tcap2021DM

August 27, 2021

1 How to perform a dark matter analysis with ctools

In this tutorial you will learn how to use ctools to constrain the parameters of a dark-matter model (mass and annihilation cross section). We will consider as an example the dwarf spheroidal galaxy Aquarius II. Credits for the notebook:Alejandra Aguirre-Santaella, Tarek Hassan, and Sergio Hernández Cadena

Lets start by importing the required Python modules

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import numpy as np

import gammalib
import ctools
import cscripts

import time

import sys
import os
from show_residuals import plot_residuals
```

This notebook was tested using python 3 (more specifically 3.7.6) ctools/gammalib version 1.7.4. If you have errors, please check first if they dissapear when using this version!

[2]:
```
print(ctools.__version__)
print(gammalib.__version__)
```

```
1.7.4
1.7.4
```

1.1 Modelling the source

For simplicity we assume the galaxy is a point source for CTA.

We take the value of the astrophysical J factor from Pace & Strigari, 2018 (units: $\mathrm{GeV}^2\ \mathrm{cm}^{-5}$). Coordinates for the Aquarius II galaxy are taken from Torealba et al., 2016.

[3]:
```
srcname = 'AquariusII'
jfactor = 1.8621e+18

ra      = 338.4813
dec     = -9.3274
```

The dark-matter annihilation spectrum is modelled using the tables from Cirelli et al., 2011. The tables were formatted in such a way that they can be ingested by ctools. You can check this example to see how to create the file from the literature material. We provide two files depending if the spectrum include or not electroweak corrections. The example FITS file contains the tables for different annihilation channels, gamma-ray energies between 30 GeV and 100 TeV, and dark-matter masses in the range $[0.1, 100]$ TeV. Using the spectrum in this way allow us to only change the flux normalization for other galaxies. See below.

[4]:
```
dmspecEW = 'DMModelAnnihilationAquariusIIEW1.fits'
```

For our example we will set the DM mass (GeV) and annihilation channel to τ's as follows.

[5]:
```
mass    = 5.0e+3
channel = 8
```

For the moment we will set the value of the annihilation cross section ($\langle\sigma_\chi v\rangle$ in cm^3 s^{-1}) based on the upper limits obtained with data from the H.E.S.S. experiment: Search for dark matter signals towards recently detected DES dwarf galaxy satellites.

[6]:
```
sigmav = 2.7e-22
```

We can now combine J factor, mass, and annhilihation cross section to derive the overall flux normalization for this model.

[7]:
```
fluxnorm  = jfactor * sigmav / (8*gammalib.pi*mass**2)
fluxnorm *= 1.0e-3
```

We use the GModelSpectralTable class in GammaLib to manage the spectral model. You can check gammalib Documentation about Spectral components for more information anout GModelSpectralTable. You can also follow the tutorial *How to do advanced model manipulation and fitting in Python* to learn how handle models with GammaLib.

The steps necessary to set up the spectral model are - load the fits file - set the value of the mass we will simulate - set the annihilation channel - set the normalization of the spectrum

The names of the dark-matter parameters are set within the FITS file in the PARAMETERS extension.

[8]:
```
dmspec = gammalib.GModelSpectralTable()
dmspec.load(dmspecEW)
dmspec['Mass'].value(mass)
dmspec['Channel'].value(channel)
dmspec['Channel'].scale(1)
dmspec['Normalization'].value(fluxnorm)
```

```
dmspec['Normalization'].range(0.0, 1.0e+20)
```

The normalization is a free parameter by default. In this example, we will aditionally set free the mass

[9]:
```
dmspec['Mass'].free()
```

Now, we can take a look at the spectral information of our source. Note that we will use the spectrum with electroweak corrections.

[10]:
```
print(dmspec)
```

