Gammapy: A Python package for gamma-ray astronomy

Axel Donath[7], Christoph Deil[14], Régis Terrier[3], Johannes King[23], Jose Enrique Ruiz[18], Quentin Remy[19], Léa Jouvin[30], Atreyee Sinha[27], Matthew Wood[12], Fabio Pintore[15], Manuel Paz Arribas[30], Laura Olivera[19], Luca Giunti[3], Bruno Khelifi[4], Ellis Owen[22], Brigitta Sipőcz[6], Olga Vorokh[30], Julien Lefaucheur[30], Fabio Acero[8], Thomas Robitaille[2], David Fidalgo[30], Jonathan D. Harris[30], Cosimo Nigro[16], Lars Mohrmann[19], Dirk Lennarz[1], Jalel Eddine Hajlaoui[3], Alexis de Almeida Coutinho[28], Matthias Wegenmat[30], Dimitri Papadopoulos[30], Maximilian Nöthe[26], Nachiketa Chakraborty[30], Michael Droettboom[21], Helen Poon[30], Arjun Voruganti[30], Jason Watson[9], Thomas Armstrong[30], Vikas Joshi[13], Erik Tollerud[25], Erik M. Bray[30], Domenico Tiziani[30], Gabriel Emery[30], Hubert Siejkowski[30], Kai Brügge[24], Luigi Tibaldo[17], Arpit Gogia[30], Ignacio Minaya[30], Marion Spir-Jacob[30], Yves Gallant[30], Andrew W. Chen[10], Roberta Zanin[5], Jean-Philippe Lenain[30], Larry Bradley[25], Kaori Nakashima[13], Anne Lemière[3], Mathieu de Bony[30], Matthew Craig[30], Lab Saha[7], Zé Vinicius[30], Kyle Barbary[30], Thomas Vuillaume[29], Adam Ginsburg[30], Daniel Morcuende[30], José Luis Contreras[30], Laura Vega Garcia[30], Oscar Blanch Bigas[30], Víctor Zabalza[30], Wolfgang Kerzendorf[20], Rolf Buehler[11], Sebastian Panny[30], Silvia Manconi[30], Stefan Klepser[11], Peter Deiml[30], Johannes Buchner[30], Hugo van Kemenade[30], Eric O. Lebigot[30], Benjamin Alan Weaver[30], Debanjan Bose[30], Rubén López-Coto[30], and Sam Carter[30]

(Affiliations can be found after the references)

December 7, 2022

ABSTRACT

Historically, the data and the analysis software of γ-ray astronomy experiments are proprietary to the collaborations operating them. The next generation of γ-ray instruments, as the Cherenkov Telescope Array (CTA) will be operated as open observatories, with public data. This introduces the need to have a high-level analysis software available to the astronomical community. In this article we present the first major release of Gammapy, the Long-term Support (LTS) v1.0, a community-developed open-source Python package for γ-ray astronomy. We present its general design and provide an overview of the analysis methods and features it implements. Starting from event lists and a description of the specific instrument response functions (IRF) stored in open FITS based data formats, Gammapy implements the reduction of the input data and instrument response to binned WCS, HEALPix or region based data structures. Thereby it handles the dependency of the IRFs with time, energy as well as position on the sky. It offers a variety of background estimation methods for spectral, spatial and spectro-morphological analysis. Counts, background and IRFs data are bundled in datasets and can be serialised, rebinned and stacked. Gammapy supports to model binned data using Poisson maximum likelihood fitting. It comes with built-in spectral, spatial and temporal models, support for particle radiation emission spectra as well as support for custom user models, to model e.g., energy dependent morphology of γ-ray sources. Multiple datasets can be combined in a joint-likelihood approach to either handle time dependent IRFs, different classes of events or combination of data from multiple instruments. Gammapy also implements methods to estimate flux points, likelihood profiles per energy bin, light curves as well as flux and signficance maps in energy bins. We further describe the general development approach and how Gammapy integrates into ecosystem of other scientific and astronomical Python packages. We also present analysis examples with simulated CTA data and provide results of scientific validation analyses using data of existing instruments such as H.E.S.S. and *Fermi*-LAT.

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

1. Introduction

γ-ray astronomy is a rather young field of research. The γ-ray range of the electromagnetic spectrum provides us with insight to the most energetic process in the universe such as surroundings of black holes and remnants of supernova explosions. As in other branches of astronomy, γ-rays can be observed by both satellite as well as ground based instruments. Ground based instrument use the Earth's atmosphere as a particle detector. Very high-energy cosmic γ-rays interact in the atmosphere and create a large shower of secondary particles that can be observed from the ground. Ground-based γ-ray astronomy relies on this phenomenon

to detect the primary γ-ray photons and measure their direction and energy. It covers the energy range from fews tens of GeV up to the PeV. There are two main categories of ground based instruments: Imaging Atmospheric Cherenkov Telescopes (IACT) and Water Cherenkov observatories. IACTs make images of atmospheric showers by detecting the Cherenkov radiation emitted by the cascading charged particles 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 (de Naurois & Mazin 2015).

Water Cherenkov observatories detect directly particles from the tail of the shower when it reaches the ground. These instruments have very large field-of-view, large duty-cycle but higher energy threshold and usually have lower signal to noise ratios compared to IACTs.

Ground based γ-ray astronomy has been historically structured by experiments run by independent collaborations relying on their own proprietary data and analysis software developed as part of the instrument. While this model has been very 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 explotation of the available γ-ray data, especially because the different detection techniques have complementary properties in terms of covered energy rage, dutu cycle and spatial resolution.

The Cherenkov Telescope Array (CTA) will be the first instrument to be operated as an open observatory in the domain. Its high level data (e.g. the event list) 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 existing instruments and their interoperability requires 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 observatory specific instrument response functions (IRF). One the data is present in the form of a list of events, the information is independent on the data reduction process, and eventually of the detection technique. This implies, for example, that data from IACT and particle samplers can be represented within the same model. The efforts to prototype a format usable by various instruments converged in the so-called *Data Format for γ-ray Astronomy* initiative (Deil et al. 2017; Nigro et al. 2021a), abbreviated in `gamma-astro-data-format` (GADF). The latter 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 gamma-like events with their measured quantities (energy, direction, arrival time) and a parametrisation of the response of the system.

