Gammapy: A Python package for gamma-ray astronomy

Paper Authors, Axel Donath[28], Régis Terrier[20], Quentin Remy[17], Atreyee Sinha[15], Cosimo Nigro[24], Fabio Pintore[26], Bruno Khélifi[20], Laura Olivera-Nieto[17], Jose Enrique Ruiz[23],
Coordination Committee⋆, Fabio Acero[1], David Berge[6,7], Catherine Boisson[9], Jose Louis Contreras[15], Axel Donath[28], Stefan Funk[19], Christopher van Eldik[19], Matthias Füßling[18], Jim Hinton[17], Bruno Khélifi[20], Rubén López-Coto[23], Fabio Pintore[26], Régis Terrier[20], Roberta Zanin[18],
Gammapy Project Contributors, Arnau Aguasca-Cabot[2,3,4,5], Pooja Bhattacharjee[8], Kai Brügge[10,11], Johannes Buchner[12], David Carreto Fidalgo[13], Andrew Chen[14], Mathieu de Bony de Lavergne[8], José Vinícius de Miranda Cardoso[16], Christoph Deil[17], Luca Giunti[20], Léa Jouvin[32], Johannes King[30,17], Julien Lefaucheur[21,20], Marianne Lemoine-Goumard[31], Jean-Philippe Lenain[22], Maximilian Linhoff[11], Lars Mohrmann[17], Daniel Morcuende[15], Sebastian Panny[25], Maxime Regeard[20], Lab Saha[15], Hubert Siejkowski[27], Aneta Siemiginowska[28], Brigitta M Sipőcz[29], Tim Unbehaun[19], and Thomas Vuillaume[8]

(Affiliations can be found after the references)

March 21, 2023

ABSTRACT

Context. Traditionally, TeV-γ-ray astronomy has been conducted by experiments employing proprietary data and analysis software. However, the next generation of γ-ray instruments, such as the Cherenkov Telescope Array Observatory (CTAO), will be operated as open observatories. Alongside the data, they will also make associated software tools available to a wider community. This necessity prompted the development of open, high-level astronomy software customised for high-energy astrophysics.
Aims. In this article, we present Gammapy, an open-source Python package for the analysis of astronomical γ-ray data, and illustrate the functionalities of its first long-term-support release, version 1.0. Built on the modern Python scientific ecosystem, Gammapy provides a uniform platform for reducing and modelling data from different γ-ray instruments for many analysis scenarios. Gammapy complies with several well-established data conventions in high-energy astrophysics, providing serialised data products that are interoperable with other software packages.
Methods. Starting from event lists and instrument response functions, Gammapy provides the functionalities for reducing data binned in energy and sky coordinates. Several techniques for background estimation are implemented in the package to handle the residual hadronic background. After the data are binned, the flux and morphology of one or more γ-ray sources can be estimated using Poisson maximum likelihood fitting and assuming a variety of spectral, temporal, and spatial models. Estimation of flux points, likelihood profiles and light curves is also supported.
Results. After describing the structure of the package, we show the capabilities of Gammapy in multiple traditional and novel γ-ray analysis scenarios using public data, such as spectral and spectro-morphological modelling and estimations of a spectral energy distribution and a light curve. Its flexibility and its power are displayed in a final multi-instrument example, where datasets from different instruments, at different stages of data reduction, are simultaneously fitted with an astrophysical flux model.

Key words. Gamma rays: general - Astronomical instrumentation, methods and techniques - Methods: data analysis

1. Introduction

The γ-ray range of the electromagnetic spectrum provides us insights into the most energetic processes in the universe such as those accelerating particles in the surroundings of black holes, and remnants of supernova explosions. As in other branches of astronomy, γ-rays can be observed by satellite as well as ground-based instruments. Ground-based instruments use the Earth's atmosphere as a particle detector. Very-high-energy (VHE) cosmic γ-rays interact in the atmosphere and create large showers of secondary particles that can be observed from the ground. Ground-based γ-ray astronomy relies on these extensive air showers to detect the primary γ-ray photons and infer their incident direction and energy. VHE γ-ray astronomy covers the energy range from few tens of GeV up to the PeV. There are two main categories of ground-based instruments:

Imaging Atmospheric Cherenkov Telescopes (IACTs) obtain images of the atmospheric showers by detecting the Cherenkov radiation emitted by charged particles in the cascade and use these images to reconstruct the properties of the incident particle. Those instruments have a limited field of view (FoV) and duty cycle, but good energy and angular resolution.

Water Cherenkov Detectors (WCDs) detect particles directly from the tail of the shower when it reaches the ground. These instruments have a very large FoV, and large duty-cycle, but a higher energy threshold and lower signal-to-noise ratios compared to IACTs (de Naurois & Mazin 2015).

Ground-based γ-ray astronomy has been historically conducted through experiments operated by independent

⋆ Corresponding author: GAMMAPY-COORDINATION-L@IN2P3.FR

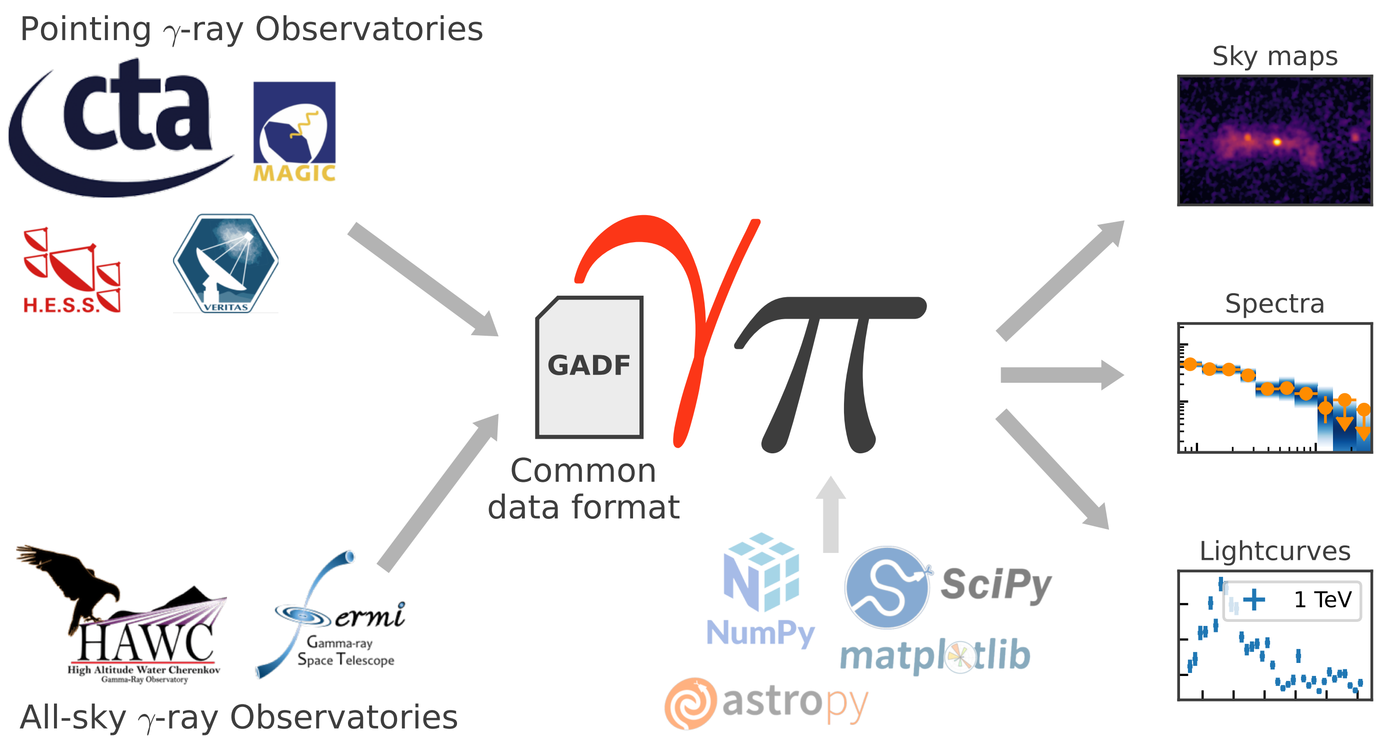

Fig. 1. Core idea and relation of Gammapy to different γ-ray instruments and the gamma astro data format (GADF). The top left shows the group of current and future pointing instruments based on the imaging atmospheric Cherenkov technique (IACT). This includes instruments such as the Cherenkov Telescope Array Observatory (CTAO), the High Energy Stereoscopic System (H.E.S.S.), the Major Atmospheric Gamma Imaging Cherenkov Telescopes (MAGIC), and the Very Energetic Radiation Imaging Telescope Array System (VERITAS). The lower left shows the group of all-sky instruments such as the Fermi Large Area Telescope (Fermi-LAT) and the High Altitude Water Cherenkov Observatory (HAWC). The calibrated data of all those instruments can be converted and stored into the common GADF data format. Gammapy can read data stored in the GADF format. The Gammapy package is a community-developed project that provides a common interface to the data and analysis of all these γ-ray instruments. This way users can also easily combine data from different instruments and perform a joint analysis. Gammapy is built on the scientific Python ecosystem, and the required dependencies are shown below the Gammapy logo.

collaborations, each relying on their own proprietary data and analysis software developed as part of the instrument. While this model has been successful so far, it does not permit easy combination of data from several instruments and therefore limits the interoperability of existing facilities. This lack of interoperability currently limits the full exploitation of the available γ-ray data, especially because the different instruments often have complementary sky coverages, and the various detection techniques have complementary properties in terms of energy range covered, duty cycle and spatial resolution.

The Cherenkov Telescope Array Observatory (CTAO) will be the first ground-based γ-ray instrument to be operated as an open observatory. Its high-level data[1] will be shared publicly after some proprietary period, and the software required to analyze it will be distributed as well. To allow the re-usability of data from existing instruments and their interoperability, it is required to use open data formats and open tools that can support the various analysis methods commonly used in the field.

In practice, the data reduction workflow of all γ-ray observatories is remarkably similar. After data calibration, shower events are reconstructed and gamma/hadron separation is applied to build lists of γ-ray-like events. The lists of γ-ray events are then used to derive scientific results, such as spectra, sky maps or light curves, taking into account the observation specific instrument response functions (IRFs). Once the data is reduced to a list of events with reconstructed physical properties of the primary particle, the information is independent of the data-reduction process, and, eventually, of the detection technique. This implies, for example, that high-level data from IACTs and WCDs can be represented with the same data model. The efforts to create a common format usable by various instruments converged in the so-called *Data Formats for γ-ray Astronomy* initiative (Deil et al. 2017; Nigro et al. 2021), abbreviated to `gamma-astro-data-formats` (GADF). This proposes prototypical specifications to produce files based on the flexible image transport system (FITS) format (Pence et al. 2010) encapsulating this high-level information. This is realized by storing a list of γ-ray-like events with their reconstructed and observed quantities such as energy, incident direction and arrival time and a parametrisation of the IRFs associated with the event list data.

In the past decade observing the γ-ray sky has transitioned from a niche in the field of particle physics to an established branch of astronomy, completing the view of the sky in high energies. At the same time *Python* has become

[1] The lowest reduction level of data published by CTAO will be reconstructed event lists and corresponding instrument response functions.

extremely popular as a scientific programming language, in particular in the field of data sciences. This success is mostly attributed to the simple and easy to learn syntax, the ability to act as a "glue" language between different programming languages and last but not least the rich ecosystem of packages and its open and supportive community (Momcheva & Tollerud 2015).

In the sub-field of astronomy, the Astropy project (Astropy Collaboration et al. 2013) was created in 2012 to build a community-developed core Python package for astronomy. It offers basic functionalities that astronomers of many fields need, such as representing and transforming astronomical coordinates, manipulating physical quantities including units as well as reading and writing FITS files.

The Gammapy project was started following the model of Astropy, with the objective of building a common software library for very high-energy γ-ray data analysis (Donath et al. 2015). The core of the idea is illustrated in Figure 1. The various γ-ray instruments can export their data to a common data format (GADF) and then these data can be combined and analysed using a common software library. The Gammapy package is an independent community-developed software project, it has been selected to be the core library for the Science Analysis tools of CTAO but also involves contributors associated to other instruments. The Gammapy package is built on the scientific Python ecosystem: it uses Numpy (Harris et al. 2020) for n-dimensional data structures, Scipy (Virtanen et al. 2020) for numerical algorithms, Astropy (Astropy Collaboration et al. 2013) for astronomy-specific functionality, iminuit (Dembinski & et al. 2020) for numerical minimisation and Matplotlib (Hunter 2007) for visualization.

With the public availability of the GADF format specifications and the Gammapy package, some experiments started to make limited subsets of their γ-ray data publicly available for testing and validating Gammapy. For example, the H.E.S.S. collaboration released a limited test dataset (about 50 hours of observations taken between 2004 and 2008) based on the GADF DL3 format (H.E.S.S. Collaboration 2018a). This data release served as a basis for validation of open analysis tools, including Gammapy (see e.g. Mohrmann et al. 2019). The HAWC collaboration also released a limited test dataset of the Crab Nebula, which was used to validate the Gammapy package in Albert, A. et al. (2022). The increased availability of public data that followed the definition of a common data format, and the development of Gammapy as a community-driven open software, led the way toward a more open science in the very-high-energy γ-ray Astronomy domain. In future CTAO will be an open observatory committed to follow the FAIR (Findable, Accessible, Interoperable and Reusable) principles (Wilkinson et al. 2016; Barker et al. 2022) that define the key requirements for open science.

In this article, we describe the general structure of the Gammapy package, its main concepts and organisational structure. We start in Section 2 with a general overview of the data analysis workflow in very high-energy γ-ray astronomy. Then we show how this workflow is reflected in the structure of the Gammapy package in Section 3, while also describing the various subpackages it contains. Section 4 presents a number of applications, while Section 5 finally discusses the project organization.

2. Gamma-ray Data Analysis

The data analysis process in γ-ray astronomy is usually split into two parts. The first one deals with the data processing from detector measurement, calibration, event reconstruction and selection to yield a list of reconstructed γ-ray event candidates. This part of the data reduction sequence, sometimes referred to as low-level analysis, is usually very specific to a given observation technique and even to a given instrument.

The other sequence, referred to as high-level analysis, deals with the extraction of physical quantities related to γ-ray sources and the production of high-level products such as spectra, light curves and catalogs. The methods applied here are more generic and are broadly shared across the field. The similarity in the high-level analysis would also allow for combining data from multiple instruments.

To extract physically relevant information, such as the flux, spatial or spectral shape of one or more sources, an analytical model is commonly adopted to describe the intensity of the radiation from gamma-ray sources as a function of the energy, $E_{\rm true}$, and of the position in the FoV, $p_{\rm true}$:

$$\Phi(p_{\rm true}, E_{\rm true}; \hat{\theta}), [\Phi] = \mathrm{TeV}^{-1}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1} \quad (1)$$

where $\hat{\theta}$ is a set of model parameters that can be adjusted in a fit. To convert this analytical flux model into a prediction on the number of gamma-ray events detected by an instrument, $N_{\rm pred}$, with their estimated energy E and position p, the model is convolved with the response function of the instrument.