```
=== GModelSpectralTable ===
 Table file …: DMModelAnnihilationAquariusIIEW1.fits
 Number of parameters …: 3
  Normalization …: 8.00178532734829e-16 +/- 0 [0,1e+20]
(free,scale=1,gradient)
  Mass …: 5000 +/- 0 [100,100000]  (free,scale=1000)
  Channel …: 8 [0,27]  (fixed,scale=1)
 Mass values …: 101 [100, 100000] (logarithmic)
 Channel values …: 28 [0, 27] (linear)
 Energies …: 450 [30 GeV, 99.999997952 TeV]
 Spectra array dimension …: 3
 Number of spectra …: 2828
 Number of spectral bins …: 450
```

We then create the spatial model and the source model using the appropriate GammaLib classes.

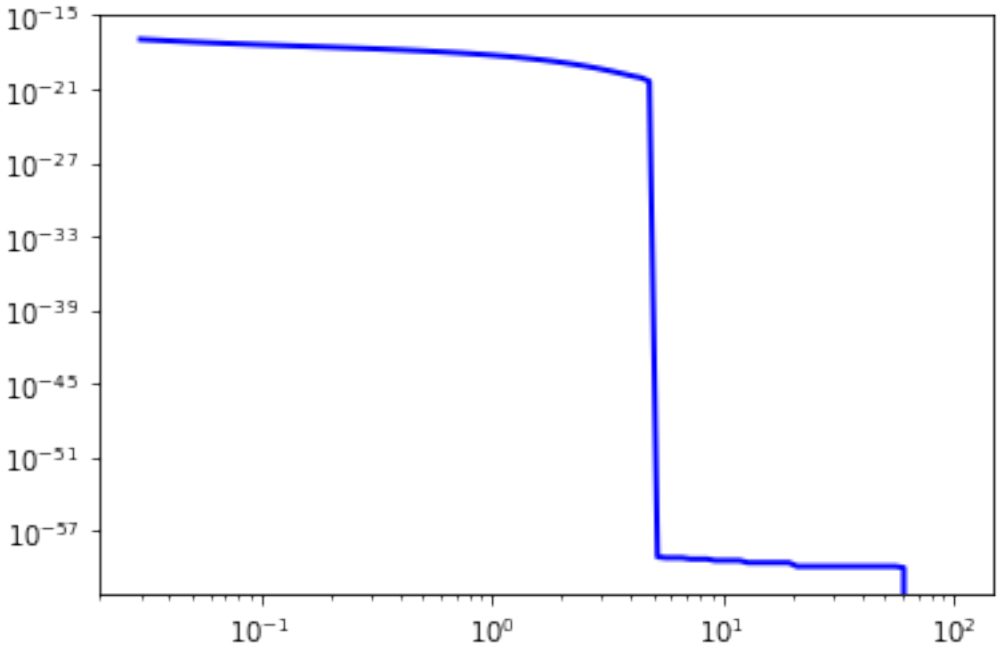
[11]:
```
spatial = gammalib.GModelSpatialPointSource(ra, dec)

dmmodel = gammalib.GModelSky(spatial, dmspec)
dmmodel.name(srcname)
dmmodel.tscalc(True) # calculate TS for this source
```

1.1.1 Questions

1. Mention the features of photon spectra from annihilation of dark matter particles.
2. Plot the spectrum of the source we will be simulating. Hint: use the dmspec.eval() function, which needs gammalib.GEnergy as input.

[12]:
```
energies = np.logspace(np.log10(0.03), np.log10(100), 100)
fluxes = [dmspec.eval(gammalib.GEnergy(e,'TeV')) for e in energies]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
line, = ax.plot(energies, fluxes, color='blue', lw=2)
ax.set_xscale('log')
ax.set_yscale('log')
```



1.2 Simulating CTA Observations

1.2.1 Sky model

We also need a model for the instrumental background, that we will read from the IRFs with a power-law spectral correction.

[13]:
```
# spectral correction
spectral = gammalib.GModelSpectralPlaw(1, 0, gammalib.GEnergy(1, 'TeV'))

# background model
bkgmodel = gammalib.GCTAModelIrfBackground(spectral)
bkgmodel.name('Background')
bkgmodel.instruments('CTA')
```

We combine the two models in a container and save it to disk for later use.

[14]:
```
inmodel = "inmodel.xml"
models  = gammalib.GModels()
models.append(dmmodel)
models.append(bkgmodel)
models.save(inmodel)
```

1.2.2 Exercise: Observation simulation

We simulate an observation of 30 min pointed at Aquarius II using the instrument response functions for CTA South.

[15]:
```
emin  = 0.03 # TeV
emax  = 100. # TeV
caldb = 'prod3b-v2'
irf   = 'South_z40_50h'

