UPDATE ON CTA DARK MATTER ANALYSIS WITH GAMMAPY

The extension of the Dark Matter module

Judit Pérez-Romero, judit.perez@uam.es, IFT UAM-CSIC

Jose Enrique Ruiz, jer@iaa.es, IAA-CSIC

Miguel Á. Sánchez-Conde, miguel.sanchezconde@uam.es, IFT UAM-CSIC

UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

PROPOSAL FOR GAMMAPY

DM community needs

Create a standard analysis pipeline for DM searches

Based on DM CTA Conventions

https://forge.in2p3.fr/projects/dmep-swg-standards-and-conventions-for-cta-dm-analyses/wiki

Gammapy Implementations

- **Create tutorial notebooks** for users

Implement new features ⟲ Identify needs in notebooks

- **Extend the existing dm-module**
  - Add some new classes
  - Refactor code

REVIEW OF DARK MATTER SCIENCE

- Goal of the CTA Dark Matter (DM) SWG:

  Perform an state-of-the-art study for the sensitivity of CTA for DM searches

General need of a software to perform the standard DM analysis → $\gamma\pi$

- Open
- Flexible
- Fast

https://github.com/gammapy/gammapy-meetings/blob/master/coding-sprints/2019-02_Paris/slides/extension_dark_matter_module.pdf

- WIMP DM scenario:

$$\frac{d\Phi_{DM}}{dE}(E, l.o.s, \Delta\Omega, z) = \frac{< \sigma v >}{2m_{DM}^2}\frac{dN}{dE}(E, z) \times J(l.o.s, \Delta\Omega, z)$$

Total DM flux coming from an astrophysical object

E = Energy
l.o.s = Line of sight
ΔΩ = Angular extension
z = redshift

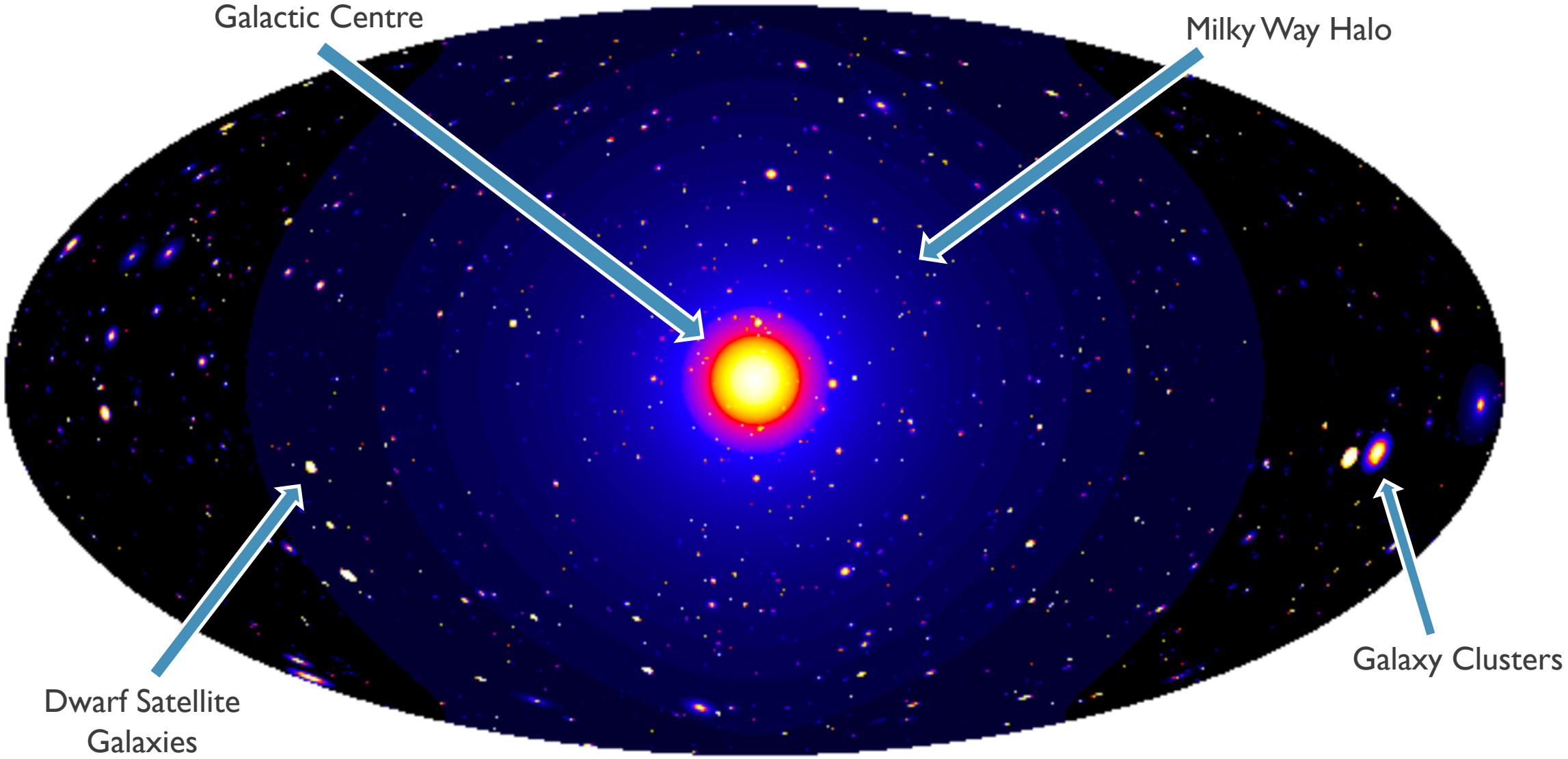
$\frac{d\phi}{dE}$ **Particle Physics Model**

