The Southern Wide-field Gamma-ray Observatory

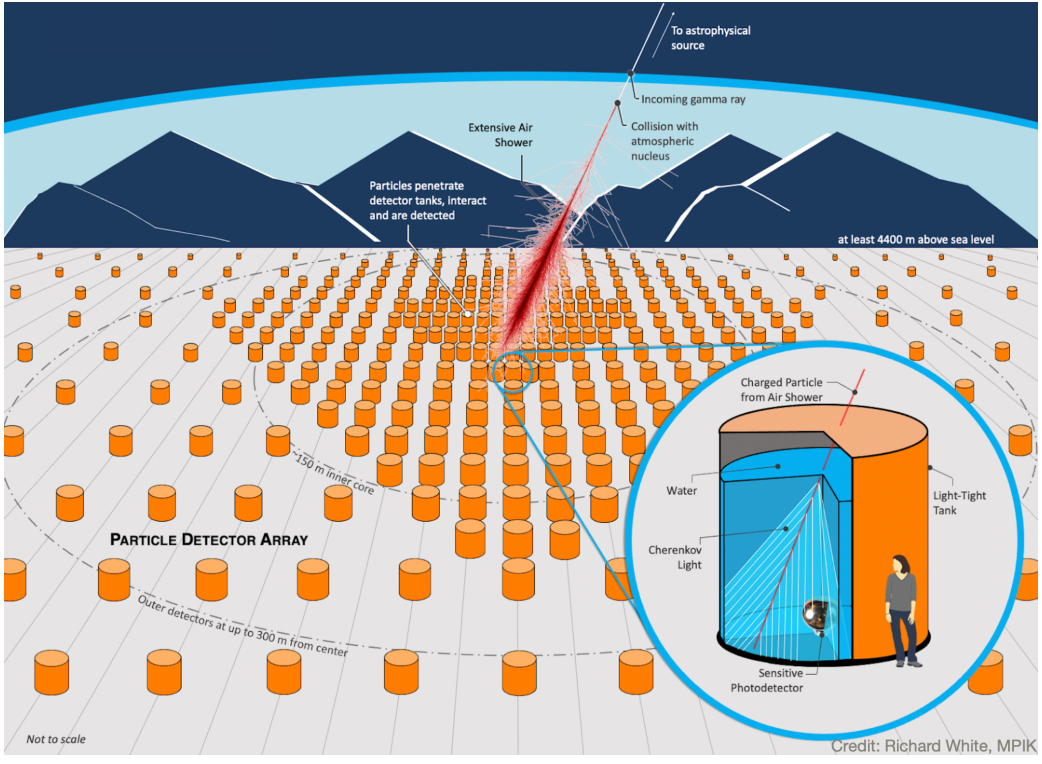
Gammapy in SWGO

Gammapy coding sprint in Heidelberg

10/06/2024

Laura Olivera-Nieto

Intro to SWGO



- In R&D phase
- Water cherenkov particle detection
- In South america
- 100s of GeV up to PeV
- Bigger and better than HAWC

SWGO and Gammapy

(Draft) SWGO Software Concept

M. Peresano[1], H. Schoorlemmer[2], J. Hinton[3], L. Olivera Nieto[3], F. Werner[3], add your name[X]

[1] *University of Turin & INFN, Italy*

[2] *Radboud University Nijmegen, Netherlands*

[3] *MPIK Heidelberg, Germany*

SWGO
The Southern Wide-field Gamma-ray Observatory

2. Data analysis tools
   - Read the data and IRFs and produce higher level data products such as sky-maps, spectra, light-curves, flux maps...
   - Multi-source modeling and support for different event classes essential.
   - Support for many different types of analysis (from large-scale structures to transients)
   - Define and implement basic sanity checks and diagnostics to be routinely applied to standard analyses.
   - Provide tools to simulate data (binned or at event list level) for a given set of IRFs.