obssim = ctools.ctobssim()
obssim['inmodel'] = inmodel
obssim['caldb'] = caldb
obssim['irf'] = irf
obssim['seed'] = int(time.time())
obssim['ra'] = ra
obssim['dec'] = dec
obssim['rad'] = 3.0
obssim['tmin'] = '2021-01-01T00:00:00'
obssim['tmax'] = '2021-01-01T00:30:00'
obssim['emin'] = emin
obssim['emax'] = emax
obssim.run()
```

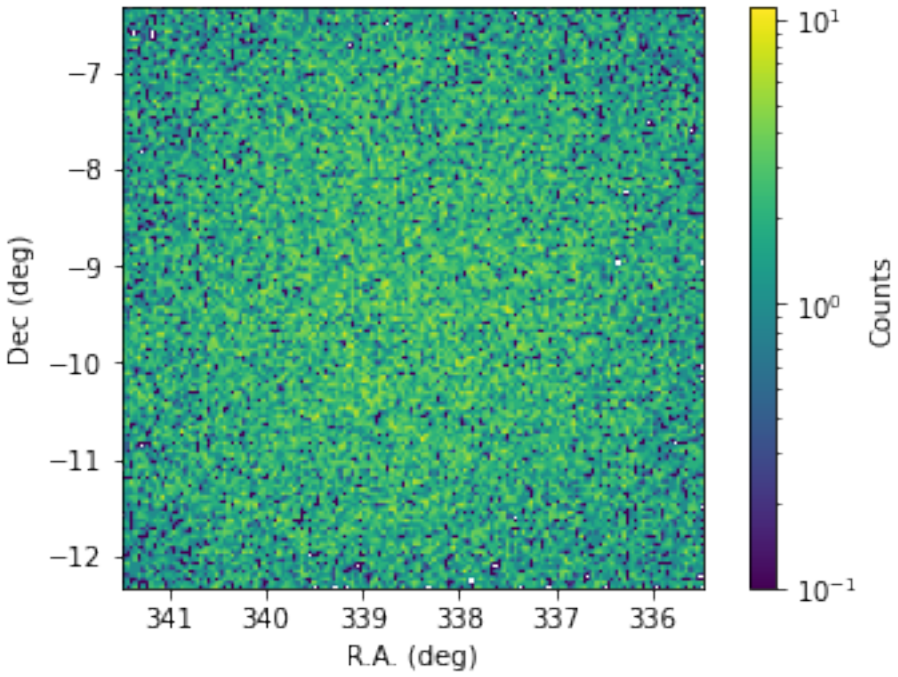
1.2.3 Exercise: creating a skymap

Peek at the simulated events by creating a skymap.

[16]:
```
skymap = ctools.ctskymap(obssim.obs())
skymap['coordsys'] = 'CEL'
skymap['proj'] = 'CAR'
skymap['xref'] = ra
skymap['yref'] = dec
skymap['binsz'] = 0.02
skymap['nxpix'] = 150
skymap['nypix'] = 150
skymap['emin'] = emin
skymap['emax'] = emax
skymap['bkgsubtract'] = 'NONE'
skymap.run()
```

[17]:
```
ax = plt.subplot()
plt.imshow(skymap.skymap().array(),origin='lower',
extent=[ra+0.02*150,ra-0.02*150,dec-0.02*150,dec+0.02*150],
# boundaries of the coord grid
norm=LogNorm(vmin=0.1))
ax.set_xlabel('R.A. (deg)')
ax.set_ylabel('Dec (deg)')
```