- Difficult to parametrize analytically
- WIMP models tabulated in *Cirelli+12*

Astrophysical Model

- Models the DM distribution of the object
- State-of-the-art computations: customized programs/simulations

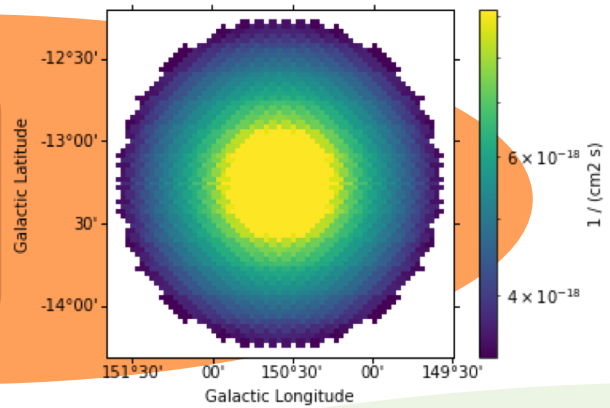
Dark Matter induced gamma-ray sky simulation



Pieri+09, arXiv:0908.0195

Input J-factor (Clumpy/own cooked file):

- Point-like: one value
- 2D skymap
- 2D skymap + mask



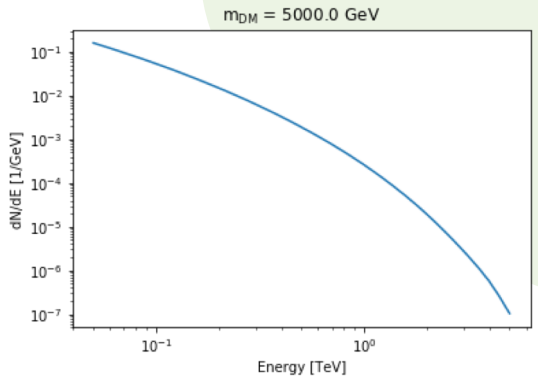
Colour legend:

- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

J

+

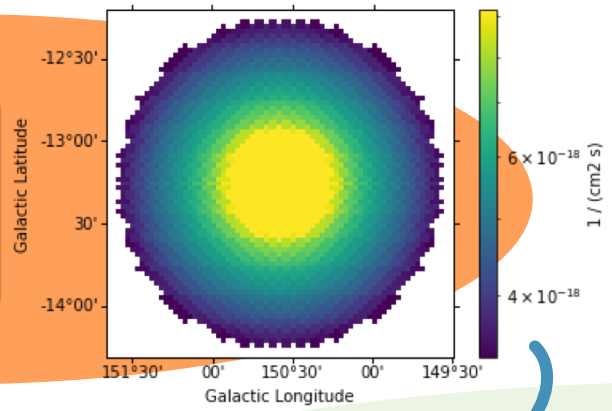
$\frac{d\phi}{dE}$ Particle Physics Model in Gammapy



Determined by:

- DM Mass
- Annihilation channel

Input J-factor (Clumpy/own cooked file):
- Point-like: one value
- 2D skymap
- 2D skymap + mask

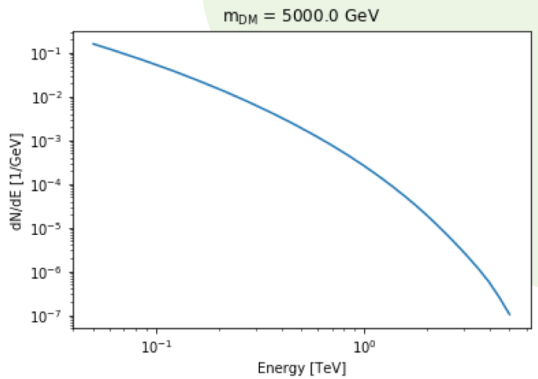


Colour legend:
- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

J

+

$\frac{d\phi}{dE}$ Particle Physics Model in Gammapy



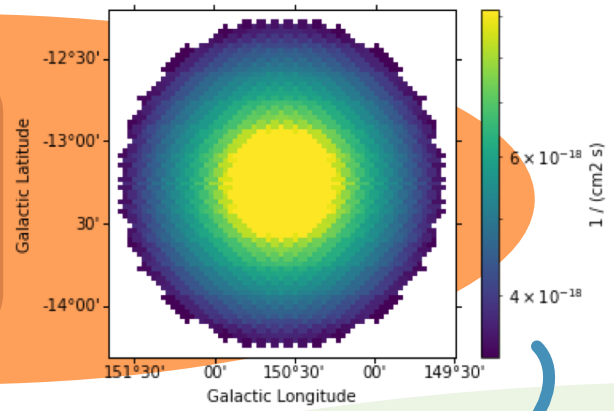
Determined by:
- DM Mass
- Annihilation channel

$\frac{d\Phi_{ann}}{dE}$

Total DM flux:
- Spectral information
- **DM Annihilation Spectral Model**

- Need to introduce 2D templates for the spatial information

Input J-factor (Clumpy/own cooked file):
- Point-like: one value
- 2D skymap
- 2D skymap + mask

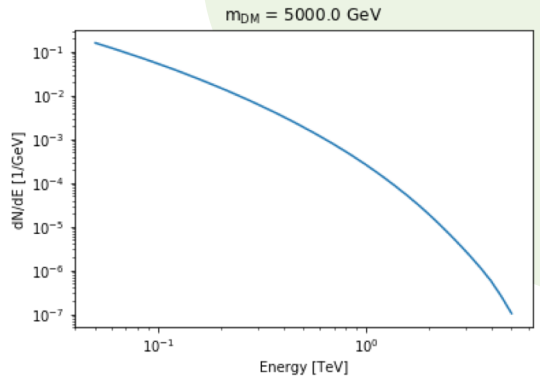


Colour legend:
- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

J

+

$\frac{d\phi}{dE}$ Particle Physics Model in Gammapy



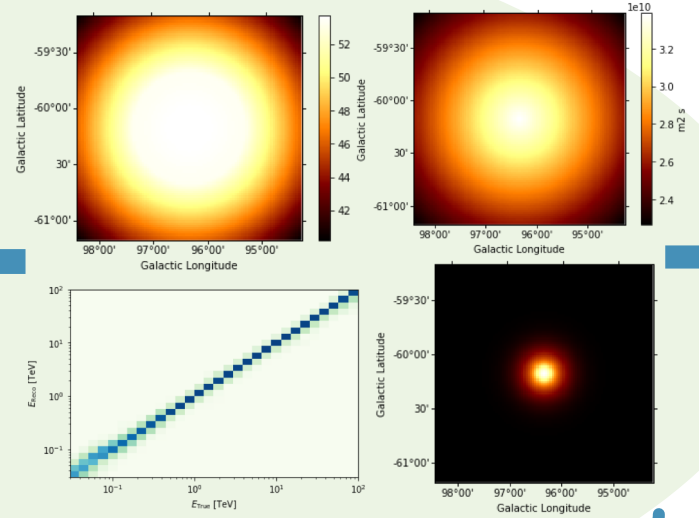
Determined by:
- DM Mass
- Annihilation channel

