GammaLib and ctools

Jürgen Knödlseder, M. Mayer, C. Deil, J.-B. Cayrou, E. Owen, N. Kelley-Hoskins, C.-C. Lu, R. Buehler, F. Forest, T. Louge, et al.

A&A 593, A1 (2016)
DOI: 10.1051/0004-6361/201628822

Astronomy & Astrophysics

GammaLib and ctools

A software framework for the analysis of astronomical gamma-ray data

J. Knödlseder[1], M. Mayer[2], C. Deil[3], J.-B. Cayrou[1], E. Owen[3], N. Kelley-Hoskins[4], C.-C. Lu[3], R. Buehler[4], F. Forest[1], T. Louge[1], H. Siejkowski[5], K. Kosack[6], L. Gerard[4], A. Schulz[4], P. Martin[1], D. Sanchez[7], S. Ohm[4], T. Hassan[8], and S. Brau-Nogué[1]

[1] Institut de Recherche en Astrophysique et Planétologie, 9 avenue Colonel-Roche, 31028 Toulouse Cedex 4, France
[2] Department of Physics, Humboldt University Berlin, Newtonstr. 15, 12489 Berlin, Germany
[3] Max-Planck-Institut für Kernphysik, Saupfercheckweg 1, 69117 Heidelberg, Germany
[4] Deutsches Elektronen-Synchrotron, Platanenallee 6, 15738 Zeuthen, Germany
[5] AGH University of Science and Technology, ACC Cyfronet AGH, ul. Nawojki 11, PO Box 386, 30-950 Kraków 23, Poland
[6] CEA/IRFU/SAp, CEA Saclay, Bat 709, Orme des Merisiers, 91191 Gif-sur-Yvette, France
[7] Laboratoire d'Annecy-le-Vieux de Physique des Particules, 9 chemin de Bellevue, BP 110, 74941 Annecy-le-Vieux Cedex, France
[8] Institut de Fisica d'Altes Energies, The Barcelona Institute of Science and Technology, Campus UAB, 08193 Bellaterra, Spain

Received 29 April 2016 / Accepted 1 June 2016

ABSTRACT

The field of gamma-ray astronomy has seen important progress during the last decade, yet to date no common software framework has been developed for the scientific analysis of gamma-ray telescope data. We propose to fill this gap by means of the GammaLib software, a generic library that we have developed to support the analysis of gamma-ray event data. GammaLib was written in C++ and all functionality is available in Python through an extension module. Based on this framework we have developed the ctools software package, a suite of software tools that enables flexible workflows to be built for the analysis of Imaging Air Cherenkov Telescope event data. The ctools are inspired by science analysis software available for existing high-energy astronomy instruments, and they follow the modular ftools model developed by the High Energy Astrophysics Science Archive Research Center. The ctools were written in Python and C++, and can be either used from the command line via shell scripts or directly from Python. In this paper we present the GammaLib and ctools software versions 1.0 that were released at the end of 2015. GammaLib and ctools are ready for the science analysis of Imaging Air Cherenkov Telescope event data, and also support the analysis of *Fermi*-LAT data and the exploitation of the COMPTEL legacy data archive. We propose using ctools as the science tools software for the Cherenkov Telescope Array Observatory.

Key words. methods: data analysis – virtual observatory tools

1. Introduction

The last decade has seen important progress in the field of gamma-ray astronomy thanks to significant improvements in the performance of ground-based and space-based gamma-ray telescopes. Gamma-ray photons are currently studied over more than eight decades in energy, from a few 100 keV up to more than 10 TeV. The technologies used for observing gamma rays are very diverse and cover indirect imaging devices, such as coded mask and Compton telescopes, and direct imaging devices, such as pair creation telescopes, and air or water Cherenkov telescopes.

Despite this technical diversity, the high-level data that is produced by the instruments for scientific analysis have great similarities. Generally, the data is comprised of lists of individual events, each of which is characterised by temporal, directional and energy information. Many space-based missions (e.g. CGRO, INTEGRAL, *Fermi*) provide event data using the Flexible Image Transport System (FITS) data format (Pence et al. 2010) and follow the Office of Guest Investigator Programs (OGIP) conventions (Corcoran et al. 1995), and efforts are also underway to convert data from existing ground-based observatories (H.E.S.S., VERITAS, MAGIC, HAWC) into the same framework[1]. The next-generation ground-based Cherenkov Telescope Array (CTA) Observatory, a world-wide project to create a large and sustainable Imaging Air Cherenkov Telescope (IACT) observatory (Acharya et al. 2013), will also follow that path. The CTA will provide all-sky coverage by implementing two IACT arrays, one in the northern hemisphere that will be located in La Palma, and one in the southern hemisphere that will be located in Chile, equipped in total with more than a hundred IACTs of three different size classes to cover the energy range 20 GeV−300 TeV. The CTA Observatory will distribute the high-level event data in form of FITS files to the astronomical community (Knödlseder et al. 2015).

Despite the standardisation in the data formats, there are no common tools yet for the scientific analysis of gamma-ray data. So far, each instrument comes with its suite of software tools, which often requires costly development cycles and maintenance efforts, and which puts the burden on the astronomer to learn how to use each of them for scientific research. For X-ray astronomy, the High Energy Astrophysics Science Archive Research Center (HEASARC) has developed standards (such as XSELECT or XSPEC) that unify the data analysis tasks, making

[1] http://gamma-astro-data-formats.readthedocs.org

X-ray data more accessible to the astronomical community at large.

We aim to follow a similar path for gamma-ray astronomy, and therefore developed the GammaLib software package as a general framework for the analysis of gamma-ray event data (Knödlseder et al. 2011). On top of this framework we developed the ctools software package, a suite of software tools that enables building flexible workflows for the analysis of IACT event data (Knödlseder et al. 2016). The development of the ctools and GammaLib software packages has recently been driven by the needs for CTA, but the interest in these packages is also growing within the collaborations of existing IACTs. We therefore want to provide a reference publication for both software packages that makes the community at large aware of their existence, but that also describes in some detail their content.

Both software packages are developed as open source codes[2] under the GNU General Public License version 3. GammaLib and ctools are version controlled using the Git system that is managed through a GitLab front end[3]. The code development is managed using Redmine[4] and we use a Jenkins-based multi-platform continued integration system to assure the integrity and the functionality of the software[5]. Code quality is controlled using SonarQube[6]. GammaLib and ctools are built and tested on a large variety of Linux distributions, on Mac OS X (10.6−10.11), on FreeBSD and on OpenSolaris. Usage on Windows is not supported.

GammaLib is mostly written in C++ and provides Application Programming Interfaces (APIs) for C++ and Python. GammaLib is an object oriented library that comprises over 200 classes for a total volume of nearly 120 000 lines of code. The ctools package is written in Python and C++ and comprises nearly 30 analysis tools that make up almost 20 000 lines of code. The interface between C++ code and Python is generated for both packages using the Simplified Wrapper and Interface Generator (SWIG)[7]. Both Python 2 (from version 2.3 on) and Python 3 are supported.

This paper describes GammaLib and ctools release version 1.0. Sections 2 and 3 present the GammaLib and ctools software packages, respectively. Section 4 summarises the performance of the software. We conclude with an outlook on future developments in Sect. 5.

2. GammaLib

2.1. Overview

GammaLib is a single shared library that contains all C++ classes, support functions, and some global variables that are needed to analyse gamma-ray event data. A central feature of GammaLib is that all functionalities that are necessary for the analysis of gamma-ray event data are implemented natively, reducing thus external dependencies to a strict minimum. This is essential to keep the long-term software maintenance cost down, and helps in assuring independence from operating systems and user-friendly installation. The price to pay for this feature was a larger initial development effort that we had to invest at the beginning of the project. The only external library GammaLib relies on is HEASARC's cfitsio library[8] that is however available as binary package on all modern Linux and Mac OS X systems. An optional dependency is the readline and ncurses libraries that enhance the Image Reduction and Analysis Facility (IRAF) command line parameter interface that is implemented as a user interface, but GammaLib is also fully functional if these libraries are not available.

All GammaLib classes start with an capital "G", followed by a capitalised class name. If the class name is composed of several words, each word is capitalised (CamelCase). Examples of GammaLib class names are `GModels`, `GSkyMap` or `GFitsBinTable`. Names of classes that contain lists of objects (container classes) are generally formed by appending an "s" to the class name of the objects they contain. For example, `GModels` is a container class for `GModel` objects. GammaLib functions or global variables are defined within the `gammalib` namespace. For example, `gammalib::expand_env()` is a function that expands the environment variables in a string, or `gammalib::MeV2erg` is a multiplier that converts energies from units of MeV to units of ergs.

2.2. Software layout

The GammaLib classes are organised into three software layers, each of which comprises several modules (see Fig. 1). The top layer provides support for instrument-independent high-level data analysis, comprising the handling of observations, models, and sky maps. Also the support for creating ftool applications is part of this layer. Core services related to numerical computations and function optimisation are implemented in the second layer. The third layer is an interface layer that allows handling of data in FITS, XSPEC and XML formats, and that implements support for Virtual Observatory (VO) interoperability.

2.2.1. Observation module

The observation module contains the abstract `GObservation` base class that defines the interface for a generic gamma-ray observation. In GammaLib, an observation is defined as a period in time during which an instrument was taking data, in a stable configuration that can be described by a fixed instrument response function (the time period does not need to be contiguous). The data is represented by events that may either be provided in a list or that are binned in a n-dimensional data cube.

Each event or event bin is characterised by three fundamental properties: instrument direction $\boldsymbol{p}'$, measured energy E', and trigger time t' (we use in the following primed symbols to denote reconstructed or measured quantities, and unprimed symbols to denote true quantities). We note that the instrument direction is not necessarily given in sky coordinates, but could be for instance the detector number or any other instrument related property that characterises the arrival direction of an event. While energy and time are handled in a unit or reference independent way, it should be noted that GammaLib stores energies internally in MeV and defines the time zero at 1st January 2010, 00:00:00 (TT).

The observation module contains also the abstract `GResponse` base class that represents the instrument response function $R(\boldsymbol{p}', E', t'|\boldsymbol{p}, E, t)$ that describes the transformation

[2] The source code can be downloaded from `http://cta.irap.omp.eu/ctools/download.html` which also provides access to Mac OS X binary packages. Up to date user documentation for both packages can be found at `http://cta.irap.omp.eu/gammalib` and `http://cta.irap.omp.eu/ctools`

[3] `https://cta-gitlab.irap.omp.eu`

[4] `https://cta-redmine.irap.omp.eu`

[5] `https://cta-jenkins.irap.omp.eu`

[6] `https://cta-sonar.irap.omp.eu`

[7] `http://www.swig.org`

[8] `http://heasarc.gsfc.nasa.gov/fitsio/`

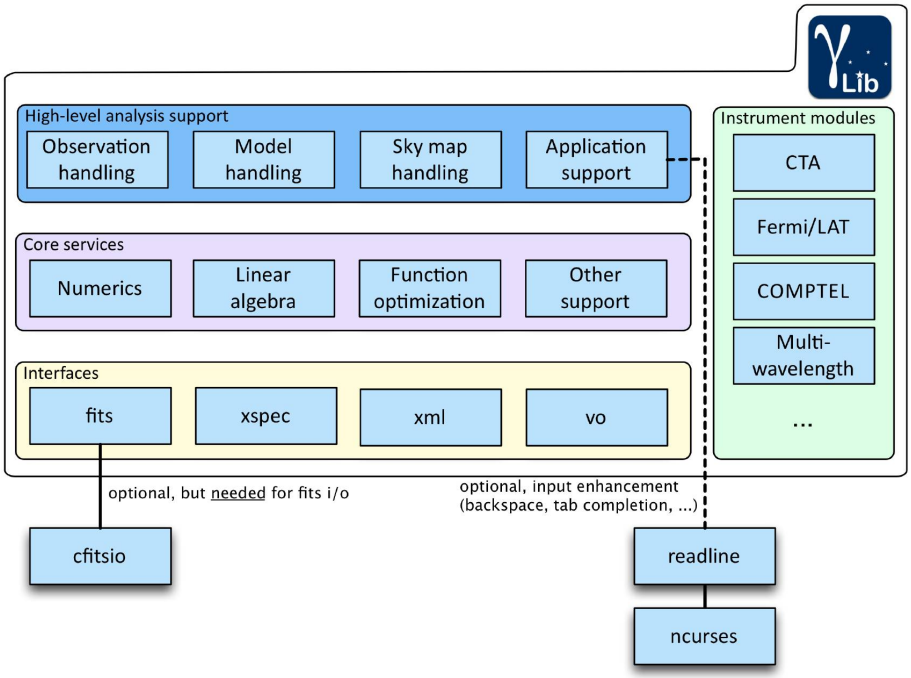

Fig. 1. Organisation scheme of the GammaLib library (see text for a description of the entities shown).

from the physical properties of a photon (sky direction $\boldsymbol{p}$, energy E, and time t) to the measured characteristics of an event (the instrument response function is given in units of cm^2 sr^{-1} s^{-1} MeV^{-1}). The instrument response function and the events are the basic constitutents of a GammaLib observation, which implies that every observation can hold its proper (and independent) instrument response function. We will use in the following the index i to indicate that a function applies to a specific observation.

A fundamental function for each observation is the likelihood function $L_i(M)$ that quantifies the probability that the data collected during a given observation is drawn from a particular model M (see Sect. 2.2.2 for a description of the model handling in GammaLib). The formulae used for the likelihood computation depend on the type of the data (binned or unbinned) and the assumed underlying statistical law. For event lists, the Poisson formula

$$-\ln L_i(M) = e_i(M) - \sum_k \ln P_i(\boldsymbol{p}'_k, E'_k, t'_k|M) \quad (1)$$

is used, where the sum is taken over all events k, characterised by the instrument direction $\boldsymbol{p}'_k$, the measured energy E'_k and the trigger time t'_k. $P_i(\boldsymbol{p}', E', t'|M)$ is the probability density that given the model M, an event with instrument direction $\boldsymbol{p}'$, measured energy E' and trigger time t' occurs. $e_i(M)$ is the total number of events that are predicted to occur during an observation given the model M, computed by integrating the probability density over the trigger time, measured energy and instrument direction:

$$e_i(M) = \int_{\mathrm{GTI}} \int_{\mathrm{Ebounds}} \int_{\mathrm{ROI}} P_i(\boldsymbol{p}', E', t'|M)\, \mathrm{d}\boldsymbol{p}'\, \mathrm{d}E'\, \mathrm{d}t'. \quad (2)$$

The temporal integration boundaries are defined by so-called good time intervals (GTIs) that define contiguous periods in time during which data was taken. The spatial integration boundaries are defined by a so-called region of interest (ROI).

For binned data following a Poisson distribution the formula

$$-\ln L_i(M) = \sum_k e_{k,i}(M) - n_{k,i} \ln e_{k,i}(M) \quad (3)$$

is used, where the sum over k is now taken over all data cube bins. $n_{k,i}$ is the number of events in bin k observed during observation i, and

$$e_{k,i}(M) = P_i(\boldsymbol{p}'_k, E'_k, t'_k|M) \times \Omega_k \times \Delta E_k \times \Delta T_k \quad (4)$$

is the predicted number of events from model M in bin k of observation i. The probability density is evaluated for the reference instrument direction $\boldsymbol{p}'_k$, measured energy E'_k and trigger time t'_k of bin k, typically taken to be the values at the bin centre, and multiplied by the solid angle Ω_k, the energy width ΔE_k and the exposure time (or ontime) ΔT_k of bin k[9]. Alternatively, if the data follow a Gaussian distribution the formula

$$-\ln L_i(M) = \frac{1}{2} \sum_k \left(\frac{n_{k,i} - e_{k,i}(M)}{\sigma_{k,i}} \right)^2 \quad (5)$$

is used, where $\sigma_{k,i}$ is the statistical uncertainty in the measured number of events for bin k.

Observations are collected in the `GObservations` container class, which is the central object that is manipulated in a GammaLib data analysis. By summing for a given model M over the negative log-likelihood values of all observations in the container using

$$-\ln L(M) = -\sum_i \ln L_i(M), \quad (6)$$

the joint maximum likelihood is computed, enabling the combination of an arbitrary number of observations to constrain the parameters of a model M. This opens the possibility of performing

[9] Any dead time correction is taken into account by the instrument response function.

multi-instrument and multi-wavelength analyses of event data with GammaLib by combining observations performed by different instruments in a single observation container. If OpenMP support is available, Eq. (6) will be parallelised and hence benefits from the availability of multi-core and/or multi-processor infrastructures to speed up computations.

2.2.2. Model module

The model module collects all classes needed to describe event data in a parametrised way. The generic abstract base class of all models is the `GModel` class, which provides methods to compute the probability density $P_i(\boldsymbol{p}', E', t'|M_j)$ of observing an event with instrument direction $\boldsymbol{p}'$, measured energy E', and trigger time t' during an observation i for a given model M_j. We note that we used here the index j to indicate a specific model. Models can be combined using the `GModels` container class, which in turn computes the sum

$$P_i(\boldsymbol{p}', E', t'|M) = \sum_j P_i(\boldsymbol{p}', E', t'|M_j) \tag{7}$$

of the probability densities of all models. `GModels` is a member of the `GObservations` container class, so that all information needed for gamma-ray event data analysis is contained in a single object.

There are two basic classes that derive from the abstract `GModel` base class: the `GModelSky` class which implements a factorised representation of the spatial, spectral, and temporal components of a celestial source, and the abstract `GModelData` base class which defines the interface for any instrument specific description of events, and which is generally used to model instrumental backgrounds. While a celestial model $M_j^S(\boldsymbol{p}, E, t)$ is defined as function of true quantities, a data model $M_{j,i}^D(\boldsymbol{p}', E', t')$ is defined as function of reconstructed or measured quantities, and may also depend on the observation i. The event probability density for a celestial source model is computed by convolving the model with the instrument response function of the observation using

$$P_i(\boldsymbol{p}', E', t'|M_j) = \int_{\boldsymbol{p},E,t} R_i(\boldsymbol{p}', E', t'|\boldsymbol{p}, E, t) \times M_j^S(\boldsymbol{p}, E, t)\, \mathrm{d}\boldsymbol{p}\, \mathrm{d}E\, \mathrm{d}t \tag{8}$$

while for a data model, the event probability density is directly given by the model using

$$P_i(\boldsymbol{p}', E', t'|M_j) = M_{j,i}^D(\boldsymbol{p}', E', t'). \tag{9}$$

The factorisation of a celestial source model is given by

$$M^S(\boldsymbol{p}, E, t) = M_\mathrm{S}(\boldsymbol{p}|E, t) \times M_\mathrm{E}(E|t) \times M_\mathrm{T}(t), \tag{10}$$

where $M_\mathrm{S}(\boldsymbol{p}|E, t)$, $M_\mathrm{E}(E|t)$, and $M_\mathrm{T}(t)$ are the spatial, spectral, and temporal components of the model (we drop the indices j from now on). We note that this definition allows for energy- and time-dependent spatial model components, and for time-dependent spectral model components. So far, however, only model components that are constant in time are implemented.

The spatial component can be either modelled as a point source, a radially symmetric source, an elliptical source, or a diffuse source. For the latter, options comprise an isotropic intensity distribution on the sky, an arbitrary intensity distribution in form of a sky map, or an arbitrary energy-dependent intensity distribution provided in form of a map cube. Several model components exist for radial or elliptical sources. Disk models describe uniform intensity distributions within radial or elliptical boundaries. The radial Gaussian model represents an intensity distribution given by

$$M_\mathrm{S}(\boldsymbol{p}|E, t) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{1}{2}\frac{\theta^2}{\sigma^2}\right), \tag{11}$$

where θ is the angular separation from the centre of the distribution, and σ is the Gaussian width of the distribution. The radial shell model represents a spherical shell projected on the sky given by

$$M_\mathrm{S}(\boldsymbol{p}|E, t) = n_0 \begin{cases} \sqrt{\sin^2\theta_\mathrm{out} - \sin^2\theta} - \sqrt{\sin^2\theta_\mathrm{in} - \sin^2\theta} & \text{if } \theta \le \theta_\mathrm{in} \\ \sqrt{\sin^2\theta_\mathrm{out} - \sin^2\theta} & \text{if } \theta_\mathrm{in} < \theta \le \theta_\mathrm{out} \\ 0 & \text{if } \theta > \theta_\mathrm{out} \end{cases} \tag{12}$$

where θ_in and θ_out are the apparent inner and outer shell radii on the sky, respectively, and

$$\begin{aligned} n_0 = \frac{1}{2\pi}\Bigg(&\frac{\sqrt{1-\cos 2\theta_\mathrm{out}} - \sqrt{1-\cos 2\theta_\mathrm{in}}}{2\sqrt{2}} \\ &+ \frac{1+\cos 2\theta_\mathrm{out}}{4} \ln\left(\frac{\sqrt{2}\cos\theta_\mathrm{out}}{\sqrt{2} + \sqrt{1-\cos 2\theta_\mathrm{out}}}\right) \\ &- \frac{1+\cos 2\theta_\mathrm{in}}{4} \ln\left(\frac{\sqrt{2}\cos\theta_\mathrm{in}}{\sqrt{2} + \sqrt{1-\cos 2\theta_\mathrm{in}}}\right)\Bigg)^{-1} \end{aligned} \tag{13}$$

is a normalisation constant. Finally, the elliptical Gaussian model represents an intensity distribution given by

$$M_\mathrm{S}(\theta, \phi|E, t) = n_0 \times \exp\left(-\frac{\theta^2}{2r_\mathrm{eff}^2}\right), \tag{14}$$

where the effective ellipse radius r_eff towards a given position angle is given by

$$r_\mathrm{eff} = \frac{ab}{\sqrt{(a\sin(\phi-\phi_0))^2 + \sqrt{(b\cos(\phi-\phi_0))^2}}} \tag{15}$$

and a is the semi-major axis of the ellipse, b is the semi-minor axis, ϕ_0 is the position angle of the ellipse, counted counterclockwise from North, and ϕ is the azimuth angle with respect to celestial North. The normalisation constant n_0 is given by

$$n_0 = \frac{1}{2\pi \times a \times b}. \tag{16}$$

The spectral model components $M_\mathrm{E}(E|t)$ include a power law model

$$M_\mathrm{E}(E|t) = k_0 \left(\frac{E}{E_0}\right)^\gamma \tag{17}$$

where k_0 is a prefactor, E_0 is the pivot energy, and γ is the spectral index. We note that in GammaLib spectral indices are defined including the sign, hence typical gamma-ray sources have

values of $\gamma = -2 \ldots -4$. A variant of the power law model that replaces the pivot energy and prefactor by the integral flux N over an energy range $[E_{\rm min}, E_{\rm max}]$ is given by

$$M_{\rm E}(E|t) = \frac{N(\gamma+1)E^{\gamma}}{E_{\rm max}^{\gamma+1} - E_{\rm min}^{\gamma+1}}. \quad (18)$$

A broken power law is implemented by

$$M_{\rm E}(E|t) = k_0 \times \begin{cases} \left(\frac{E}{E_{\rm b}}\right)^{\gamma_1} & \text{if } E < E_{\rm b} \\ \left(\frac{E}{E_{\rm b}}\right)^{\gamma_2} & \text{otherwise} \end{cases} \quad (19)$$

with $E_{\rm b}$ being the break energy, and γ_1 and γ_2 being the spectral indices before and after the break, respectively. An exponentially cut-off power law is implemented by

$$M_{\rm E}(E|t) = k_0 \left(\frac{E}{E_0}\right)^{\gamma} \exp\left(\frac{-E}{E_{\rm cut}}\right) \quad (20)$$

with $E_{\rm cut}$ being the cut-off energy. A variant of this model is the super exponentially cut-off power law model, defined by

$$M_{\rm E}(E|t) = k_0 \left(\frac{E}{E_0}\right)^{\gamma} \exp\left(-\left(\frac{E}{E_{\rm cut}}\right)^{\alpha}\right) \quad (21)$$

which includes an additional power law index α on the cut-off term. A log parabola model is defined by

$$M_{\rm E}(E|t) = k_0 \left(\frac{E}{E_0}\right)^{\gamma+\eta\ln(E/E_0)}, \quad (22)$$

where η is a curvature parameter. And a Gaussian function that can be used to model gamma-ray lines is defined by

$$M_{\rm E}(E|t) = \frac{N_0}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(E-\bar{E})^2}{2\sigma^2}\right), \quad (23)$$

where $\bar{E}$ is the centre energy, and σ is the line width. An arbitrary spectral model is defined by a file function based on energy and intensity values specified in an ASCII file from which a piece-wise power law model is constructed. The file function can be adjusted to the data by applying a global scaling factor. Alternatively, the spectral node model is allowing for adjusting each pair of energy and intensity values as free parameters of a piece-wise power law model.

2.2.3. Sky map module

The sky map module collects classes that are used for handling sky maps. The central class of the module is the `GSkyMap` class which transparently handles sky maps provided either in the FITS World Coordinate Systems (WCS; Greisen & Calabretta 2002) or all-sky maps that are defined on the HEALPix grid (Górski et al. 2005). Currently, seven WCS projections are implemented (Aitoff, zenithal/azimuthal perspective, cartesian, Mercator's, Mollweide, stereographic and gnomonic), and we plan to implement in the future the full set of projections that is available in the wcslib library[10]. We note that sky pixels in GammaLib are defined at the bin centre, hence the first pixel of a WCS map covers the pixel range $[-0.5, +0.5]$ in the x- and the y-direction.

The sky map module also contains classes to define and handle arbitrary regions on the sky. Methods exist to test whether a given sky direction is contained in a sky region, or whether a region overlaps or is contained in another sky region. The format for defining sky regions is compatible with that used by DS9[11], enabling the loading of DS9 region files into GammaLib. So far, only a circular sky region has been implemented.

2.2.4. Application module

The last module of the high-level analysis support layer is the application module that provides classes that support building of ftools-like analysis tools, and which specifically establish the link to the ctools software package (see Sect. 3). The central class of this module is the `GApplication` base class, which upon construction grants access to user parameters provided in the IRAF command line parameter interface, a format that is already widely used for high-energy astronomy analysis frameworks, including ftools, the *Chandra* CIAO package, the INTEGRAL OSA software, or the *Fermi*-LAT science tools (see Appendix D).

The `GApplication` class also contains a logger that assures that all information provided during the execution of a tool will be presented to the user in a uniform way. By default, the logger will write output into an ASCII file, but simultaneous logging into the console can be enabled upon request.

2.2.5. Core modules

A number of services that are central to many GammaLib classes are collected into four core modules (cf. Fig. 1). Services useful for numerical computations are in the numerics module, and include classes for numerical integration and differentiation, as well as mathematical functions (e.g. error function, gamma function) and constants (e.g. π, $\ln 2$, $\sqrt{2}$). Classes that allow vector and matrix operations are collected in the linear algebra module, including classes to manage symmetric or sparse matrices. Classes that are used for function minimisation are collected in the optimisation module. Function minimisation is done using an optimiser, which will adjust the parameters of a function to minimise the function value. The standard optimiser for GammaLib is based on the iterative Levenberg-Marquardt method (Marquardt 1963). There is provision for including alternative optimisers. Additional services are collected in the support module. This includes classes for linear and bilinear interpolation, random number generation using a high-quality long-period natively implemented generator (Marsaglia & Zaman 1994), the handling of comma-separated value tables in ASCII files, and filename handling.

2.2.6. Interfaces

GammaLib provides a number of interfaces to support input and output of files and information. This includes in particular an interface to FITS files which is implemented in the fits module. The central class of this module is `GFits` which provides an in-memory representation of a FITS file. Each FITS file contains a list of Header Data Units (HDUs) composed of header keywords and either an image or a table. Both ASCII and binary tables are supported. To read and write FITS files, GammaLib relies on the cfitsio library.

GammaLib also includes an interface that allows manipulating of data provided in the XSPEC format used for X-ray astronomy (Arnaud 1996). GammaLib also includes a module that

[10] http://www.atnf.csiro.au/people/mcalabre/WCS/wcslib/

[11] http://ds9.si.edu/doc/ref/region.html

enables input and output of ASCII files in Extensible Markup Language (XML) format. Finally, GammaLib includes a module than enables the exchange of data and information with other Virtual Observatory (VO) compatible applications, such as for example the Aladin interactive sky atlas that can be used for sky map display. VO support is still experimental in the release 1.0.

2.2.7. Instrument modules

All modules that have been described so far are completely instrument independent, and are used to handle data obtained with any kind of gamma-ray telescope. To support the analysis of a specific instrument, instrument-specific modules have been added that implement a number of pure virtual base classes of the observation module. The general naming convention for instrument-specific classes is to prefix the class names after the initial "G" with a unique instrument code, i.e. `GCTAObservation` for the implementation of the CTA observation class. The GammaLib package includes so far instrument-specific modules for CTA (code `CTA`, Sect. 2.2.8), *Fermi*-LAT (code `LAT`, Sect. 2.2.9), and COMPTEL (code `COM`, Sect. 2.2.10), as well as a generic interface for multi-wavelength data (code `MWL`, Sect. 2.2.11). In the following sections we will describe the instrument-specific modules that exist so far in the GammaLib package.

2.2.8. CTA module

While the initial driver for developing the CTA module was to support event data analysis for the upcoming Cherenkov Telescope Array Observatory, the recent developments were also motivated by the needs of existing IACTs, such as H.E.S.S., VERITAS, and MAGIC. Maximum likelihood techniques are conventionally used for the analysis of data from medium-energy and high-energy instruments (e.g. Mattox et al. 1996; de Boer et al. 1992; Diehl et al. 2003), but the technique is rather new in the field of very-high-energy astronomy, and consequently its validation by applying it to data obtained with existing IACTs is extremely valuable for the preparation of CTA. To enable the joint analysis of data from all IACTs, specific instrument codes have been implemented (`CTA`, `HESS`, `VERITAS`, and `MAGIC`) and should be used in the observation definition XML file (see Appendix B), although all IACTs will make use of the same classes of the CTA module.

CTA event data is provided as a FITS file, containing for each event the reconstructed photon arrival direction, the reconstructed energy, and the arrival time. In addition, each event is enumerated by an identifier, and optionally may be characterised by instrument coordinates and an event phase (in case the data has been folded with the ephemerides of a pulsar or a binary system).

The instrument response for CTA is assumed to factorise into

$$R(\boldsymbol{p}', E', t'|\boldsymbol{p}, E, t) = A_{\rm eff}(\boldsymbol{p}, E, t) \times PSF(\boldsymbol{p}'|\boldsymbol{p}, E, t) \times E_{\rm disp}(E'|\boldsymbol{p}, E, t) \quad (24)$$

where $A_{\rm eff}(\boldsymbol{p}, E, t)$ is the effective area in units of cm^2, $PSF(\boldsymbol{p}'|\boldsymbol{p}, E, t)$ is the point spread function that satisfies

$$\int PSF(\boldsymbol{p}'|\boldsymbol{p}, E, t)\, {\rm d}\boldsymbol{p}' = 1 \quad (25)$$

and $E_{\rm disp}(E'|\boldsymbol{p}, E, t)$ is the energy dispersion that satisfies

$$\int E_{\rm disp}(E'|\boldsymbol{p}, E, t)\, {\rm d}E' = 1. \quad (26)$$

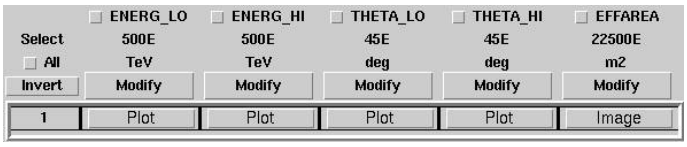

Fig. 2. Screenshot of effective area information stored in Response Table format.

In addition, the expected instrumental background rate $B_{\rm rate}(\boldsymbol{p}', E', t')$, given in units of counts s^{-1} MeV^{-1} sr^{-1}, is treated as the fourth response component.

The format for storing the CTA response information has been inspired by the FITS file format that is used by the *Fermi*-LAT science tools, and is based on so-called response tables. A response table consists of a FITS binary table with a single row and a number of vector columns that stores an arbitrary number of n-dimensional data cubes of identical size. The axes of each cube dimension are defined using pairs of columns that specify the lower and upper bin boundaries of all axis bins. The names of the columns are composed by appending the suffixes `_LO` and `_HI` to the axes names. The axes columns are followed by an arbitrary number of data cubes. Figure 2 illustrates how an energy and off-axis dependent effective area response is stored in this format. The first four columns define the lower and upper bin boundaries of the energy axis (`ENERG_LO` and `ENERG_HI`) and the off-axis angle axis (`THETA_LO` and `THETA_HI`), followed by a fifth column that holds the effective area data cube (`EFFAREA`).

For the point spread function, two variants exist that both depend on energy and off-axis angle, but differ in the functional form that is used to describe the PSF. The first variant implements a superposition of three 2D Gaussian functions that are each characterised by a width and a relative amplitude. Alternatively, a King profile defined by

$$PSF(\boldsymbol{p}'|\boldsymbol{p}, E, t) = \frac{1}{2\pi\sigma^2}\left(1 - \frac{1}{\gamma}\right)\left(1 + \frac{1}{2\gamma}\frac{\delta^2}{\sigma^2}\right)^{-\gamma}, \quad (27)$$

can be used, where δ is the angular separation between the true and measured photon directions $\boldsymbol{p}$ and $\boldsymbol{p}'$, respectively, σ describes the width and γ the tail of the distribution. In both cases, the energy and off-axis angle dependent functional parameters are stored in the data cubes of the respective Response Tables.

The energy dispersion is stored as a three-dimensional data cube spanned by true energy, the ratio of reconstructed over true energy, and off-axis angle. The background rate is stored as a three-dimensional data cube spanned by the detector coordinates `DETX` and `DETY` and the reconstructed energy.

In addition to the fundamental factorisation (Eq. (24)) of the CTA instrument response, there exists a specific response definition that is used in a so-called stacked analysis. In a stacked analysis, data from multiple observations is combined into a single counts cube, and consequently, an average response needs to be computed based on a proper weighting of the individual instrument response functions for each observation. The response for a stacked analysis is composed of an exposure cube, a point spread function cube and a background cube; the handling of energy dispersion is not supported so far. The exposure cube is computed using

$$X_{\rm cube}(\boldsymbol{p}, E) = \sum_i A_{{\rm eff},i}(\boldsymbol{p}, E, t) \times \tau_i, \quad (28)$$

where $A_{{\rm eff},i}(\boldsymbol{p}, E, t)$ is the effective area and τ_i the livetime of observation i, and the sum is taken over all observations. The

point spread function cube is computed using

$$PSF_{\rm cube}(\boldsymbol{p}, E, \delta) = \frac{\sum_i PSF_i(\boldsymbol{p}'|\boldsymbol{p}, E, t) \times A_{{\rm eff},i}(\boldsymbol{p}, E, t) \times \tau_i}{\sum_i A_{{\rm eff},i}(\boldsymbol{p}, E, t) \times \tau_i}, \quad (29)$$

and the background cube is computed using

$$B_{\rm cube}(\boldsymbol{p}', E') = \frac{\sum_i B_i(\boldsymbol{p}', E', t') \times \tau_i}{\sum_i \tau_i}. \quad (30)$$

The CTA module also contains some models to describe the distribution of the instrumental background in the data. All models are factorised using

$$B(\boldsymbol{p}', E', t') = B_{\rm S}(\boldsymbol{p}'|E', t') \times B_{\rm E}(E'|t') \times B_{\rm T}(t'), \quad (31)$$

where $B_{\rm S}(\boldsymbol{p}'|E', t')$ is the spatial, $B_{\rm E}(E'|t')$ the spectral, and $B_{\rm T}(t')$ the temporal component of the model. We note that this factorisation is similar to the one used for celestial source models (Eq. (10)) with all true quantities being replaced by reconstructed quantities. In fact, the spectral and temporal model components $M_{\rm E}(E'|t')$ and $M_{\rm T}(t')$ that are provided by the model module can be used as spectral and temporal components of the CTA background model, and only the spatial component needs an instrument specific implementation (this is related to the fact that the instrument direction $\boldsymbol{p}'$ is instrument specific, while energy E' and time t' are generic quantities).

The available background models for CTA differ in the implementation of the spatial component $B_{\rm S}(\boldsymbol{p}'|E', t')$. A first option consists of using the energy-dependent background rate templates that are stored in the instrument response function to describe the spatial component of the model, i.e.

$$B_{\rm S}(\boldsymbol{p}'|E', t') = B_{\rm rate}(\boldsymbol{p}', E', t'). \quad (32)$$

Alternatively, the spatial distribution of the background rate can be modelled using the effective area of the instrument, i.e.

$$B_{\rm S}(\boldsymbol{p}'|E', t') = A_{\rm eff}(\boldsymbol{p}', E', t'), \quad (33)$$

with the true arrival direction, energy and time being replaced by the reconstructed quantities. A specific model exists also to model the background for a stacked analysis, using the background cube as spatial component, i.e.

$$B_{\rm S}(\boldsymbol{p}'|E', t') = B_{\rm cube}(\boldsymbol{p}', E'). \quad (34)$$

Finally, a radially symmetric parametric background model can be used that defines the off-axis dependence of the background rate as function of the offset angle θ. Three implementations for the radial dependency exist: a Gaussian in offset angle squared, defined by

$$B_{\rm S}(\boldsymbol{p}'|E', t') = \exp\left(-\frac{1}{2}\frac{\theta^4}{\varsigma^2}\right), \quad (35)$$

where ς is a width parameter, a radial profile defined by

$$B_{\rm S}(\boldsymbol{p}'|E', t') = \left(1 + \left(\frac{\theta}{c_0}\right)^{c_1}\right)^{-c_2/c_1}, \quad (36)$$

where c_0 is the width of the radial profile, c_1 is the width of the central plateau, and c_2 is the size of the tail of the radial distribution, and a polynomial function

$$B_{\rm S}(\boldsymbol{p}'|E', t') = \sum_{l=0}^{m} c_l \theta^l, \quad (37)$$

where c_l are polynomial coefficients, and m is the degree of the polynomial.

2.2.9. *Fermi*-LAT module

The *Fermi*-LAT module provides support to include event data collected with the Large Area Telescope (LAT) aboard NASA's *Fermi* satellite into a joint maximum likelihood analysis, but it relies on the official *Fermi* science tools[12] to prepare the data in the appropriate format. So far, the module supports only the analysis of data processed with the Pass 6 and Pass 7 event-level reconstructions, but the implementation of Pass 8 analysis support is in progress. Also, only a binned maximum likelihood analysis has been implemented so far. Such an analysis requires on input a livetime cube, prepared using the `gtltcube` *Fermi*-LAT science tool, and a source map that also includes a counts map, created using the `gtsrcmaps` *Fermi*-LAT science tool.

The *Fermi*-LAT response function is factorised using

$$R(\boldsymbol{p}', E', t'|\boldsymbol{p}, E, t) = A_{\rm eff}(E, \theta) \times PSF(\delta|E, \theta) \times E_{\rm disp}(E'|E, \theta) \quad (38)$$

where $A_{\rm eff}(E, \theta)$ is the effective area in units of cm^2, $PSF(\delta|E, \theta)$ is the point spread function that satisfies

$$\int PSF(\delta|E, \theta)\, {\rm d}\delta = 1, \quad (39)$$

$E_{\rm disp}(E'|E, \theta)$ is the energy dispersion that satisfies

$$\int E_{\rm disp}(E'|E, \theta)\, {\rm d}E' = 1, \quad (40)$$

θ is the inclination angle with respect to the LAT z-axis, and δ is the angular separation between the true and measured photon directions $\boldsymbol{p}$ and $\boldsymbol{p}'$, respectively. The instrument response function is independent of time. Two functional forms are available for the point spread function which are both composed of a superposition of two King functions:

$$\begin{aligned} PSF_1(\delta|E, \theta) &= n_{\rm c}\left(1 - \frac{1}{\gamma_{\rm c}}\right)\left(1 + \frac{1}{2\gamma_{\rm c}}\frac{\delta^2}{\sigma^2}\right)^{-\gamma_{\rm c}} \\ &\quad + n_{\rm t}\left(1 - \frac{1}{\gamma_{\rm t}}\right)\left(1 + \frac{1}{2\gamma_{\rm t}}\frac{\delta^2}{\sigma^2}\right)^{-\gamma_{\rm t}}, \end{aligned} \quad (41)$$

and

$$\begin{aligned} PSF_3(\delta|E, \theta) &= n_{\rm c}\left(\left(1 - \frac{1}{\gamma_{\rm c}}\right)\left(1 + \frac{1}{2\gamma_{\rm c}}\frac{\delta^2}{s_{\rm c}^2}\right)^{-\gamma_{\rm c}}\right. \\ &\quad \left. + n_{\rm t}\left(1 - \frac{1}{\gamma_{\rm t}}\right)\left(1 + \frac{1}{2\gamma_{\rm t}}\frac{\delta^2}{s_{\rm t}^2}\right)^{-\gamma_{\rm t}}\right). \end{aligned} \quad (42)$$

The parameters $n_{\rm c}$, $n_{\rm t}$, $s_{\rm c}$, $s_{\rm t}$, σ, $\gamma_{\rm c}$ and $\gamma_{\rm t}$ depend on energy E and off-axis angle θ. The energy dispersion is so far not used.

The LAT events are partitioned into exclusive event types that for Pass 6 and Pass 7 data correspond to pair conversions located in either the front or the back section of the tracker (for Pass 8 the event partitioning has been generalised to other event types). For each event type, a specific response function exists that will be designated in the following with the superscript α.

The livetime cube is a means to speed up the exposure calculations in a *Fermi*-LAT analysis and contains the integrated livetime as a function of sky position and inclination angle with respect to the LAT z-axis. This livetime, denoted by $\tau(\boldsymbol{p}, \theta)$, is the time that the LAT observed a given position on the sky at a given inclination angle, and includes the history of the LAT's orientation during the entire observation period. A *Fermi*-LAT

[12] http://fermi.gsfc.nasa.gov/ssc/data/analysis/

livetime cube includes also a version of the livetime information that is weighted by the livetime fraction (i.e. the ratio between livetime and ontime) and that allows correction of inefficiencies introduced by so-called ghost events, and that we denote here by $\tau_{\rm wgt}(\boldsymbol{p}, \theta)$. The exposure for a given sky direction $\boldsymbol{p}$, photon energy E and event type α is then computed using

$$X^{\alpha}(\boldsymbol{p}, E) = f_1^{\alpha}(E) \int_{\theta} \tau(\boldsymbol{p}, \theta)\, A_{\rm eff}^{\alpha}(E, \theta)\, {\rm d}\theta + f_2^{\alpha}(E) \int_{\theta} \tau_{\rm wgt}(\boldsymbol{p}, \theta)\, A_{\rm eff}^{\alpha}(E, \theta)\, {\rm d}\theta, \quad (43)$$

and the exposure weighted point spread function is computed using

$$PSF^{\alpha}(\delta|\boldsymbol{p}, E) = f_1^{\alpha}(E) \int_{\theta} \tau(\boldsymbol{p}, \theta)\, A_{\rm eff}^{\alpha}(E, \theta)\, PSF^{\alpha}(\delta|E, \theta)\, {\rm d}\theta + f_2^{\alpha}(E) \int_{\theta} \tau_{\rm wgt}(\boldsymbol{p}, \theta)\, A_{\rm eff}^{\alpha}(E, \theta)\, PSF^{\alpha}(\delta|E, \theta)\, {\rm d}\theta, \quad (44)$$

where $f_1^{\alpha}(E)$ and $f_2^{\alpha}(E)$ are energy and event type dependent efficiency factors.

Finally, the point spread function for a point source is computed using

$$\overline{PSF}(\delta|\boldsymbol{p}, E) = \frac{\sum_{\alpha} PSF^{\alpha}(\delta|\boldsymbol{p}, E)}{\sum_{\alpha} X^{\alpha}(\boldsymbol{p}, E)}, \quad (45)$$

where the sum is taken over all event types.

2.2.10. COMPTEL module

The COMPTEL module provides support to include event data collected with the Compton Telescope aboard NASA's CGRO mission into a joint maximum likelihood analysis. The module accepts high-level data available at HEASARC's archive for high-energy astronomy missions[13], and is to our knowledge the only software that can be used today to exploit the legacy COMPTEL data in that archive. There are three basic data files in FITS format that are used to describe a COMPTEL observation for a given energy range and that are available from HEASARC: a DRE file, containing the binned event data, a DRX file, containing the exposure as function of sky direction, and a DRG file, containing geometry information. In addition, the energy-dependent instrument response is described by IAQ files that however are not available in the HEASARC archive. Therefore, IAQ files applying to the standard energy ranges of COMPTEL (0.75–1, 1–3, 3–10 and 10–30 MeV) are included in the GammaLib package.

COMPTEL measured photons using two detector planes separated by 1.5 m, where an incoming photon interacts first by Compton scattering with a detector of the upper plane before being absorbed in a detector of the lower plane. A COMPTEL event is characterised by an instrument direction spanned by the angles $(\chi, \psi, \bar{\varphi})$. (χ, ψ) is the direction of the photon after scattering in the upper detector plane, which is determined from the photon interaction locations in both detector planes, and

$$\bar{\varphi} = \arccos\left(1 - \frac{m_{\rm e}c^2}{E_2} + \frac{m_{\rm e}c^2}{E_1 + E_2}\right) \quad (46)$$

is the Compton scattering angle as inferred from the energy deposits E_1 and E_2 in the upper and lower detector planes, respectively. The measured energy of the photon is estimated from the sum

$$E' = E_1 + E_2 \quad (47)$$

of the energy deposits in both detector planes. The probability that a photon which interacted in the upper detector plane will encounter a detector of the lower plane is described by $DRG(\chi, \psi, \bar{\varphi})$, which also includes any cuts related to the removal of events coming from the Earth limb.

The COMPTEL response is factorised using

$$R(\boldsymbol{p}', E', t'|\boldsymbol{p}, E, t) = \frac{DRX(\boldsymbol{p})}{T} \times DRG(\chi, \psi, \bar{\varphi}) \times IAQ(\chi, \psi, \bar{\varphi}|\boldsymbol{p}, E), \quad (48)$$

where $DRX(\boldsymbol{p})$ is the exposure in units of cm^2 s, T is the ontime in units of s, and $IAQ(\chi, \psi, \bar{\varphi}|\boldsymbol{p}, E)$ quantifies the interaction probability for a Compton scattering in the upper detector plane followed by an interaction in the lower detector plane. We note that $IAQ(\chi, \psi, \bar{\varphi}|\boldsymbol{p}, E)$ is azimuthally symmetric about the source direction, and the IAQ file is stored as a 2D FITS image providing the interaction probabilities as function of $\bar{\varphi}$ and $\varphi_{\rm geo}$ for a given energy range, where $\varphi_{\rm geo}$ is the angular separation between (χ, ψ) and $\boldsymbol{p}$.

The `GCOMObservation` class implements a COMPTEL observation for a given energy range. Performing a spectral analysis for COMPTEL thus implies appending an observation per energy range to the observation container. A COMPTEL observation also contains a DRB data cube that provides an estimate of the instrumental background. This instrumental background model can be adjusted to the data by maximum likelihood fitting of its $\bar{\varphi}$ distribution. Since DRB files are not provided by HEASARC, the user may specify the DRG file as a first order approximation of the instrumental background distribution in COMPTEL data. It is planned to implement more accurate background models for COMPTEL in the future.

2.2.11. Multi-wavelength module

The multi-wavelength module provides support to add flux points that were obtained by external analyses to constrain a joint maximum likelihood fit. A typical example would be to constrain a synchrotron component using data obtained at radio wavebands or in the optical or X-ray bands. Another example is the addition of gamma-ray flux points in case the original event data is not publicly available. Since the multi-wavelength module handles data in physics space, the response function has the trivial form

$$R(\boldsymbol{p}', E', t'|\boldsymbol{p}, E, t) = 1. \quad (49)$$

So far, flux points need to be specified in a FITS table composed of at least 2 columns. If the table contains two columns it is assumed that they respectively contain the energy and the flux information. For three columns it is assumed that the third column contains the statistical uncertainty in the flux measurement. For four or more columns it is assumed that the first four columns respectively contain the energy, the energy uncertainty, the flux and the flux uncertainty. Energy (or wavelength) information can be provided in units of erg(s), keV, MeV, GeV, TeV, or Ångström. Flux information can be provided in units of ph cm^{-2} s^{-1} MeV^{-1} or erg cm^{-2} s^{-1}. The FITS table unit keywords are analysed to infer the proper units of the energy and flux axes. We plan to connect in the future the multi-wavelength module to the Virtual Observatory interface to support the interoperability with VO services.

[13] http://heasarc.gsfc.nasa.gov/docs/journal/cgro7.html

3. ctools

3.1. Overview

The ctools package has been written with the goal to provide a user-friendly set of software tools allowing for the science analysis of IACT event data. The software operates on lists of reconstructed IACT events that have been calibrated in energy and from which most of the particle background has been removed based on air Cherenkov shower image characteristics. The software also requires IACT instrument response functions describing the transformation from physical properties of photons to measured characteristics of events. We propose to use ctools as the science tools software for CTA, yet the software also supports the analysis of data from existing IACTs such as H.E.S.S., VERITAS, or MAGIC, provided that the data and response functions are converted into the proper format.

The ctools package allows for the creation of images, spectra and light curves of gamma-ray sources, providing the results in FITS format that is compatible with standard astronomical tools that can be used for their display. Tools to graphically display the analysis results are therefore not included in ctools, but a number of Python plotting scripts based on the matplotlib Python module (Hunter 2007) are available in the `examples` folder of the ctools package for visualisation of results.

Each ctool performs a single, well-defined analysis step, so that scientists can combine the modular tools into a customised workflow that matches the specific needs of an analysis. The ctools philosophy is very similar to the rational behind the ftools (Pence et al. 1993), which are widely used in X-ray astronomy, and have also inspired the science analysis frameworks of INTEGRAL and *Fermi*-LAT.

The ctools package is based on the GammaLib analysis framework, which is the only external library dependency of the software. This assures the seamless installation of the software on a large variety of operating systems and platforms, and keeps the long-term software maintenance cost at a manageable level. Each tool of the package is a class that derives from GammaLib's `GApplication` class, providing thus a standard user interface and common functionalities and behavior to all tools. A tool can be implemented as compiled executable written in C++ or as a Python script. To distinguish both, we call the former a ctool and the latter a cscript. Names of ctools start with "ct" while names of cscripts start with "cs". ctools and cscripts expose identical user interfaces, and are largely indistinguishable to the user. Our current philosophy is to use ctools for the basic building blocks of the package, while cscripts are used for high-level tasks, calling eventually several of the ctools. We will use the terms tool or tools in this paper if we make no distinction between a ctool or a cscript.

All tools can be called from the command line using the IRAF Command Language parameter interface (see Appendix D). All tools are also available as Python classes in the `ctools` and `cscripts` Python modules that are compliant with Python 2 (from version 2.3 on) and Python 3. Within Python, observation containers can be passed from one tool to another, avoiding the need for storing intermediate results on disk. This enables the creation of pure in-memory analysis workflows, circumventing potential I/O bottlenecks and profiting from the continuously growing amount of memory that is available on modern computers.

The ctools package ships with a calibration database that contains instrument response functions that are needed to simulate and analyse CTA event data. The calibration database is organised following HEASARC's calibration database (CALDB) format[14], which places all calibration relevant information into a directory tree starting from the path defined by the `CALDB` environment variable. Instrument response functions are specified for ctools by the `caldb` and `irf` parameters, where the first gives the name of the calibration database (which is `caldb=prod2` for the IRFs shipped with ctools), and the second gives the name of the IRF (one of `North_0.5h`, `North_5h`, `North_50h`, `South_0.5h`, `South_5h`, or `South_50h`, labelling response functions for the northern and southern CTA arrays, with variants that have been optimised for exposure times of 0.5 h, 5 h and 50 h).

3.2. Available tools

Figure 3 provides a summary of the available tools in release 1.0 of the ctools package, grouped according to functionality, and arranged according to the typical usage in a workflow. Workflow examples can be found in the online ctools user manual[15]. Simulation of event data is supported by the `csobsdef` script and the `ctobssim` tool, event data selection is done using the `ctselect` tool, and event binning and related response preparation is supported by the `ctbin`, `ctcubemask`, `ctexpcube`, `ctpsfcube`, and `ctbkgcube` tools. For maximum likelihood analysis there are the `ctlike`, `ctbutterfly`, `ctulimit`, `cttsmap` and `ctmodel` tools and the `csresmap` script. For imaging analysis there exists the `ctskymap` tool, spectral analysis is done using the `csspec` script, and for timing analysis there is the `cslightcrv` script. In addition, there are the utility scripts `cscaldb` to inspect the IRF database, `cssens` to determine a sensitivity curve, `cspull` to generate pull distributions, and `cstsdist` to investigate Test Statistics distributions. In the following we provide a brief description of the tools.

3.2.1. csobsdef

The `csobsdef` script generates an observation definition XML file from a list of pointings that can be used as a starting point for the simulation of IACT observations (see Appendix B for a description of the XML file format). The pointing list is a comma-separated value (CSV) ASCII file with header keywords in the first row followed by a list of pointings, with one pointing per row. For example, the ASCII file

```
name,id,ra,dec,duration,emin,emax,rad,deadc,caldb,irf
Crab,01,83.63,22.01,1800,0.1,100,5,0.95,prod2,South_0.5h
Crab,02,83.63,21.01,1800,0.2,100,5,0.95,prod2,South_0.5h
Crab,03,83.63,23.01,1800,0.3,100,5,0.95,prod2,South_0.5h
```

will produce an observation definition XML file containing 3 observations of 1800 s duration, wobbling around the Crab position in Declination, and with energy thresholds, as specified by the `emin` column, increasing from 0.1 TeV to 0.3 TeV. Alternatively to Right Ascension and Declination, Galactic longitude and latitude can be specified using the `lon` and `lat` keywords. Angles are specified in units of degrees, energies in units of TeV. Only the keywords `ra` and `dec` (or `lon` and `lat`) are mandatory. If no `duration` keyword is provided the `csobsdef` script will query a value and apply that duration to all pointings in the list. All other keywords are optional and default values will be assumed for all observations, unless the keyword is explicitly specified as a parameter to `csobsdef`.

[14] http://heasarc.gsfc.nasa.gov/docs/heasarc/caldb/caldb_intro.html
[15] http://cta.irap.omp.eu/ctools/user_manual/getting_started/quickstart.html

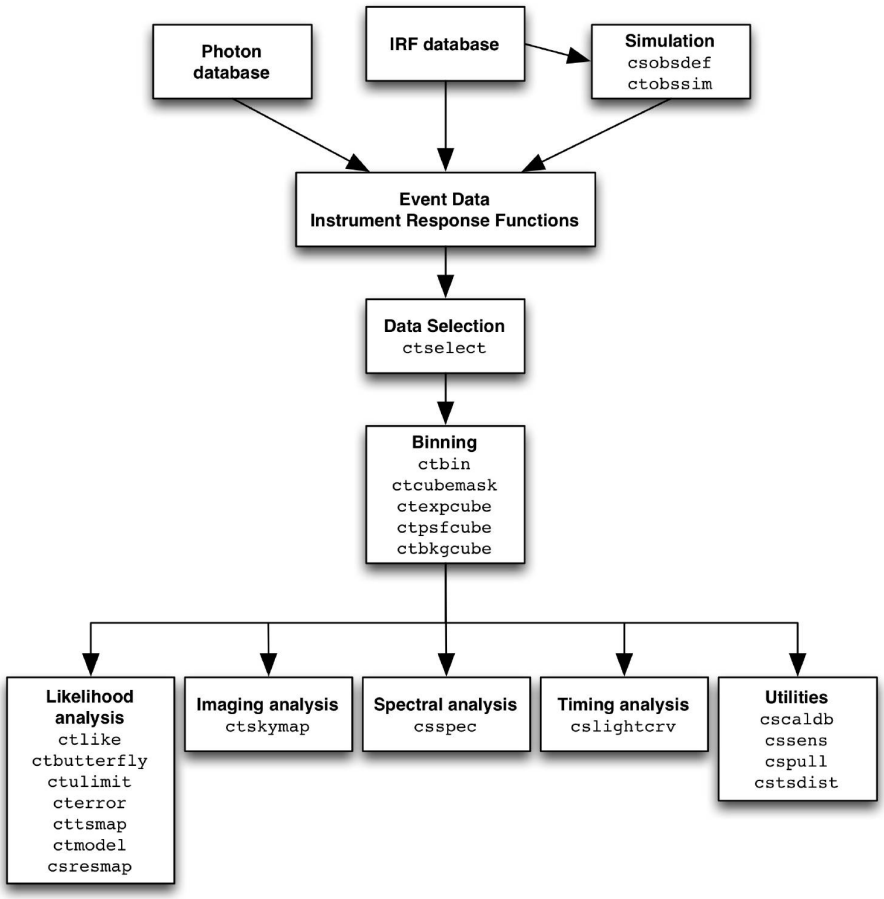

Fig. 3. Overview over ctools.

3.2.2. ctobssim

The `ctobssim` tool simulates IACT event list(s) based on an input model and the instrument characteristics described by the instrument response function(s). The simulation includes photon events from astrophysical sources and background events that are drawn from the input model using the numerical random number generator provided by GammaLib. The seed value for the random number generator can be specified through the `seed` parameter. By default, `ctobssim` simulates a single IACT pointing and produces a single event list, but by specifying explicitly an observation definition XML file using the `inobs` parameter, the tool can be instructed to generate an event list for each observation that is defined in the XML file. If OpenMP support is available, `ctobssim` will parallelise the computations and spread the simulations of multiple observations over all available computing cores. `ctobssim` creates FITS file(s) comprising the event list and their Good Time Intervals.

3.2.3. ctselect

The `ctselect` tool selects from an event list only those events whose reconstructed arrival directions fall within a circular acceptance region, and whose reconstructed energies and trigger times fall within specified boundaries. In addition, arbitrary event selection criteria can be defined by using the cfitsio row filtering syntax[16]. Optionally, `ctselect` applies save energy thresholds that are specified in the effective area component of the instrument response function via the `LO_THRES` and `HI_THRES` FITS header keywords, or user supplied energy thresholds that are given in an observation definition XML file through the `emin` and `emax` attributes (see Appendix B). If `ctselect` is applied to a single event list, the tool outputs a new FITS file that only contains the selected events. The tool can also be applied to a list of observations by specifying an observation definition XML file on input. In that case, `ctselect` will create one events FITS file per observation, and outputs a new observation definition XML file that references the new event files.

3.2.4. ctbin

This `ctbin` tool creates a counts cube that is filled with events from event list(s). A counts cube is a three-dimensional data cube spanned by Right Ascension or Galactic longitude, Declination or Galactic latitude, and reconstructed energy. The events are either taken from a single event list file or from the event lists that are specified in an observation definition XML file. If multiple event lists are given in the observation definition XML file, the tool will loop over all event lists and stack their events into a single counts cube. `ctbin` creates a counts cube FITS file comprising the counts cube data, the counts cube energy boundaries,

[16] https://heasarc.gsfc.nasa.gov/docs/software/fitsio/c/c_user/node97.html

and the Good Time Intervals of all event lists that have been filled into the counts cube.

3.2.5. ctcubemask

The `ctcubemask` tool masks out specific regions from a counts cube by setting the corresponding bin values to −1, since bins with negative values will be ignored by GammaLib in a maximum likelihood analysis. The tool applies a spatial mask that is comprised of a circular selection region (bins outside this region will be ignored) and a list of circular exclusion regions (bins inside these regions will be ignored). The circular exclusion regions are specified by an ASCII file in a format that is inspired by the region format used by DS9[17]. Specifically, the ASCII file contains one row per exclusion region, given in the format

```
circle(83.63, 21.5, 0.4)
```

where `83.63` and `21.5` are the Right Ascension and Declination of the region centre and `0.4` is the radius (in degrees) of the exclusion circle. The tool also only selects counts cube energy layers that are fully contained within a specified energy interval. `ctcubemask` creates a counts cube FITS file that is a copy of the input counts cube where all masked bins will be set to values of −1. Users can of course mask additional bins of the cube manually to implement more complex masking schemes by setting individual counts cube bins to values of −1.

3.2.6. ctexpcube

The `ctexpcube` tool generates an exposure cube for a stacked maximum likelihood analysis according to Eq. (28). An exposure cube is a three-dimensional data cube spanned by Right Ascension or Galactic longitude, Declination or Galactic latitude, and energy, which gives the exposure as function of true sky direction and energy. `ctexpcube` creates an exposure cube FITS file comprising the exposure cube, its energy boundaries, and the Good Time Intervals of all observations that have been used for the computation of the exposure cube.

3.2.7. ctpsfcube

The `ctpsfcube` tool generates a point spread function cube for a stacked maximum likelihood analysis according to Eq. (29). A point spread function cube is a four-dimensional data cube spanned by true Right Ascension or Galactic longitude, true Declination or Galactic latitude, true energy, and offset angle between true and measured arrival direction of a photon. `ctpsfcube` creates a point spread function cube FITS file comprising the point spread function cube, its energy boundaries, and the definition of the angular separation axis.

3.2.8. ctbkgcube

The `ctbkgcube` tool generates a background cube for a stacked maximum likelihood analysis according to Eq. (30). A background cube is a three-dimensional data cube spanned by reconstructed Right Ascension or Galactic longitude, Declination or Galactic latitude, and energy. An input model is used to predict the expected number of background counts in each background cube bin. `ctbkgcube` creates a background cube FITS file comprising the background rate per cube bin, and the energy boundaries of the background cube. The tool also creates an output model XML file that should be used as input for a maximum likelihood analysis.

[17] http://ds9.si.edu/doc/ref/region.html

3.2.9. ctlike

The `ctlike` tool is the main engine of the ctools package, allowing for the determination of the flux, spectral index, position or extent of gamma-ray sources using maximum likelihood model fitting of IACT event data. The analysis can be done using an unbinned or binned formulation of the log-likelihood function (Eqs. (1), (3), and (5)), and the tool is able to perform a joint maximum likelihood analysis of data collected in separate observations or with different instruments.

Multiple observations, including data collected with different instruments, can be handled by specifying an observation definition XML file on input (see Appendix B). Each of the observations will be kept separately and associated with its appropriate instrument response function, as opposed to a stacked analysis where average response functions are used. If an observation definition XML file is provided, `ctlike` will use the joint likelihood of all the observations for parameter optimisation (see Eq. (6)).

By default, `ctlike` will use the Poisson statistics for likelihood computation, but for binned analysis also Gaussian statistics can be requested. All model parameters which are flagged by the attribute `free="1"` in the model XML file, including spatial parameters, will be adjusted by `ctlike`. For all model components j for which the attribute `tscalc="1"` is specified in the model XML file, `ctlike` will also compute the Test Statistics (TS) value that is defined by

$$\mathrm{TS} = 2 \ln L(M) - 2 \ln L(M_{-j}), \tag{50}$$

where $L(M)$ is the maximum likelihood value for the full model M and $L(M_{-j})$ is the maximum likelihood value for a model from which the component j has been removed. Under the hypothesis that the model M provides a satisfactory fit of the data, TS follows a χ^2_p distribution with p degrees of freedom, where p is the number of model parameters in component j (Cash 1979).

`ctlike` creates an output model XML file that contains the values of the best fitting model parameters. For all free parameters, an `error` attribute is added that provides the statistical uncertainty in the parameter estimate. If for a model component the computation of the TS value has been requested, a `ts` attribute providing the TS value is added. The output model can be used as an input model for other ctools.

3.2.10. ctbutterfly

The `ctbutterfly` tool calculates a butterfly diagram for a specific source with power law spectral model. The butterfly diagram is the envelope of all power law models that are within a given confidence limit compatible with the data (by default a confidence level of 68% is used). The tool computes the envelope by evaluating for each energy the minimum and maximum intensity of all power law models that fall within the error ellipse of the prefactor and index parameters. The error ellipse is derived from the covariance matrix of a maximum likelihood fit. The butterfly diagram can be displayed using the `show_butterfly.py` script that is provided with the ctools package. For illustration, Fig. 4 shows the output of the `show_butterfly.py` script, obtained for a simulated source with a flux of 10 mCrab, observed with the southern CTA array for 30 min.

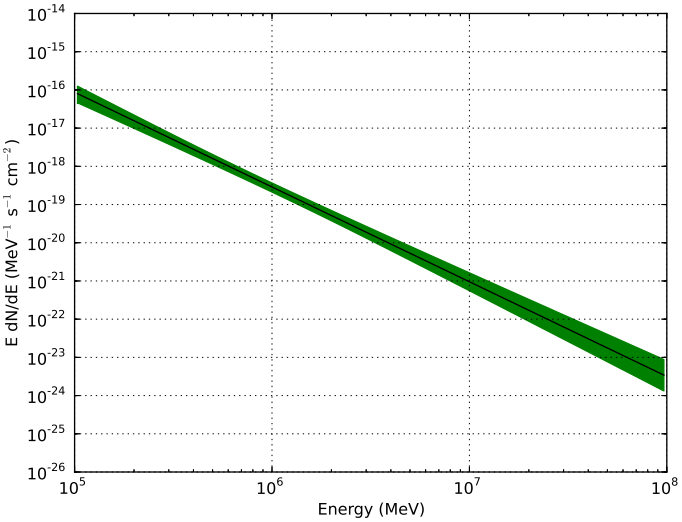

Fig. 4. Butterfly diagram for a simulated source with a flux of 10 mCrab, observed with the southern CTA array for 30 min.

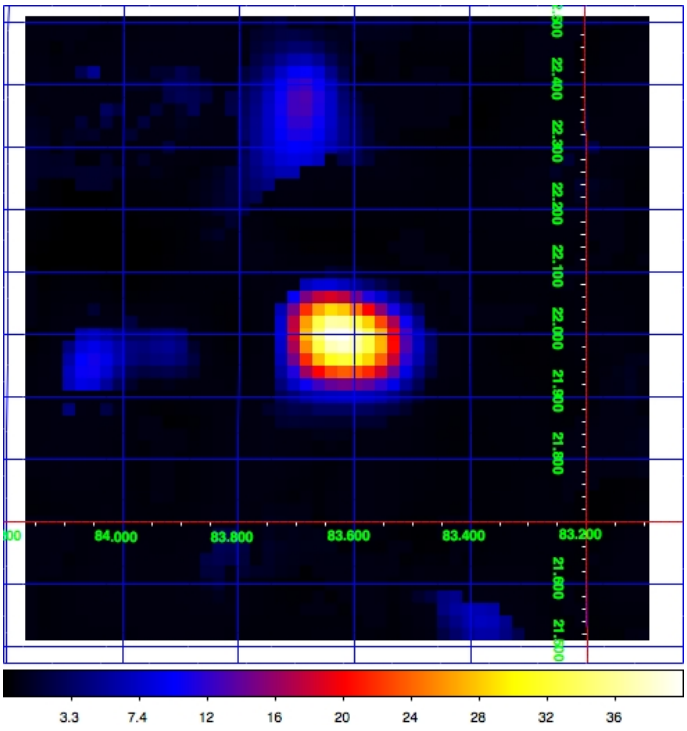

Fig. 5. TS map for a simulated source with a flux of 10 mCrab, observed with the southern CTA array for 30 min.

3.2.11. ctulimit

The `ctulimit` tool computes the upper flux limit for a specific source model. Except for the node function, all spectral models are supported. Starting from the maximum likelihood model parameters, the tool finds the model flux that leads to a decrease of the likelihood that corresponds to a given confidence level (by default a confidence level of 95% is used). `ctulimit` writes the differential upper flux limit at a given reference energy and the integrated upper flux limit into the log file.

3.2.12. cterror

The `cterror` tool computes the parameter errors for a specific source model using the likelihood profiles. Starting from the maximum likelihood model parameters, the tool finds the minimum and maximum model parameters that lead to a decrease of the likelihood that corresponds to a given confidence level (by default a confidence level of 68% is used). `cterror` creates an output model XML file that contains the values of the best fitting model parameters. For all free parameters, an `error` attribute is added that provides the statistical uncertainty in the parameter estimate as obtained from the likelihood profile. While `cterror` computes asymmetrical errors, which are written into the log file, the XML file will contain the mean error that is obtained by averaging the negative and positive parameter errors.

3.2.13. cttsmap

The `cttsmap` tool generates a TS map for a specific source model with point source, radial or elliptical spatial component. The tool displaces the specified source on a grid of sky positions and computes for each position the TS value (see Eq. (50)) by fitting the remaining free parameters of the source model. If the only remaining free parameter of the source model is the source flux, the square-root of the TS values corresponds to the pre-trial detection significance of the source in Gaussian sigma. We note that the TS values are only valid if at least a few events are actually detected towards a grid position, which may not always be the case for high energies and/or short observing times (irrespectively of whether these events actually come from the source or from instrumental background). `cttsmap` creates a FITS file comprising a sky map of TS values, followed by one image extension per free parameter that contain sky maps of the fitted parameter values. Figure 5 shows a TS map that has been obtained for a simulated source with a flux of 10 mCrab, observed with the southern CTA array for 30 min. Events between 0.1 and 100 TeV have been considered.

3.2.14. ctmodel

The `ctmodel` tool generates a model cube based on a model definition XML file (see Appendix C). A model cube is a three-dimensional data cube providing the number of predicted counts for a model as function of reconstructed Right Ascension or Galactic longitude, Declination or Galactic latitude, and energy. `ctmodel` creates a model cube FITS file comprising the predicted number of events per bin, the energy boundaries of the model cube, and the Good Time Intervals of all observations that have been used to compute the model cube.

3.2.15. csresmap

The `csresmap` scripts generates a residual map for a given model. It works for event lists, counts cubes or observation definition XML files. For event lists, parameters that define the spatial and spectral binning need to be provided so that the script can bin the data internally. The model is then convolved with the instrumental response function for that binning and used for residual computation. Before residual computation, the counts and model cubes are collapsed into maps by summing over all energies. Three options exist then for residual computation: the subtraction of the model from the counts (`algorithm=SUB`), the subtraction and division by the model (`algorithm=SUBDIV`), and the subtraction and division by the square root of the model (`algorithm=SUBDIVSQRT`). By default `algorithm=SUBDIV` is

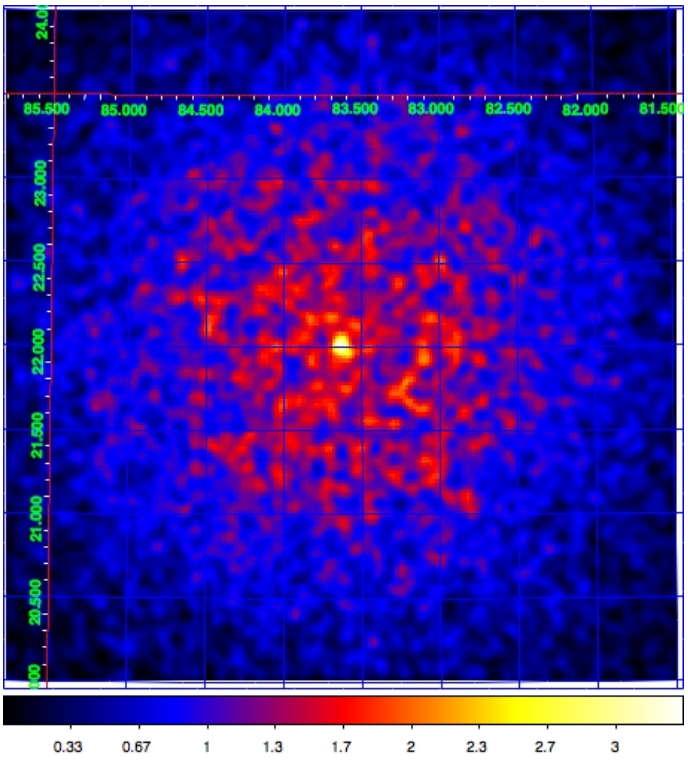

Fig. 6. Sky map of the observed events above 100 GeV for a simulated source with a flux of 10 mCrab, observed with the southern CTA array for 60 min. We note that the map is not background subtracted.

applied. `csresmap` creates a FITS file containing a sky map of the residuals.

3.2.16. ctskymap

The `ctskymap` tool generates a sky map from either a single event list or the event lists specified in an observation definition XML file. The tool will loop over all event lists that are provided and fill all events into a single sky map. So far, only the generation of maps of the measured number of counts are supported. `ctskymap` creates a FITS file comprising a sky map. Figure 6 shows a sky map created by `ctskymap` for an observation of a simulated source with a flux of 10 mCrab, observed with the southern CTA array for 60 min.

3.2.17. csspec

The `csspec` script extracts the spectrum of a gamma-ray source by fitting a model in a given set of spectral bins to the data. The model fits are performed using `ctlike` and the script computes the source flux and its statistical uncertainty in each spectral bin, as well as the significance of the source detection. Optionally, `csspec` computes also upper flux limits for each spectral bin. The script works on event list(s) or counts cube(s). `csspec` creates a FITS file containing a table with the fitted source spectrum, comprising one row per spectral bin. The spectrum can be displayed using the `show_spectrum.py` script that is provided with the ctools package. For illustration, Fig. 7 shows the output of the script that was obtained for a simulated source with a flux of 1 Crab, observed with the southern CTA array for 30 min.

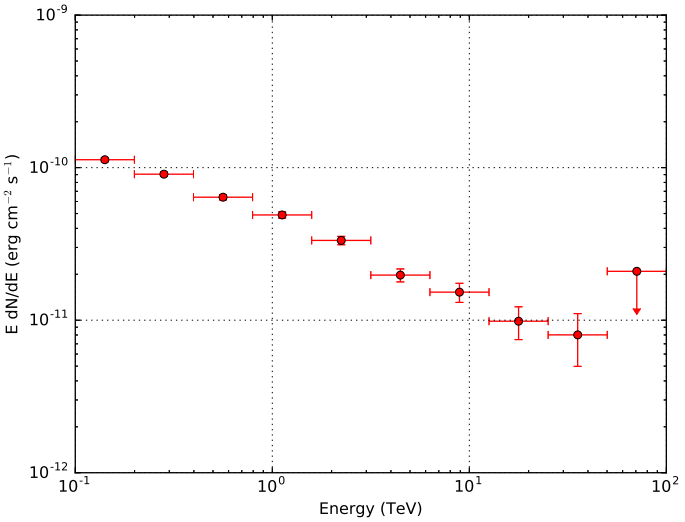

Fig. 7. Spectrum obtained using `csspec` for a simulated source with a flux of 1 Crab, observed with the southern CTA array for 30 min.

3.2.18. cslightcrv

The `cslightcrv` script computes a light curve by performing a maximum likelihood fit using `ctlike` in a series of time bins. The time bins can be either specified in an ASCII file, as an interval divided into equally sized time bins, or can be taken from the Good Time Intervals of the observation(s). The format of the ASCII file is one row per time bin, each specifying the start of stop value of the bin, separated by a whitespace. Times are specified in Modified Julian Days (MJD). `cslightcrv` creates a FITS file containing a table with the fitted model parameters and their statistical errors, the statistical significance of the detection as expressed by the TS value, and the upper flux limit, with one row per time bin.

3.2.19. cssens

The `cssens` script computes the differential or integrated CTA sensitivity using maximum likelihood fitting of a test source. The differential sensitivity is determined for a number of energy bins, the integral sensitivity is determined for a number of energy thresholds and an assumed source spectrum. The test source is fitted to simulated data using `ctlike` to determine its detection significance as a function of source flux. The source flux is then varied until the source significance achieves a given level, specified by the significance parameter `sigma`, and set by default to 5σ. To reduce the impact of variations between individual Monte Carlo simulations, a sliding average is applied in the significance computation. The significance is estimated using the TS value defined by Eq. (50). The simplified assumption is made that the significance (in Gaussian sigma) is the square root of the TS values. The sensitivity curve can be displayed using the `show_sensitivity.py` script that is provided with the ctools package. Figure 8 shows the differential sensitivity curve that has been obtained using the `cssens` script for the southern CTA array after 30 min of observations. Please note that the high-energy sensitivity is slightly better than the one published by CTA[18], as the latter includes an additional constraint

[18] https://portal.cta-observatory.org/Pages/CTA-Performance.aspx

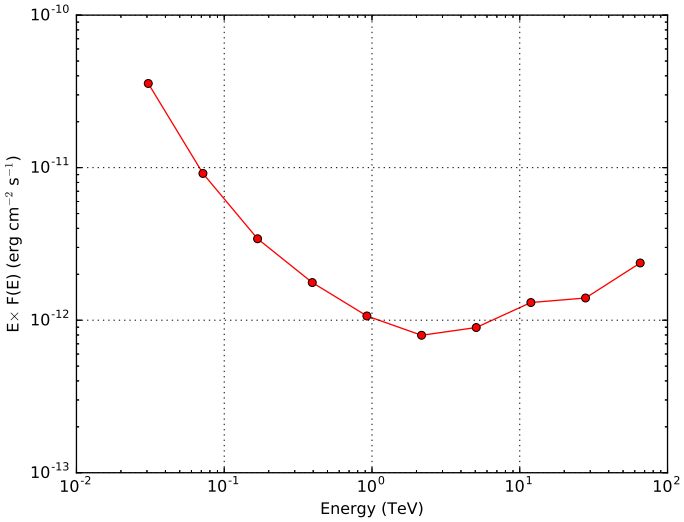

Fig. 8. On-axis differential sensitivity for a point source obtained using the `cssens` script for the southern CTA array (obtained for an observation time of 30 min).

of detecting a minimum of 10 photons from a source, while for a TS computation the presence of a few photons may be sufficient to reach a significance of 5σ.

3.2.20. cspull

The `cspull` script generates pull distributions for all free model parameters. The pull is defined by

$$g = \frac{x - \mu}{\sigma} \tag{51}$$

where x is the fitted model parameter, μ is its true value, and σ is the statistical uncertainty in the fitted model parameter. For an unbiased and correct estimate of the model parameter and its statistical error, the pull will be distributed as a standard Gaussian with mean zero and unit width. The `cspull` script will perform `ntrials` statistically independent computations of the pull for each free model parameter by simulating events using `ctobssim` followed by a maximum likelihood model fitting using `ctlike`. From the output file, pull distribution plots can be generated using the `show_pull_histogram.py` script that is included in the package. Another script named `show_pull_evolution.py` can be used to plot the evolution of the mean and standard deviation of the pull distribution as function of the number of trials.

3.2.21. cstsdist

The `cstsdist` script generates TS distributions (see Eq. (50)) for a given source by repeatedly computing the TS value for `ntrials` simulated data sets. This script supports unbinned or binned data, support for a stacked analysis is not yet implemented.

4. Performance

4.1. Numerical accuracy

The GammaLib and ctools packages have been developed with the goal to achieve an accuracy of better than 1% in all numerical computations. This means that if ctools are used to determine for example the flux received from a gamma-ray source or the spectral points of a spectral energy distribution (SED), the relative precision of the flux or the spectral points is better than 1%. The same is true for spatial parameters, such as source position or source extension. For many cases the actual numerical precision is in fact much better than 1%, but in any case, it should never be worse. Note, however, that this does not imply that source parameters can be determined with an IACT with an accuracy of 1%. The accuracy depends in the end on the precision to which the instrument response function is known, which is currently more in the 10–20% range.

The user should also be aware that bins are always evaluated at their centre, which can lead to biases when the binning is chosen too coarse. This is particularly important when performing a binned or stacked analysis, where the spatial and spectral binning needs to be sufficiently fine grained to fully sample the variation of the model. In particular, the spatial binning should be better than the best angular resolution over the energy range of interest. Typically, a value of 0.02° per pixel should be used for the spatial binning, and at least 10 bins per decade for the spectral binning of IACT data.

There is an issue with the fit of the broken power law spectral model, which has unreliable statistical errors, specifically for the prefactor and the break value. Errors are in general too large, which is related to the fact that the model gradient is discontinuous in energy.

The user should also avoid fitting the pivot energies E_0 of spectral models. The pivot energy is not an independent parameter of the spectral models, and consequently, in case all other spectral parameters are free, the pivot energy is unconstrained. The pivot energy should therefore always be kept fixed, or other parameters of the spectral model need to be fixed to assure the non-degeneracy of the free model parameters.

Finally, we note that when the width of the radial shell model becomes comparable to or smaller than the angular resolution, the shell width tends to be overestimated while the shell radius tends to be underestimated. The fitted shell width and radius should thus not be over-interpreted when the width is close to the angular resolution of the IACT. Also we note that the convergence of the elliptical Gaussian model can be slow, and in some situations requires on the order of 20 iterations before the fit converges. Nevertheless, the numerical accuracy of the model fitting results for the elliptical Gaussian model is satisfactory.

4.2. Science verification

Science verification of GammaLib and ctools is performed by verifying that the pull distributions of all spatial and spectral models follow a standard Gaussian with mean zero and unit width. This verification is done using the `cspull` script. Figure 9 shows as an example the pull distribution of the prefactor and the index of a power law spectral model, obtained for 10 000 trials and based on a Crab-like point source observed during an observation duration of 30 min for an unbinned analysis.

Science verification is also actively ongoing within collaborations of existing IACTs. The confrontation of the software to real IACT data is fundamental in verifying the concepts that have been implemented. They will also play a key role in validating the handling of instrumental backgrounds using parametric models, and feed back into their improvements. Initial results are promising, but since the data is proprietary to the respective collaborations, we will not publish them here.

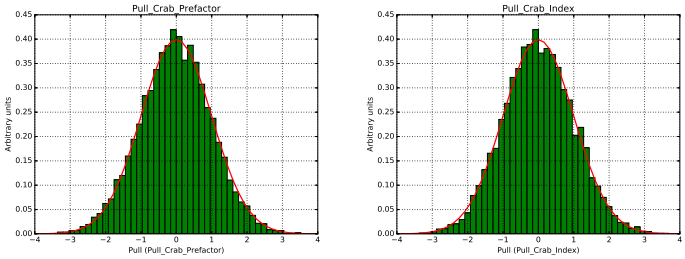

Fig. 9. Pull distributions for the prefactor and the index of a power law spectral model for an unbinned analysis.

Table 1. Benchmark of ctools unbinned, binned and stacked analysis pipelines as function of the observation duration.

Duration	Unbinned		Binned		Stacked	
	full	`ctlike`	full	`ctlike`	full	`ctlike`
(h)	(s)	(s)	(s)	(s)	(s)	(s)
0.5	7	1	41	35	35	28
1	8	2	41	35	35	27
2	9	3	42	35	36	28
4	13	6	43	36	36	27
8	17	8	44	36	38	28
16	28	16	48	37	42	29
32	51	31	53	37	48	29
64	94	61	64	37	60	29
128	184	124	86	38	86	30
256	359	244	129	38	131	31

Notes. Quoted numbers give the spent processor time in seconds. Processor times are quoted for the full pipeline and for the `ctlike` maximum likelihood fitting step only.

4.3. Benchmarking

Computing benchmarks of simple ctools analysis pipelines have been generated using the `benchmark_cta_analysis.py` script that is part of the ctools package. The benchmarks have been obtained on a Dell PowerEdge R815 server equipped with four 12-core AMD Opteron 6174 2.2 GHz processors running a CentOS 5 operating system (no parallel computing has been used in the pipeline setup). The unbinned pipeline executes in sequence the `ctobssim`, `ctselect`, and `ctlike` tools for a region of interest of 3°. The binned pipeline executes the `ctobssim`, `ctbin`, and `ctlike` tools for 200 × 200 spatial bins of 0.02° × 0.02° in size, and 40 energy bins. The stacked pipeline executes the `ctobssim`, `ctbin`, `ctexpcube`, `ctpsfcube`, `ctbkgcube`, and `ctlike` tools for the same binning. The source model consisted of a single point source with a power law spectrum on top of the instrumental background. Only the spectral parameters of the source and the background model have been fitted. The simulation was done for the southern CTA array using the `South_50h` IRF. Events in the energy range 100 GeV−100 TeV were simulated and analysed.

Table 1 summarises the spent processor time in seconds in each of the analysis pipelines for observation durations ranging from 30 min to 256 h. Figure 10 provides a graphical representation of the results, where solid lines give the total time spent in the pipelines while dashed lines represent the time spent in the `ctlike` tool only. The processor time for `ctlike` in a binned or stacked analysis is essentially independent of the duration of the observation, which is expected since the number of operations

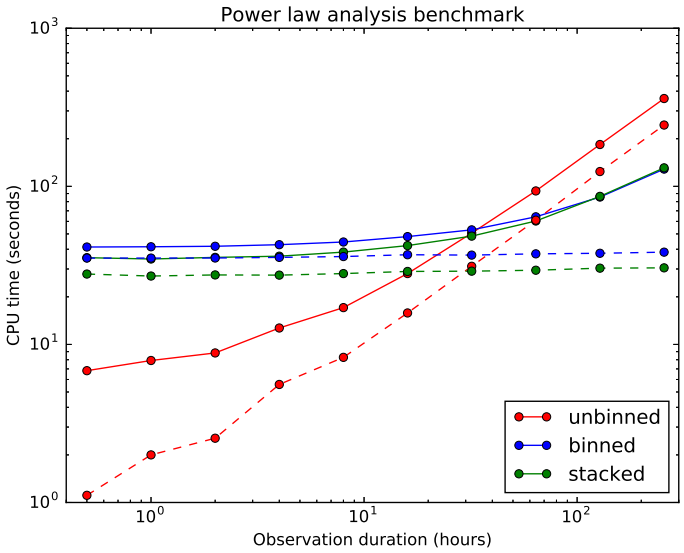

Fig. 10. Graphical representation of the ctools computing benchmark results for the fit of the spectrum of a single point source. Solid lines give the total time spent in the pipelines while dashed lines represent the time spent in the `ctlike` tool only.

for these analyses depend only on the number of bins and not the number of events (see Eq. (3)). In contrast, for an unbinned analysis the number of required operations increases with the number of events (see Eq. (1)), and the processor time increases accordingly. The crossover where the time spent in `ctlike` is equal for unbinned and binned or stacked analysis is for an observation duration of ∼30 h. The additional processor time for the full pipeline comes essentially from the event simulation step using `ctobssim`, which in all cases increases with the duration of the observing time. Above ∼60 h, the time spent in event simulation exceeds the time spent in maximum likelihood model fitting for a binned or stacked analysis. For unbinned analysis, the time spent in maximum likelihood fitting dominates for observation durations longer than ∼5 h.

Processor time benchmarks have also been obtained for all spectral and spatial models. Table 2 summarises the results for an observation duration of 32 h. The first part of the table provides benchmarks for various spectral models that have been combined with a point source spatial component with fixed position. While the performance of the stacked analysis is relatively insensitive of the spectral model that is used, unbinned and binned analyses show some variations. The broken power law, super exponentially cut off power law, and the log parabola models take longest in an unbinned analysis, making a stacked analysis more efficient for observation durations longer than 6−7 h. For the broken power law, an unbinned analysis takes substantially more fit iterations than a binned or a stacked analysis, which is related to the discontinuity of the model gradients mentioned above (cf. Sect. 4.1). The super exponentially cut-off power law and the log parabola model are expensive in the evaluation, which is a drawback for unbinned analysis where the model is evaluated for each event, in contrast to a binned or stacked analysis, where the model is effectively only evaluated once for each energy layer thanks to an internal value caching mechanism.

The second part of Table 2 summarises benchmarks for various spatial models that have been combined with a power law spectral model with free prefactor and index. All spatial parameters (position, size or orientation) have been adjusted by the fit.

Table 2. Benchmark of ctools unbinned, binned and stacked analysis pipelines for an observation duration of 32 h as function of the spectral and spatial model.

Model	Unbinned/stacked crossover duration (h)	Unbinned		Binned		Stacked	
		full (s)	`ctlike` (s)	full (s)	`ctlike` (s)	full (s)	`ctlike` (s)
Power law	30	51	31	53	37	48	29
Power law 2	25	53	33	57	40	49	29
Broken power law	7	201	183	76	60	54	35
Exponentially cut-off power law	22	65	40	60	39	54	30
Super exponentially cut-off power law	6	242	166	117	51	102	31
Log parabola	7	204	125	106	38	108	34
File function	30	50	31	55	39	51	31
Node function	33	48	30	57	41	53	35
Point source	30	97	78	127	111	89	70
Isotropic diffuse source	22	590	478	3515	3407	469	358
Sky map source ($4° \times 9°$ map)	0.5	613	595	4434	4419	43	25
Map cube source ($1° \times 1°$ map)	22	922	900	5053	5043	662	640
Radial disk source ($r = 0.2°$)	2	3289	3268	592	574	320	299
Radial Gaussian source ($\sigma = 0.2°$)	15	19 089	19 068	3404	3386	2683	2661
Radial shell source ($\theta_{\rm in} = 0.3°$, $\theta_{\rm out} = 0.4°$)	3	32 530	32 509	8439	8420	1327	1305
Elliptical disk source ($a = 0.2°$, $b = 0.4°$)	6	11 864	11 840	2324	2302	2893	2869
Elliptical Gaussian source ($a = 0.2°$, $b = 0.4°$)	3	154 337	154 316	6559	6540	9904	9882

Notes. The first column gives the model type (including the size parameters of the spatial model components), the second columns indicates the observation duration from which on a stacked analysis is faster than an unbinned analysis. The remaining numbers give the spent processor time in seconds. Processor times are quoted for the full pipeline and for the `ctlike` maximum likelihood fitting step only.

Fits of diffuse models take generally somewhat longer than fits of a point source, the only exception being the use of a sky map in a stacked analysis which is relatively fast due to an efficient internal caching of response values. We note that all diffuse model response computations benefit from an internal caching mechanism that evaluates the response only once for each event or event bin, taking advantage of the absence of spatial parameters that modify the emission morphology for diffuse models. Fits of radial or elliptical models take generally substantially longer than fits of a point source or of diffuse models, owing to the larger number of parameters that need to be adjusted. As spatial parameters have no analytical gradient, the derivatives have to be computed numerically, multiplying by three the number of function evaluations that are needed for each spatial parameter. Also, no simple response caching can be performed. The large number of iterations needed for the convergence of the elliptical Gaussian source (see Sect. 4.1) explains why the processing time is excessively large for this spatial model. Nevertheless, future code developments will aim at reducing the processing times for the radial and elliptical spatial models, as they will be central to many science analyses of IACT data.

For the time being, a stacked analysis is faster than an unbinned analysis for observation durations that are longer than a few hours, although the precise crossover duration is very model dependent. With the exception of the elliptical Gaussian source model, the `ctlike` processing time for a stacked analysis is always less than an hour, and can be as fast as 5 min for a radial disk model. A stacked analysis is therefore very often the method of choice for fitting radial or elliptical spatial models to IACT data.

5. Outlook

Following several years of development and testing, we have now released the GammaLib and ctools software packages for the scientific analysis of gamma-ray event data. So far, the software mainly targets the analysis of Imaging Air Cherenkov Telescope event data, but it can also be used for the analysis of *Fermi*-LAT data or the exploitation of the COMPTEL legacy archive.

Future developments will focus on the expansion of the ctools package to cover all features that are needed for a fully developed CTA science tools software package. This includes support for classical very-high-energy analysis techniques, such as aperture photometry and on-off fitting, the addition of tools for source extraction and identification, as well as for the timing analysis of pulsars and binaries. It is also planned to enhance the support for imaging analysis, for example by adding a tool that enables the spatial deconvolution of the data to produce super-resolved sky maps.

The GammaLib package will be expanded to also cover the analysis of Pass 8 data for *Fermi*-LAT, to enhance the COMPTEL module by an improved treatment of the instrumental background, and to implement full interoperability with Virtual Observatory tools. Including additional instrument modules to broaden the telescope coverage is also under investigation.

We want to conclude with the reminder that ctools and GammaLib are open source software packages that benefit from a dynamic and enthusiastic community of researchers. The projects are open to new contributions and enhancements, and if you have suggestions or ideas on how to expand the existing software, you are warmly welcomed to join the development team.

Acknowledgements. We would like to acknowledge highly valuable discussions with members of the CTA Consortium, the *Fermi*-LAT Collaboration, the H.E.S.S. Collaboration, the VERITAS Collaboration, and the MAGIC Collaboration that all contributed to improve and consolidate the GammaLib and ctools software packages. We also want to thank W. Collmar for making the COMPTEL instrument response functions available for inclusion into GammaLib. This work has been carried out thanks to the support of the OCEVU Labex (ANR-11-LABX-0060) and the A*MIDEX project (ANR-11-IDEX-0001-02) funded by the "Investissements d'Avenir" French government program managed by the ANR.

References

Acharya, B. S., Actis, M., Aghajani, T., et al. 2013, Astropart. Phys., 43, 3

Arnaud, K. A. 1996, in Astronomical Data Analysis Software and Systems V, eds. G. H. Jacoby, & J. Barnes, ASP Conf. Ser., 101, 17

Cash, W. 1979, ApJ, 228, 939

Corcoran, M. F., Angelini, L., George, I., et al. 1995, in Astronomical Data Analysis Software and Systems IV, eds. R. A. Shaw, H. E. Payne, & J. J. E. Hayes, ASP Conf. Ser., 77, 219

de Boer, H., Bennett, K., Bloemen, H., et al. 1992, in Data Analysis in Astronomy, eds. V. di Gesu, L. Scarsi, R. Buccheri, & P. Crane (Springer Verlag), 241

Diehl, R., Baby, N., Beckmann, V., et al. 2003, A&A, 411, L117

Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 759

Greisen, E. W., & Calabretta, M. R. 2002, A&A, 395, 1061

Hunter, J. D. 2007, Comput. Sci. Eng., 9, 90

Knödlseder, J., Mayer, M., Deil, C., et al. 2011, Astrophysics Source Code Library [record ascl:1110.007]

Knödlseder, J., Beckmann, V., Boisson, C., et al. 2015, ArXiv e-prints [arXiv:1508.06078]

Knödlseder, J., Mayer, M., Deil, C., et al. 2016, Astrophysics Source Code Library [record ascl:1601.005]

Marquardt, D. W. 1963, J. Soc. Industr. Appl. Math., 11, 431

Marsaglia, G., & Zaman, A. 1994, Comput. Phys., 8, 117

Mattox, J. R., Bertsch, D. L., Chiang, J., et al. 1996, ApJ, 461, 396

Pence, W., Blackburn, J. K., & Greene, E. 1993, in Astronomical Data Analysis Software and Systems II, eds. R. J. Hanisch, R. J. V. Brissenden, & J. Barnes, ASP Conf. Ser., 52, 541

Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., & Stobie, E. 2010, A&A, 524, A42

Appendix A: Installing the software

GammaLib is distributed as source code releases that can be downloaded from http://cta.irap.omp.eu/gammalib/download.html. Alternatively, the trunk of the source code can be accessed by cloning from the Git repository using

```
$ export GIT_SSL_NO_VERIFY=true
$ git clone https://cta-gitlab.irap.omp.eu/gammalib/gammalib.git
```

Building and installing the code follows the standard procedure for GNU software packages:

```
$ ./autogen.sh
$ ./configure
$ make
$ make check
$ [sudo] make install
```

The first command is only required when the source code has been cloned using Git, the (optional) `sudo` command is necessary when admin privileges are needed to install the library into the default `/usr/local/gamma` directory.

All instrument-specific modules are compiled by default when building the package, but can optionally be disabled during the configuration step. For instance

```
$ ./configure --without-lat --without-com --without-mwl
```

configures GammaLib to exclude the *Fermi*-LAT, COMPTEL and multi-wavelength modules.

The ctools package is distributed as source code releases that can be downloaded from http://cta.irap.omp.eu/ctools/download.html. Alternatively, the trunk of the source code can be accessed by cloning from the Git repository

```
$ export GIT_SSL_NO_VERIFY=true
$ git clone https://cta-gitlab.irap.omp.eu/ctools/ctools.git
```

Building and installing the code follows the same procedure as for GammaLib. To get started with GammaLib and ctools, refer to the "Quickstart" and "Using ctools from Python" sections of the user documentation at http://cta.irap.omp.eu/ctools.

Science verification is part of the continuous testing strategy implemented for ctools, and a user can exercise the standard science verification pipeline by typing the

```
$ make science-verification
```

command in the ctools source package after having installed and configured the software. Be aware, however, that this standard test will take more than a day for completion.

Appendix B: Observation definition XML format

The possibility of analysing data from different instruments with the same software framework opens up the opportunity to use GammaLib and ctools for multi-instrument and multi-wavelength analyses. To facilitate multi-instrument event data analysis, the `GObservations` class can directly load observations based on information provided in an observation definition XML file. In Python, the syntax for loading a observation definition XML file is

```
$ python
>>> import gammalib
>>> obs = gammalib.GObservations("myobservations.xml")
```

The format of the observation definition XML file is illustrated below:

```
<?xml version="1.0" standalone="no"?>
<observation_list title="observation library">
  <observation name="Crab" id="1" instrument="COM">
    <parameter name="DRE" file="m50439_dre.fits"/>
    <parameter name="DRB" file="m34997_drg.fits"/>
    <parameter name="DRG" file="m34997_drg.fits"/>
    <parameter name="DRX" file="m32171_drx.fits"/>
    <parameter name="IAQ" file="ENERG(1.0-3.0)MeV"/>
  </observation>
  <observation name="Crab" id="1" instrument="LAT">
    <parameter name="CountsMap"    file="srcmap.fits"/>
    <parameter name="ExposureMap"  file="expmap.fits"/>
    <parameter name="LiveTimeCube" file="ltcube.fits"/>
    <parameter name="IRF"          value="P7SOURCE_V6"/>
  </observation>
  <observation name="Crab" id="1" instrument="CTA">
    <parameter name="EventList"   file="cta_events.fits"/>
    <parameter name="Calibration" database="prod2"
                                  response="South_0.5h"/>
  </observation>
</observation_list>
```

The definition of the observations is enclosed in a single `<observation_list>` element that can contain an arbitrary number of `<observation>` elements. In the example above, three `<observation>` elements are present, and the `instrument` attribute specifies the instrument code for each of these observations. Based on that attribute, GammaLib appends instrument-specific observations to the `GObservations` container, and dispatches the reading of the `<observation>` XML elements to the respective `GObservation::read()` methods that interpret the instrument specific content.

Optionally, energy thresholds for a given CTA observation can be specified using the `emin` and `emax` attributes.

```
<observation name="Crab" id="1" instrument="CTA"
                                        emin="0.1" emax="100">
```

The units of the energy threshold are TeV. The `ctselect` tool will automatically apply these energy thresholds to the data if the parameter `usethres`=USER is specified.

Appendix C: Model definition XML format

Models can be defined by manipulating the C++ or Python model classes, but alternatively a model definition can be specified in form of an XML file that can directly be loaded by the `GModels` class. In Python, the syntax for loading a model definition XML file is

```
$ python
>>> import gammalib
>>> models = gammalib.GModels("mymodels.xml")
```

The format of the model definition XML file is inspired from, and is compatible with, the format used by the *Fermi*-LAT science tools[19]. The general structure of a model definition XML file is

```
<?xml version="1.0" standalone="no"?>
<source_library title="source library">
  <source name="Crab" type="PointSource">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" .../>
      <parameter name="Index"     .../>
      <parameter name="Scale"     .../>
```

[19] http://fermi.gsfc.nasa.gov/ssc/

```
    </spectrum>
    <spatialModel type="SkyDirFunction">
      <parameter name="RA"  .../>
      <parameter name="DEC" .../>
    </spatialModel>
  </source>
  <source name="Bkg" type="CTAIrfBackground" instrument="CTA">
    <spectrum type="PowerLaw">
      <parameter name="Prefactor" .../>
      <parameter name="Index"     .../>
      <parameter name="Scale"     .../>
    </spectrum>
  </source>
</source_library>
```

The collection of models is contained in a single `<source_library>` element that in turn can contain an arbitrary number of `<source>` elements. In the example above, two `<source>` elements are present which each correspond to one model. The `type` attribute of each `<source>` element allows GammaLib to identify the appropriate model class. In the example, the first model is a celestial point source model with a power law spectral component, and the second model is an instrumental background model for CTA. Each model component contains a number of model parameters of the form

```
<parameter name=".." scale=".." value=".." min=".." max=".."
                                                 free=".."/>
```

where

- `name` is the model parameter name that is unique within a model component;
- `scale` is a scale factor that will be multiplied with `value` to provide a model parameter (this pre-scaling allows all `value` attributes to be of the same order, which is needed to guarantee the numerical stability when the model is fitted to the data);
- `value` is the pre-scaled model value (see above);
- `min` is the lower limit for the `value` term;
- `max` is the upper limit for the `value` term;
- `free` is a flag that specifies whether a model parameter should be fitted (`free="1"`) or kept fixed (`free="0"`) when then model is fitted to the data.

Please refer to http://cta.irap.omp.eu/gammalib/user_manual/modules/model.html for a detailed description of the syntax of the model definition XML file for each model type.

Appendix D: IRAF command-line parameter interface

GammaLib implements the IRAF command line parameter interface, a format for specifying application parameters that is already widely used for high-energy astronomy analysis frameworks, including ftools, the *Chandra* CIAO package, the INTEGRAL OSA software, or the *Fermi*-LAT science tools. Following the IRAF standard, user parameters are specified in a so-called parameter file, which is an ASCII file that lists one parameter per row in the format

```
name, type, mode, value, minimum, maximum, prompt
```

where

- `name` specifies a unique user parameter name.
- `type` specifies the type of the user parameter and is one of "b", "i", "r", "s", "f", "fr", "fw", "fe", "fn", which stands for boolean, integer, real (or floating point), string, and file name. The various file name types test for read access, write access, file existence, and file absence, respectively.
- `mode` specifies the user parameter mode and is one of "a", "h", "l", "q", "hl", "ql", "lh", "lq", where "a" stands for "automatic", "h" for "hidden", "l" for "learn", and "q" for "query" (the other possibilities are combinations of modes); "automatic" means that the mode is inferred from the "mode" parameter in the parameter file, "hidden" means that the parameter is not queried, but can be specified by explicitly setting its value on the command line or from within Python, "l" means that the value entered by the user will be written to the parameter file on disk, so that the next time it will be used as the default value, "q" means that the parameter will be queried.
- `value` specifies the user parameter value. Possible values for boolean parameters are case-insensitive "y", "n", "yes", "no", "t", "f", "true", and "false". The special case-insensitive values "indef", "none", "undef" or "undefined" are used to signal that a value is not defined. If "inf", "infinity" or "nan" is specified and the parameter is of integer type, its value will be set to the largest possible integer value. If the parameter is of floating point type, its status will be set to "Not A Number".
- `minimum` specifies either a list of options, or the minimum user parameter value for integer or floating point types, provided that `maximum` is also specified. Options are separated by "|" characters.
- `maximum` specifies for integer or floating point types the maximum user parameter value, provided that `minimum` is also specified.
- `prompt` specifies the text that will be prompted when querying for user parameters.

User parameters can be either queried interactively or can be specified on the command line which inhibits an application to query the specified parameters. Command line parameters are given by specifying the parameter name, followed without whitespace by an equality symbol and the parameter value (hyphens are not needed). For example

```
$ ctobssim infile=myfilefits ra=83.0 dec=22.0
```

will set the `infile`, `ra`, and `dec` parameters of the `ctobssim` tool. Using

```
$ ctobssim debug=yes
```

will instruct the `ctobssim` to log any output into the console. And help text about `ctobssim` can be displayed using

```
$ ctobssim --help
```