```
cbar = plt.colorbar()
cbar.set_label('Counts')
```



1.3 Exercise: Likelihood Analysis

Run the likelihood analysis

[18]:
```
like = ctools.ctlike(obssim.obs())
like.run()
```

Let's check the TS of the source

[19]:
```
like.obs().models()[srcname].ts()
```

[19]: 67.19385399844032

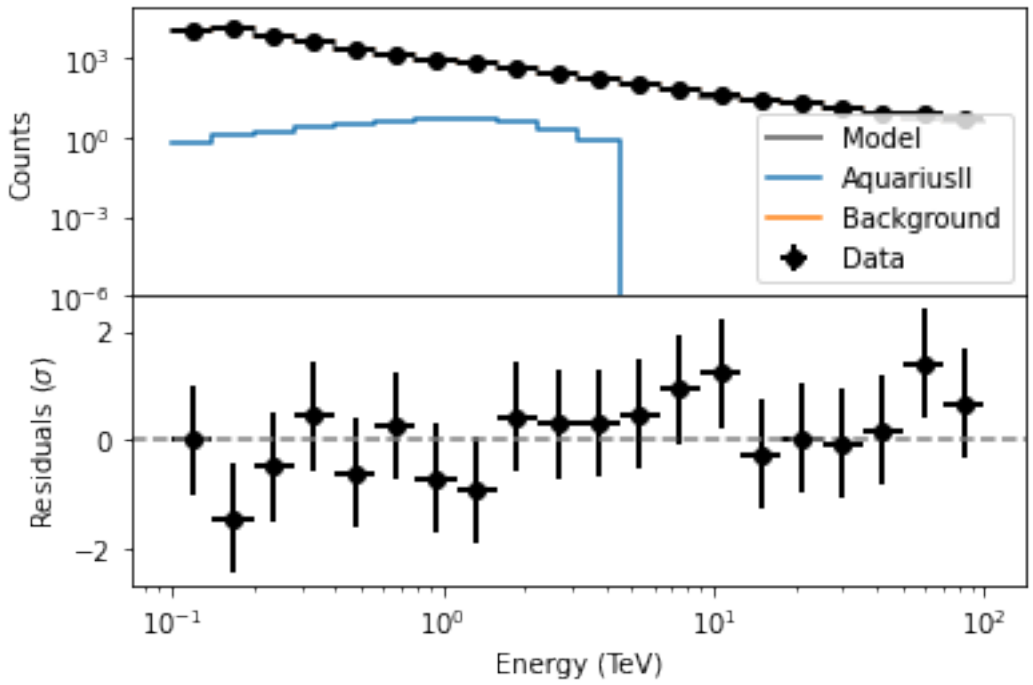
1.3.1 Exercise

Generate residual map and spectrum to verify that the fitted model reproduces the data well.

[20]:
```
resspec = cscripts.csresspec(like.obs())
resspec['algorithm'] = 'SIGNIFICANCE'
resspec['mask'] = False
resspec['components'] = True
resspec['outfile'] = 'resid_spectrum.fits'
```

```
resspec.execute()
```

[21]:
```
plot_residuals('resid_spectrum.fits','',0)
```



1.3.2 Results of the analysis

So the model we simulated yields a significant detection of the dark-matter signal!

We can check the fitted model to get the value of the mass and compare with the input.

[22]:
```
print(like.obs().models()[srcname])
```

```
=== GModelSky ===
 Name …: AquariusII
 Instruments …: all
 Test Statistic …: 67.1938539984403
 Observation identifiers …: all
 Model type …: PointSource
 Model components …: "PointSource" * "TableModel" * "Constant"
 Number of parameters …: 6
 Number of spatial par's …: 2
  RA …: 338.4813 deg (fixed,scale=1)
  DEC …: -9.3274 deg (fixed,scale=1)
 Number of spectral par's ..: 3
  Normalization …: 1.14385808093738e-15 +/- 4.01085128162733e-16
```

