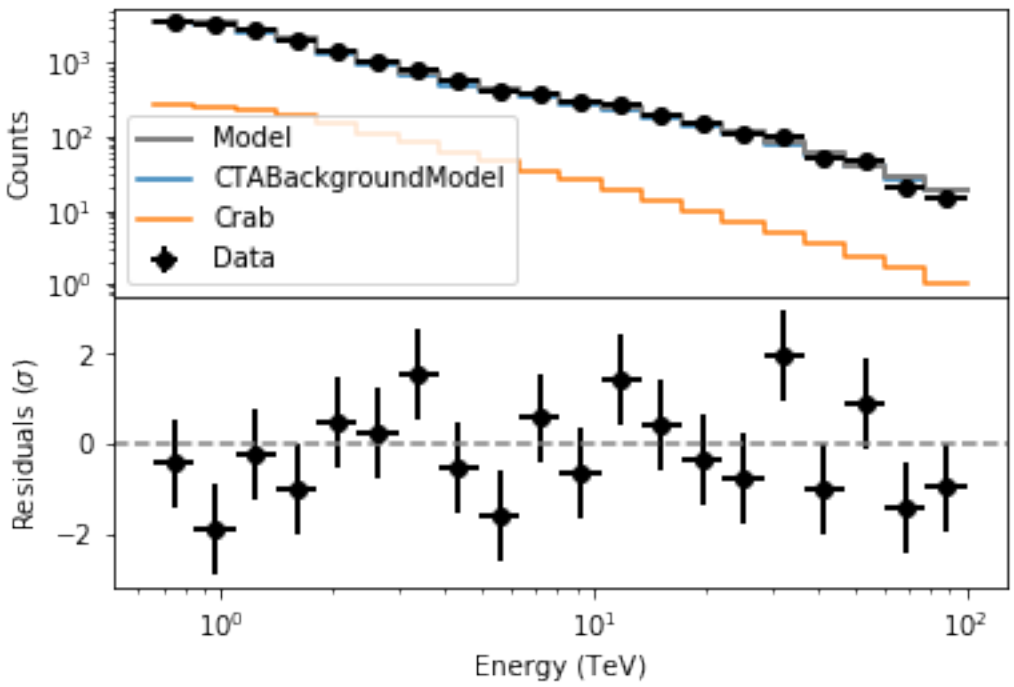
1.5 Inspecting the fit residuals

After having done a maximum likelihood fit it is good practice to inspect the fit residuals. You can inspect the spectral residuals, and, if you have run a 3D fit, also the spatial residuals. Let's start with the spectral residuals, for which we use the csresspec tool.

```
In [18]: resspec = cscripts.csresspec(like.obs())
         resspec['algorithm'] = 'SIGNIFICANCE'
         resspec['mask'] = False
         resspec['components'] = True
         resspec['outfile'] = 'resid_spectrum.fits'
         resspec.execute()
```

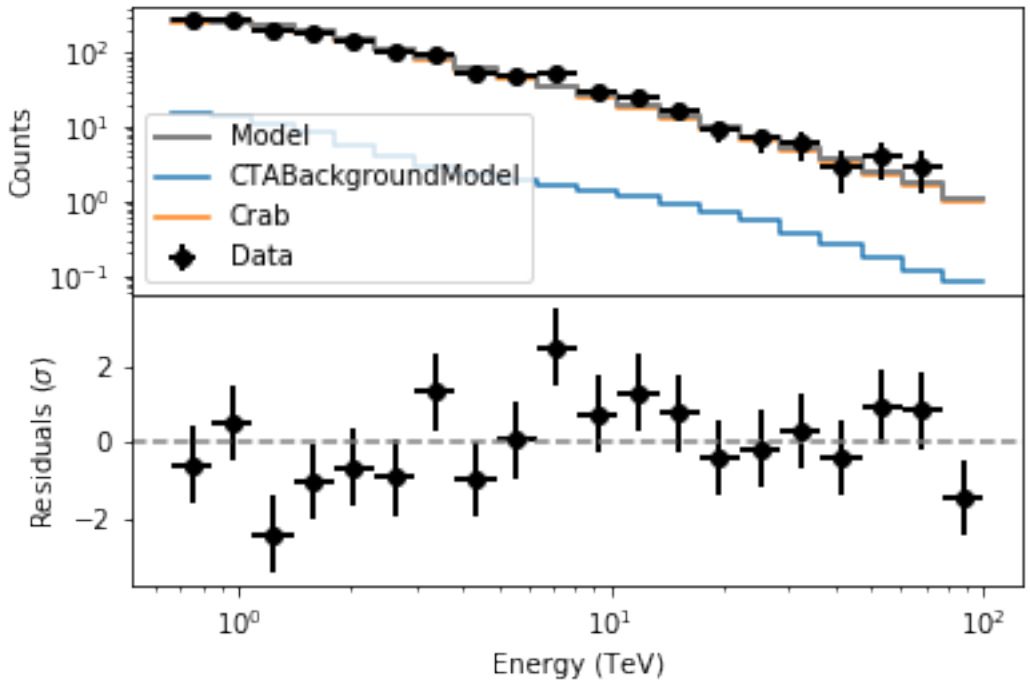
We have chosen to compute the residuals as number of Gaussian sigma's based on the calculation of significance from the likelihood ratio test for Poisson statistics. We have set the 'mask' parameter to False, that is, we are integrating the residuals over the whole area covered by the events. We have also set the 'components' parameter to True to that we will see the contribution of the different components to the model. We can visualize the residuals using an example script provided with the ctools installation

```
In [19]: import sys
         import os
         sys.path.append(os.environ['CTOOLS']+'/share/examples/python/')

         from show_residuals import plot_residuals
         plot_residuals('resid_spectrum.fits','',0)
```



Over the whole field of view the data are largely dominated by background events. What we care mostly about, however, is how good our model is in the region of the source. We will therefore use the mask to only look at the residuals in a 0.2 deg region around the Crab nebula.

```
In [20]: resspec = cscripts.csresspec(like.obs())
         resspec['algorithm'] = 'SIGNIFICANCE'
         resspec['mask'] = True
         resspec['ra'] = 83.63
         resspec['dec'] = 22.01
         resspec['rad'] = 0.2
         resspec['components'] = True
         resspec['outfile'] = 'resid_0p2deg_spectrum.fits'
         resspec.execute()

         plot_residuals('resid_0p2deg_spectrum.fits','',0)
```



You can appreciate that the models provides a good spectral description of the data both around the source and over the entire field of view.

Next we will look at the spatial residuals by creating a map of residuals using the csresmap tool.

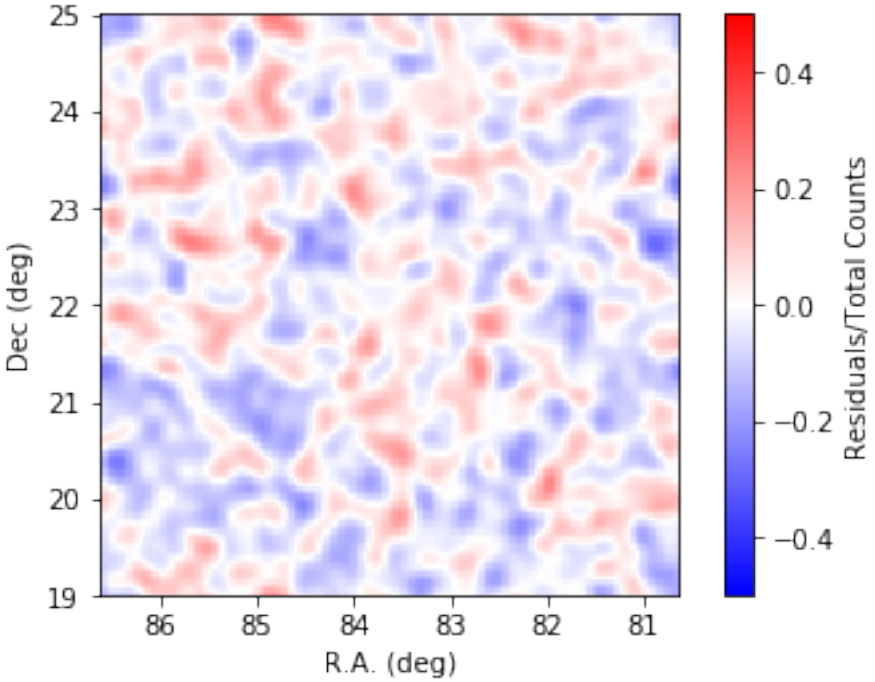
```
In [21]: resmap = cscripts.csresmap(like.obs())
         resmap['algorithm'] = 'SUBDIV'
```

```
resmap['coordsys'] = 'CEL'
resmap['proj'] = 'CAR'
resmap['xref'] = 83.63
resmap['yref'] = 22.01
resmap['binsz'] = 0.02
resmap['nxpix'] = 150
resmap['nypix'] = 150
resmap['emin'] = 0.1
resmap['emax'] = 100.
resmap.run()
```

Here we have chosen the SUBDIV algorithm, that is, we will be looking as residuals as a fraction of the total events number.

We can now inspect the residual map, again with a slight smoothing to suppress statistical fluctuations.

```
In [22]: resmap._resmap.smooth('GAUSSIAN',0.05)
         ax  = plt.subplot()
         plt.imshow(resmap._resmap.array(),origin='lower',
                    cmap='bwr',vmin=-0.5,vmax=0.5,
                    extent=[83.63+0.02*150,83.63-0.02*150,22.01-0.02*150,22.01+0.02*150])
                    # Boundaries of the coord grid
         ax.set_xlabel('R.A. (deg)')
         ax.set_ylabel('Dec (deg)')
         cbar = plt.colorbar()
         cbar.set_label('Residuals/Total Counts')
```



Also the spatial residuals are flat over the field of view.

1.6 On/Off spectral analysis

In classical IACT analyses the background is normally derived from the data, by defining On (source) and Off (background) regions. An On/Off analysis is recommended if you want to assure minimal dependency on the Monte Carlo background model.

For an On/Off spectral analysis you need first to identify the On and Off regions, and to build binned count spectra for them. This is accomplished via the csphagen tool (the count spectra are stored in a format from X-ray analysis called PHA).

By default, csphagen calculates the background counts using the REFLECTED algorithm, in which, for each individual observation the background regions have the same shape as the source region, and are rotated around the center of the camera keeping the same offset. As many reflected regions as possible are used, excluding the area of the camera near the source position. Since the background rates are expected to be approximately radially symmetric in camera coordinates, this method minimizes the impact of the background rate modeling from Monte Carlo. An optional exclusion map (in FITS WCS format) can be provided as input through the hidden inexclusion parameter if other regions of significant gamma-ray emission ought to be excluded from the background computation.

```
In [23]: phagen  = cscripts.csphagen(sel.obs())
         phagen['ebinalg'] = 'LOG'
         phagen['emin'] = 0.1
         phagen['emax'] = 100.0
         phagen['enumbins'] = 30
         phagen['coordsys'] = 'CEL'
         phagen['srcname'] = 'Crab'
         phagen['ra']            = 83.63
         phagen['dec']           = 22.01
         phagen['rad']           = 0.2
         phagen['bkgmethod']     = 'REFLECTED'
         phagen['use_model_bkg'] = False
         phagen.run()
```

Note that we have used the 'use_model_bkg' parameter to ignore the input background model and derive all background information from the data (assuming that the background event rate per solid angle unit as a function fo energy is the same in the On and Off regions). On the other hand we have specified the source name, Crab, to ensure to properly account for the PSF in the evaluation of the On/Off response. If you set this parameter to NONE a pointlike source at the center of the On region will be considered.

Let's peek at the resulting observations and models.

```
In [24]: print(phagen.obs()[0])
         print(phagen.obs().models())
```

```
=== GCTAOnOffObservation ===
 Name ...:
 Identifier ...: 000001
 Instrument ...: CTAOnOff
```

```
 Statistic ...: wstat
 Ontime ...: 1800.00000017881 s
 Livetime ...: 1764.00000017324 s
 Deadtime correction ...: 0.979999999998889
=== GPha ===
 Exposure ...: 1764.00000017324 s
 Number of bins ...: 30
 Energy range ...: 100 GeV - 100 TeV
 Observation energy range ..: 100 GeV - 100 TeV
 Total number of counts ...: 6390
 Underflow counts ...: 0
 Overflow counts ...: 0
 Outflow counts ...: 0
=== GPha ===
 Exposure ...: 1764.00000017324 s
 Number of bins ...: 30
 Energy range ...: 100 GeV - 100 TeV
 Observation energy range ..: 100 GeV - 100 TeV
 Total number of counts ...: 5360
 Underflow counts ...: 0
 Overflow counts ...: 0
 Outflow counts ...: 0
=== GArf ===
 Number of bins ...: 101
 Energy range ...: 50 GeV - 120 TeV
=== GRmf ===
 Number of true energy bins : 101
 Number of measured bins ...: 30
 True energy range ...: 50 GeV - 120 TeV
 Measured energy range ...: 100 GeV - 100 TeV
=== GModels ===
 Number of models ...: 1
 Number of parameters ...: 6
=== GModelSky ===
 Name ...: Crab
 Instruments ...: all
 Observation identifiers ...: all
 Model type ...: PointSource
 Model components ...: "PointSource" * "PowerLaw" * "Constant"
 Number of parameters ...: 6
 Number of spatial par's ...: 2
  RA ...: 83.6331 [-360,360] deg (fixed,scale=1)
  DEC ...: 22.0145 [-90,90] deg (fixed,scale=1)
 Number of spectral par's ..: 3
  Prefactor ...: 5.7e-16 +/- 0 [1e-23,1e-13] ph/cm2/s/MeV (free,scale=1e-16,gradient)
  Index ...: -2.48 +/- 0 [-0,-5]  (free,scale=-1,gradient)
  PivotEnergy ...: 300000 [10000,1000000000] MeV (fixed,scale=1000000,gradient)
 Number of temporal par's ..: 1
```

```
  Normalization ...: 1 (relative value) (fixed,scale=1,gradient)
 Number of scale par's ...: 0
```

The observation entails four objects, two GPha objects for the On and Off spectra, respectively with ancillary information, notably the scaling factor to be applied to the background spectrum and the boundaries of the nergy bins. The GArf object contains the spectral response of the instrument extracted from the instrument response functions, including effective area for gamma-ray detection and background rates. The GRmf object contains the remaining part of the instrument response, i.e., an energy redistribution matrix. Note that we are performing a 1D analysis: the effect of the PSF is already folded into the spectral response computation taking into account the morphology of the source as specified in the input model. Note that the fit statistic is set to wstat, that is, we are not using a background model and the background rates will be treated in a fit as nuisance parameters.

We are now ready to run the On/Off fit. For this it's sufficient to pass to ctlike the On/Off observations container.

```
In [25]: like = ctools.ctlike(phagen.obs())
         like.run()
```

And we can look at the results.

```
In [26]: print(like.opt())
         print(like.obs().models())
```

```
=== GOptimizerLM ===
 Optimized function value ..: 17.318
 Absolute precision ...: 0.005
 Acceptable value decrease .: 2
 Optimization status ...: converged
 Number of parameters ...: 6
 Number of free parameters .: 2
 Number of iterations ...: 2
 Lambda ...: 1e-05
=== GModels ===
 Number of models ...: 1
 Number of parameters ...: 6
=== GModelSky ===
 Name ...: Crab
 Instruments ...: all
 Observation identifiers ...: all
 Model type ...: PointSource
 Model components ...: "PointSource" * "PowerLaw" * "Constant"
 Number of parameters ...: 6
 Number of spatial par's ...: 2
  RA ...: 83.6331 [-360,360] deg (fixed,scale=1)
  DEC ...: 22.0145 [-90,90] deg (fixed,scale=1)
 Number of spectral par's ..: 3
  Prefactor ...: 5.65475655212032e-16 +/- 8.13522205014716e-18 [1e-23,1e-13] ph/cm2/s/MeV (free
```

```
 Index ...: -2.47165180054983 +/- 0.0124181735856028 [-0,-5]  (free,scale=-1,gradient)
 PivotEnergy ...: 300000 [10000,1000000000] MeV (fixed,scale=1000000,gradient)
Number of temporal par's ..: 1
 Normalization ...: 1 (relative value) (fixed,scale=1,gradient)
Number of scale par's ...: 0
```

The source model spectral parameters are compatible with the true Monte Carlo values and the results from the unbinned 3D analysis.