SWGO and Gammapy

6.2 Early thoughts on implementation

6.2.1 Data selection

The performance of the data selection tools will be very dependent on the packaging and metadata chosen when packaging reconstructed data. The need to open and go through files with large number of event entries should be avoided, and rather use file header information. Experience of the Fermi-LAT data center might be a useful reference in terms of providing access to users to a large amount of data efficiently and in a well documented manner.

The way to read the selected data and IRFs inside the science tools must be defined early and follow existing conventions (like this proposal).

6.2.2 Data analysis

Should aim to make maximum use of existing tools shared with the rest of the community (e.g. Gammapy). Specific needs of SWGO could either be taken into account by preserving the current role of SWGO members in the Gammapy steering and development process, or developing a SWGO-dedicated package that uses Gammapy as base library. In both cases, compatibility with community standards and feasibility of joint analysis with other instruments should be preserved.

The large field of view of SWGO will require support of HEALPix pixelization schemes in every step of the data analysis process in order to allow for full-sky analyses and large structure searches. This is currently partially lacking (although planned) in Gammapy for example.

Support of analyses for which the input data is not pre-selected for gamma-likeness should be designed carefully and take into account the increased file sizes of these datasets. Also the case at the data selection step.

6.3 Interfaces

- We want to be able to use Gamapy
- Both for SWGO-only analysis but also for joint analysis with e.g. CTA

SWGO and Gammapy

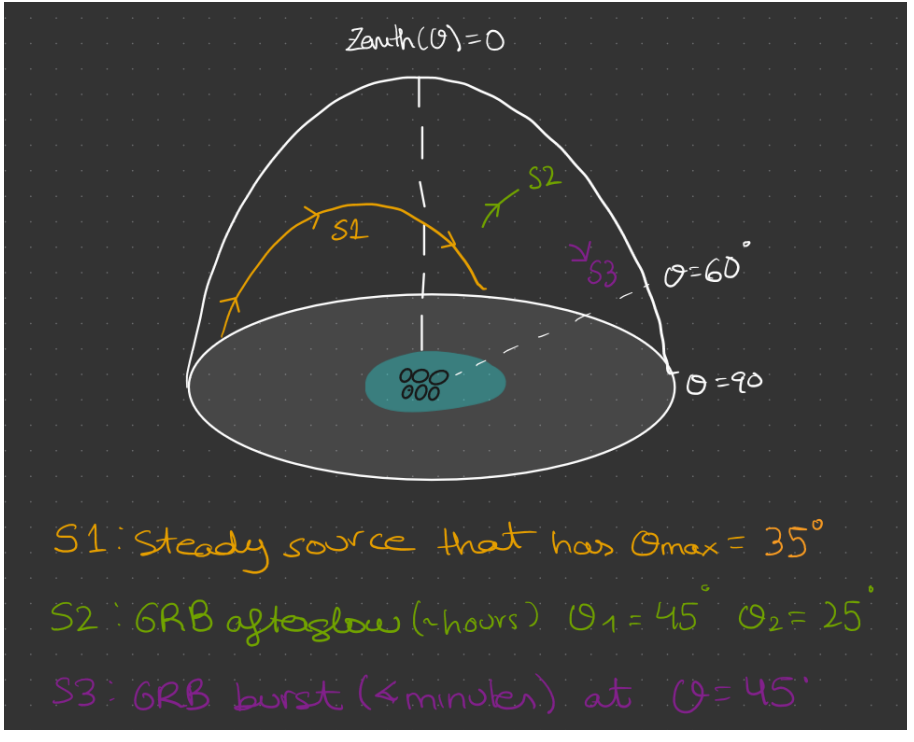
- In fact we already use Gammapy!
- IRFs are stored into gammapy.irf objects
- Science benchmarks derived using Gammapy

2 Benchmark Derivation

Except for the cosmic ray benchmark, all benchmarks were derived using the latest available version (*pass3_v7*) of the simulations and instrument response functions (IRFs) and the software *Gammapy* version 1.1.

SWGO science cases

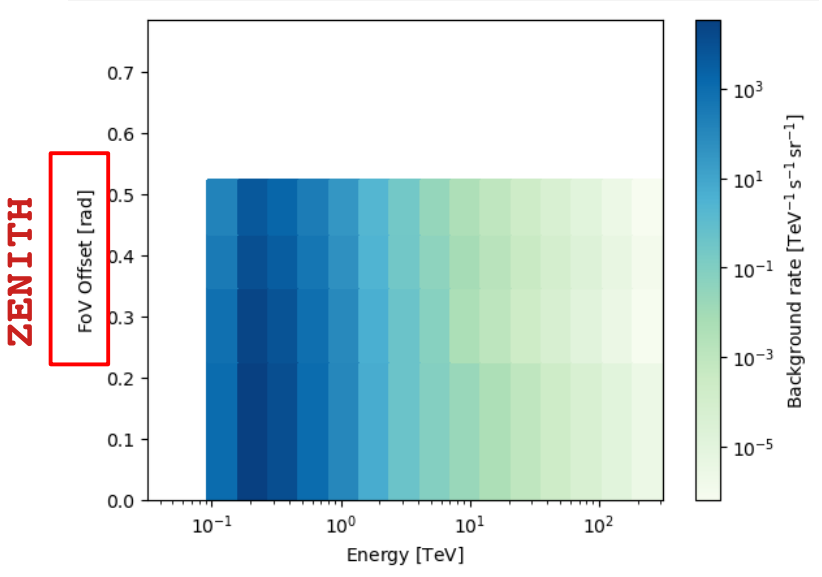
Science Case
Transient Sources: Gamma-ray Bursts
Galactic Accelerators: PeVatron Sources
Galactic Accelerators: PWNe and TeV Halos
Galactic Accelerators: Source Confusion
Diffuse Emission: Fermi Bubbles
Fundamental Physics: Dark Matter from Galactic Halo
Cosmic-rays: Mass-resolved dipole/multipole anisotropy



SWGO IRFs

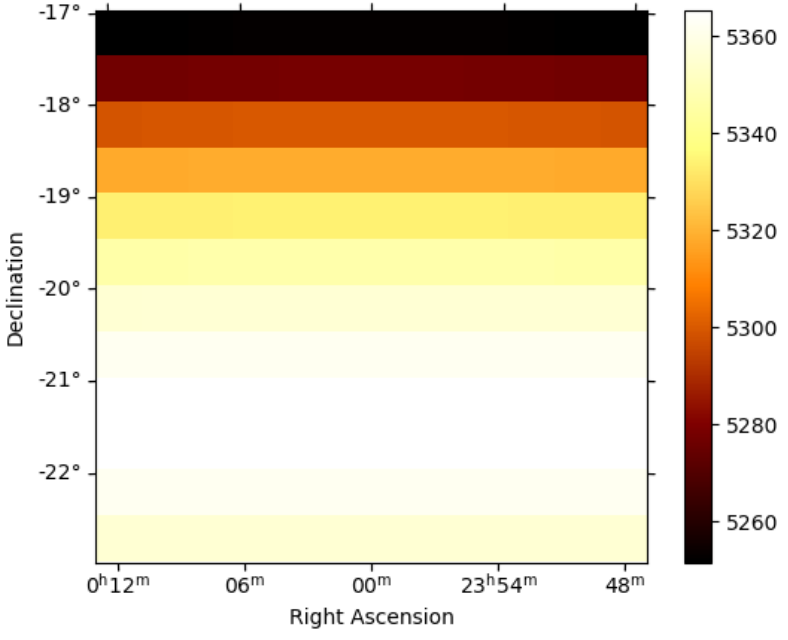
"Table IRF"

Instantaneous response binned in zenith and energy (true or reco, depends on which IRF)



"Map IRF"

Sky map with response integrated over a source transit (so,integrated in zenith).



Case 1 - short transient

- Zenith barely changes, assume that neither do IRFs.
- Can use SpectrumDaset!
- Works because of this bit in gammapy/makers/utils.py#L31

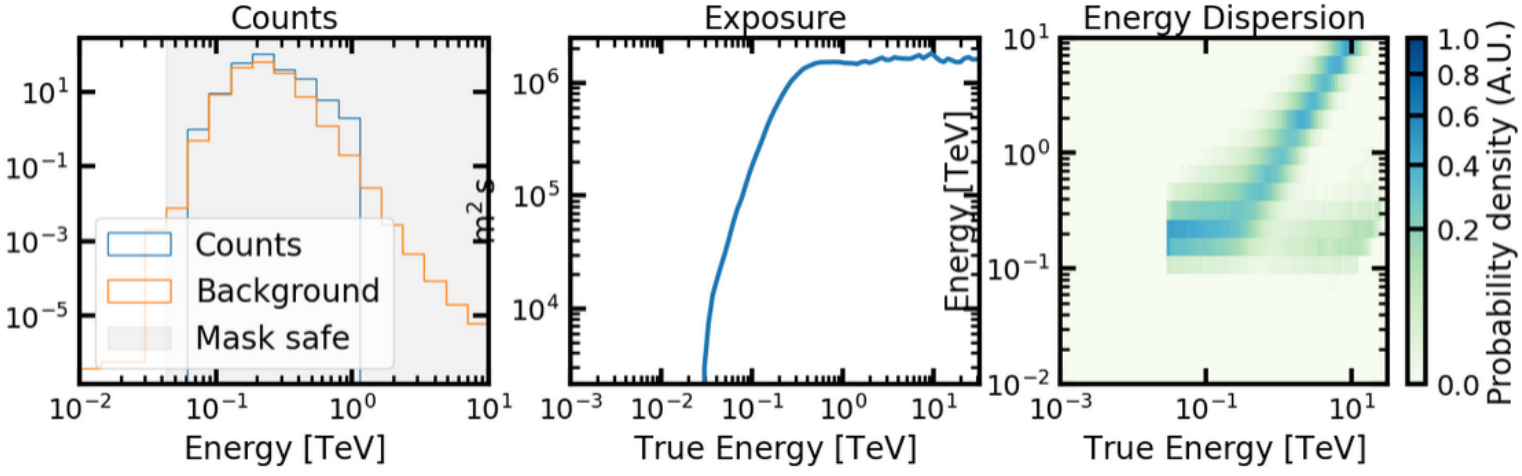
```python
if irf.fov_alignment == FoVAlignment.ALTAZ:
    if not isinstance(pointing, FixedPointingInfo) and isinstance(
        irf, BackgroundIRF
    ):
        raise TypeError(
            "make_map_background_irf requires FixedPointingInfo if "
            "BackgroundIRF.fov_alignement is ALTAZ",
        )

    # for backwards compatibility, obstime should be required
    if obstime is None:
        warnings.warn(
            "Future versions of gammapy will require the obstime keyword for this function",
            DeprecationWarning,
        )
        obstime = pointing.obstime

    pointing_altaz = pointing.get_altaz(obstime)
    altaz_coord = sky_coord.transform_to(pointing_altaz.frame)

    # Compute FOV coordinates of map relative to pointing
    fov_lon, fov_lat = sky_to_fov(
        altaz_coord.az, altaz_coord.alt, pointing_altaz.az, pointing_altaz.alt
    )
```

Case 1 – short transient



Cases 2/3 – long transient or steady source

- The IRFs are combined weighting each zenith bin by how much the source spends in it.
- Results in Map IRFs (HAWC style)
- Can use in MapDatasetMaker almost completely out of the box
- For case 2, IRFs would be "custom" made for that analysis
- For case 3 we multiply the bkg and effective exposure made for 1 full transit by the required number of transits
- Question is how much of this goes into gammapy and at what point

Case 3 – exposure calculation

- You have IRFs for 1 transit and a bunch of events and GTIs.
- How many transits are in the data? You can figure it out with the GTIs
- Currently done outside gammapy
- Note this is not a problem for SWGO yet, as there is no data! But it is for HAWC

Clunky things

- **Event types handling in data reduction.** Right now we save separate HDU tables to the same file and perform data reduction in a loop.

- **Exposure calculation from GTIs.** Easy for case 1, unclear for case 2/3.

- **Metadata in MapDataset maker** (#4281)

```
# DEFINING THE MISSING MAPDATASET ATTIBUTES (SEE https://github.com/gammapy/gammapy/pull/4281)
info = {}
info["OBS_MODE"] = 'DRIFT'
info['ALT_PNT'] = 90
info['AZ_PNT'] = 0
info['TSTART'] = 0
info['TSTOP'] = 1
info['DURATION'] = 1
info['MJDREFI'] = 44244
info['MJDREFF'] = 0
info['GEOLON'] = 0
info['GEOLAT'] = 12
info['ALTITUDE'] = 4000
```

Clunky things

- **Default rad axis is too small.**

```
RAD_MAX = 0.66
RAD_AXIS_DEFAULT = MapAxis.from_bounds(
    0, RAD_MAX, nbin=66, node_type="edges", name="rad", unit="deg"
)
```

- **maxaeff mask safe only a lower bound.**

- **Background map interpolation:** We save background map as a rate per sr so that it can safely be interpolated to any geometry. Would be nice if MapDatasetMaker did this if the background units are $TeV^{-1} \cdot sr^{-1}$

```
def get_background_counts(bkg_map, geom):
    """Extract background counts from the map with a rate per sr and TeV.

    Parameters
    ----------
    bkg_map : :class:`~gammapy.maps.WcsNDMap`
        Background rate (1/TeV/sr) map.
    geom : :class:`~gammapy.maps.WcsGeom`
        Geometry on which to interpolate the
        input map.

    Returns
    -------
    bkg_counts : :class:`~gammapy.maps.WcsNDMap`
        Background counts map.
    """
    # Interpolate it to the geometry
    bkg_rate_this_geom = bkg_map.interp_to_geom(geom, preserve_counts=False)

    # Create a map containing the bin sizes
    bin_sizes = Map.from_geom(geom, unit="sr TeV")

    # Take into account both solid angle size (sr) and energy size (TeV)
    bin_sizes.data = geom.bin_volume().to_value("sr TeV")

    bkg_counts = bkg_rate_this_geom * bin_sizes

    # If something went wrong with the units, this line would fail
    bkg_counts.quantity = bkg_counts.quantity.to_value("")

    return bkg_counts
```

Clunky things

- **Constructing a DRIFT observation.** (this one might be my own incompetence)

```
def construct_observation(irfs, observatory_frame, time, deltaT, obs_id):

    pointing = FixedPointingInfo(mode = PointingMode.DRIFT,
                                 fixed_altaz= SkyCoord(0*u.deg, 90*u.deg, frame=observatory_frame),
                                location=observatory_frame.location,
                                time_start =time,
                                time_stop = time + deltaT)

    observation = Observation.create(pointing = pointing,
                    location = observatory_frame.location,
                    obs_id = obs_id,
                    livetime = deltaT,
                    tstart = time,
                    tstop = time + deltaT,
                    irfs = irfs)
    return observation
```

Missing things

- **Joint significance maps (@Quentin)**. Ideally fast as well :-)
- **Joint differential sensitivity curves.** Not really missing but not yet in gammapy.

```
class SensitivityEstimator:
    """Estimate sensitivity from map datasets.

```

- **Energy-dependent observation time.** Useful for GRBs. Also region size, but this is doable by adjusting bkg rate
- **Transit information metadata** (related to exposure mentioned earlier). Similar to livetime but not really…
- **Full Healpix support** (I did some effort but stopped short)

Conclusion

- SWGO plans to use Gammapy for data analysis
  - directly gammapy or a gammapy-based package?
- It already basically works out of the box with minor awkwardness
  - need involvement in development to make sure it stays this way this is a note to myself :-')
- SWGO will use event types: joint analyses are critical. This includes maps and sensitivity calculations.
- HealPix support for full-sky needs
- Thoughts??