```
[0,1e+20]  (free,scale=1,gradient)
  Mass …: 3909.36922722194 +/- 861.909322734889 [100,100000]
(free,scale=1000)
  Channel …: 8 [0,27]  (fixed,scale=1)
 Number of temporal par's ..: 1
  Normalization …: 1 (relative value) (fixed,scale=1,gradient)
 Number of scale par's …: 0
```

For the annihilation cross-section $\langle\sigma_\chi v\rangle$ we need to take into account the J factor. The easiest way is to derive the annihilation cross-section based on the ratio ξ between the flux measured and the flux expected for the reference cross-section value. We will consider the integrated flux from the minimum energy of the observations up to the dark-matter mass. The measured annihilation cross-section therefore is:

$\langle\sigma_\chi v\rangle_{\rm meas} = \xi \times \langle\sigma_\chi v\rangle_{\rm ref}$

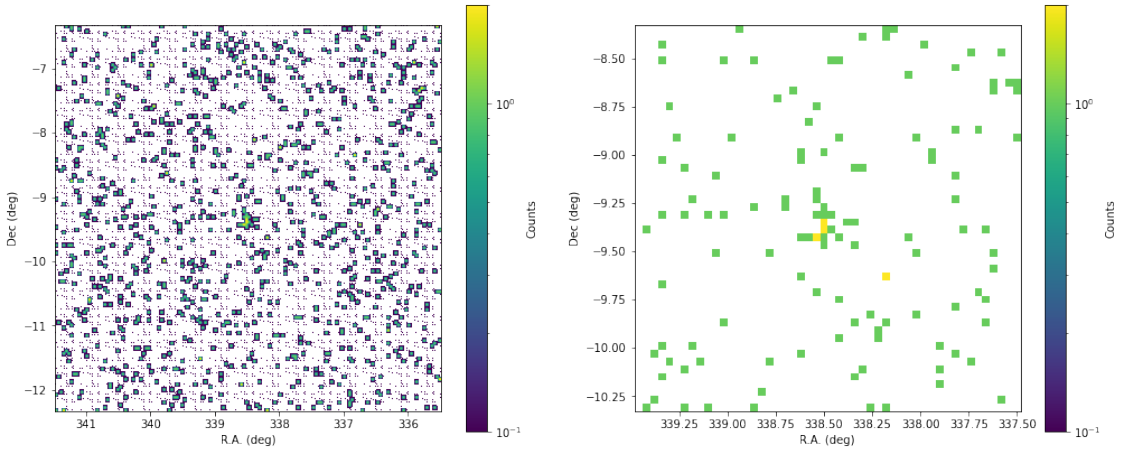
1.3.3 Exercise:

1. Did you see anything in the skymap? We actually didn't… Why not?
2. Can you come up with any way to see the point-like source we are simulating? ctools is detecting it, so it must be somewhere…

[23]:
```
sm = ctools.ctskymap(obssim.obs())
sm['coordsys'] = 'CEL'
sm['proj'] = 'CAR'
sm['xref'] = ra
sm['yref'] = dec
sm['binsz'] = 0.02
sm['nxpix'] = 150
sm['nypix'] = 150
sm['emin'] = 0.7
sm['emax'] = emax
sm['bkgsubtract'] = 'NONE'
sm.run()
fig = plt.figure(figsize=[17, 7])
ax = fig.add_subplot(1, 2, 1)
im = ax.imshow(sm.skymap().array(),origin='lower',
extent=[ra+0.02*150,ra-0.02*150,dec-0.02*150,dec+0.02*150],
# boundaries of the coord grid
norm=LogNorm(vmin=0.1))
cbar = fig.colorbar(im) # Add a colorbar to the figure based on the image
cbar.set_label('Counts')
ax.set_xlabel('R.A. (deg)')
ax.set_ylabel('Dec (deg)')
ax2 = fig.add_subplot(1, 2, 2)
im2 = ax2.imshow(sm.skymap().array(),origin='lower',
extent=[ra+0.02*150,ra-0.02*150,dec-0.02*150,dec+0.02*150],
# boundaries of the coord grid
norm=LogNorm(vmin=0.1))
```

```
cbar2 = fig.colorbar(im2) # Add a colorbar to the figure based on the image
cbar2.set_label('Counts')
ax2.set_xlabel('R.A. (deg)')
ax2.set_ylabel('Dec (deg)')
ax2.set_xlim([ra+1, ra-1])
ax2.set_ylim([dec-1, dec+1])
```

[23]: (-10.3274, -8.3274)



1.3.4 Exercise:

1. Derive the uncertainty on $\langle \sigma_\chi v \rangle$ from the relative uncertainty of the normalization parameter.
2. What is the "observed" value for the mass of the particle?

[24]:
```
# Define energy range where the integrated flux is computed
gemin = gammalib.GEnergy(emin, 'TeV')
gemax = gammalib.GEnergy(mass, 'GeV')
# Get the reference flux
ref_flux = dmmodel.spectral().flux(gemin, gemax)
# Get the measured flux
meas_flux = like.obs().models()[srcname].spectral().flux(gemin, gemax)
# Measured cross-section
csi = meas_flux / ref_flux
meas_sigmav = csi * sigmav
print(meas_sigmav)
```

```
3.6957321729807284e-22
```

[25]:
```
obs_spec = like.obs().models()[srcname].spectral()
print(obs_spec['Normalization'].error() / obs_spec['Normalization'].value())
```