$\frac{d\Phi_{ann}}{dE}$

Total DM flux:
- Spectral information
- `DM Annihilation Spectral Model`

+

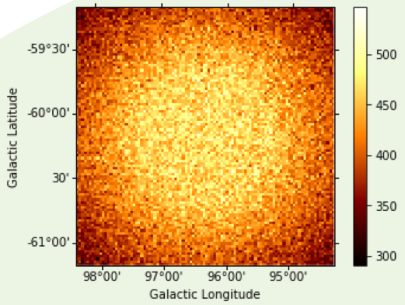
CTA IRFs



- Need to introduce 2D templates for the spatial information

`DM Dataset ON/OFF`

DM Dataset

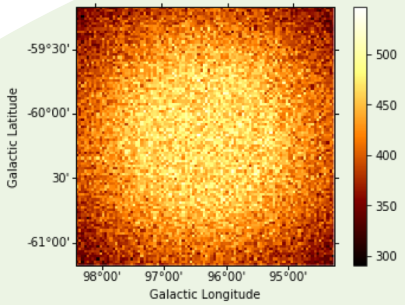


- Random Poisson realization
  `dataset.fake()`

Colour legend:

- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

DM Dataset



- Random Poisson realization
  `dataset.fake()`

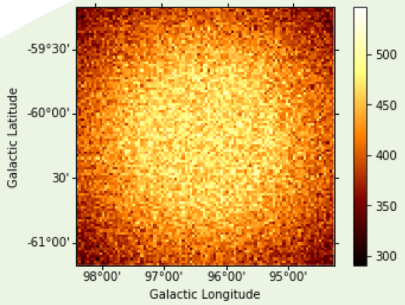
Fit the dataset, our "observation":

- Range in masses for different annihilation channels
- Used likelihood: **wstat** (ON/OFF observation) + **nuisance parameters**
- Need O(100) realizations to have good statistics
- Test Statistics (TS) versus null hypothesis

Colour legend:

- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

DM Dataset



- Random Poisson realization
  `dataset.fake()`

Fit the dataset, our "observation":

- Range in masses for different annihilation channels
- Used likelihood: **wstat** (ON/OFF observation) + **nuisance parameters**
- Need O(100) realizations to have good statistics
- Test Statistics (TS) versus null hypothesis

Signal!:

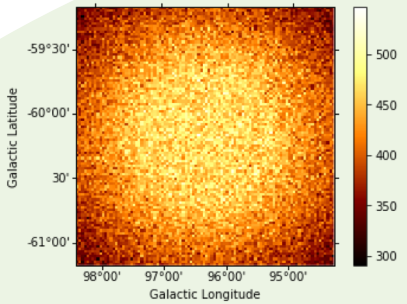
- Treat as an usual astrophysical source
- 3D analysis

No Signal
(historically happened in DM searches)

Colour legend:

- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

DM Dataset



- Random Poisson realization
  `dataset.fake()`

Fit the dataset, our "observation":

- Range in masses for different annihilation channels
- Used likelihood: **wstat** (ON/OFF observation) + **nuissance parameters**
- Need O(100) realizations to have good statistics
- Test Statistics (TS) versus null hypothesis

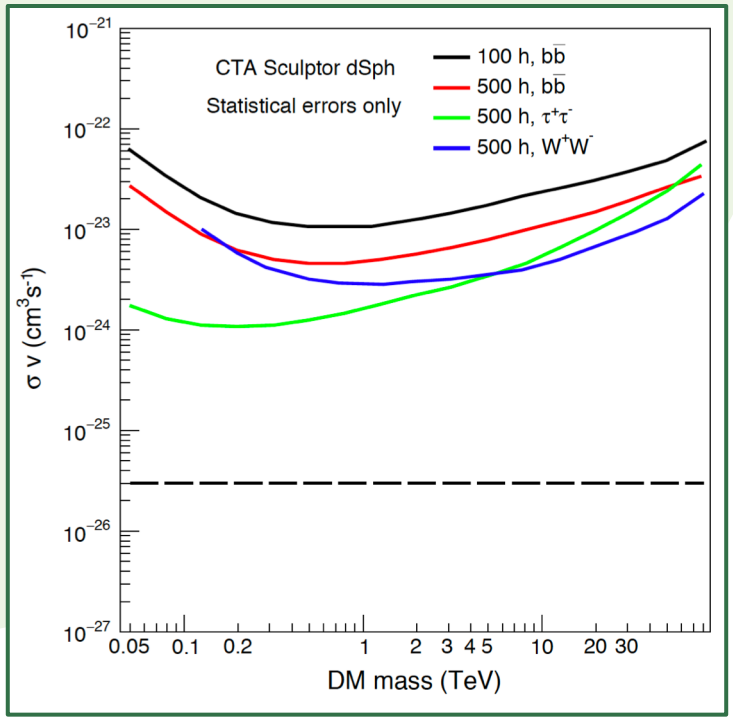
No Signal
(historically happened in DM searches)

TS Confidence Levels on DM parameter space
$\langle\sigma v\rangle$ vs DM mass

Signal!:

- Treat as an usual astrophysical source
- 3D analysis

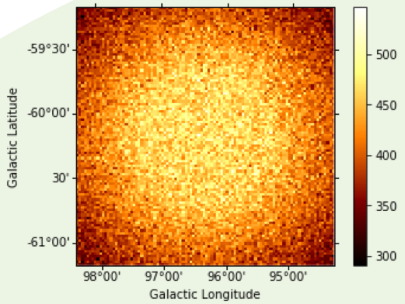
Acharya+17 [*CTA Cons.*], arXiv:1709.07997



Colour legend:

- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

DM Dataset



- Random Poisson realization
  `dataset.fake()`

Fit the dataset, our "observation":

- Range in masses for different annihilation channels
- Used likelihood: **wstat** (ON/OFF observation) + **nuisance parameters**
- Need O(100) realizations to have good statistics
- Test Statistics (TS) versus null hypothesis

No Signal
(historically happened in DM searches)

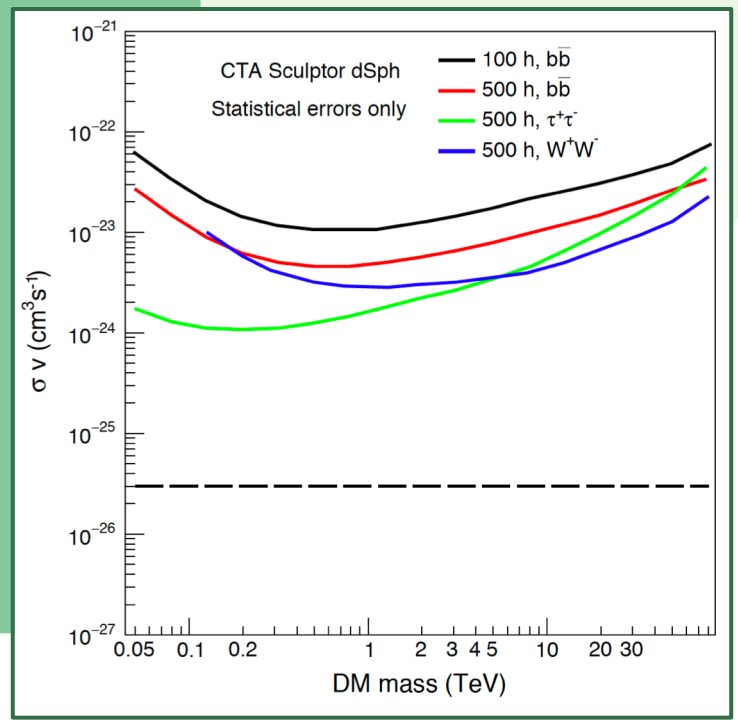
TS Confidence Levels on DM parameter space
$\langle\sigma v\rangle$ vs DM mass

`SigmaVEstimator`

Signal!:

- Treat as an usual astrophysical source
- 3D analysis

Acharya+17 [*CTA Cons.*], arXiv:1709.07997



Colour legend:

- Already implemented in Gammapy (or minor improvements)
- Needs to be implemented in Gammapy (or major improvements)
- Created outside Gammapy

Deconstructing the Dark Matter SigmaVEstimator

Setup

```
from gammapy.irf import load_cta_irfs
from gammapy.spectrum import CountsSpectrum
from gammapy.astro.darkmatter.utils import SigmaVEstimator, DMDatasetOnOff
from gammapy.astro.darkmatter import DarkMatterAnnihilationSpectralModel
```

Define parameters for simulated observation

```
# -
livetime = 100 * u.h
offset = 0.5 * u.deg
FOVLON=0 * u.deg
FOVLAT=0 * u.deg

energy = np.logspace(-1.8, 1.5, 20) * u.TeV

# DMAnnihilation Model
JFAC = 3.41e19 * u.Unit("GeV2 cm-5") # point source
mDM = 5000*u.Unit("GeV")
channel = "b"
redshift = 0
```

psf
aeff
bkg

Get IRFs

```
# Load IRFs
filename = (
    "$GAMMAPY_DATA/cta-1dc/caldb/data/cta/1dc/bcf/South_z20_50h/irf_file.fits"
)
cta_irf = load_cta_irfs(filename)
```

Build the background counts

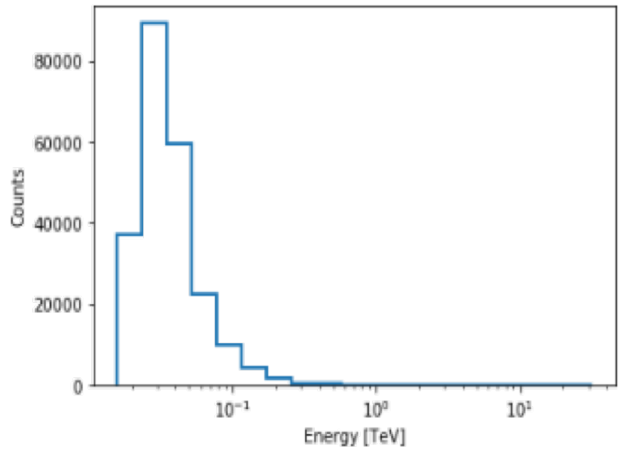
```
containment = 0.68
energies = np.sqrt(energy[1:] * energy[:-1])
psf = cta_irf["psf"].to_energy_dependent_table_psf(theta=offset)
on_radii = psf.containment_radius(energy=energies, fraction=containment)
solid_angles = 2 * np.pi * (1 - np.cos(on_radii)) * u.sr

#
aeff.data.data *= containment
```

```
bkg_data = cta_irf["bkg"].evaluate_integrate(
    fov_lon=FOVLON, fov_lat=FOVLAT, energy_reco=energy
)
bkg = CountsSpectrum(
    energy[:-1],
    energy[1:],
    data=(bkg_data * solid_angles).to_value("h-1")*livetime
)
```

```
bkg.plot_hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11cd12278>
```

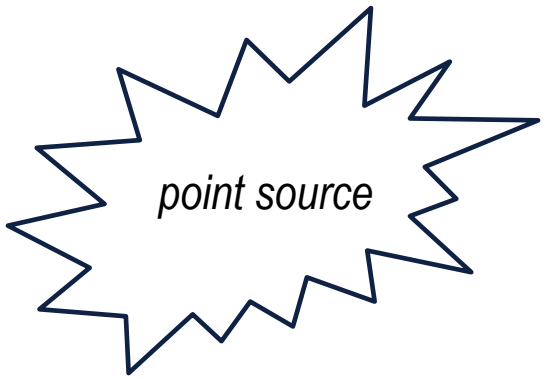


https://github.com/peroju/dm-gammapy/blob/master/notebooks/DarkMatterUseCaseSigmaVEstimator.ipynb

DarkMatterAnnihilationSpectralModel
astro/darkmatter/spectra.py

```
# DMAnnihilation Model
JFAC = 3.41e19 * u.Unit("GeV2 cm-5") # point source
mDM = 5000*u.Unit("GeV")
channel = "b"
redshift = 0

# IRF terms
aeff.data.data *= containment
bkg = CountsSpectrum
livetime = 100 * u.h
```



Simulate OnOff observation with DM emission model

```
# DM Flux
flux_model = DarkMatterAnnihilationSpectralModel(
    mass=mDM,
    channel=channel,
    jfactor=JFAC,
    z=redshift
)
```

```
#from gammapy.modeling.models import Absorption, AbsorbedSpectralModel
#absorption_model = Absorption.read_builtin('dominguez')
#flux_model = AbsorbedSpectralModel(flux_model, absorption_model, redshift)
```

```
dataset = DMDatasetOnOff(
    aeff=aeff,
    model=flux_model,
    livetime=livetime,
    acceptance=1,
    acceptance_off=5,
)
```

```
dataset.fake(bkg)
```

```python
nuissance = dict(
    j=JFAC,
    jobs=JFAC,
    sigmaj=0.1*JFAC,
    sigmatau=0.01,
    #width=5,
    #steps=15,
)
dataset.nuissance = nuissance
```

DMDatasetOnOff
astro/darkmatter/utils.py


```python
class DMDatasetOnOff(SpectrumDatasetOnOff):
    """Dark matter dataset OnOff with nuissance parameters and likelihood."""

    def __init__(self, nuissance=None, **kwargs):
        super().__init__(**kwargs)
        if nuissance is None:
            nuissance = {
                "j": None,
                "jobs": None,
                "sigmaj": None,
                "sigmatau": None,
                "width": None,
                "steps": None,
            }
        self.nuissance = nuissance

    def likelihood(self):
        wstat = super().likelihood()
        liketotal = wstat
        if self.check_nuissance():
            liketotal += self.jnuissance() + self.gnuissance()
        return liketotal

    def jnuissance(self):
        exp_up = (np.log10(self.nuissance["j"].value) - np.log10(self.nuissance["jobs"].value)) ** 2
        exp_down = 2 * (np.log(self.nuissance["sigmaj"].value) ** 2)
        up = np.exp(-1 * exp_up / exp_down)
        down = (
            np.log(10)
            * self.nuissance["jobs"].value
            * np.sqrt(2 * np.pi)
            * np.log10(self.nuissance["sigmaj"].value)
        )
        res = up / down
        return -2 * np.log(res)

    def gnuissance(self):
        res = 1 / (np.sqrt(2* np.pi) * self.nuissance["sigmatau"])
        return -2 * np.log(res)
```

The SigmaVEstimator *(fitting / likelihood profiling in nested loops)*

Enable inspection

```
import logging
logging.basicConfig()
#logging.getLogger("gammapy.astro.darkmatter.utils").setLevel("WARNING")
logging.getLogger("gammapy.astro.darkmatter.utils").setLevel("DEBUG")
```

Instatiate estimator

```
masses = [100, 200, 500, 1000, 5000, 10000, 50000]*u.GeV
channels = ["b", "tau", "Z"]
estimator = SigmaVEstimator(dataset, masses, channels, background_model=bkg, jfactor=JFAC)
```

Run estimator and fetch results

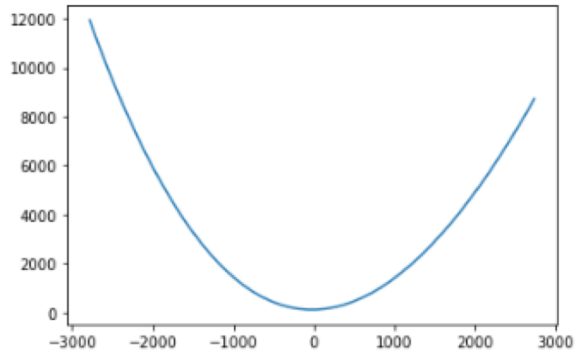
```
%%time
likelihood_profile_opts=dict(bounds=50, nvalues=50)
result = estimator.run(runs=10, nuisance=True)
#if nuisance = True the process takes the nuisance parameters into account.
```

logs / inspection
nuisance loops
~ 1min per run

Plot likelihood profile for a specific fit

```
idx = np.argwhere(masses.value==100)
profile = result["runs"]["b"][0]["likeprofile"][idx][0][0]
```

```
plt.plot(profile["values"], profile["likelihood"]);
```

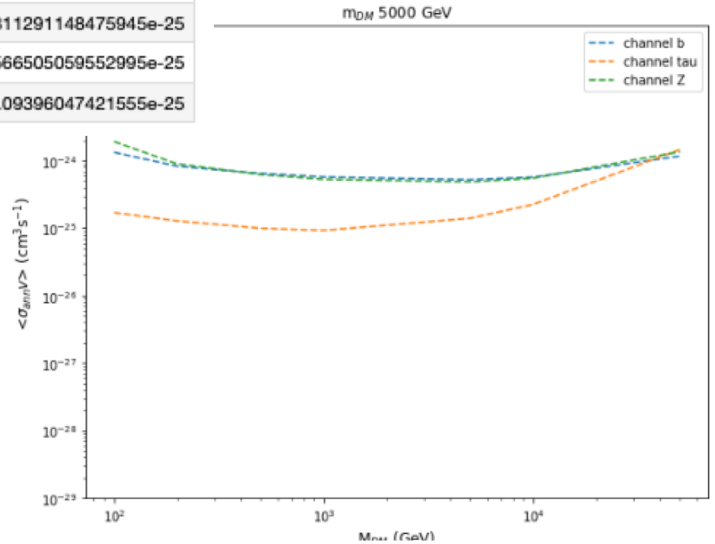


Obtained results

```
result["mean"]["b"]
```

Table length=7

mass	sigma_v	std
GeV	cm3 / s	cm3 / s
float64	float64	float64
100.0	1.333896839289899e-24	3.935980530464652e-25
200.0	8.271522632489434e-25	2.1810917821389594e-25
500.0	6.470707992306648e-25	2.121894489820506e-25
1000.0	5.741742758759052 5e-25	2.1744794124605246e-25
5000.0	5.226677235832766e-25	2.0811291148475945e-25
10000.0	5.704487304999477e-25	2.7566505059552995e-25
50000.0	1.1701358121695634e-24	6.09396047421555e-25



Display results for channel b and run 0

```
cols = ["mass", "sigma_v", "sv_best", "sv_ul", "likeprofile"]
result["runs"]["b"][0][cols]
```

The SigmaVEstimator *(fitting / likelihood profiling in nested loops)*

```
nuissance = dict(
    j=JFAC,
    jobs=JFAC,
    sigmaj=0.1*JFAC,
    sigmatau=0.01,
    #width=5,
    #steps=15,
)
dataset.nuissance = nuissan
```

SigmaVEstimator
astro/darkmatter/utils.py

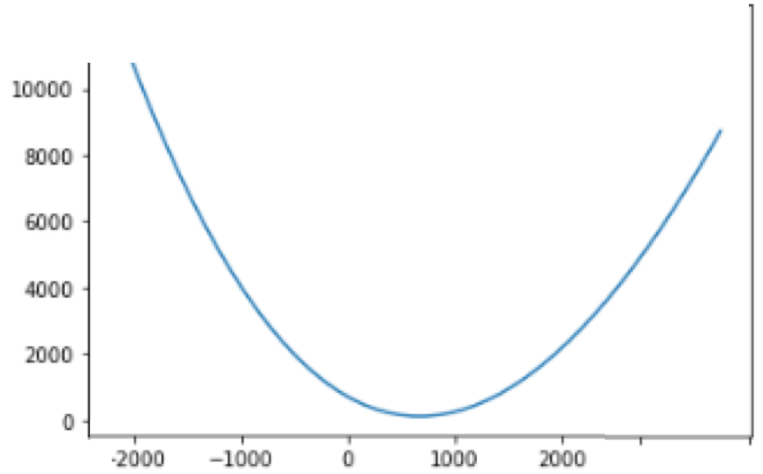
The SigmaVEstimator class estimates σv for a list of annihilation channels and particle masses.

In order to have statistics the overall process below is repeated for a given number of `runs`.

- For each run a **random poisson realization** of a `DMDatasetOnOff` is created as a simulated observation.
- For each run we consider a list of annihilation channels.
  - For each channel we consider a list of masses.
    - For each mass in each channel:
      - A specific `DarkMatterAnnihilationSpectralModel` is instantiated with parameters `JFactor`, `channel`, `mass`.
      - This specific annhilation flux model is attached to the `DMDatasetOnOff` dataset instance.
      - A fit of this specific annhilation model to the simulated observation of the run is performed for each of the `steps` values that the nuisance parameter `j` takes in a range of ± `sigmaj` * `width`.
      - A fit profle is chosen: the one giving the minimum value for the likelihood.
      - A check (L_0 - L_{min} ≤ 25) is done.
      - The value of the scale parameter in the flux model `sv` giving ΔL=2.71 in the likelihood profile is chosen.
      - If the value of `sv` is not in the range of the physical region (>=0) or does not reach ΔL=2.71 we skip the run and take the next one.
      - σv is the chosen value of `sv` is multiplied by the thermal relic cross section.
- Profile likelihoods and values calculated for `sv` and σv are provided for each run, channel and mass.
- For each channel a table of mean values across all runs of σv vs. mass is provided.
- Messages are displayed according to the logging level sonfiguration defined.

```
masses = [100, 200, 500, 1000, 5000, 10000, 50000]*u.GeV
channels = ["b", "tau", "Z"]
estimator = SigmaVEstimator(dataset, masses, channels, background_model=bkg, jfactor=JFAC)
```

```
%%time
likelihood_profile_opts=dict(bounds=50, nvalues=50)
result = estimator.run(runs=10, nuisance=True)
#if nuisance = True the process takes the nuisance parameters into account.
```



SUMMARY

- The need in the DM community is still there: some CTA-DM projects are waiting to the DM-module!
- A lot of progress since Coding Sprint in February in Paris:
  - New class for the DM annihilation spectrum: `DarkMatterAnnihilationSpectralModel`
  - New DM Dataset for ON/OFF observations: `DMDatasetOnOff`
  - Nuisance parameters introduced in the likelihood
  - New function for fitting the DM Datasets and producing upper limits for DM parameters: `SigmaVEstimator`
  - Preliminary notebook reproducing the standard DM analysis pipeline
- Work to do:
  - Test the current pipeline
  - Introduce 2D templates in the analysis: spectral model class, datasets, likelihood…
  - Suggestions and comments are welcome!