Python has become extremely popular as a scientific programming language in particular in the field of data sciences. The 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 eco-system of packages and its open and supportive community, from core numerical analysis packages such as Numpy (Harris et al. 2020) and Scipy (Virtanen et al. 2020), visualization libraries, e.g. Matplotlib (Hunter 2007), or optimization packages such as IMinuit(Dembinski & et al. 2020).

The Astropy (Astropy Collaboration et al. 2013) was created in 2012 to build a community-developed Python package for astronomical reasearch. Among other things, it offers functionalities for transforming and representing

astronomical coordinates, manipulating physical quantities and their units as well as reading and writing FITS files.

The Gammapy project started in 2015 with the objective of building a flexible and efficient software library for very high-energy γ-ray data analysis (Donath et al. 2015). This is possible thanks to the definition of a common data format such as GADF, and thanks to an open development with a broad community of contributors. The relation of Gammapy to the different instruments and the GADF data format is illustrated in Figure **??**.

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 (Collaboration 2018). This data release served as a basis for analysis tools validation (Mohrmann et al. 2019, see e.g.).

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 and also describe 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 reduction process in γ-ray astronomy is usually split into two parts. The first one deals with the data processing from camera measurement, calibration, event reconstruction and selection to yield a list of reconstructed γ-ray event candidates. This 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. They also frequently imply joint analysis of multi-instrument data. To extract physically relevant information, the measured counts on an instrument are usually compared to a parametrised model of the true intensity distribution of γ-ray source in a given field-of-view, which is "forward folded" through a model of the instrument repsonse to get a prediction for the expected counts measured by the instrument. Parameters of the model are then estimated using statistical techniques such as maximum likelihood estimation (MLE).

In the most general way we can write the expected number of detected events at measured position p and energy E:

$$N(p, E)\mathrm{d}p\mathrm{d}E = t_{\mathrm{obs}} \int_{E_{\mathrm{true}}} \int_{p_{\mathrm{true}}} R(p, E | p_{\mathrm{true}}, E_{\mathrm{true}}) \quad (1)$$

$$\times \, \Phi(p_{\mathrm{true}}, E_{\mathrm{true}})\mathrm{d}E_{\mathrm{true}}\mathrm{d}p_{\mathrm{true}} \quad (2)$$

where

- $R(p, E | p_{\mathrm{true}}, E_{\mathrm{true}})$ is the instrument response
- $\Phi(p_{\mathrm{true}}, E_{\mathrm{true}})$ is the parametrised sky flux model
- t_{obs} is the observation time

A common assumption is that the instrument response can be simplified as the product of three independent functions:
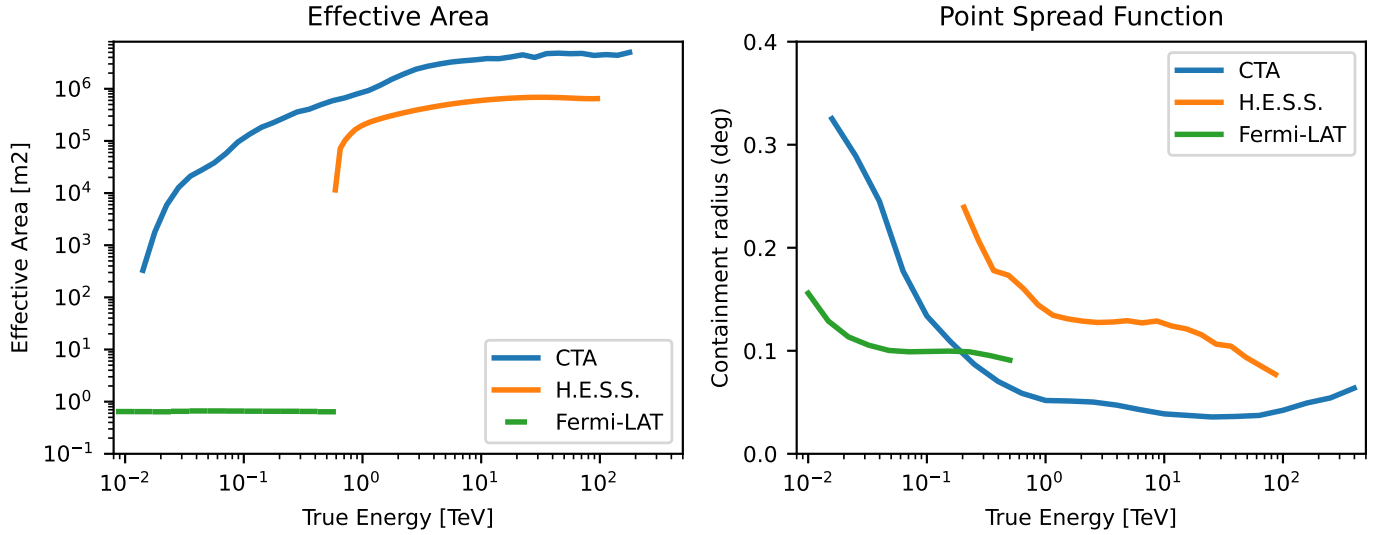
Fig. 1. Example instrument response functions of some experiments and observatories for which Gammapy can analyse data. The CTA IRFs are from the prod5 production. The H.E.S.S. IRFs are from the DL3 DR1, using observation ID 033787. The point spread function shows the 65& containment radius of the PSF. The *Fermi*-LAT IRFs are from *pass8*. TODO: Add more instruments? HAWC? MAGIC?

$$R(p, E|p_{\text{true}}, E_{\text{true}}) = A_{\text{eff}}(p_{\text{true}}, E_{\text{true}}) \times \qquad (3)$$
$$PSF(p|p_{\text{true}}, E_{\text{true}}) \times \qquad (4)$$
$$E_{\text{disp}}(E|p_{\text{true}}, E_{\text{true}}) \qquad (5)$$

where:

- $A_{\text{eff}}(p_{\text{true}}, E_{\text{true}})$ 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_{true} and position p_{true}.
- $PSF(p|p_{\text{true}}, E_{\text{true}})$ is the point spread function. It gives the probability of measuring a direction p when the true direction is p_{true} and the true energy is E_{true}. γ-ray instruments consider the probability density of the angular separation between true and reconstructed directions $\delta p = p_{\text{true}} - p$, i.e. $PSF(\delta p|p_{\text{true}}, E_{\text{true}})$.
- $E_{\text{disp}}(E|p_{\text{true}}, E_{\text{true}})$ is the energy dispersion. It gives the probability to reconstruct the photon at energy E when the true energy is E_{true} and the true position p_{true}. γ-ray instruments consider the probability density of the migration $\mu = \frac{E}{E_{\text{true}}}$, i.e. $E_{\text{disp}}(\mu|p_{\text{true}}, E_{\text{true}})$.

γ-ray data at the Data Level 3 (DL3) therefore consists in lists of gamma-like events and their corresponding instrument response functions (IRFs). The latter include the aforementioned effective area, point spread function (PSF), energy dispersion and residual hadronic background. The handling of DL3 data is performed by classes and methods in the `gammapy.data` (see Section 3.2) and the `gammapy.irf` (see Section 3.3) subpackages.

The first step in the analysis is the selection and extraction of observations based of their metadata including information such as pointing direction, observation time and observation conditions.

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 a spatial dimension, which either defines a spherical projection from celestial coordinates to an pixelised image space (for 3-dimensional or cube analysis) or a region on the sky (for regular spectral analysis). The `gammapy.maps` subpackage provides general multidimensional geometry objects (`Geom`) and the associated data structures (`Maps`) (see Section 3.4).

After all data has been projected into the same coordinate system, it is typically required to improve the residual hadronic background estimate. As residual hadronic background models can be subject to significant systematic uncertainties, the background estimated 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 field-of-view background techniques or background measurements performed within, e.g. reflected regions (Berge et al. 2007). Data measured at the field-of-view or energy boundaries of the instrument is typically associated with a sytematic 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 data level 4 (DL4). Those 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 modeling and fitting. Different variations of such datasets support different analysis methods and fit statistics (e.g. Poisson statistics with known background or with OFF background measurements). The datasets are used to perform joint-likelihood fitting allowing 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 next step is then typically to model and fit the datasets, either individually, or in a joint likelihood anal-

ysis. For this purpose Gammapy provides a uniform interface to multiple fitting backends. It also provides a variety of built in models. This includes spectral, spatial and temporal model classes to describe the γ-ray emission in the sky. Where spectral models can be simple analytical models or more complex ones from radiation mechanisms of accelerated particle populations (e.g. inverse Compton or π^o decay). Independently or subsequently to the global modelling, the data can be re-grouped to compute flux points, light curves and flux as well as significance maps in energy bands.

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 in the data reduction workflow described in the previous section. Sub-packages either contain data structures representing data at different data reduction levels or contain algorithms to transition between the different data reduction 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 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 create source catalogs. 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 functionality 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 of the GADF data format. DL3 data are typically bundled into individual observations, which corresponds to stable periods of data aqusition. For IACT data analysis, for which the GADF data model and Gammapy were initially conceived for, these are usually $20 - 30\,\mathrm{min}$ 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, which contains the DL3 data index file. The `DataStore` object gathers a collection of observations and providing 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 additionally 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 class 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 functionality to handle IRFs in a variety of formats. Usually, IRFs store instrument properties in the form of multidimensional tables, with quantities expressed in terms of energy (true or reconstructed), off-axis angles or cartesian detector coordinates. The main information stored in the common γ-ray IRFs are the effective area, energy dispersion, point spread function 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 in different kinds, 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 angles, or even plot the multidimensional effective area table.

3.3.2. Point Spread Function

Gammapy allows user to treat different kinds of PSFs, in particular, parametric multi-dimensional Gaussians (`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 of offset form the poiting 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 en-
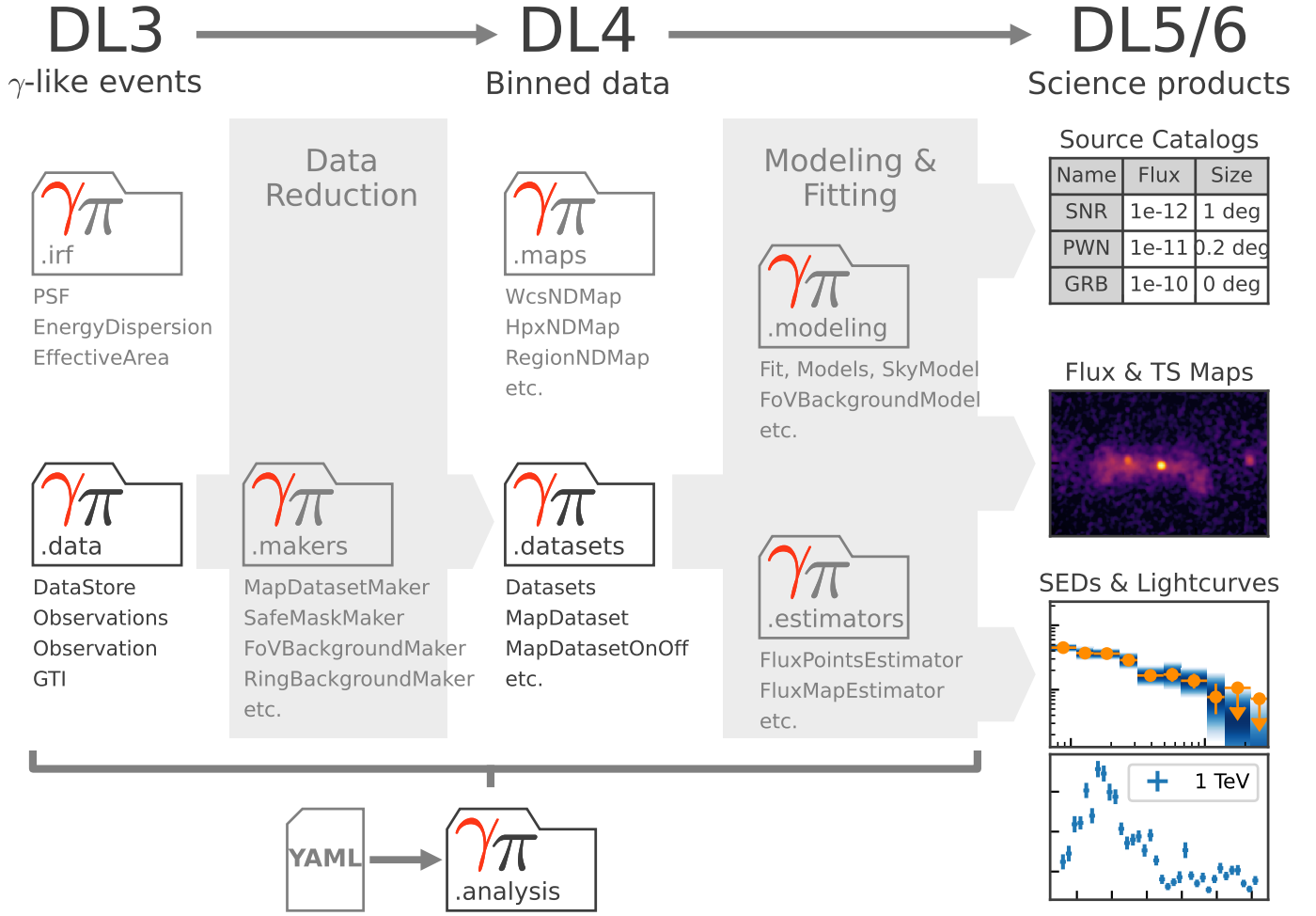
Fig. 2. Gammapy sub-package structure and data analysis workflow. The top row defines the groups for the different data levels and reduction steps from raw gamma-like events on the left, to high level science products on the right. The direction of the data flow is illustrated with the grey arrows. The gray folder icons represent the different sub-packages in Gammapy and their names. Below each icon there is a list of the most important objects defined in the sub-package.

ergy and position on the sky. During the modeling 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 used for the prediction of counts from that model component.

3.3.3. Energy Dispersion

For IACTs, the energy dispersion is typically parameterised 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, 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. Then there is the `EDispKernelMap`, which contains an energy disperion matrix per sky position. I.e., a 4-dimensional WCS map where at each sky-position is associated an energy dispersion matrix. The latter is a representation of the energy resolution

as a function of the true energy only repesented by the sub-class `EDispKernel`.

3.3.4. Background model

The backgropund rate can be represented in Gammapy as either a 2-dimensional data structure (`Background2D`) of count rate normalised per steradians and energy at different reconstructed energies and offset angles or as rate per steradians and energy, as a function of reconstructed energy and detector coordinates (`Background3D`). In the former, the background is expected to follow a radially symmetric shape, while in the latter, it can be more complex.

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

```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 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 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-lineary 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 observation time-stamps. 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 4). This allows uses to perform the same higher level operations on maps independent of the underlying pixelisation scheme.

3.4.1. WCS Maps

The FITS WCS pixelization supports a different number of 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.

```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. 4. 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`.

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 definiton of the HEALPix pixelisation and thus allows for easy up 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.

```python
from pathlib import Path
from gammapy.datasets import (
    MapDataset,
    SpectrumDatasetOnOff,
    FluxPointsDataset,
    Datasets
)

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 / "tests/spectrum/flux_points/diff_flux_points.fits",
    name="flux-points-dataset",
)


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

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

Fig. 5. 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 access by name later in the code. The environment variable `$GAMMAPY_DATA` is autromtically resolved by Gammapy.

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. Typically they are is 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 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.

3.5. gammapy.datasets

The `gammapy.datasets` subpackage contains classes to bundle together binned data along with associated models and the likelihood, which provides an interface to the `Fit` class (Sec 3.8.2) for modeling and fitting purposes. Depending upon the type of analysis and the associated statistics, different types of Datasets are supported. `MapDataset` is used for 3D (spectral and morphological) fitting, and a 1D spectral fitting is done using `SpectrumDataset`. While the default statistics for both of these is *Cash*, their corresponding on off versions are adapted for the case where the background is measured from real off counts, and support *WStat* statistics. The predicted counts are computed by convolution of the models with the associated IRFs. Fitting of precomputed flux points is enabled through `FluxPointsDataset`, using chi2 statistics. Multiple datasets of same or different types can be bundled together in `Datasets` (e.g., Figure 5), 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.

3.6. gammapy.makers

The data reduction step includes all tasks required to process and prepare γ-ray data from the DL3 level to modeling and fitting. The `gammapy.makers` sub-package contains the various classes and functions required to do so. First, events are binned and IRFs are interpolated and projected onto the chosen analysis geometry. The end product of the data reduction process are 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, see Fig 6.

Because the background models 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 field-of-view 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 step.

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 field of view 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 associated statistical estimators that are commonly used in γ-ray astronomy. In general, γ-ray observations count Poisson-distributed events at various sky positions, and contain both signal and background events. Estimation of the number of signal events is done through likelihood maximization. In Gammapy, the fit statistics are Poisson log-likelihood functions normalized like chi-squares, i.e., they follow the expression $2 \log \mathcal{L}$, where $\mathcal{L}$ is the likelihood function used. When the expected number of background events is known, the used statistic function is the *Cash* statistic (Cash 1979). It is used by datasets using background templates such as the `MapDataset`. When the num-

```python
import astropy.units as u
from gammapy.data import DataStore
from gammapy.maps import MapAxis, WcsGeom
from gammapy.datasets import MapDataset
from gammapy.makers import (
    MapDatasetMaker,
    FoVBackgroundMaker,
    SafeMaskMaker,
)

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=3 * u.deg,
    axes=[energy_axis],
)

empty = MapDataset.create(geom=geom)

maker = MapDatasetMaker()

mask_maker = SafeMaskMaker(
    methods=["offset-max", "aeff-default"],
    offset_max="2.3 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)
print(dataset)
```

Fig. 6. 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 consitently define `.run()` methods, which take a dataset object and optionally an `Observation` object. This way `Maker` classes can be chained to define more complex data reduction "pipelines".

ber 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 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

```python
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(stat.n_sig)

# Significance
print(stat.sqrt_ts)

# Asymmetrical rrors
print(stat.compute_errn(1.))
print(stat.compute_errp(1.))
```

Fig. 7. Using `gammapy.stats` to compute statistical quantities such as excess, signficance and assymetric errors from counts based data. The data is passed on initialisation of the `WStatCountsStatistic` class. The quantities are the computed ON excess of the corresponding class attributes such as `stat.n_sig` and `stat.sqrt_ts`.

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 7 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 modeling and fitting data. This includes spectral, spatial and temporal model classes, as well as the fit and parameter API.

3.8.1. Models

Source models in Gammapy are four dimensional models which support two spatial dimensions define by the sky coordinates ℓ, b, and energy dimension E as well as the time dimension t. To simplify the the definition of the models, Gammapy uses a factorised representation of the total source model:

$$f(l, b, E, t) = F(E) \cdot G(l, b, E) \cdot H(t, E) \qquad (6)$$

Where the spectral model F implicitly includes an *amplitude* parameter for the total flux of the model. The spatial model G optionally depends on energy and allows to support e.g. energy dependent morphology models. The temporal model component H optionally also supports an energy variable to allow for spectral variations of the model with time.

Following the factorisation the models objects are grouped into the following categories:

— `SpectralModel`: models to describe spectral shapes of sources,
— `SpatialModel`: models to describe spatial shapes (morphologies) of sources,

– `TemporalModel`: models to describe temporal flux evolution of sources, such as light and phase curves.

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 must be used along with another spectral model, as a multiplicative correction factor according to their spectral shape. They can be typically used for adjusting template based models, or adding taking into account the absorbtion effect on γ-ray spectra caused by the extra-galactic background light (EBL) (`EBLAbsorptionNormSpectralModel`) model. Gammapy supports a variety of EBL absorbtion models, such as Franceschini et al. (2008), Finke et al. (2010) and Domínguez et al. (2011).

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

The `SkyModel` object represents factorised model that combines the spectral, spatial and temporal model components (the spatial and temporal components being optional). `SkyModel` objects represents additive emission components, usually sources or diffuse emission, although a single source can also be modeled by multiple components. To handle list of multiple `SkyModel` objects, Gammapy has 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 modeling framework can be easily extended with user-defined models. For example agnpy models that describe leptonic radiative processes in jetted Active Galactic Nuclei (AGN) can be wrapped into Gammapy (see Section 3.5 of Nigro et al. 2021b) .

3.8.2. Fit

The `Fit` class provides methods to fit i.e., optimise parameters and estimate parameter 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 do a joint fit to multiple IACT datasets, or to a joint IACT and *Fermi*-LAT dataset. Many examples are given in the tutorials.

TODO: Add fit code example...?

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)

Note that, for now, covariance matrixand errors are computed only for the fitting with IMinuit. However depending on the problem other optimizers can better perform, so sometimes it can be useful to run a pre-fit with

```python
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. 8. 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 addtional `frame` parameter which allow users to define the coordinate frame of the position of the model.

alternative optimization methods. In future we plan to extend the supported fitting backends, including for example solutions based on Markov chain Monte Carlo. [1]

3.9. gammapy.estimators

By fitting parametric models to the data, the total integrated flux and overall temporal, spectral and morphological shape of the γ-ray emission can be constrained. In many cases 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

---

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

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, the fit statistics of the null hypothesis and the difference between both, the so-called TS value. From the TS value the significance of the significance of the measured signal and associated flux can be derived.

Optionally it can also compute more advanced quantities such as assymmetric 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 for inspecting the quality of a fit, for which asymptomatically a parabolic shape of the profile is 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 the 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 user to conveniently transform between different SED types. Table 3.9 shows an overview and definitions of the supported SED types. The actual flux values for each SED type are obtained by mutiplication 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 (reference?) format, FITS based ND skymaps and formats compatible with Astropy's `Table` and `BinnedTimeSeries` data structures. This allows users to directly further analyse the results, e.g. standard algorithms for time analysis such as computing Lomb-Scargle periodograms or Bayesian blocks for time series. So far Gammapy does not support 'unfolding' of γ-ray spectra. Methods for this will be implemented in a future version of Gammapy.

The code example shown in Figure 9 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. This way, a full analysis can be executed via a single command line taking the configuration file as input.

The `Analysis` class has the responsibility of orchestrating of 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: observa-

```python
from gammapy.datasets import MapDataset
from gammapy.estimators import TSMapEstimator
from astropy import units as u

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

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()
```

Fig. 9. Using the `TSMapEstimator` object from `gammapy.estimators` to compute a 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 and flux upper limit maps respectively.

tions 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 for false color image representation. Those colormaps have been used historically 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 modeling 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 Nebula (PWN) during the

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

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

```python
from gammapy.catalog import CATALOG_REGISTRY

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

source = catalog["PKS 2155-304"]
model = source.sky_model()
source.flux_points.plot()

source.lightcurve().plot()
```

Fig. 10. 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

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.

While the present list of use cases is rather preliminary, this can be enriched with time with 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 accesed 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. This module works independently from the rest of the package, and the required catalogs are supplied in `GAMMAPY_DATA` repository. The overview of currently supported catalogs, the corresponding Gammapy classes and references are shown in Table 3.13. Newly released relevant catalogs will be added in future.

4. Applications

Gammapy can be used for a variety of science cases by different IACT experiments. (Refer to publications). The analysis examples shown in the following section are limited by the availability of public data. In general, there is no limitation on performing a specific analysis on 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 11. 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 users 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 on top of the map of counts. After reduction of the data it was modelled using a forward-folding method and a power law spectral shape, using the `PowerLawSpectralModel` and `Fit` class. Based on this best fit model the final flux points and corresponding log-likelihood profiles are 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 field of view 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. Additionally, the 1D approach neglects the energy-dependence of the PSF.

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.

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`.
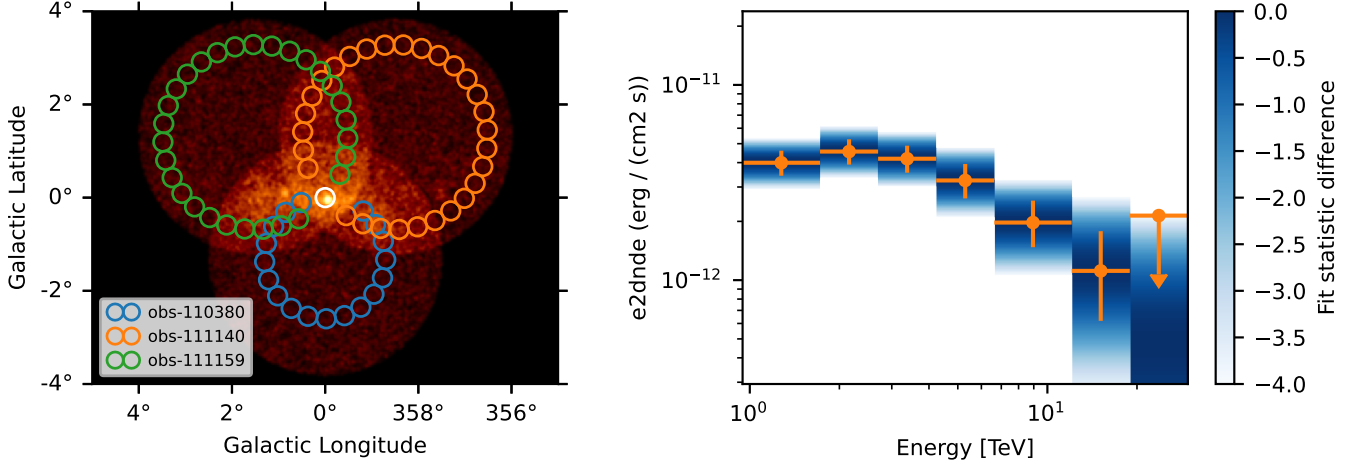
Fig. 11. Example of 1D spectral analysis of the Galactic Center for three simulated CTA observations for the 1DC dataset. The left image shows the maps of counts with the measurement region in white and background regions overlaid in different colors. The right image shows the resulting spectral points and their corresponding log-likelihood profiles.

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 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 σ=0.2°and a log parabola spectral shape.
3. A large shell-like structure centered around (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 12. 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 in the middle panel of Figure 12, there are no regions above or below 5σ, 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 12. 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 left panel of Figure 12.

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 4.1.
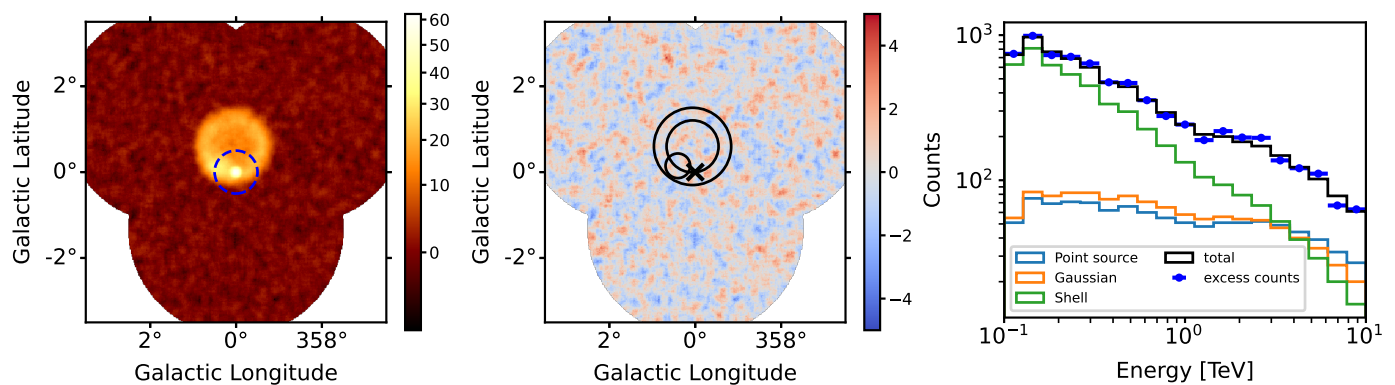
Fig. 12. Example of a 3D analysis for simulated sources with point-like, Gaussian and shell-like morphology. The simulation uses *prod5* IRFs from CTA. The left image shows a significance map where the three simulated sources can be seen. The middle figure shows another significance map, but this time 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.

4.3. Temporal Analysis

A common use case in most 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 13 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), and shipped with `GAMMAPY_DATA`. 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 - 1.5 TeV, and 1.5 - 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. It is prepared using the standard *fermitools* (Fermi Science Support Development Team 2019) and selecting a region of 5°x4° 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. It consists of 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 $80\,\text{GeV} - 20\,\text{TeV}$. They are reduced to ON/OFF dataset before being fitted. HAWC flux points data are estimated in HAWC Collaboration (2019) with 2.5 years of data and span an energy range $300\,\text{GeV} - 300\,\text{TeV}$, and directly read with Gammapy.

In the case the HAWC data is included from the DL5 data level, meaning flux points and associated errors provided by HAWC Collaboration (2019).

Gammapy automatically generates a likelihood including three different types of terms, two Poissonian likelihoods for the *Fermi*-LAT map and the ON/OFF counts, 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 to the almost five decades in energy $10\,\text{GeV} - 300\,\text{TeV}$.

The result of the joint fit is displayed in Figure 14. 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.
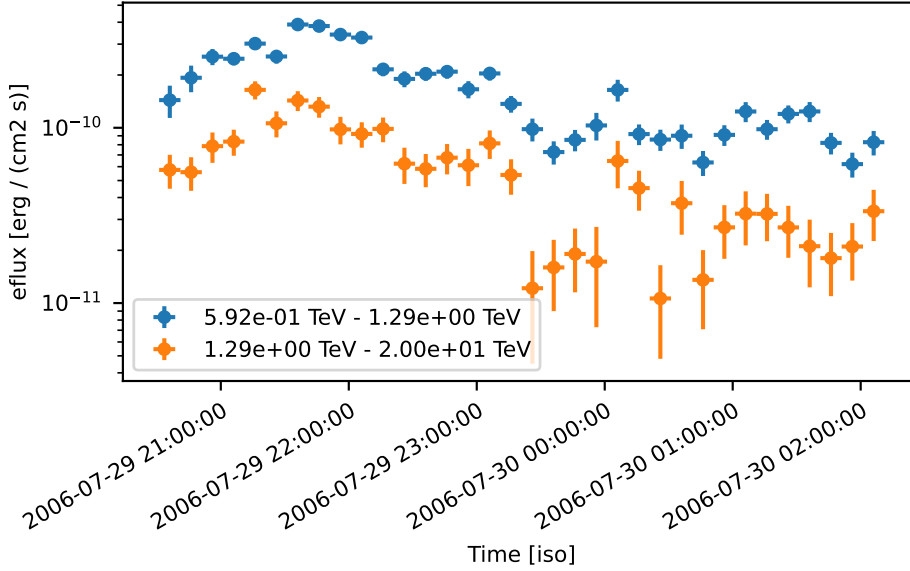
Fig. 13. Binned light curves in two different energy bands for the source PKS 2155-304 in two energy bands, (500 GeV - 1.5 TeV, and 1.5 TeV to 20 TeV) as observed by the H.E.S.S. telescopes in 2006. The coloured markers show the flux points in the different energy bands. The horizontal error illustrates the width of the time bin of 10 min. The vertical error bars show the associated assymetrical flux errors. The marker is set to the center of the time bin.
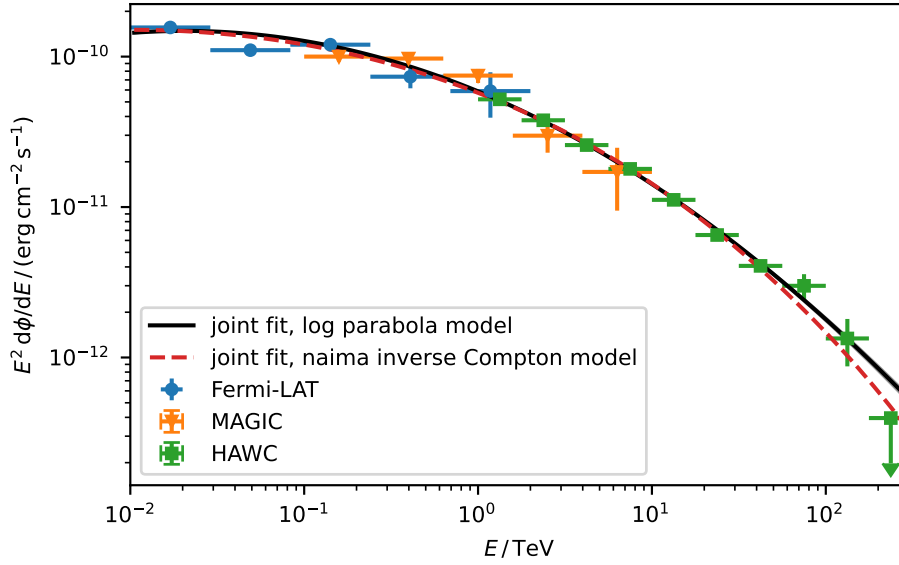

Fig. 14. A multi-instrument spectral energy distribution (SED) and combined model fit of the Crab Nebula. The colored markers show the flux points computed from the data of the different listed instruments. The horizontal error bar illustrates the width of the chosen energy band E_{Min}, E_{Max}. The 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 to small be visible.

4.5. Broadband SED Modeling

By combining Gammapy with astrophysical modelling codes, users can also fit astrophysical spectral models to γ-ray data. In γ-ray astronomy one typically observes two radiation production mechanisms, the so-called hadronic and leptonic scenarios. 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. 2022), Naima (Zabalza 2015), Jetset (Tramacere 2020) and Gamera (Hahn et al. 2022). Tyically those emission models predict broadband emission from radio, up to the very high-energy γ-ray range. 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 of this application, we use the same multi-instrument dataset described in the previous section and we fit it with an inverse Compton model computed with naima and wrapped in the Gammapy fitting routine with the NaimaSpectraModel class. We consider a log-parabolic electron distribution scattering: the synchrotron radiation produced by the same electrons; near and far infrared photon fields and the cosmic microwave background (CMB). We adopted the prescription on the target photon fields provided in the documentation of the *Naima* package [2]. The best-fit inverse Compton spectrum is represented with a red dashed line in Figure 14.

4.6. Surveys, Catalogs, and Population Studies

Sky surveys have a large potential for new source detections, and new phenomena discovery 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 of the preparation of the H.E.S.S. Galactic plane survey catalog (HGPS, H. E. S. S. Collaboration

---

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

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

The IACTs data reduction step is done in the same way than described in the previous sections but scale up to few thousands of observations. The object detection step consists a minima in finding local maxima in the significance (or TS) maps, given 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 choice, 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, that 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 an e.g. easily be achieved by relying on coordinate handling from Astropy or affiliated packages such as *AstroML* (Vanderplas et al. 2012) or *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[3]. So we can improve performances, optimize our analyses strategies, and identify the needs in term of parallelisation to process the large datasets provided by the surveys.

5. 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[4]. 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, ensures 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 punctual 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*[5] hosts different repositories related with the project. The software codebase may be found in the *gammapy* repository (see Table 3 and Figure 15 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[6], blocking the processes and alerting developers when fails occur. This is the case of the continuous integration workflow, which monitors the execution of the test coverage suite[7] using datasets from the *gammapy-data* repository[8]. 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.

---

[3] 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.

[4] `https://gammapy.org/team.html`

[5] `https://github.com/gammapy`

[6] `https://github.com/features/actions`

[7] `https://pytest.org`

[8] `https://github.com/gammapy/gammapy-data`

Language	files	blank	comment	code
Python API	272	14893	26382	34703
Python Tests	123	6432	1061	22555
DocStrings	272	0	0	26382
reStructuredText	133	4058	3149	9576
YAML	16	33	31	781
JSON	1	0	0	367
SVG	4	4	4	298
CSS	1	64	8	257
make	2	39	18	210
Batch	1	21	1	150
Markdown	5	31	0	126
INI	1	14	0	95
Cython	1	16	35	68
TOML	1	1	0	28
Total	438	19174	29628	46659

Table 3. Overview of used programming languages and distribution of code across the different file categories in the Gammapy code base. The most right column list the total number of lines in a file. The *comment* column lists the number of comments in the files (including method and class docstrings), the *blank* column lists the number of blank lines in the file. The uppermost rows distinguish between code that implements actual functionality and code the implements tests. All documentataion in Gammapy is implemented as docstrings, reStructuredText (RST) files or notebooks.
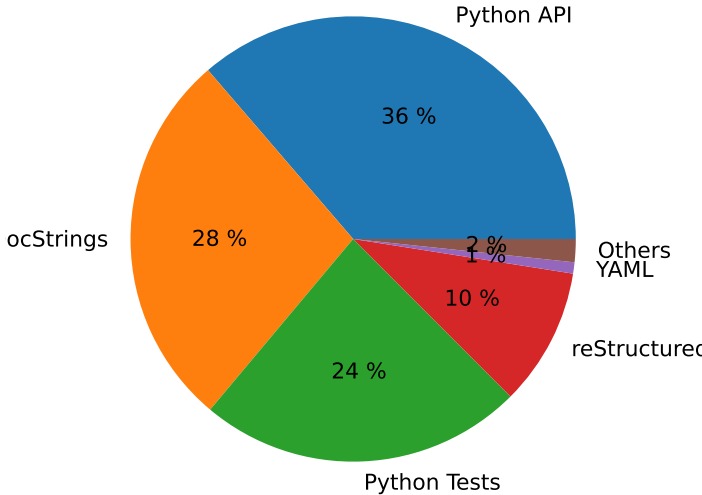

Fig. 15. Overview of used programming languages and distribution of code across the different file categories in the Gammapy code base.

Other automated tasks, executing in the *gammapy-benchmarks*[9] 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 doc-

umentation building and publishing workflow within the *gammapy-docs* repository[10] (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.

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[11] and as a binary package to the *conda-forge* and *astropy* Anaconda repository[12] channels. At this time, Gammapy is also available as a Debian Linux package[13]. 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 Astronomy Source Code Library[14] and Zenodo[15] digital libraries for software. The Zenodo record is synchronised with the codebase GitHub repository so that every release triggers the update of the versioned record. In addition, the next release of Gammapy will be added to the Open-source scientific Software and Service Repository[16] and indexed in the European Open Science Cloud catalog [17].

In addition Gammapy is also listed in the *SoftWare Heritage* [18] (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 [19], is using the Gammapy SWHIDs to register the releases as scientific products [20] of open science.

5.4. Documentation and User-support

Gammapy provides its user community with a tested and versioned up-to-date online documentation[21] (Boisson et al. 2019) built with Sphinx[22] scanning the codebase Python scripts, as well as a set of RST files and Jupyter notebooks.

---

[9] https://github.com/gammapy/gammapy-benchmarks

[10] https://github.com/gammapy/gammapy-docs
[11] https://pypi.org
[12] https://anaconda.org/anaconda/repo
[13] https://packages.debian.org/sid/python3-gammapy
[14] https://ascl.net/1711.014
[15] https://zenodo.org/record/5721467o
[16] https://projectescape.eu/ossr
[17] https://eosc-portal.eu
[18] https://softwareheritage.org
[19] https://hal.archives-ouvertes.fr
[20] https://hal.archives-ouvertes.fr/hal-03713410v1
[21] https://docs.gammapy.org
[22] https://www.sphinx-doc.org

The documentation includes a handwritten 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 (Óscar 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 as Jupyter notebooks that do not fit in the Gammapy core documentation but which may be relevant as specific use cases. Contributions happen via pull requests to the *gammapy-recipes* GitHub repository and merged after a short review. All notebooks in the repository are tested and published in the Gammapy recipes webpage[23] automatically using GitHub actions.

A growing community of users is gathering around the Slack messaging[24] and GitHub discussions[25] support forums, providing valuable feedback on the Gammapy functionalities, interface and documentation. Other communication channels have been set like mailing lists, a Twitter account[26], regular public coding sprint meetings, hands-on session 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[27] and Astropy's APE (Greenfield 2013) system. PIG are self-contained documents which outline a set of larger changes to the Gammapy code base. This includes larger 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 the 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 [28].

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; we basically follow 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 user by runtime warnings first and then implemented in the subsequent minor release. This approach we consider 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).

6. 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 [29]. 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 software as well as 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[30]. 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 as-

---

[23] `https://gammapy.github.io/gammapy-recipes`

[24] `https://gammapy.slack.com`

[25] `https://github.com/gammapy/gammapy/discussions`

[26] `https://twitter.com/gammapyST`

[27] `https://peps.python.org/pep-0001/`

[28] `https://docs.gammapy.org/0.20/development/pigs/index.html`

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

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

tronomy, which relies on the scientific Python ecosystem, including Numpy and 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 description 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 milestone 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.

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 selected as the base library for the future science tools for CTA, 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 HAWC, 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 and

a common vision of open and reproducible science for the future.

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

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

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

Collaboration, H. 2018, H.E.S.S. first public test data release, For terms of use, see README.txt and hess__dl3_dr1.pdf

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

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)

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

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

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

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

MAGIC Collaboration. 2016, Astroparticle Physics, 72, 76

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

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

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

Nigro, C., Sitarek, J., Gliwny, P., et al. 2021b, arXiv e-prints, arXiv:2112.14573

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

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

Vanderplas, J., Connolly, A., Ivezić, Ž., & Gray, A. 2012, in Conference on Intelligent Data Understanding (CIDU), 47 –54

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

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

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

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

[1] Amazon, Atlanta GA, US
[2] Aperio Software, Insight House, Riverside Business Park, Stoney Common Road, Stansted, Essex, CM24 8PL, UK
[3] Astroparticle and Cosmology Laboratory CNRS, Université de Paris, 10 Rue Alice Domon et Léonie Duquet, 75013 Paris, France
[4] CNRS
[5] CTA Observatory gGmbH, Via Piero Gobetti 93/3 40129 Bologna, Italy
[6] California Institute of Technology, Pasadena CA, US
[7] Center for Astrophysics | Harvard & Smithsonian, CfA, 60 Garden St., 02138 Cambridge MA, US
[8] DAp/AIM, CEA-Saclay/CNRS, France
[9] DESY
[10] Department of Physics, University of the Free State, PO Box339, Bloemfontein 9300, South Africa
[11] Deutsches Elektronen-Synchrotron, DESY, Platanenallee 6 15738 Zeuthen, Germany
[12] Facebook, 1 Hacker Way, Menlo Park, CA 94025, US
[13] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen Centre for Astroparticle Physics, Erwin-Rommel-Str. 1 91058 Erlangen, Germany
[14] HeidelbergCement AG, Berliner Straße 6, 69120 Heidelberg, Germany
[15] INAF/IASF, via Ugo La Malfa 153, 90146 Palermo, Italy
[16] Institut de Fisica d'altes Energies, IFAE, Edifici Cn, Campus UAB, 08193 Bellaterra (Barcelona), Spain
[17] Institute de Recherche en Astrophysique et Planetologie, IRAP,
[18] Instituto de Astrofísica de Andalucía - CSIC, Gta. de la Astronomía, 18008 Granada, Spain
[19] Max Planck Institut für Kernphysik, MPIK, Saupfercheckweg 1, 69117 Heidelberg, Germany
[20] Michigan State University, Department of Physics and Astronomy, 428 S Shaw Ln East Lansing, MI 48824, US
[21] Mozilla, Mountain View, CA
[22] National Tsing Hua University Institute of Astronomy, Hsinchu, Taiwan
[23] Pamyra
[24] Point 8 GmbH, Rheinlanddamm 201, 44139 Dortmund, Germany
[25] Space Telescope Science Institute, STScI, 3700 San Martin Drive, 21218 Baltimore MD, US
[26] TU Dortmund University, Otto-Hahn-Str. 4a 44227 Dortmund, Germany
[27] Universidad Complutense de Madrid EMFTEL, UCM, Plaza de la Ciencias, 28040 Madrid, Spain
[28] Universidade de São Paulo, Brazil
[29] Université Savoie Mont Blanc, CNRS, Laboratoire d'Annecyde Physique des Particules - IN2P3, 74000 Annecy, France
[30] unknown