```
0.35064238723919816
```

[26]:
```
rel_error = obs_spec['Normalization'].error() / obs_spec['Normalization'].
 ↪value()
sigma_v_error = meas_sigmav * rel_error
```

[27]:
```
print('Annihilation cross-section from observations:\n')
print('\t{:.3e} +/- {:.3e} cm**3/s'.format(meas_sigmav, sigma_v_error))
```

```
Annihilation cross-section from observations:

        3.696e-22 +/- 1.296e-22 cm**3/s
```

1.4 Upper limits

Unfortunately the dark-matter signal may not be so strong that CTA can detect it. In this case you will want to set upper limits on the annihilation cross-section. To explore this scenario we will perform a new simulation with different parameters.

[28]:
```
sigmav = 5.0e-24

fluxnorm  = jfactor * sigmav / (8*gammalib.pi*mass**2)
fluxnorm *= 1.0e-3
```

[29]:
```
models[srcname].spectral()['Normalization'].value(fluxnorm)
models[srcname].spectral()['Mass'].fix()
models.save(inmodel)
```

1.4.1 Exercise: Simulating new observations

We regenerate the observations for the new cross-section value. (Again, 30 min)

[30]:
```
obssim = ctools.ctobssim()
obssim['inmodel'] = inmodel
obssim['caldb'] = caldb
obssim['irf'] = irf
obssim['seed'] = int(time.time())
obssim['ra'] = ra
obssim['dec'] = dec
obssim['rad'] = 3.0
obssim['tmin'] = '2021-01-01T00:00:00'
obssim['tmax'] = '2021-01-01T00:30:00'
obssim['emin'] = emin
obssim['emax'] = emax
obssim.run()
```

We perform again a likelihood fit and check the TS value.

[31]:
```
like = ctools.ctlike(obssim.obs())
like.run()
```

[32]:
```
like.obs().models()[srcname].ts()
```

[32]:
```
1.0617628769250587
```

The small value of TS means that there is no evidence of any dark-matter signal. Therefore we proceed to compute upper limits on the annihilation cross-section.

In order to do so let's fix the mass and calculate an upper limit on the normalization. For example let's consider a mass of 5 TeV.

[33]:
```
print(like.obs().models()[srcname])
```

```
=== GModelSky ===
 Name …: AquariusII
 Instruments …: all
 Test Statistic …: 1.06176287692506
 Observation identifiers …: all
 Model type …: PointSource
 Model components …: "PointSource" * "TableModel" * "Constant"
 Number of parameters …: 6
 Number of spatial par's …: 2
  RA …: 338.4813 deg (fixed,scale=1)
  DEC …: -9.3274 deg (fixed,scale=1)
 Number of spectral par's ..: 3
  Normalization …: 6.52191516617015e-17 +/- 7.88109617179099e-17
[0,1e+20]  (free,scale=1,gradient)
  Mass …: 5000 [100,100000]  (fixed,scale=1000)
  Channel …: 8 [0,27]  (fixed,scale=1)
 Number of temporal par's ..: 1
  Normalization …: 1 (relative value) (fixed,scale=1,gradient)
 Number of scale par's …: 0
```

1.4.2 Exercise

We can use ctulimit to compute an upper limit on the source normalization.

Tip: set wisely the energy range for the calculation of the integral flux upper limits.

[34]:
```
ulimit = ctools.ctulimit(obssim.obs())
ulimit['srcname'] = srcname
### energy range for flux calculation #############
ulimit['emin'] = emin
ulimit['emax'] = mass * 1.e-3
ulimit.run()
```

1.4.3 Exercise

As we did before we can compare the flux upper limit and reference flux to obtain an upper limit on the annhilation cross-section. Derive the UL to the annihilation cross-secion

[35]:
```
ulimit_flux = ulimit.flux_ulimit()
ref_flux    = models[srcname].spectral().flux(gemin, gammalib.GEnergy(mass,␣
 ↪'GeV'))
csi = ulimit_flux / ref_flux
ul_sigmav = csi * sigmav
```

[36]:
```
print(ul_sigmav)
```