In the most general way, we can write the expected number of detected events from the sky model Φ at measured position p and energy E, for a given set of parameters $\hat{\theta}$, as:

$$\begin{aligned} N(p, E, \hat{\theta})\,\mathrm{d}p\,\mathrm{d}E = & t_{\rm obs} \int_{E_{\rm true}} \int_{p_{\rm true}} R(p, E|p_{\rm true}, E_{\rm true}) \\ & \cdot \Phi(p_{\rm true}, E_{\rm true}, \hat{\theta}) \mathrm{d}E_{\rm true} \mathrm{d}p_{\rm true} \end{aligned} \quad (2)$$

where $R(p, E|p_{\rm true}, E_{\rm true})$ is the instrument response and $t_{\rm obs}$ is the observation duration.

A common assumption is that the instrument response can be simplified as the product of three independent functions:

$$\begin{aligned} R(p, E|p_{\rm true}, E_{\rm true}) = & A_{\rm eff}(p_{\rm true}, E_{\rm true}) \\ & \cdot PSF(p|p_{\rm true}, E_{\rm true}) \\ & \cdot E_{\rm disp}(E|p_{\rm true}, E_{\rm true}) \end{aligned} \quad (3)$$

where:

- $A_{\rm eff}(p_{\rm true}, E_{\rm true})$ is the effective collection area of the detector. It is the product of the detector collection area times its detection efficiency at true energy $E_{\rm true}$ and position $p_{\rm true}$.

- $PSF(p|p_{\rm true}, E_{\rm true})$ is the point spread function (PSF). It gives the probability density of measuring a direction p when the true direction is $p_{\rm true}$ and the true energy is $E_{\rm true}$. γ-ray instruments typically consider radial symmetry of the PSF. With this assumption the probability density $PSF(\Delta p|p_{\rm true}, E_{\rm true})$ only depends on the angular separation between true and reconstructed direction defined by $\Delta p = p_{\rm true} - p$.

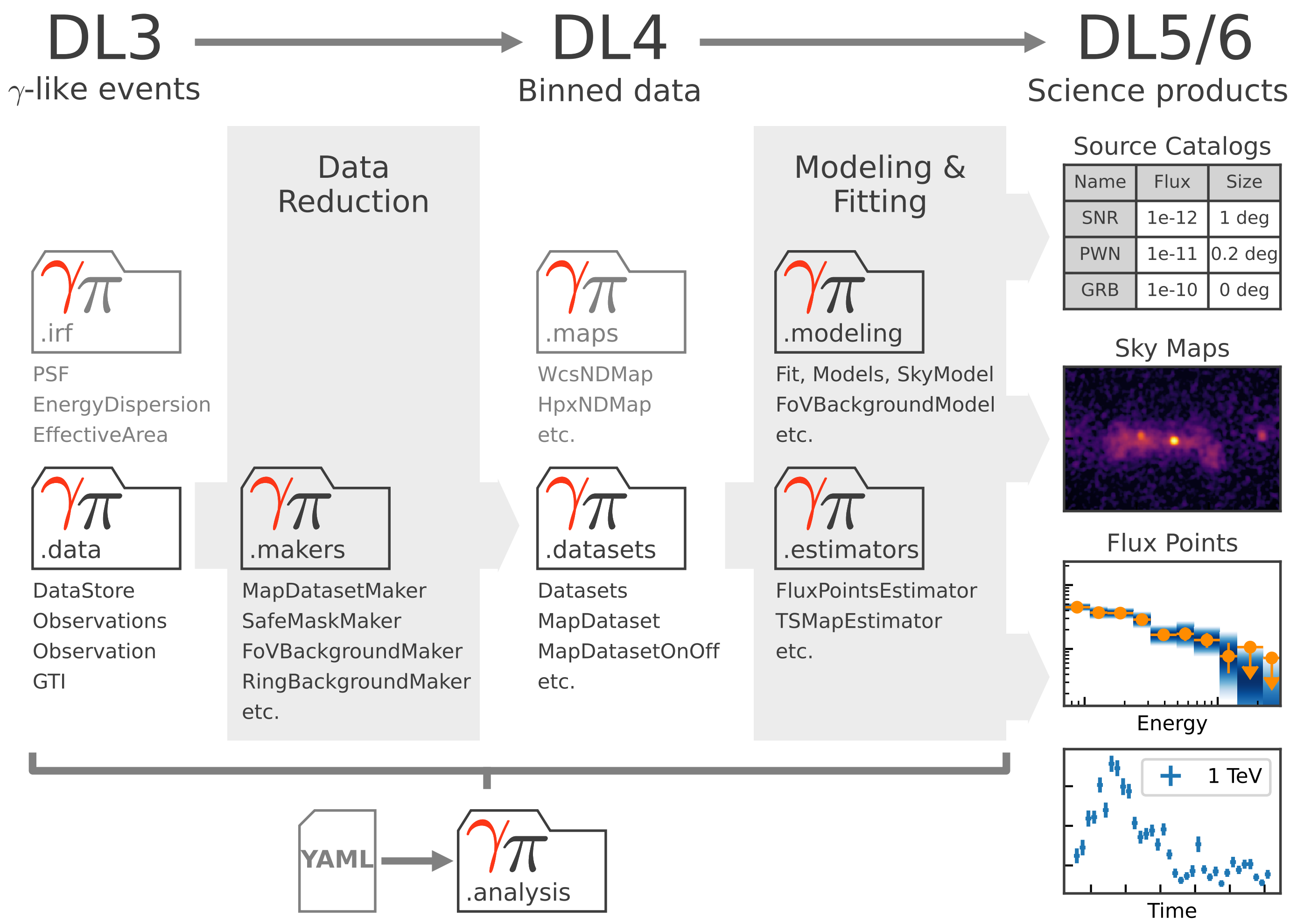

Fig. 2. Gammapy sub-package structure and data analysis workflow. The top row defines the different levels of data reduction, from lists of γ-ray-like events on the left (DL3), to high-level scientific products (DL5) on the right. The direction of the data flow is illustrated with the gray arrows. The gray folder icons represent the different sub-packages in Gammapy and names given as the corresponding Python code suffix, e.g. `gammapy.data`. Below each icon there is a list of the most important objects defined in the sub-package. The light grey folder icons show the subpackages for the most fundamental data structures such as maps and IRFs. The bottom of the figure shows the high-level analysis sub-module with its dependcy on the YAML file format.

– $E_{\rm disp}(E|p_{\rm true}, E_{\rm true})$ is the energy dispersion. It gives the probability to reconstruct the photon at energy E when the true energy is $E_{\rm true}$ and the true position $p_{\rm true}$. γ-ray instruments consider $E_{\rm disp}(\mu|p_{\rm true}, E_{\rm true})$, the probability density of the event migration, $\mu = \frac{E}{E_{\rm true}}$.

γ-ray data at the Data Level 3 (DL3) therefore consist of lists of γ-ray-like events and their corresponding instrument response functions. The latter include the effective area ($A_{\rm eff}$), PSF and energy dispersion ($E_{\rm disp}$). In general, IRFs depend on the geometrical parameters of the detector, e.g. location of an event in the FoV or the elevation angle of the incoming direction of the event. Consequently IRFs might be parametrised as functions of detector specific coordinates too.

An additional component of DL3 IRFs is the residual hadronic background model Bkg. It represents the intensity of charged particles misidentified as γ-rays that are expected during an observation. It is defined as a function of the measured position in the FoV and measured energy.

In total, the expected number of events in a γ-ray observation is given by:

$$N(p, E; \hat{\theta})\ {\rm d}p\ {\rm d}E = E_{\rm disp} \circledast \left[PSF \circledast \left(A_{\rm eff} \cdot t_{\rm obs} \cdot \Phi(\hat{\theta}) \right) \right] + Bkg(p, E) \cdot t_{\rm obs} \tag{4}$$

Finally, predicted and observed events, N_{obs}, can be combined in a likelihood function, $\mathcal{L}(\hat{\theta}, N_{obs})$, usually Poissonian, that is maximised to obtain the best-fit parameters of the flux model, $\hat{\theta}$.

2.1. Gammapy data analysis workflow

The first step in γ-ray data analysis is the selection and extraction of observations based on their metadata including information such as pointing direction, observation time and observation conditions. The access to the events data and instrument reponse per observation is supported by classes and methods in the `gammapy.data` (see Section 3.2) and the `gammapy.irf` (see Section 3.3) subpackages.

The next step of the analysis is the data reduction, where all observation events and instrument responses are filled into or projected onto a common physical coordinate system, defined by a map geometry. The definition of the map geometry typically consists of a spectral dimension defined by a binned energy axis and of spatial dimensions, which either define a spherical projection from celestial coordinates to a pixelised image space or a single region on the sky. The `gammapy.maps` subpackage provides general multidimensional geometry objects and the associated data structures (see Section 3.4).

After all data have been projected into the same geometry, it is typically required to improve the residual hadronic background estimate. As residual hadronic background models can be subject to significant systematic uncertainties, these models can be improved by taking into account actual data from regions without known γ-ray sources. This includes methods such as the ring or the FoV background techniques or background measurements performed within, e.g. reflected regions (Berge et al. 2007). Data measured at the FoV or energy boundaries of the instrument are typically associated with a systematic uncertainty in the IRF. For this reason this part of the data is often excluded from subsequent analysis by defining regions of "safe" data in the spatial as well as energy dimension. All of these data reduction steps are performed by classes and functions implemented in the `gammapy.makers` subpackage (see Section 3.6).

The counts data and the reduced IRFs in the form of maps are bundled into datasets that represent the fourth data level (DL4). These reduced datasets can be written to disk, in a format specific to Gammapy to allow users to read them back at any time later for modelling and fitting. Different variations of such datasets support different analysis methods and fit statistics. The datasets can be used to perform a joint-likelihood fit, allowing one to combine different measurements, e.g. from different observations but also from different instruments or event classes. They can also be used for binned simulation as well as event sampling to simulate DL3 events data. The various DL4 objects and the associated functionalities are implemented in the `gammapy.datasets` subpackage (see Section 3.5).

The next step is then typically to model and fit the datasets, either individually, or in a joint likelihood analysis. For this purpose Gammapy provides a uniform interface to multiple fitting backends. In addition to providing a variety of built-in models, including spectral, spatial and temporal model classes to describe the γ-ray emission in the sky, custom user-defined models are also supported. Spectral models can be simple analytical models or more complex ones from radiation mechanisms of accelerated particle populations (e.g. inverse Compton or π^0 decay). Independently or subsequently to the global modelling, the data can be re-grouped to compute flux points, light curves and flux maps as well as significance maps in different energy bands. The modelling and fitting functionalities are implemented in the `gammapy.modeling`, `gammapy.estimators` and `gammapy.stats` subpackages (see respectively Section 3.8, 3.9 and 3.7).

3. Gammapy Package

3.1. Overview

The Gammapy package is structured into multiple sub-packages. The definition of the content of the different sub-packages follows mostly the stages of the data reduction workflow described in the previous section. Sub-packages either contain structures representing data at different reduction levels or algorithms to transition between these different levels.

Figure 2 shows an overview of the different sub-packages and their relation to each other. The `gammapy.data` and `gammapy.irf` sub-packages define data objects to represent DL3 data, such as event lists and IRFs as well as functionality to read the DL3 data from disk into memory. The `gammapy.makers` sub-package contains the functionality to reduce the DL3 data to binned maps. Binned maps and datasets, which represent a collection of binned maps, are defined in the `gammapy.maps` and `gammapy.datasets` sub-packages, respectively. Parametric models, which are defined in `gammapy.modeling`, are used to jointly model a combination of datasets, for example, to compute a spectrum using data from several facilities. Estimator classes, which are contained in `gammapy.estimators`, are used to compute higher level science products such as flux and significance maps, light curves or flux points. Finally there is a `gammapy.analysis` sub-package which provides a high-level interface for executing analyses defined from configuration files. In the following sections, we will introduce all sub-packages and their functionalities in more detail.

3.2. gammapy.data

The `gammapy.data` sub-package implements the functionality to select, read, and represent DL3 γ-ray data in memory. It provides the main user interface to access the lowest data level. Gammapy currently only supports data that is compliant with `v0.2` and `v0.3` of the GADF data format. DL3 data are typically bundled into individual observations, corresponding to stable periods of data acquisition. For IACT data analysis, for which the GADF data model and Gammapy were initially conceived, these are usually $20 - 30$ min long. Each observation is assigned a unique integer ID for reference.

A typical usage example is shown in Figure 3. First a `DataStore` object is created from the path of the data directory. The directory contains an observation as well as a FITS HDU [2] index file which assigns the correct data and IRF FITS files and HDUs to the given observation ID. The `DataStore` object gathers a collection of observations and provides ancillary files containing information about the telescope observation mode and the content of the data unit of each file. The `DataStore` allows for selecting a list of observations based on specific filters.

The DL3 level data represented by the `Observation` class consist of two types of elements: first, a list of γ-ray events with relevant physical quantities such as estimated energy, direction and arrival times, which is represented by the `EventList` class. Second, a set of associated IRFs, providing the response of the system, typically factorised in independent components as described in Section 3.3. The separate handling of event lists and IRFs addition-

[2] Header Data Unit

```python
from gammapy.data import DataStore

data_store = DataStore.from_dir(
    base_dir="$GAMMAPY_DATA/hess-dl3-dr1"
)

obs_ids = [23523, 23526, 23559, 23592]

observations = data_store.get_observations(
    obs_id=obs_ids, skip_missing=True
)

for obs in observations:
    print(f"Observation id: {obs.obs_id}")
    print(f"N events: {len(obs.events.table)}")
    print(f"Max. area: {obs.aeff.quantity.max()}")
```

Fig. 3. Using `gammapy.data` to access DL3 level data with a `DataStore` object. Individual observations can be accessed by their unique integer observation id number. The actual events and instrument response functions can be accessed as attributes on the `Observation` object, such as `.events` or `.aeff` for the effective area information. The output of the code example is shown in Figure A.1.

ally allows for data from non-IACT γ-ray instruments to be read. For example, to read *Fermi*-LAT data, the user can read separately their event list (already compliant with the GADF specifications) and then find the appropriate IRF classes representing the response functions provided by *Fermi*-LAT, see example in Section 4.4.

3.3. gammapy.irf

The `gammapy.irf` sub-package contains all classes and functionalities to handle IRFs in a variety of formats. Usually, IRFs store instrument properties in the form of multi-dimensional tables, with quantities expressed in terms of energy (true or reconstructed), off-axis angles or cartesian detector coordinates. The main quantities stored in the common γ-ray IRFs are the effective area, energy dispersion, PSF and background rate. The `gammapy.irf` sub-package can open and access specific IRF extensions, interpolate and evaluate the quantities of interest on both energy and spatial axes, convert their format or units, plot or write them into output files. In the following, we list the main classes of the sub-package:

3.3.1. Effective Area

Gammapy provides the class `EffectiveAreaTable2D` to manage the effective area, which is usually defined in terms of true energy and offset angle. The class functionalities offer the possibility to read from files or to create it from scratch. The `EffectiveAreaTable2D` class can also convert, interpolate, write, and evaluate the effective area for a given energy and offset angle, or even plot the multi-dimensional effective area table.

3.3.2. Point Spread Function

Gammapy allows users to treat different kinds of PSFs, in particular, parametric multi-dimensional Gaussian distributions (`EnergyDependentMultiGaussPSF`) or King profile functions (`PSFKing`). The `EnergyDependentMultiGaussPSF` class is able to handle up to three Gaussians, defined in terms of amplitudes and sigma given for each true energy and offset angle bin. Similarly, `PSFKing` takes into account the gamma and sigma parameters. The general `ParametricPSF` class allows users to create a custom PSF with a parametric representation different from Gaussian(s) or King profile(s). The generic `PSF3D` class stores a radial symmetric profile of a PSF to represent non-parametric shapes, again depending on true energy and offset from the pointing position.

To handle the change of the PSF with the observational offset during the analysis the `PSFMap` class is used. It stores the radial profile of the PSF depending on the true energy and position on the sky. During the modelling step in the analysis, the PSF profile for each model component is looked up at its current position and converted into a 3d convolution kernel which is used for the prediction of counts from that model component.

3.3.3. Energy Dispersion

For IACTs, the energy resolution and bias, sometimes called energy dispersion, is typically parametrised in terms of the so-called migration parameter (μ), which is defined as the ratio between the reconstructed energy and the true energy. By definition, the mean of this ratio is close to unity for a small energy bias and its distribution can be typically described by a Gaussian profile. However, more complex shapes are also common. The migration parameter is given at each offset angle and reconstructed energy. The main sub-classes are the `EnergyDispersion2D` which is designed to handle the raw instrument description, and the `EDispKernelMap`, which contains an energy disperion matrix per sky position. I.e., a 4-dimensional sky map where each position is associated to an energy dispersion matrix. The energy dispersion matrix is a representation of the energy resolution as a function of the true energy only and implemented in Gammapy by the sub-class `EDispKernel`.

3.3.4. Instrumental Background

The instrumental background rate can be represented as either a 2-dimensional data structure named `Background2D` or a 3-dimensional one named `Background3D`. The background rate is stored as a differential count rate, normalised per solid angle and energy interval at different reconstructed energies and offset angles. In the `Background2D` case, the background is expected to follow a radially symmetric shape and changes only with the offset angle from FoV center. In the `Background3D` case, the background is allowed to vary with longitude and latitude of a tangential FoV coordinates system.

Some example IRFs read from public data files and plotted with Gammapy are shown in Figure 4.

3.4. gammapy.maps

The `gammapy.maps` sub-package provides classes that represent data structures associated with a set of coordinates or a region on a sphere. In addition it allows to handle an arbitrary number of non-spatial data dimensions, such as time or energy. It is organized around three types of struc-

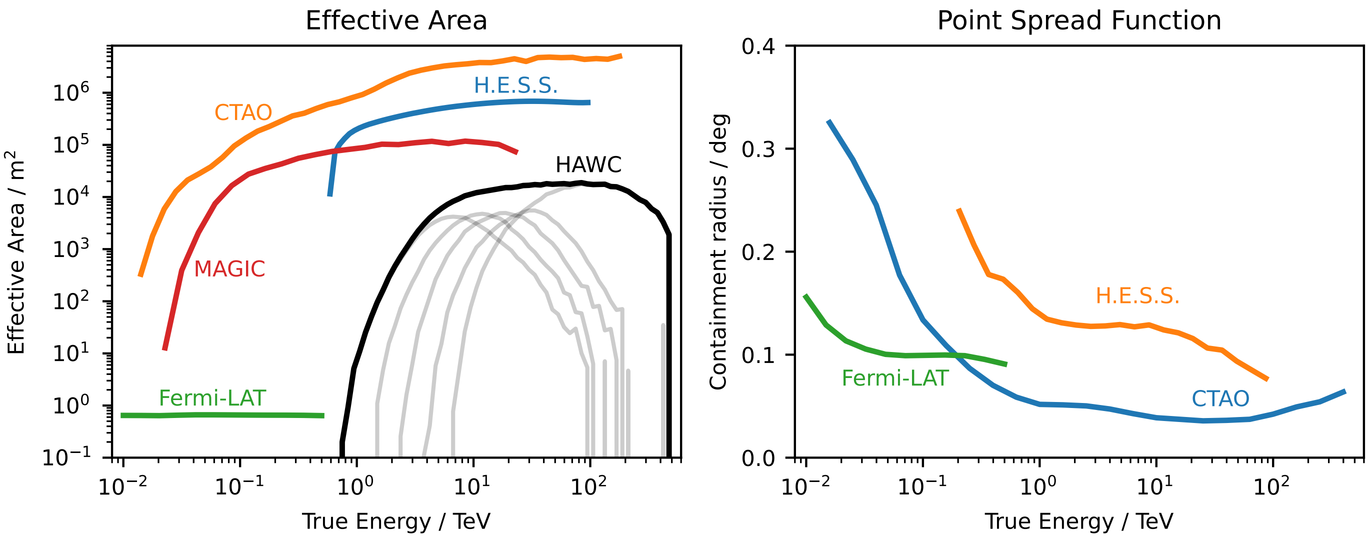

Fig. 4. Using `gammapy.irf` to read and plot instrument response functions. The left panel shows the effective area as a function of energy for the CTA, H.E.S.S., MAGIC, HAWC and *Fermi*-LAT instruments. The right panel shows the 68% containment radius of the PSF as a function of energy for the CTA, H.E.S.S. and *Fermi*-LAT instruments. The CTA IRFs are from the *prod5* production. The H.E.S.S. IRFs are from the DL3 DR1, using observation ID 033787. The MAGIC effective area is computed for a 20 min observation at the Crab Nebula coordinates. The *Fermi*-LAT IRFs use *pass8* data and are also taken at the position of the Crab Nebula. The HAWC effective area is shown for the event classes $N_{Hit} = 5 - 9$ as light gray lines along with the sum of all event classes as a black line. The HAWC IRFs are taken from the first public release of events data by the HAWC collaboration. All IRFs do not correspond to the latest performance of the instruments, but still are representative of the detector type and energy range. We exclusively relied on publicly available data provided by the collaborations. The data is also available in the `gammapy-data` repository.

tures: geometries, sky maps and map axes, which inherit from the base classes `Geom`, `Map` and `MapAxis` respectively.

The geometry object defines the pixelization scheme and map boundaries. It also provides methods to transform between sky and pixel coordinates. Maps consist of a geometry instance defining the coordinate system together with a Numpy array containing the associated data. All map classes support a basic set of arithmetic and boolean operations with unit support, up and downsampling along extra axes, interpolation, resampling of extra axes, interactive visualisation in notebooks and interpolation onto different geometries.

The `MapAxis` class provides a uniform application programming interface (API) for axes representing bins on any physical quantity, such as energy or angular offset. Map axes can have physical units attached to them, as well as define non-linearly spaced bins. The special case of time is covered by the dedicated `TimeMapAxis`, which allows time bins to be non-contiguous, as it is often the case with observational times. The generic class `LabelMapAxis` allows the creation of axes for non-numeric entries.

To handle the spatial dimension the sub-package exposes a uniform API for the FITS World Coordinate System (WCS), the HEALPix pixelization and region-based data structure (see Figure 5). This allows users to perform the same higher level operations on maps independent of the underlying pixelisation scheme. The `gammapy.maps` package is also used by external packages such as *FermiPy* (Wood et al. 2017)

3.4.1. WCS Maps

The FITS WCS pixelization supports a number of different projections to represent celestial spherical coordinates in a regular rectangular grid. Gammapy provides full support to data structures using this pixelization scheme. For details see Calabretta & Greisen (2002). This pixelisation is typically used for smaller regions of interests, such as pointed observations and is represented by a combination of the `WcsGeom` and `WcsNDMap` class.

3.4.2. HEALPix Maps

This pixelization scheme (Calabretta & Greisen 2002) provides a subdivision of a sphere in which each pixel covers the same surface area as every other pixel. As a consequence, however, pixel shapes are no longer rectangular, or regular. This pixelisation is typically used for all-sky data, such as data from the HAWC or *Fermi*-LAT observatory. Gammapy natively supports the multiscale definition of the HEALPix pixelisation and thus allows for easy upsampling and downsampling of the data. In addition to the all-sky map, Gammapy also supports a local HEALPix pixelisation where the size of the map is constrained to a given radius. For local neighbourhood operations, such as convolution, Gammapy relies on projecting the HEALPix data to a local tangential WCS grid. This data structure is represented by the `HpxGeom` and `HpxNDMap` classes.

3.4.3. Region Maps

In this case, instead of a fine spatial grid dividing a rectangular sky region, the spatial dimension is reduced to a single bin with an arbitrary shape, describing a region in the sky with that same shape. Region maps are typically used together with a non-spatial dimension, for example an energy axis, to represent how a quantity varies in that dimension inside the corresponding region. To avoid the

```python
from gammapy.maps import Map, MapAxis
from astropy.coordinates import SkyCoord
from astropy import units as u

skydir = SkyCoord("0d", "5d", frame="galactic")

energy_axis = MapAxis.from_energy_bounds(
    energy_min="1 TeV", energy_max="10 TeV", nbin=10
)

# Create a WCS Map
m_wcs = Map.create(
    binsz=0.1,
    map_type="wcs",
    skydir=skydir,
    width=[10.0, 8.0] * u.deg,
    axes=[energy_axis])


# Create a HEALPix Map
m_hpx = Map.create(
    binsz=0.1,
    map_type="hpx",
    skydir=skydir,
    axes=[energy_axis]
)

# Create a region map
region = "galactic;circle(0, 5, 1)"
m_region = Map.create(
    region=region,
    map_type="region",
    axes=[energy_axis]
)

print(m_wcs, m_hpx, m_region)
```

Fig. 5. Using `gammapy.maps` to create a WCS, a HEALPix and a region based data structures. The initialisation parameters include consistently the positions of the center of the map, the pixel size, the extend of the map as well as the energy axis definition. The energy minimum and maximum values for the creation of the `MapAxis` object can be defined as strings also specifying the unit. Region definitions can be passed as strings following the DS9 region specifications `http://ds9.si.edu/doc/ref/region.html`. The output of the code example is shown in Figure A.3.

complexity of handling spherical geometry for regions, the regions are projected onto the local tangential plane using a WCS transform. This approach follows Astropy's *Regions* package (Bradley et al. 2022), which is both used as an API to define regions for users as well as handling the underlying geometric operations. Region based maps are represented by the `RegionGeom` and `RegionNDMap` classes.

3.5. gammapy.datasets

The `gammapy.datasets` subpackage contains classes to bundle together binned data along with the associated models and likelihood function, which provides an interface to the `Fit` class (Sec 3.8.2) for modelling and fitting purposes. Depending upon the type of analysis and the associated statistic, different types of Datasets are supported. The `MapDataset` is used for combined spectral and morphological (3D) fitting, while a 1D spectral fitting can be performed using the `SpectrumDataset`. While the default fit statistics for both of these classes is the *Cash* (Cash 1979) statistic, there are other classes which support analyses where the background is measured from control regions, so called "off" obervations. Those require the use of a different fit statistics, which takes into account the uncertainty of the background measurement. This case is covered by the `MapDatasetOnOff` and `SpectrumDatasetOnOff` classes, which use the *WStat* (Arnaud et al. 2022) statistic.

```python
from pathlib import Path

from gammapy.datasets import (
    Datasets,
    FluxPointsDataset,
    MapDataset,
    SpectrumDatasetOnOff,
)

path = Path("$GAMMAPY_DATA")

map_dataset = MapDataset.read(
    path / "cta-1dc-gc/cta-1dc-gc.fits.gz",
    name="map-dataset",
)

spectrum_dataset = SpectrumDatasetOnOff.read(
    path / "joint-crab/spectra/hess/pha_obs23523.fits",
    name="spectrum-datasets",
)

flux_points_dataset = FluxPointsDataset.read(
    path / "hawc_crab/HAWC19_flux_points.fits",
    name="flux-points-dataset",
)


datasets = Datasets([
    map_dataset,
    spectrum_dataset,
    flux_points_dataset
])

print(datasets["map-dataset"])
```

Fig. 6. Using `gammapy.datasets` to read existing reduced binned datasets. After the different datasets are read from disk they are collected into a common `Datasets` container. All dataset types have an associated name attribute to allow a later access by name in the code. The environment variable `$GAMMAPY_DATA` is automtically resolved by Gammapy. The output of the code example is shown in Figure A.2.

The predicted counts are computed by convolution of the models with the associated IRFs. Fitting of precomputed flux points is enabled through `FluxPointsDataset`, using χ^2 statistics. Multiple datasets of same or different types can be bundled together in `Datasets` (e.g., Figure 6), where the likelihood from each constituent member is added, thus facilitating joint fitting across different observations, and even different instruments across different wavelengths. Datasets also provide functionalities for manipulating reduced data, e.g. stacking, sub-grouping, plotting. Users can also create their customized datasets for implementing modified likelihood methods.

```python
import astropy.units as u

from gammapy.data import DataStore
from gammapy.datasets import MapDataset
from gammapy.makers import (
    FoVBackgroundMaker,
    MapDatasetMaker,
    SafeMaskMaker
)
from gammapy.maps import MapAxis, WcsGeom

data_store = DataStore.from_dir(
    base_dir="$GAMMAPY_DATA/hess-dl3-dr1"
)

obs = data_store.obs(23523)

energy_axis = MapAxis.from_energy_bounds(
    energy_min="1 TeV",
    energy_max="10 TeV",
    nbin=6,
)

geom = WcsGeom.create(
    skydir=(83.633, 22.014),
    width=(4, 3) * u.deg,
    axes=[energy_axis],
    binsz=0.02 * u.deg,
)

empty = MapDataset.create(geom=geom)

maker = MapDatasetMaker()

mask_maker = SafeMaskMaker(
    methods=["offset-max", "aeff-default"],
    offset_max="2.0 deg",
)

bkg_maker = FoVBackgroundMaker(
    method="scale",
)

dataset = maker.run(empty, observation=obs)
dataset = bkg_maker.run(dataset, observation=obs)
dataset = mask_maker.run(dataset, observation=obs)
dataset.peek()
```

Fig. 7. Using `gammapy.makers` to reduce DL3 level data into a `MapDataset`. All `Maker` classes represent a step in the data reduction process. They take the configuration on initialisation of the class. They also consistently define `.run()` methods, which take a dataset object and optionally an `Observation` object. In this way, `Maker` classes can be chained to define more complex data reduction pipelines. The output of the code example is shown in Figure A.5.

3.6. gammapy.makers

The `gammapy.makers` sub-package contains the various classes and functions required to process and prepare γ-ray data from the DL3 to the DL4, representing the input for modelling and fitting. First, events are binned and IRFs are interpolated and projected onto the chosen analysis geometry. The end product of the data reduction process is a set of binned counts, background exposure, psf and energy dispersion maps at the DL4 level. The `MapDatasetMaker` and `SpectrumDatasetMaker` are responsible for this task for 3D and 1D analyses, respectively (see Figure 7).

Because background models usually suffer from strong uncertainties, it is required to correct them from the data themselves. Several techniques are commonly used in TeV γ-ray astronomy such as FoV background normalization or background measurement in reflected regions, see Berge et al. (2007). Specific `Makers` such as the `FoVBackgroundMaker` or the `ReflectedRegionsBackgroundMaker` are in charge of this process.

Finally, to limit other sources of systematic uncertainties, a data validity domain is determined by the `SafeMaskMaker`. It can be used to limit the extent of the FoV used, or to limit the energy range to, e.g., a domain where the energy reconstruction bias is below a given value.

3.7. gammapy.stats

The `gammapy.stats` subpackage contains the fit statistics and the associated statistical estimators commonly adopted in γ-ray astronomy. In general, γ-ray observations count Poisson-distributed events at various sky positions and contain both signal and background events. To estimate the number of signal events in the observation one typically uses Poisson maximum likelihood estimation (MLE). In practice this is done by minimizing a fit statistic defined by $-2\log\mathcal{L}$, where $\mathcal{L}$ is the likelihood function used. Gammapy uses the convention of a factor of 2 in front, such that a difference in log-likelihood will approach a χ^2 distribution in the statistial limit.

When the expected number of background events is known, the statistic function is the so called *Cash* statistic (Cash 1979). It is used by datasets using background templates such as the `MapDataset`. When the number of background events is unknown, and an "off" measurement where only background events are expected is used, the statistic function is *WStat*. It is a profile log-likelihood statistic where the background counts are marginalized parameters. It is used by datasets containing "off" counts measurements such as the `SpectrumDatasetOnOff`, used for classical spectral analysis.

To perform simple statistical estimations on counts measurements, `CountsStatistic` classes encapsulate the aforementioned statistic functions to measure excess counts and estimate the associated statistical significance, errors and upper limits. They perform maximum likelihood ratio tests to estimate significance (the square root of the statistic difference) and compute likelihood profiles to measure errors and upper limits. The code example 8 shows how to compute the Li & Ma significance (Li & Ma 1983) of a set of measurements.

3.8. gammapy.modeling

`gammapy.modeling` contains all the functionality related to modelling and fitting data. This includes spectral, spatial and temporal model classes, as well as the fit and parameter API.

```
from gammapy.stats import WStatCountsStatistic

n_on = [13, 5, 3]
n_off = [11, 9, 20]
alpha = [0.8, 0.5, 0.1]
stat = WStatCountsStatistic(n_on, n_off, alpha)

# Excess
print(f"Excess: {stat.n_sig}")

# Significance
print(f"Significance: {stat.sqrt_ts}")

# Asymmetrical errors
print(f"Error Neg.: {stat.compute_errn(n_sigma=1.0)}")
print(f"Error Pos.: {stat.compute_errp(n_sigma=1.0)}")
```

Fig. 8. Using `gammapy.stats` to compute statistical quantities such as excess, significance and asymetric errors from counts based data. The data array such as `counts`, `counts_off` and the background efficency ratio `alpha` are passed on initialisation of the `WStatCountsStatistic` class. The derived quantities are then computed dynamically from the corresponding class attributes such as `stat.n_sig` for the excess and `stat.sqrt_ts` for the significance. The output of the code example is shown in Figure A.4.

3.8.1. Models

Source models in Gammapy (Eq. 1) are four-dimensional analytical models which support two spatial dimensions defined by the sky coordinates ℓ, b, an energy dimension E, and a time dimension t. To simplify the definition of the models, Gammapy uses a factorised representation of the total source model:

$$\phi(\ell, b, E, t) = F(E) \cdot G(\ell, b, E) \cdot H(t, E). \tag{5}$$

The spectral component $F(E)$, described by the `SpectralModel` class, always includes an *amplitude* parameter to adjust the total flux of the model. The spatial component $G(\ell, b, E)$, described by the `SpatialModel` class, also depends on energy, in order to consider energy-dependent sources morphology. Finally, the temporal component $H(t, E)$, described by the `TemporalModel` class, also supports an energy dependency in order to consider spectral variations of the model with time.

The models follow a naming scheme which contains the category as a suffix to the class name. The spectral models include a special class of normed models, named using the `NormSpectralModel` suffix. These spectral models feature a dimension-less *norm* parameter instead of an *amplitude* parameter with physical units. They can be used as an energy-dependent multiplicative correction factor to another spectral model. They are typically used for adjusting template-based models, or, for example, to take into account the absorbtion effect on γ-ray spectra caused by the extra-galactic background light (EBL) (`EBLAbsorptionNormSpectralModel`). Gammapy supports a variety of EBL absorption models, such as those from Franceschini et al. (2008), Finke et al. (2010), and Domínguez et al. (2011).

The analytical spatial models are all normalized such that they integrate to unity over the entire sky. The template spatial models may not, so in that special case they have to be combined with a `NormSpectralModel`.

```
from gammapy.modeling.models import (
    SkyModel,
    PowerLawSpectralModel,
    PointSpatialModel,
    ConstantTemporalModel,
)

# define a spectral model
pwl = PowerLawSpectralModel(
    amplitude="1e-12 TeV-1 cm-2 s-1", index=2.3
)

# define a spatial model
point = PointSpatialModel(
    lon_0="45.6 deg",
    lat_0="3.2 deg",
    frame="galactic"
)

# define a temporal model
constant = ConstantTemporalModel()

# combine all components
model = SkyModel(
    spectral_model=pwl,
    spatial_model=point,
    temporal_model=constant,
    name="my-model",
)
print(model)
```

Fig. 9. Using `gammapy.modeling.models` to define a source model with a spectral, spatial and temporal component. For convenience the model parameters can be defined as strings with attached units. The spatial model takes an additional `frame` parameter which allow users to define the coordinate frame of the position of the model. The output of the code example is shown in Figure A.8.

The `SkyModel` class represents the factorised model in Eq. 5 (the spatial and temporal components being optional). A `SkyModel` object can represent the sum of several emission components: either, for example, from multiple sources and from a diffuse emission, or from several spectral components within the same source. To handle a list of multiple `SkyModel` objects, Gammapy implements a `Models` class.

The model gallery provides a visual overview of the available models in Gammapy. Most of the analytic models commonly used in γ-ray astronomy are built-in. We also offer a wrapper to radiative models implemented in the Naima package (Zabalza 2015). The modelling framework can be easily extended with user-defined models. For example, the radiative models of jetted Active Galactic Nuclei (AGN) implemented in Agnpy, can be wrapped into Gammapy (see Section 3.5 of Nigro et al. 2022a).

3.8.2. Fit

The `Fit` class provides methods to fit, i.e. optimise, model parameters and estimate their errors and correlations. It interfaces with a `Datasets` object, which in turn is connected to a `Models` object containing the model parameters in its `Parameters` object. Models can be unique for a given dataset, or contribute to multiple datasets, allowing e.g., to perform a joint fit to multiple IACT datasets, or to jointly fit IACT and *Fermi*-LAT datasets. Many examples are given in the tutorials.

The `Fit` class provides a uniform interface to multiple fitting backends:

- iminuit (Dembinski & et al. 2020)
- `scipy.optimize` (Virtanen et al. 2020)
- Sherpa (Refsdal et al. 2011; Freeman et al. 2001)

Note that, for now, covariance matrix and errors are computed only for the fitting with iminuit. However, depending on the problem other optimizers can perform better, so sometimes it can be useful to run a pre-fit with alternative optimization methods. In the future, we plan to extend the supported fitting backends, including for example solutions based on Markov chain Monte Carlo methods. [3]

3.9. gammapy.estimators

By fitting parametric models to the data, the total γ-ray flux and its overall temporal, spectral and morphological components can be constrained. In many cases though, it is useful to make a more detailed follow-up analysis by measuring the flux in smaller spectral, temporal or spatial bins. This possibly reveals more detailed emission features, which are relevant for studying correlation with counterpart emissions.

The `gammapy.estimators` sub-module features methods to compute flux points, light curves, flux maps and flux profiles from data. The basic method for all these measurements is equivalent. The initial fine bins of `MapDataset` are grouped into larger bins. A multiplicative correction factor (the *norm*) is applied to the best fit reference spectral model and is fitted in the restricted data range, defined by the bin group only.

In addition to the best-fit flux *norm*, all estimators compute quantities corresponding to this flux. This includes: the predicted number of total, signal and background counts per flux bin; the total fit statistics of the best fit model (for signal and background); the fit statistics of the null hypothesis (background only); and the difference between both, the so-called test statistic value (TS). From this TS *value*, a significance of the measured signal and associated flux can be derived.

Optionally, the estimators can also compute more advanced quantities such as asymmetric flux errors, flux upper limits and one-dimensional profiles of the fit statistic, which show how the likelihood functions varies with the flux *norm* parameter around the fit minimum. This information is useful in inspecting the quality of a fit, for which a parabolic shape of the profile is asymptomatically expected at the best fit values.

The base class of all algorithms is the `Estimator` class. The result of the flux point estimation are either stored in a `FluxMaps` or `FluxPoints` object. Both objects are based on an internal representation of the flux which is independent of the Spectral Energy Distribution (SED) type. The flux is represented by a reference spectral model and an array of normalisation values given in energy, time and spatial bins, which factorises the deviation of the flux in a given bin from the reference spectral model. This allows users to conveniently transform between different SED types. Table 1 shows an overview and definitions of the supported SED types. The actual flux values for each SED type are obtained by multiplication of the *norm* with the reference flux.

Both result objects support the possibility to serialise the data into multiple formats. This includes the GADF SED format [4], FITS-based ND sky maps and other formats compatible with Astropy's `Table` and `BinnedTimeSeries` data structures. This allows users to further analyse the results with Astropy, for example using standard algorithms for time analysis, such as the Lomb-Scargle periodogram or the Bayesian blocks. So far, Gammapy does not support unfolding of γ-ray spectra. Methods for this will be implemented in future versions of Gammapy.

```python
from astropy import units as u

from gammapy.datasets import MapDataset
from gammapy.estimators import TSMapEstimator

dataset = MapDataset.read("$GAMMAPY_DATA/cta-1dc-gc/cta-1dc-gc

estimator = TSMapEstimator(
    energy_edges=[0.1, 1, 10] * u.TeV,
    n_sigma=1,
    n_sigma_ul=2,
)

maps = estimator.run(dataset)
maps["sqrt_ts"].plot_grid(add_cbar=True)
```

Fig. 10. Using the `TSMapEstimator` object from `gammapy.estimators` to compute a flux, flux upper limits and TS map. The additional parameters `n_sigma` and `n_sigma_ul` define the confidence levels (in multiples of the normal distribution width) of the flux error and flux upper limit maps respectively. The output of the code example is shown in Figure A.6.

The code example shown in Figure 10 shows how to use the `TSMapEstimator` objects with a given input `MapDataset`. In addition to the model, it allows to specify the energy bins of the resulting flux and TS maps.

3.10. gammapy.analysis

The `gammapy.analysis` sub-module provides a high-level interface (HLI) for the most common use cases identified in γ-ray analyses. The included classes and methods can be used in Python scripts, notebooks or as commands within `IPython` sessions. The HLI can also be used to automatise workflows driven by parameters declared in a configuration file in YAML format. In this way, a full analysis can be executed via a single command line taking the configuration file as input.

[3] a prototype is available in gammapy-recipes, https://gammapy.github.io/gammapy-recipes/_build/html/notebooks/mcmc-sampling-emcee/mcmc_sampling.html

[4] https://gamma-astro-data-formats.readthedocs.io/en/latest/spectra/flux_points/index.html

Type	Description	Unit Equivalency
dnde	Differential flux at a given energy	$\mathrm{TeV^{-1}\,cm^{-2}\,s^{-1}}$
e2dnde	Differential flux at a given energy	$\mathrm{TeV\,cm^{-2}\,s^{-1}}$
flux	Integrated flux in a given energy range	$\mathrm{cm^{-2}\,s^{-1}}$
eflux	Integrated energy flux in a given energy range	$\mathrm{erg\,cm^{-2}\,s^{-1}}$

Table 1. Definition of the different SED types supported in Gammapy.

The `Analysis` class has the responsibility for orchestrating the workflow defined in the configuration `AnalysisConfig` objects and triggering the execution of the `AnalysisStep` classes that define the identified common use cases. These steps include the following: observations selection with the `DataStore`, data reduction, excess map computation, model fitting, flux points estimation, and light curves production.

3.11. gammapy.visualization

The `gammapy.visualization` sub-package contains helper functions for plotting and visualizing analysis results and Gammapy data structures. This includes, for example, the visualization of reflected background regions across multiple observations, or plotting large parameter correlation matrices of Gammapy models. It also includes a helper class to split wide field Galactic survey images across multiple panels to fit a standard paper size.

The sub-package also provides `matplotlib` implementations of specific colormaps. Those colormaps have been historically used by larger collaborations in the very high-energy domain (such as MILAGRO or H.E.S.S.) as "trademark" colormaps. While we explicitly discourage the use of those colormaps for publication of new results, because they do not follow modern visualization standards, such as linear brightness gradients and accessibility for visually impaired people, we still consider the colormaps useful for reproducibility of past results.

3.12. gammapy.astro

The `gammapy.astro` sub-package contains utility functions for studying physical scenarios in high-energy astrophysics. The `gammapy.astro.darkmatter` module computes the so called J-factors and the associated γ-ray spectra expected from annihilation of dark matter in different channels, according to the recipe described in Cirelli et al. (2011).

In the `gammapy.astro.source` sub-module, dedicated classes exist for modelling galactic γ-ray sources according to simplified physical models, e.g. Supernova Remnant (SNR) evolution models (Taylor 1950; Truelove & McKee 1999), evolution of Pulsar Wind Nebulae (PWNe) during the free expansion phase (Gaensler & Slane 2006) or computation of physical parameters of a pulsar using a simplified dipole spin-down model.

In the `gammapy.astro.population` sub-module there are dedicated tools for simulating synthetic populations based on physical models derived from observational or theoretical considerations for different classes of Galactic very high-energy γ-ray emitters: PWNe, SNRs Case & Bhattacharya (1998), pulsars Faucher-Giguère & Kaspi (2006); Lorimer et al. (2006); Yusifov & Küçük (2004) and γ-ray binaries.

```python
import matplotlib.pyplot as plt

from gammapy.catalog import CATALOG_REGISTRY

catalog = CATALOG_REGISTRY.get_cls("4fgl")()
print("Number of sources :", len(catalog.table))

source = catalog["PKS 2155-304"]

_, axes = plt.subplots(ncols=2)
source.flux_points.plot(ax=axes[0], sed_type="e2dnde")

source.lightcurve().plot(ax=axes[1])
```

Fig. 11. Using `gammapy.catalogs` to access the underlying model, flux points and light-curve from the *Fermi*-LAT 4FGL catalog for the blazar PKS 2155-304. The output of the code example is shown in Figure A.7.

While the present list of use cases is rather preliminary, this can be enriched with time by users and/or developers according to future needs.

3.13. gammapy.catalog

Comprehensive source catalogs are increasingly being provided by many high-energy astrophysics experiments. The `gammapy.catalog` sub-packages provides a convenient access to the most important γ-ray catalogs. Catalogs are represented by the `SourceCatalog` object, which contains the actual catalog as an Astropy `Table` object. Objects in the catalog can be accessed by row index, name of the object or any association or alias name listed in the catalog.

Sources are represented in Gammapy by the `SourceCatalogObject` class, which has the responsibility to translate the information contained in the catalog to other Gammapy objects. This includes the spatial and spectral models of the source, flux points and light curves (if available) for each individual object. Figure 11 show how to load a given catalog and access these information for a selected source. This module works independently from the rest of the package, and the required catalogs are supplied in the `GAMMAPY_DATA` repository. The overview of currently supported catalogs, the corresponding Gammapy classes and references are shown in Table 2. Newly released relevant catalogs will be added in future.

4. Applications

Gammapy is currently used for a variety of analyses by different IACT experiments and has already been employed in more than 60 scientific publications as of XX/03/2023 [5]. In

[5] List on ADS

Class Name	Shortcut	Description	Reference
`SourceCatalog3FGL`	`"3fgl"`	3rd catalog of *Fermi*-LAT sources	Acero et al. (2015)
`SourceCatalog4FGL`	`"4fgl"`	4th catalog of *Fermi*-LAT sources	Abdollahi et al. (2020)
`SourceCatalog2FHL`	`"2fhl"`	2nd catalog high-energy *Fermi*-LAT sources	Ackermann et al. (2016)
`SourceCatalog3FHL`	`"3fhl"`	3rd catalog high-energy *Fermi*-LAT sources	Ajello et al. (2017)
`SourceCatalog2HWC`	`"2hwc"`	2nd catalog of HAWC sources	Abeysekara et al. (2017)
`SourceCatalog3HWC`	`"3hwc"`	3rd catalog of HAWC sources	Albert et al. (2020)
`SourceCatalogHGPS`	`"hgps"`	H.E.S.S. Galactic Plane Survey catalog	H.E.S.S. Collaboration (2018b)
`SourceCatalogGammaCat`	`"gammacat"`	Open source data collection	Deil et al. (2022)

Table 2. Overview of supported catalogs in `gammapy.catalog`.

this section, we illustrate the capabilities of Gammapy by performing some standard analysis cases commonly considered in γ-ray astronomy. Beside reproducing standard methodologies, we illustrate the unique data combination capabilites of Gammapy by presenting a multi-instrument analysis, which is not possible within any of the current instrument private software frameworks. The examples shown are based on the data accessible in the `gammapy-data` repository, and limited by the availability of public data. We remark that, as long as the data are compliant with the GADF specifications (or its future evolutions), and hence with Gammapy's data structures, there is no limitation on performing analyses of data from a given instrument.

4.1. 1D Analysis

One of the most common analysis cases in γ-ray astronomy is measuring the spectrum of a source in a given region defined on the sky, in conventional astronomy also called *aperture photometry*. The spectrum is typically measured in two steps: first a parametric spectral model is fitted to the data and secondly flux points are computed in a pre-defined set of energy bins. The result of such an analysis performed on three simulated CTA observations is shown in Figure 12. In this case the spectrum was measured in a circular aperture centered on the Galactic Center, in γ-ray astronomy often called "on region". For such analysis the user first chooses a region of interest and energy binning, both defined by a `RegionGeom`. In a second step, the events and the IRFs are binned into maps of this geometry, by the `SpectrumDatasetMaker`. All the data and reduced IRFs are bundled into a `SpectrumDataset`. To estimate the expected background in the "on region" a "reflected regions" background method was used (Berge et al. 2007), represented in Gammapy by the `ReflectedRegionsBackgroundMaker` class. The resulting reflected regions are illustrated for all three observations overlayed on the counts map in Figure 12. After reduction, the data were modelled using a forward-folding method and assuming a point source with a power law spectral shape. The model was defined, using the `SkyModel` class with a `PowerLawSpectralModel` spectral component only. This model was then combined with the `SpectrumDataset`, which contains the reduced data and fitted using the `Fit` class. Based on this best-fit model, the final flux points and corresponding log-likelihood profiles were computed using the `FluxPointsEstimator`.

4.2. 3D Analysis

The 1D analysis approach is a powerful tool to measure the spectrum of an isolated source. However, more complicated situations require a more careful treatment. In a FoV containing several overlapping sources, the 1D approach cannot disentangle the contribution of each source to the total flux in the selected region. Sources with extended or complex morphology can result in the measured flux being underestimated, and heavily dependent on the choice of extraction region.

For such situations, a more complex approach is needed, the so-called 3D analysis. The three relevant dimensions are the two spatial angular coordinates and an energy axis. In this framework, a combined spatial and spectral model (that is, a `SkyModel`, see Section 3.8) is fitted to the sky maps that were previously derived from the data reduction step and bundled into a `MapDataset` (see Sections 3.6 and 3.5).

A thorough description of the 3D analysis approach and multiple examples that use Gammapy can be found in Mohrmann et al. (2019). Here we present a short example to highlight some of its advantages.

Starting from the IRFs corresponding to the same three simulated CTA observations used in Section 4.1, we can create a `MapDataset` via the `MapDatasetMaker`. However, we will not use the simulated event lists provided by CTA but instead, use the method `MapDataset.fake()` to simulate measured counts from the combination of several `SkyModel` instances. In this way, a DL4 dataset can directly be simulated. In particular we simulate:

1. a point source located at ($l=0°$, $b=0°$) with a power law spectral shape,
2. an extended source with Gaussian morphology located at ($l=0.4°$, $b=0.15°$) with $\sigma=0.2°$ and a log parabola spectral shape,
3. a large shell-like structure centered on ($l=0.06°$, $b=0.6°$) with a radius and width of 0.6° and 0.3° respectively and a power law spectral shape.

The position and sizes of the sources have been selected so that their contributions overlap. This can be clearly seen in the significance map shown in the left panel of Figure 13. This map was produced with the `ExcessMapEstimator` (see Section 3.9) with a correlation radius of 0.1°.

We can now fit the same model shapes to the simulated data and retrieve the best-fit parameters. To check the model agreement, we compute the residual significance map after removing the contribution from each model. This is done again via the `ExcessMapEstimator`. As can be seen

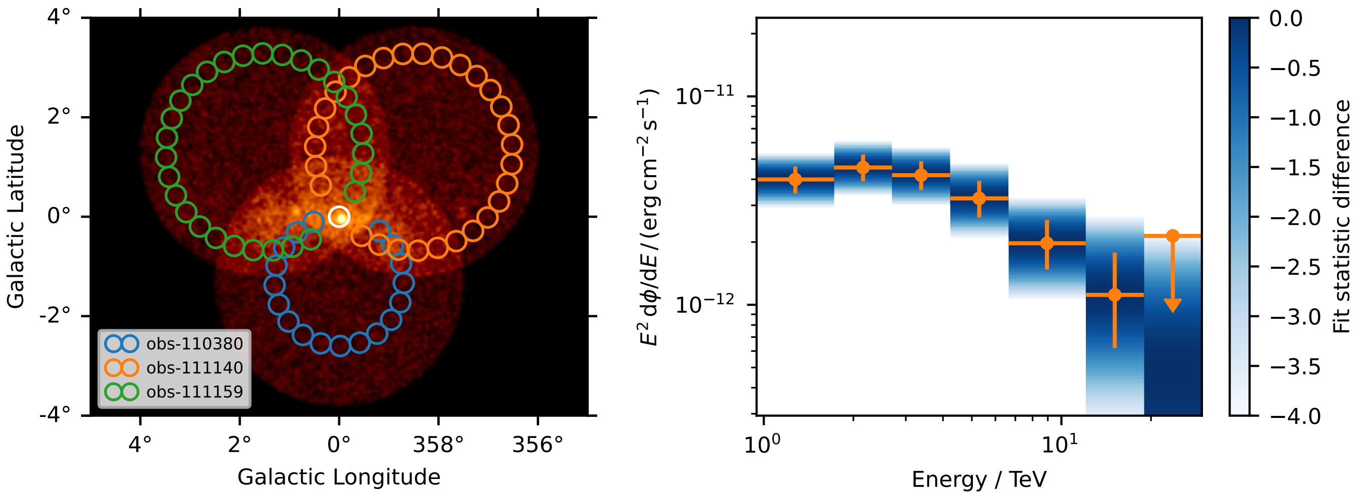

Fig. 12. Example of a one dimensional spectral analysis of the Galactic Center for three simulated CTA observations from the 1DC dataset. The left image shows the maps of counts with the signal region in white and the reflected background regions for the three different observations overlaid in different colors. The right image shows the resulting spectral flux points and their corresponding log-likelihood profiles. The flux points are shown in orange, with the horizontal bar illustrating the width of the energy bin and the vertical bar the 1 σ error. The log-likelihood profiles for each enetgy bin are shown in the background. The colormap illustrates the difference of the log-likelihood to the log-likelihood of the best fit value.

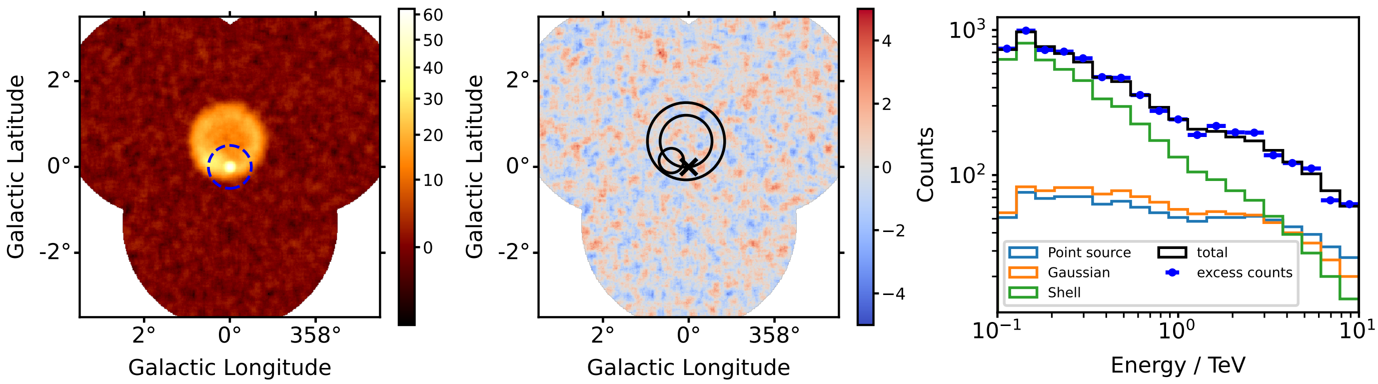

Fig. 13. Example of a 3D analysis for simulated sources with point-like, Gaussian and shell-like morphologies. The simulation uses *prod5* IRFs from CTA. The left image shows a significance map (using the *Cash* statistics) where the three simulated sources can be seen. The middle figure shows another significance map, but this time after subtracting the best-fit model for each of the sources, which are displayed in black. The right figure shows the contribution of each source model to the circular region of radius 0.5° drawn in the left image, together with the excess counts inside that region.

in the middle panel of Figure 13, there are no regions above or below $5\,\sigma$, meaning that the models describe the data sufficiently well.

As the example above shows, the 3D analysis allows to characterize the morphology of the emission and fit it together with the spectral properties of the source. Among the advantages that this provides is the ability to disentangle the contribution from overlapping sources to the same spatial region. To highlight this, we define a circular `RegionGeom` of radius 0.5° centered around the position of the point source, which is drawn in the left panel of Figure 13. We can now compare the measured excess counts integrated in that region to the expected relative contribution from each of the three source models. The result can be seen in the right panel of Figure 13.

Note that all the models fitted also have a spectral component, from which flux points can be derived in a similar way as described in Section 4.1.

4.3. Temporal Analysis

A common use case in many astrophysical scenarios is to study the temporal variability of a source. The most basic way to do this is to construct a light curve, i.e., the flux of a source in each given time bin. In Gammapy, this is done by using the `LightCurveEstimator` that fits the normalisation of a source in each time (and optionally energy) band per observation, keeping constant other parameters. For custom time binning, an observation needs to be split into finer time bins using the `Observation.select_time` method. Figure 14 shows the light curve of the blazar PKS 2155-304 in different energy bands as observed by the H.E.S.S. telescope during an exceptional flare on the night of July 29 - 30, 2006 Aharonian et al. (2009). The data are publicly available as a part of the HESS-DL3-DR1 H.E.S.S. Collaboration (2018a). Each observation is first split into 10 min smaller observations, and spectra extracted for each of these within a 0.11° radius around the source. A PowerLawSpectralModel is fit to all the datasets, leading to a reconstructed index of 3.54 ± 0.02. With this adjusted spectral model the LightCurveEstimator runs directly for two energy bands, 0.5 TeV to 1.5 TeV and 1.5 TeV to 20 TeV respectively. The obtained flux points can be analytically modelled using the available or user-implemented temporal models. Alternatively, instead of extracting a light curve, it is also possible to directly fit temporal models to the reduced datasets. By associating an appropriate SkyModel, consisting of both temporal and spectral components, or using custom temporal models with spectroscopic variability, to each dataset, a joint fit across the datasets will directly return the best fit temporal and spectral parameters.

4.4. Multi-instrument Analysis

In this multi-instrument analysis example we showcase the capabilities of Gammapy to perform a simultaneous likelihood fit incorporating data from different instruments and at different levels of reduction. We estimate the spectrum of the Crab Nebula combining data from the *Fermi*-LAT, MAGIC and HAWC instruments.

The *Fermi*-LAT data is introduced at the data level DL4, and directly bundled in a MapDataset. They have been prepared using the standard *fermitools* (Fermi Science Support Development Team 2019) and selecting a region of $5° \times 4°$ around the position of the Crab Nebula, applying the same selection criteria of the 3FHL catalog (7 years of data with energy from 10 GeV to 2 TeV, Ajello et al. 2017).

The MAGIC data is included from the data level DL3. They consist of two observations of 20 min each, chosen from the dataset used to estimate the performance of the upgraded stereo system (MAGIC Collaboration 2016) and already included in Nigro et al. (2019). The observations were taken at small zenith angles ($< 30°$) in wobble mode (Fomin et al. 1994), with the source sitting at an offset of 0.4° from the FoV center. Their energy range spans 80 GeV to 20 TeV. The data reduction for the 1D analysis is applied, and the data are reduced to a SpectrumDataset before being fitted.

HAWC data are directly provided as flux points (DL5 data level) and are read via Gammapy's FluxPoints class. They were estimated in HAWC Collaboration (2019) with 2.5 years of data and span an energy range 300 GeV to 300 TeV.

Combining the datasets in a Datasets list, Gammapy automatically generates a likelihood including three different types of terms, two Poissonian likelihoods for *Fermi*-LAT's MapDataset and MAGIC's SpectrumDataset, and a χ^2 accounting for the HAWC flux points. For *Fermi*-LAT, a three-dimensional forward folding of the sky model with the IRF is performed, in order to compute the predicted counts in each sky-coordinate and energy bin. For MAGIC, a one-dimensional forward-folding of the spectral model with the IRFs is performed to predict the counts in each estimated energy bin. A log parabola is fitted over almost five decades in energy 10 GeV to 300 TeV, taking into account all flux points from all three datasets.

The result of the joint fit is displayed in Figure 15. We remark that the objective of this exercise is illustrative. We display the flexibility of Gammapy in simultaneously fitting multi-instrument data even at different levels of reduction, without aiming to provide a new measurement of the Crab Nebula spectrum.

4.5. Broadband SED Modelling

By combining Gammapy with astrophysical modelling codes, users can also fit astrophysical spectral models to γ-ray data. There are several Python packages that are able to model the γ-ray emission, given a physical scenario. Among those packages are Agnpy (Nigro et al. 2022b), Naima (Zabalza 2015), Jetset (Tramacere 2020) and Gamera (Hahn et al. 2022). Typically those emission models predict broadband emission from radio, up to very high-energy γ-rays. By relying on the multiple dataset types in Gammapy those data can be combined to constrain such a broadband emission model. Gammapy provides a built-in NaimaSpectralModel that allows users to wrap a given astrophysical emission model from the Naima package and fit it directly to γ-ray data.

As an example application, we use the same multi-instrument dataset of the Crab Nebula, described in the previous section, and we apply an inverse Compton model computed with Naima and wrapped in the Gammapy models through the NaimaSpectralModel class. We describe the gamma-ray emission with an inverse Compton scenario, considering a log-parabolic electron distribution that scatters photons from:

- the synchrotron radiation produced by the very same electrons
- near and far infrared photon fields
- and the cosmic microwave background (CMB)

We adopt the prescription on the target photon fields provided in the documentation of the *Naima* package[6]. The best-fit inverse Compton spectrum is represented with a red dashed line in Figure 15.

4.6. Surveys, Catalogs, and Population Studies

Sky surveys have a large potential for new source detections, and discovery of new phenomena in γ-ray astronomy. They also offer less selection bias to perform source population studies over a large set of coherently detected and modelled objects. Early versions of Gammapy were developed in parallel to the preparation of the H.E.S.S. Galactic plane survey catalog (HGPS, H.E.S.S. Collaboration et al. 2018b) and the associated PWN and SNR populations studies (H.E.S.S. Collaboration et al. 2018a,c).

The increase in sensitivity and resolution provided by the new generation of instruments scales up the number of detectable sources and the complexity of models needed to represent them accurately. As an example, if we compare the results of the HGPS to the expectations from the CTA Galactic Plane survey simulations, we jump from 78 sources detected by H.E.S.S. to about 500 detectable by CTA (Remy et al. 2021). This large increase in the amount of data to analyse and increase in complexity of modelling scenarios, requires the high-level analysis software to be both scalabale as well as performant.

In short, the production of catalogs from γ-ray surveys can be divided in four main steps: data reduction; object

[6] https://naima.readthedocs.io/en/stable/examples.html#crab-nebula-ssc-model

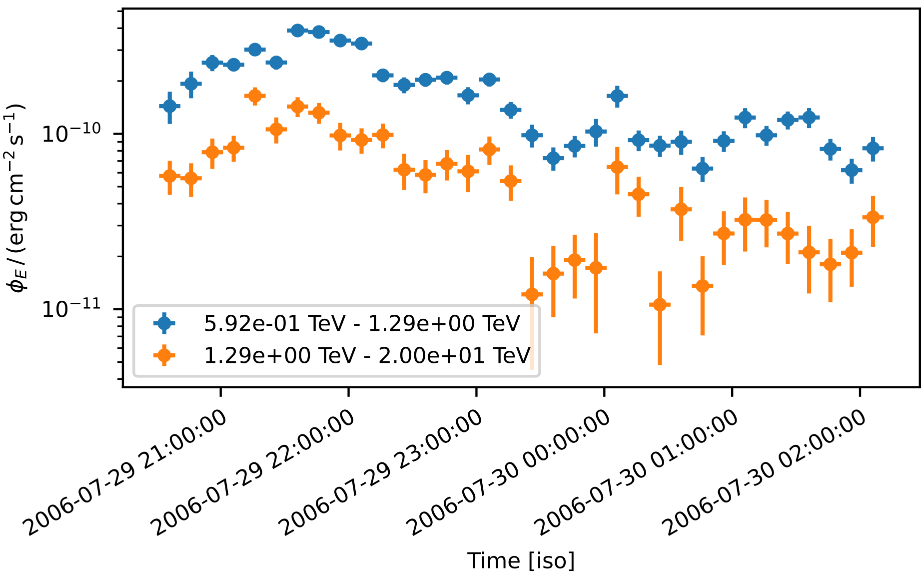

Fig. 14. Binned PKS 2155-304 light curve in two different energy bands as observed by the H.E.S.S. telescopes in 2006. The coloured markers show the flux points in the different energy bands: the range from (0.5 TeV to 1.5 TeV is shown in blue, while the range from 1.5 TeV to 20 TeV) is shown in orange. The horizontal error illustrates the width of the time bin of 10 min. The vertical error bars show the associated asymmetrical flux errors. The marker is set to the center of the time bin.

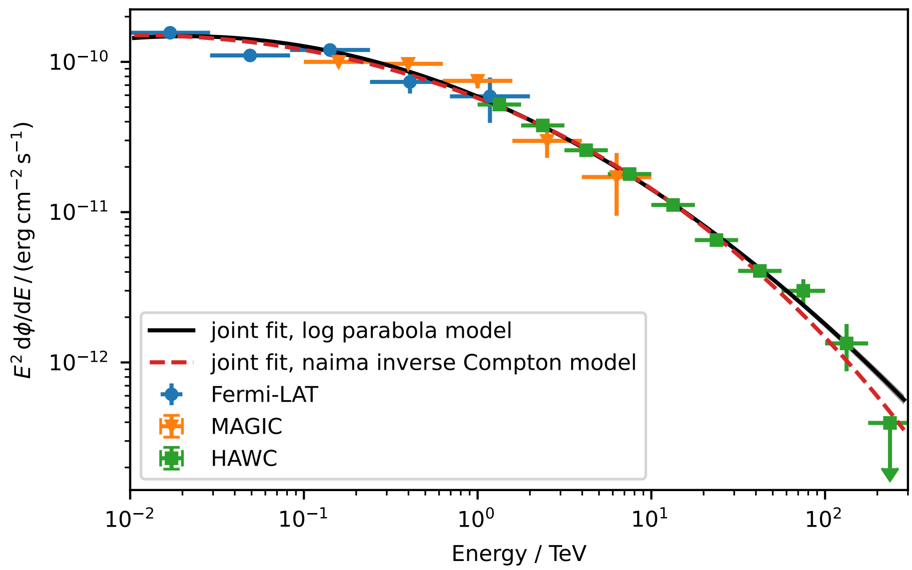

Fig. 15. A multi-instrument spectral energy distribution (SED) and combined model fit of the Crab Nebula. The colored markers show the flux points computed from the data of the different listed instruments. The horizontal error bar illustrates the width of the chosen energy band (E_{Min}, E_{Max}). The marker is set to the log-center energy of the band, that is defined by $\sqrt{E_{Min} \cdot E_{Max}}$. The vertical errors bars indicate the 1σ error of the measurement. The downward facing arrows indicate the value of 2σ upper flux limits for the given energy range. The black solid line shows the best fit model and the transparent band its 1σ error range. The band is too small be visible.

detection; model fitting and model selection; associations and classification. All steps can either be done directly with Gammapy or by relying on the seamless integration of Gammapy with the scientific Python ecosystem. This allows to rely on 3rd party functionality wherever needed.

The IACTs data reduction step is done in the same way described in the previous sections but scaled up to few thousands of observations. The object detection step typically consists in finding local maxima in the significance or TS maps, computed by the `ExcessMapEstimator` or `TSMapEstimator` respectively. Further refinements can include for example filtering and detection on these maps with techniques from the *Scikit-image* package (van der Walt et al. 2014), and outlier detection from the *Scikit-learn* package (Pedregosa et al. 2011). This allows e.g., to reduce the number of spurious detections at this stage using standard classification algorithms and then speed up the next step, as less objects will have to be fitted simultaneously. During the modelling step each object is alternatively fitted with different models in order to determine their optimal parameters, and the best-candidate model. The subpackage `gammapy.modeling.models` offers a large variety of choices, and the possibility to add custom models. Several spatial models (point-source, disk, Gaussian...), and spectral models (power law, log parabola...) may be tested for each object, so the complexity of the problem increases rapidly in regions crowded with multiple extended sources. Finally an object is discarded if its best-fit model is not significantly preferred over the null hypothesis (no source) comparing the difference in log likelihood between these two hypotheses.

For the association and classification step, which is tightly connected to the population studies, we can easily compare the fitted models to the set of existing γ-ray catalogs available in `gammapy.catalog`. Further multi-wavelength cross-matches are usually required to characterize the sources. This can easily be achieved by relying on coordinate handling from Astropy in combination with affiliated packages *Astroquery* (Ginsburg et al. 2019).

Studies performed on simulations not only offer a first glimpse on what could be the sky seen by CTA (according to our current knowledge on source populations), but also give us the opportunity to test the software on complex use cases[7]. In this way we can improve performance, optimize our analyse strategies, and identify the needs in terms of parallelisation to process the large datasets provided by surveys.

5. The Gammapy Project

In this section, we provide an overview of the organization of the Gammapy project. We briefly describe the main roles and responsibilities within the team, as well as the technical infrastructure designed to facilitate the development and maintenance of Gammapy as a high-quality software. We use common tools and services for software development of Python open-source projects, code review, testing, package distribution and user support, with a customized solution for a versioned and thoroughly-tested documentation in the form of user-friendly playable tutorials. This section concludes with an outlook on the roadmap for future directions.

5.1. Organizational Structure

Gammapy is an international open-source project with a broad developer base and contributions and commitments from mutiple groups and leading institutes in the very high-energy astrophysics domain[8]. The main development roadmaps are discussed and validated by a *Coordination Committee*, composed of representatives of the main contributing institutions and observatories. This committee is chaired by a *Project Manager* and his deputy while two *Lead Developers* manage the development strategy and organise technical activities. This institutionally-driven organisation, the permanent staff and commitment of supporting institutes ensure the continuity of the executive teams. A core team of developers from the contributing institutions is in charge of the regular development, which benefits from regular contributions of the community at large.

5.2. Technical Infrastructure

Gammapy follows an open-source and open-contribution development model based on the cloud repository service GitHub. A GitHub organization *gammapy*[9] hosts different repositories related with the project. The software codebase may be found in the *gammapy* repository (see Figure 16 for code lines statistics). We make extensive use of the pull request system to discuss and review code contributions.

Several automated tasks are set as GitHub actions[10], blocking the processes and alerting developers when failures occur. This is the case of the continuous integration workflow, which monitors the execution of the test coverage suite[11] using datasets from the *gammapy-data* repository[12]. Tests scan not only the codebase, but also the code snippets present in docstrings of the scripts and in the RST documentation files, as well as in the tutorials provided in the form of Jupyter notebooks.

Other automated tasks, executing in the *gammapy-benchmarks*[13] repository, are responsible for numerical validation tests and benchmarks monitoring. Also, tasks related with the release process are partially automated, and every contribution to the codebase repository triggers the documentation building and publishing workflow within the *gammapy-docs* repository[14] (see Sec. 5.3 and Sec. 5.4).

This small ecosystem of interconnected up-to-date repositories, automated tasks and alerts, is just a part of a bigger set of GitHub repositories, where most of them are related with the project but not necessary for the development of the software (i.e., project webpage, complementary high-energy astrophysics object catalogs, coding sprints and weekly developer calls minutes, contributions to conferences, other digital assets, etc). Finally, third-party services for code quality metrics are also set and may be found as status shields in the codebase repository.

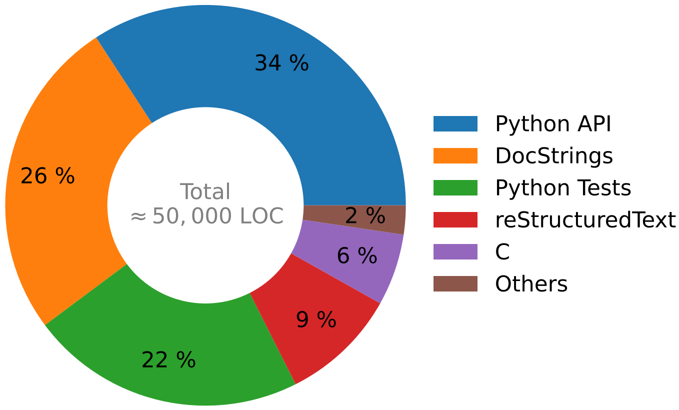

Fig. 16. Overview of used programming languages and distribution of code across the different file categories in the Gammapy code base. The total number of lines is ≈ 50000.

5.3. Software Distribution

Gammapy is distributed for Linux, Windows and Mac environments, and installed in the usual way for Python packages. Each stable release is uploaded to the Python package index[15] and as a binary package to the *conda-forge* and *astropy* Anaconda repository[16] channels. At this time, Gammapy is also available as a Debian Linux package[17]. We recommend installing the software using the *conda* installation process with an environment definition file that we provide, so to work within a virtual isolated environment with additional useful packages and ensure reproducibility.

Gammapy is indexed in the Astronomy Source Code Library[18] and Zenodo[19] digital libraries for software. The Zenodo record is synchronised with the codebase GitHub repository so that every release triggers the update of the

[7] Note that the CTA-GPS simulations were performed with the *ctools* package (Knödlseder et al. 2016) and analysed with both *ctools* and *gammapy* packages in order to cross-validate them.
[8] https://gammapy.org/team.html
[9] https://github.com/gammapy
[10] https://github.com/features/actions
[11] https://pytest.org
[12] https://github.com/gammapy/gammapy-data
[13] https://github.com/gammapy/gammapy-benchmarks
[14] https://github.com/gammapy/gammapy-docs
[15] https://pypi.org
[16] https://anaconda.org/anaconda/repo
[17] https://packages.debian.org/sid/python3-gammapy
[18] https://ascl.net/1711.014
[19] https://doi.org/10.5281/zenodo.4701488

versioned record. In addition, Gammapy has been added to the Open-source scientific Software and Service Repository[20] (Vuillaume et al. 2023) and indexed in the European Open Science Cloud catalog [21].

In addition, Gammapy is also listed in the *SoftWare Heritage* [22] (SWH) archive Cosmo (2020). The archive collects, preserves, and shares the source code of publicly available software. SWH automatically scans open software repositories, like e.g. GitHub, and projects are archived in SWH by the means of SoftWare Heritage persistent IDentifiers (SWHID), that are guaranteed to remain stable (persistent) over time. The French open publication archive, HAL [23], is using the Gammapy SWHIDs to register the releases as scientific products [24] of open science.

5.4. Documentation and User-support

Gammapy provides its user community with a tested and versioned up-to-date online documentation[25] (Boisson et al. 2019) built with Sphinx[26] scanning the codebase Python scripts, as well as a set of RST files and Jupyter notebooks. The documentation includes a user guide, a set of executable tutorials, and a reference to the API automatically extracted from the code and docstrings. The Gammapy code snippets present in the documentation are tested in different environments using our continuous integration (CI) workflow based on GitHub actions.

The Jupyter notebooks tutorials are generated using the sphinx-gallery package (Nájera et al. 2020). The resulting web published tutorials also provide links to playground spaces in *myBinder* (Project Jupyter et al. 2018), where they may be executed on-line in versioned virtual environments hosted in the myBinder infrastructure. Users may also play with the tutorials locally in their laptops. They can download a specific version of the tutorials together with the associated datasets needed and the specific conda computing environment, using the *gammapy download* command.

We have also set up a solution for users to share recipes that do not fit in the Gammapy core documentation, but which may be relevant for specific use cases, in the form of Jupyter notebooks. Contributions happen via pull requests to the *gammapy-recipes* GitHub repository and are merged after a short review. All notebooks in the repository are tested and published in the Gammapy recipes webpage[27] automatically using GitHub actions.

A growing community of users is gathering around the Slack messaging[28] and GitHub discussions[29] support forums, providing valuable feedback on the Gammapy functionalities, interface and documentation. Other communication channels have been set such as mailing lists, a Twitter account[30], regular public coding sprint meetings, hands-on sessions within collaborations, weekly development meetings, etc.

5.5. Proposals for Improving Gammapy

An important part of Gammapy's development organisation is the support for *Proposals for improving Gammapy*(PIG). This system is very much inspired by Python's PEP[31] and Astropy's APE (Greenfield 2013) system. PIG are self-contained documents which outline a set of significant changes to the Gammapy code base. This includes large feature additions, code and package restructuring and maintenance, as well as changes related to the organisational structure of the Gammapy project. PIGs can be proposed by any person in or outside the project and by multiple authors. They are presented to the Gammapy developer community in a pull request on GitHub and then undergo a review phase in which changes and improvements to the document are proposed and implemented. Once the PIG document is in a final state it is presented to the Gammapy coordination committee, which takes the final decision on the acceptance or rejection of the proposal. Once accepted, the proposed change are implemented by Gammapy developers in a series of individual contributions via pull requests. A list of all proposed PIG documents is available in the Gammapy online documentation [32].

A special category of PIGs are long-term *roadmaps*. To develop a common vision for all Gammapy project members on the future of the project, the main goals regarding planned features, maintenance and project organisation are written up as an overview and presented to the Gammapy community for discussion. The review and acceptance process follows the normal PIG guidelines. Typically roadmaps are written to outline and agree on a common vision for the next long term support release of Gammapy.

5.6. Release Cycle, Versioning, and Long-term Support

With the first long term support (LTS) release v1.0, the Gammapy project enters a new development phase. The development will change from quick feature-driven development to more stable maintenance and user support driven developement. After v1.0 we foresee a developement cycle with major, minor and bugfix releases; basically following the development cycle of the Astropy project. Thus we expect a major LTS release approximately every two years, minor releases are planned every 6 months, while bug-fix releases will happen as needed. While bug-fix releases will not introduce API-breaking changes, we will work with a deprecation system for minor releases. API-breaking changes will be announced to users by runtime warnings first and then implemented in the subsequent minor release. We consider this approach as a fair compromise between the interests of users in a stable package and the interest of developers to improve and develop Gammapy in future. The development cycle is described in more detail in PIG 23 (Terrier & Donath 2022).

[20] `https://projectescape.eu/ossr`
[21] `https://eosc-portal.eu`
[22] `https://softwareheritage.org`
[23] `https://hal.archives-ouvertes.fr`
[24] `https://hal.science/hal-03885031v1`
[25] `https://docs.gammapy.org`
[26] `https://www.sphinx-doc.org`
[27] `https://gammapy.github.io/gammapy-recipes`
[28] `https://gammapy.slack.com`
[29] `https://github.com/gammapy/gammapy/discussions`
[30] `https://twitter.com/gammapyST`
[31] `https://peps.python.org/pep-0001/`
[32] `https://docs.gammapy.org/dev/development/pigs/index.html`

6. Paper reproducibility

One of the most important goals of the Gammapy project is to support open and reproducible science results. Thus we decided to write this manuscript openly and publish the Latex source code along with the associated Python scripts to create the figures in an open repository [33]. This GitHub repository also documents the history of the creation and evolution of the manuscript with time. To simplify the reproducibility of this manuscript including figures and text, we relied on the tool *showyourwork* (Luger 2021). This tool coordinates the building process and both software and data dependencies, such that the complete manuscript can be reproduced with a single `make` command, after downloading the source repository. For this we provide detailed instructions online[34]. Almost all figures in this manuscript provide a link to a Python script, that was used to produce it. This means all example analyses presented in Sec.4 link to actually working Python source code.

7. Summary and Outlook

In this manuscript we presented the first LTS version of Gammapy. Gammapy is a Python package for γ-ray astronomy, which relies on the scientific Python ecosystem, including Numpy, Scipy, and Astropy as main dependencies. It also holds the status of an Astropy affiliated package. It supports high-level analysis of astronomical γ-ray data from intermediate level data formats, such as the FITS based GADF. Starting from lists of γ-ray events and corresponding descriptions of the instrument response users can reduce and project the data to WCS, HEALPix and region based data structures. The reduced data is bundled into datasets, which serve as a basis for Poisson maximum likelihood modelling of the data. For this purpose Gammapy provides a wide selection of built-in spectral, spatial and temporal models, as well as unified fitting interface with connection to multiple optimization backends.

With the v1.0 release, the Gammapy project enters a new development phase. Future work will not only include maintenance of the v1.0 release, but also parallel development of new features, improved API and data model support. While v1.0 provides all the features required for standard and advanced astronomical γ-ray data analysis, we already identified specific improvements to be considered in the roadmap for a future v2.0 release. This includes the support for scalable analyses via distributed computing. This will allow users to scale an analysis from a few observations to multiple hundreds of observations as expected by deep surveys of the CTA observatory. In addition the high-level interface of Gammapy is planned to be developed into a fully configurable API design. This will allow users to define arbitrary complex analysis scenarios as YAML files and even extend their workflows by user defined analysis steps via a registry system. Another important topic will be to improve the support of handling metadata for data structures and provenance information to track the history of the data reduction process from the DL3 to the highest DL5/DL6 data levels. Gammapy will also extend its functionalities for time based analyses, e.g. tests for variability in light curves, phase curves peak search, as well as improving the interoperability with other timing packages such as *Stingray* (Huppenkothen et al. 2019), *Astropy's* time series classes and *pint-pulsar* (Luo et al. 2021) for high-precision pulsar timing.

Around the core Python package a large diverse community of users and contributors has developed. With regular developer meetings, coding sprints and in-person user tutorials at relevant conferences and collaboration meetings, the community has constantly grown. So far Gammapy has seen 80 contributors from 10 different countries. With typically 10 regular contributors at any given time of the project, the code base has constantly grown its range of features and improved its code quality. With Gammapy being officially selected in 2021 as the base library for the future science tools for CTA [35], we expect the community to grow even further, providing a stable perspective for further usage, development and maintenance of the project. Besides the future use by the CTA community Gammapy has already been used for analysis of data from the H.E.S.S., MAGIC, ASTRI and VERITAS instruments.

While Gammapy was mainly developed for the science community around IACT instruments, the internal data model and software design are general enough to be applied to other γ-ray instruments as well. The use of Gammapy for the analysis of data from the High Altitude Water Cherenkov Observatory (HAWC) has been successfully demonstrated by Albert, A. et al. (2022). This makes Gammapy a viable choice for the base library for the science tools of the future Southern Widefield Gamma Ray Observatory (SWGO) and use with data from Large High Altitude Air Shower Observatory (LHAASO) as well. Gammapy has the potential to further unify the community of γ-ray astronomers, by sharing common tools, data formats and a common vision of open and reproducible science for the future.

Acknowledgements. We would like to thank the `Numpy`, `Scipy`, `IPython` and `Matplotlib` communities for providing their packages which are invaluable to the development of Gammapy. We thank the GitHub team for providing us with an excellent free development platform. We also are grateful to Read the Docs (`https://readthedocs.org/`), and Travis (`https://www.travis-ci.org/`) for providing free documentation hosting and testing respectively. A special acknowledgment has to be given to our first Lead Developer of Gammapy, Christoph Deil. Finally, we would like to thank all the Gammapy users that have provided feedback and submitted bug reports. A. Aguasca-Cabot acknowledges the financial support from the Spanish Ministry of Science and Innovation and the Spanish Research State Agency (AEI) under grant PID2019-104114RB-C33/AEI/10.13039/501100011034 and the Institute of Cosmos Sciences University of Barcelona (ICCUB, Unidad de Excelencia "María de Maeztu") through grant CEX2019-000918-M. J.L. Contreras acknowledges the funding from the ESCAPE H2020 project, GA No 824064. L. Giunti acknowledges financial support from the Agence Nationale de la Recherche (ANR-17-CE31-0014). M. Linhoff acknowledges support by the German BMBF (ErUM) and DFG (SFBs 876 and 1491). R. López-Coto acknowledges the Ramon y Cajal program through grant RYC-2020-028639-I and the financial support from the grant CEX2021-001131-S funded by MCIN/AEI/ 10.13039/501100011033. C. Nigro C.N. acknowledges support by the Spanish Ministerio de Ciencia e Innovación (MICINN), the European Union – NextGenerationEU and PRTR through the programme Juan de la Cierva (grant FJC2020-046063-I), by the the MICINN (grant PID2019-107847RB-C41), and from the CERCA program of the Generalitat de Catalunya. Q. Remy acknowledges support from the project "European Science Cluster of Astronomy & Particle Physics ESFRI Research Infrastructures" (ESCAPE), that has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 824064.

[33] `https://github.com/gammapy/gammapy-v1.0-paper`

[34] `https://github.com/gammapy/gammapy-v1.0-paper/blob/main/README.md`

[35] CTAO Press Release

J.E. Ruiz acknowledges financial support from the grant CEX2021-001131-S funded by MCIN/AEI/ 10.13039/501100011033. A. Siemiginowska was supported by NASA contract NAS8-03060 (Chandra X-ray Center). A. Sinha acknowledges support from The European Science Cluster of Astronomy & Particle Physics ESFRI Research Infrastructures funded by the European Union's Horizon 2020 research and innovation program under Grant Agreement no. 824064 and from the Spanish Ministry of Universities through the Maria Zambrano Talent Attraction Programme, 2021-2023.

References

Abdollahi, S., Acero, F., Ackermann, M., et al. 2020, The Astrophysical Journal Supplement Series, 247, 33

Abeysekara, A. U., Albert, A., Alfaro, R., et al. 2017, ApJ, 843, 40

Acero, F., Ackermann, M., Ajello, M., et al. 2015, ApJS, 218, 23

Ackermann, M., Ajello, M., Atwood, W. B., et al. 2016, ApJS, 222, 5

Aharonian, F., Akhperjanian, A. G., Anton, G., et al. 2009, A&A, 502, 749

Ajello, M., Atwood, W. B., Baldini, L., et al. 2017, ApJS, 232, 18

Albert, A., Alfaro, R., Alvarez, C., et al. 2020, ApJ, 905, 76

Albert, A., Alfaro, R., Arteaga-Velázquez, J. C., et al. 2022, A&A, 667, A36

Arnaud, K., Gordon, C., Dorman, B., & Rutkowski, K. 2022, Appendix B: Statistics in XSPEC

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33

Barker, M., Chue Hong, N. P., Katz, D. S., et al. 2022, Scientific Data, 9, 622

Berge, D., Funk, S., & Hinton, J. 2007, A&A, 466, 1219

Boisson, C., Ruiz, J. E., Deil, C., Donath, A., & Khelifi, B. 2019, in Astronomical Society of the Pacific Conference Series, Vol. 523, Astronomical Data Analysis Software and Systems XXVII, ed. P. J. Teuben, M. W. Pound, B. A. Thomas, & E. M. Warner, 357

Bradley, L., Deil, C., Patra, S., et al. 2022, astropy/regions: v0.6

Calabretta, M. R. & Greisen, E. W. 2002, A&A, 395, 1077

Case, G. L. & Bhattacharya, D. 1998, ApJ, 504, 761

Cash, W. 1979, ApJ, 228, 939

Cirelli, M., Corcella, G., Hektor, A., et al. 2011, J. Cosmology Astropart. Phys., 2011, 051

Cosmo, R. D. 2020, in Lecture Notes in Computer Science, Vol. 12097, ICMS (Springer), 362–373

de Naurois, M. & Mazin, D. 2015, Comptes Rendus Physique, 16, 610

Deil, C., Boisson, C., Kosack, K., et al. 2017, in American Institute of Physics Conference Series, Vol. 1792, 6th International Symposium on High Energy Gamma-Ray Astronomy, 070006

Deil, C., Maier, G., Donath, A., et al. 2022, Gammapy/gamma-cat: an open data collection and source catalog for Gamma-Ray Astronomy

Dembinski, H. & et al., P. O. 2020

Domínguez, A., Primack, J. R., Rosario, D. J., et al. 2011, MNRAS, 410, 2556

Donath, A., Deil, C., Arribas, M. P., et al. 2015, in International Cosmic Ray Conference, Vol. 34, 34th International Cosmic Ray Conference (ICRC2015), 789

Faucher-Giguère, C.-A. & Kaspi, V. M. 2006, ApJ, 643, 332

Fermi Science Support Development Team. 2019, Fermitools: Fermi Science Tools, Astrophysics Source Code Library, record ascl:1905.011

Finke, J. D., Razzaque, S., & Dermer, C. D. 2010, ApJ, 712, 238

Fomin, V. P., Stepanian, A. A., Lamb, R. C., et al. 1994, Astroparticle Physics, 2, 137

Franceschini, A., Rodighiero, G., & Vaccari, M. 2008, A&A, 487, 837

Freeman, P., Doe, S., & Siemiginowska, A. 2001, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 4477, Astronomical Data Analysis, ed. J.-L. Starck & F. D. Murtagh, 76–87

Gaensler, B. M. & Slane, P. O. 2006, ARA&A, 44, 17

Ginsburg, A., Sipőcz, B. M., Brasseur, C. E., et al. 2019, AJ, 157, 98

Greenfield, P. 2013, Astropy Proposal for Enhancement 1: APE Purpose and Process (APE 1)

Hahn, J., Romoli, C., & Breuhaus, M. 2022, GAMERA: Source modeling in gamma astronomy, Astrophysics Source Code Library, record ascl:2203.007

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Nature, 585, 357

HAWC Collaboration. 2019, ApJ, 881, 134

H.E.S.S. Collaboration. 2018a, H.E.S.S. first public test data release

H.E.S.S. Collaboration. 2018b, A&A, 612, A1

H.E.S.S. Collaboration, Abdalla, H., Abramowski, A., et al. 2018a, A&A, 612, A2

H.E.S.S. Collaboration, Abdalla, H., Abramowski, A., et al. 2018b, A&A, 612, A1

H.E.S.S. Collaboration, Abdalla, H., Abramowski, A., et al. 2018c, A&A, 612, A3

Hunter, J. D. 2007, Computing In Science & Engineering, 9, 90

Huppenkothen, D., Bachetti, M., Stevens, A. L., et al. 2019, apj, 881, 39

Knödlseder, J., Mayer, M., Deil, C., et al. 2016, A&A, 593, A1

Li, T. P. & Ma, Y. Q. 1983, ApJ, 272, 317

Lorimer, D. R., Faulkner, A. J., Lyne, A. G., et al. 2006, MNRAS, 372, 777

Luger, R. 2021, showyourwork, `https://github.com/rodluger/showyourwork`

Luo, J., Ransom, S., Demorest, P., et al. 2021, ApJ, 911, 45

MAGIC Collaboration. 2016, Astroparticle Physics, 72, 76

Mohrmann, L., Specovius, A., Tiziani, D., et al. 2019, A&A, 632, A72

Momcheva, I. & Tollerud, E. 2015, Software Use in Astronomy: an Informal Survey

Nigro, C., Deil, C., Zanin, R., et al. 2019, A&A, 625, A10

Nigro, C., Hassan, T., & Olivera-Nieto, L. 2021, Universe, 7, 374

Nigro, C., Sitarek, J., Gliwny, P., et al. 2022a, A&A, 660, A18

Nigro, C., Sitarek, J., Gliwny, P., et al. 2022b, A&A, 660, A18

Nájera, O., Larson, E., Estève, L., et al. 2020, sphinx-gallery/sphinx-gallery: Release v0.7.0

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825

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

Project Jupyter, Matthias Bussonnier, Jessica Forde, et al. 2018, in Proceedings of the 17th Python in Science Conference, ed. Fatih Akici, David Lippa, Dillon Niederhut, & M. Pacer, 113 – 120

Refsdal, B., Doe, S., Nguyen, D., & Siemiginowska, A. 2011, in 10th SciPy Conference, 4 – 10

Remy, Q., Tibaldo, L., Acero, F., et al. 2021, arXiv e-prints, arXiv:2109.03729

Taylor, G. 1950, Proceedings of the Royal Society of London Series A, 201, 159

Terrier, R. & Donath, A. 2022, PIG 23 - Gammapy release cycle and version numbering

Tramacere, A. 2020, JetSeT: Numerical modeling and SED fitting tool for relativistic jets, Astrophysics Source Code Library, record ascl:2009.001

Truelove, J. K. & McKee, C. F. 1999, ApJS, 120, 299

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., et al. 2014, PeerJ, 2, e453

Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, Nature Methods, 17, 261

Vuillaume, T., Al-Turany, M., F¸fling, M., et al. 2023, Open Research Europe, 3

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., et al. 2016, Scientific Data, 3, 160018

Wood, M., Caputo, R., Charles, E., et al. 2017, PoS, ICRC2017, 824

Yusifov, I. & Küçük, I. 2004, A&A, 422, 545

Zabalza, V. 2015, ArXiv e-prints [`arXiv:1509.03319`]

[1] Université Paris-Saclay, Université Paris Cité, CEA, CNRS, AIM, F-91191 Gif-sur-Yvette, France
[2] Departament de Física Quàntica i Astrofísica (FQA), Universitat de Barcelona (UB), c. Martí i Franqués, 1, 08028 Barcelona, Spain
[3] Institut de Ciències del Cosmos (ICCUB), Universitat de Barcelona (UB), c. Martí i Franqués, 1, 08028 Barcelona, Spain
[4] Institut d'Estudis Espacials de Catalunya (IEEC), c. Gran Capità, 2-4, 08034 Barcelona, Spain
[5] Departament de Física Quàntica i Astrofísica, Institut de Ciències del Cosmos, Universitat de Barcelona, IEEC-UB, Martí i Franquès, 1, 08028, Barcelona, Spain
[6] Deutsches Elektronen-Synchrotron (DESY), D-15738 Zeuthen, Germany
[7] Institute of physics, Humboldt-University of Berlin, D-12489 Berlin, Germany
[8] Université Savoie Mont Blanc, CNRS, Laboratoire d'Annecy de Physique des Particules - IN2P3, 74000 Annecy, France
[9] Laboratoire Univers et Théories, Observatoire de Paris, Université PSL, Université Paris Cité, CNRS, F-92190 Meudon, France
[10] Point8 GmbH
[11] Astroparticle Physics, Department of Physics, TU Dortmund University, Otto-Hahn-Str. 4a, D-44227 Dortmund
[12] Max Planck Institute for extraterrestrial Physics, Giessenbachstrasse, 85748 Garching, Germany
[13] Max Planck Computing and Data Facility, Gießenbachstraße 2, 85748 Garching
[14] School of Physics, University of the Witwatersrand, 1 Jan Smuts Avenue, Braamfontein, Johannesburg, 2050 South Africa
[15] IPARCOS Institute and EMFTEL Department, Universidad Complutense de Madrid, E-28040 Madrid, Spain
[16] The Hong Kong University of Science and Technology, Department of Electronic and Computer Engineering
[17] Max-Planck-Institut für Kernphysik, P.O. Box 103980, D 69029 Heidelberg, Germany
[18] Cherenkov Telescope Array Observatory gGmbH (CTAO gGmbH) Saupfercheckweg 1 69117 Heidelberg
[19] Erlangen Centre for Astroparticle Physics (ECAP), Friedrich-Alexander-Universität Erlangen-Nürnberg, Nikolaus-Fiebiger Strasse 2, 91058 Erlangen, Germany
[20] Université de Paris Cité, CNRS, Astroparticule et Cosmologie, F-75013 Paris, France
[21] Meteo France International
[22] Sorbonne Université, Université Paris Diderot, Sorbonne Paris Cité, CNRS/IN2P3, Laboratoire de Physique Nucléaire et de Hautes Energies, LPNHE, 4 Place Jussieu, F-75252 Paris, France
[23] Instituto de Astrofísica de Andalucía-CSIC, Glorieta de la Astronomía s/n, 18008, Granada, Spain
[24] Institut de Física d'Altes Energies (IFAE), The Barcelona Institute of Science and Technology, Campus UAB, Bellaterra, 08193 Barcelona, Spain
[25] Institut für Astro- und Teilchenphysik, Leopold-Franzens-Universität Innsbruck, A-6020 Innsbruck, Austria
[26] INAF/IASF Palermo, Via U. La Malfa, 153, 90146 Palermo PA
[27] Academic Computer Centre Cyfronet, AGH University of Science and Technology, Krakow, Poland
[28] Center for Astrophysics | Harvard and Smithsonian
[29] Caltech/IPAC, MC 100-22, 1200 E. California Boulevard, Pasadena, CA 91125 USA
[30] Bruno-Bauer-Straße 22, 12051 Berlin
[31] Université Bordeaux, CNRS, LP2I Bordeaux, UMR 5797
[32] IRFU, CEA, Université Paris-Saclay, F-91191 Gif-sur-Yvette, France

Appendix A: Code Examples Output

```
Observation id: 23523
N events: 7613
Max. area: 699771.0625 m2
Observation id: 23526
N events: 7581
Max. area: 623679.5 m2
Observation id: 23559
N events: 7601
Max. area: 613097.6875 m2
Observation id: 23592
N events: 7334
Max. area: 693575.75 m2
```

Fig. A.1. Output from the code example shown in Figure 3

```
MapDataset
----------

  Name                            : map-dataset

  Total counts                    : 104317
  Total background counts         : 91507.70
  Total excess counts             : 12809.30

  Predicted counts                : 91507.69
  Predicted background counts     : 91507.70
  Predicted excess counts         : nan

  Exposure min                    : 6.28e+07 m2 s
  Exposure max                    : 1.90e+10 m2 s

  Number of total bins            : 768000
  Number of fit bins              : 691680

  Fit statistic type              : cash
  Fit statistic value (-2 log(L)) : nan

  Number of models                : 0
  Number of parameters            : 0
  Number of free parameters       : 0
```

Fig. A.2. Output from the code example shown in Figure 6

```
WcsNDMap

        geom  : WcsGeom
         axes  : ['lon', 'lat', 'energy']
        shape : (100, 80, 10)
        ndim  : 3
        unit  :
        dtype : float32
 HpxNDMap

        geom  : HpxGeom
         axes  : ['skycoord', 'energy']
        shape : (3145728, 10)
        ndim  : 3
        unit  :
        dtype : float32
 RegionNDMap

        geom  : RegionGeom
         axes  : ['lon', 'lat', 'energy']
        shape : (1, 1, 10)
        ndim  : 3
        unit  :
        dtype : float32
```

Fig. A.3. Output from the code example shown in Figure 5

```
Excess: [4.2 0.5 1. ]
Significance: [0.95461389 0.18791253 0.62290414]
Error Neg.: [4.3980796  2.56480097 1.50533827]
Error Pos.: [4.63826301 2.91371256 2.11988712]
```

Fig. A.4. Output from the code example shown in Figure 8

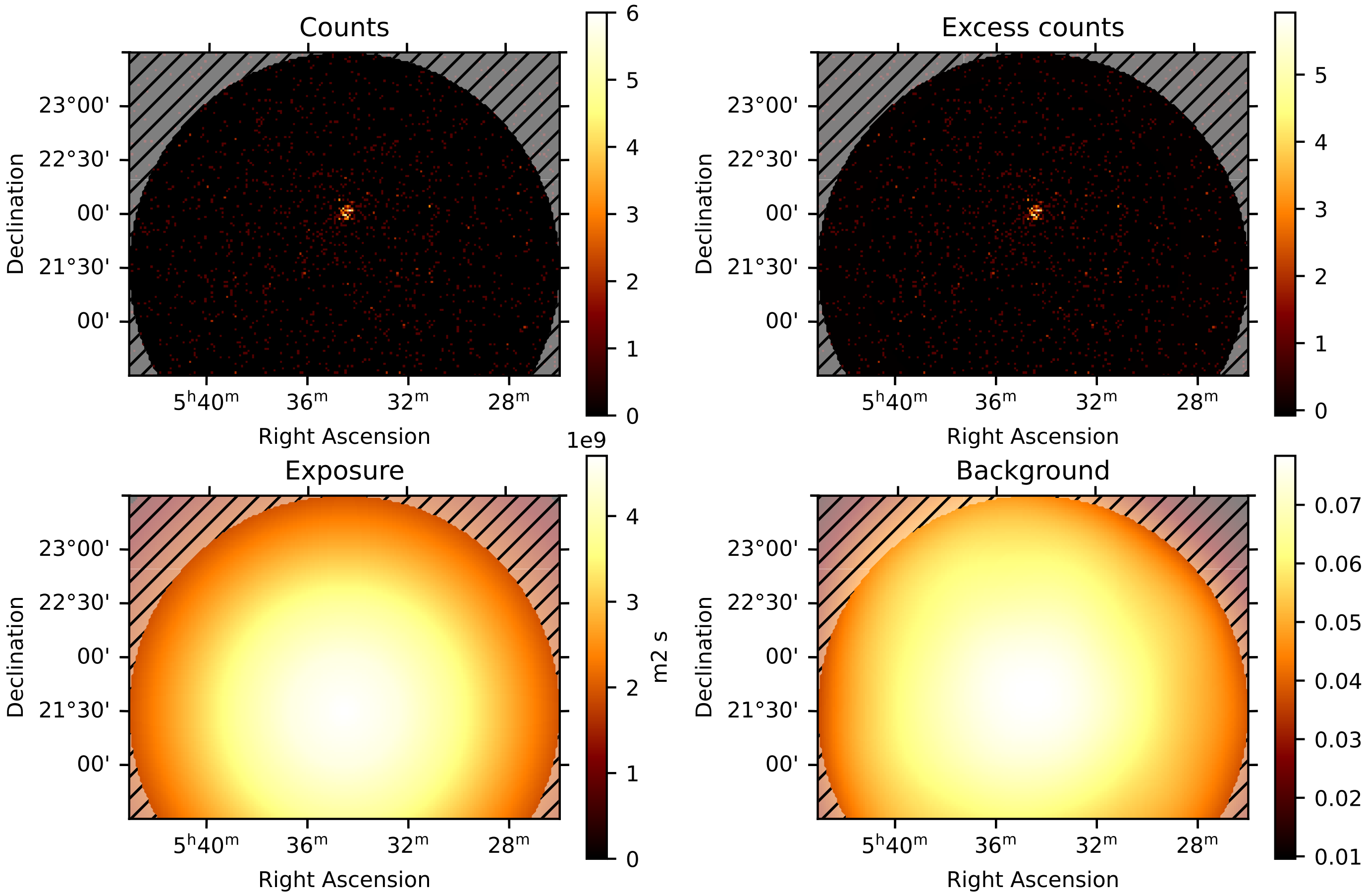

Fig. A.5. Output from the code example shown in Figure 7

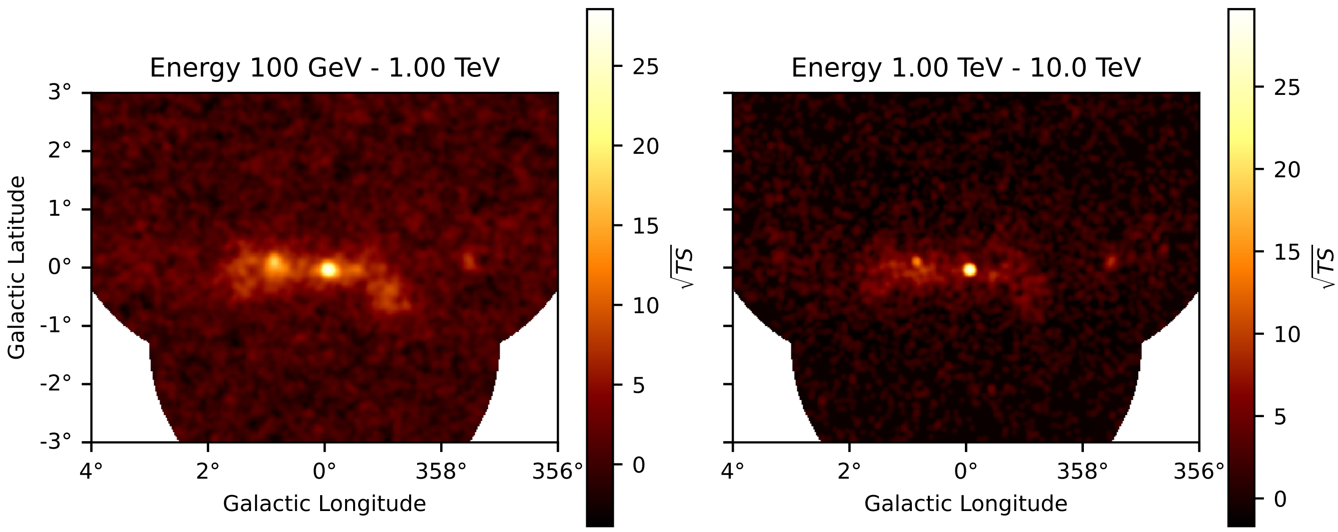

Fig. A.6. Output from the code example shown in Figure 10

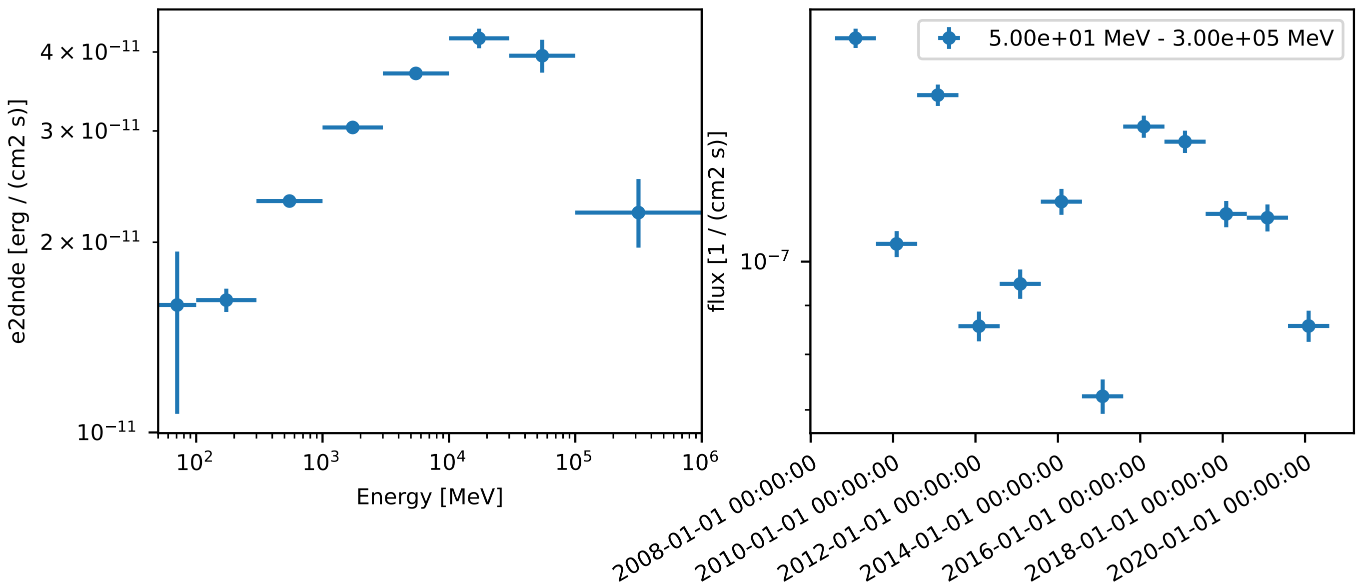

Fig. A.7. Output from the code example shown in Figure 11

```
SkyModel

  Name                      : my-model
  Datasets names            : None
  Spectral model type       : PowerLawSpectralModel
  Spatial  model type       : PointSpatialModel
  Temporal model type       : ConstantTemporalModel
  Parameters:
    index                         :      2.300   +/-    0.00
    amplitude                     :   1.00e-12   +/- 0.0e+00 1 / (cm2 s TeV)
    reference             (frozen):      1.000       TeV
    lon_0                         :     45.600   +/-    0.00 deg
    lat_0                         :      3.200   +/-    0.00 deg
```

Fig. A.8. Output from the code example shown in Figure 9