```
9.478758971099573e-23
```

1.4.4 Exercise

1. Compute the UL on the annihilation cross-section using as spectral model the spectrum without electroweak corrections. You need to load the file "DMModelAnnihilationAquariusIIEW0.fits".
   - Hint: You don't need to rerun ctobssim
   - Please be sure to pass the correct number to indicate the same annihilation channel (τ). Check Cirelli et al., 2011.

To check if there is a real difference between the two ULs, we first need to compute the average UL after perform several realizations (for example ~50).

1.4.5 Likelihood Profile

In this case we explore a simple model where the parameter of interest impact the value of the overall normalization of the gamma-ray flux. In other models, the parameter of interest change directly the shape of the spectrum, or inclusive if we want to estimate the impact of nuisance paramaters, is good practice to take a look at the likelihood profile. Let's check what the likelihood profile looks like for the normalization (related to the parameter of interest).

First, we define the range we are interested to check:

[37]:
```
norma_r = np.logspace(-28, 6)
```

Now, we fix the normalization in our model.

[38]:
```
like.obs().models()[srcname].spectral()['Normalization'].fix()
```

We compute the likelihood for every normalization value and saving to a list (take several minutes...)

[39]:
```
ll_values = []

for norma in norma_r :
    like.obs().models()[srcname].spectral()['Normalization'].value(norma)
    like.run()
    ll = like.opt().value()
    ll_values.append(ll)
```

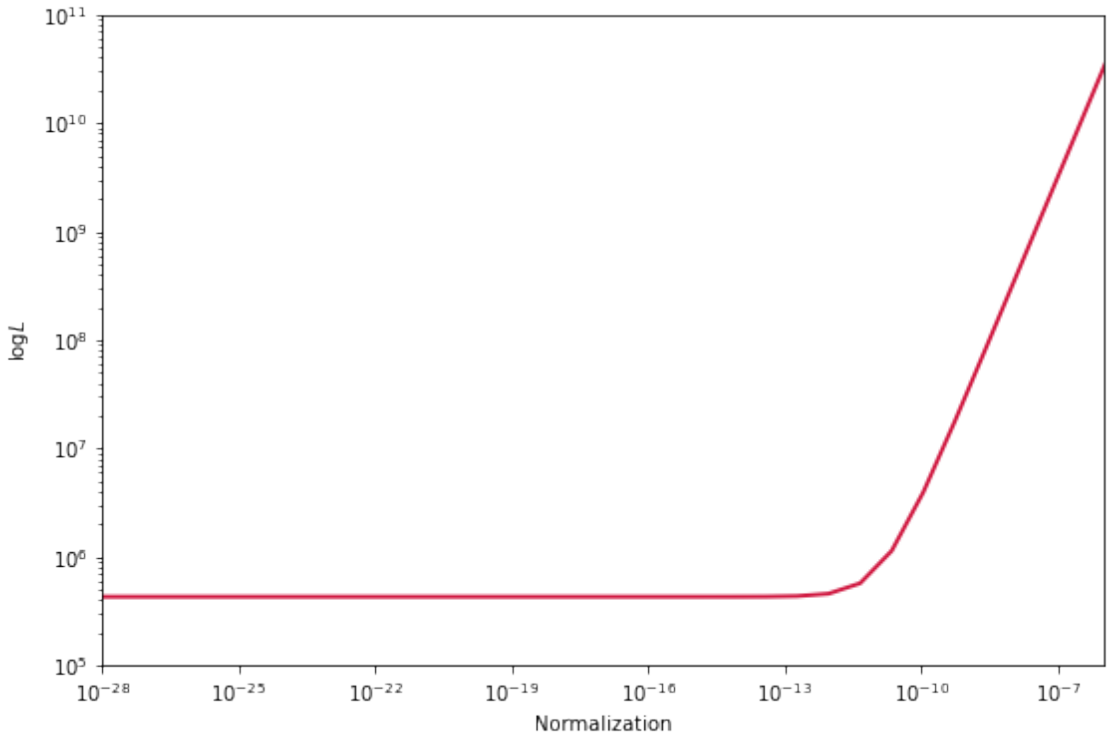
Now, let's plot the results

[40]:
```
fig, ax = plt.subplots(figsize=(9, 6))

ax.plot(norma_r, ll_values, color=( 0.82 , 0.10 , 0.26 ), lw=2)

ax.set_ylim(1.e+5, 1.e+11)
ax.set_xlim(1.e-28, 1.e-6)
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_xlabel('Normalization')
ax.set_ylabel('$\\log{L}$')
```

[40]:
```
Text(0, 0.5, '$\\log{L}$')
```



[ ]: