

## Installation Instructions :

- Install root prerequisites : (<https://root.cern/install/dependencies/>)
- Install root : ([https://root.cern/install/build\\_from\\_source/](https://root.cern/install/build_from_source/))
- installation example :
- `git clone --branch latest-stable --depth=1 https://github.com/root-project/root.git`
- `mkdir root_build root_install && cd root_build`
- `cmake -D builtin_cfitsio=ON -DCMAKE_INSTALL_PREFIX=../root_install ../root`
- `cmake --build . --install -j4` (you can change 4 by the number of core your CPU have)
- `source ../root_install/bin/thisroot.sh`
  
- Download LIVelihood (<https://lapp-owncloud.in2p3.fr/s/wp6pzKzX7GgG2bj>)
- `sh clean-install.sh`
- `source setup.sh`
- to test the install :
  - `root`
  - `.L ../LIVlib-build/lib/libLIV.so`
  - `.L scripts/Example.C`
  - `test_runner()`
- install conda if not already installed : <https://www.anaconda.com/products/distribution>
- install gammapy :
- `curl -O https://gammapy.org/download/install/gammapy-1.0-environment.yml`
- `conda env create -f gammapy-1.0-environment.yml`
- `conda activate gammapy-1.0`
- `gammapy download datasets`
- `export GAMMAPY_DATA=$PWD/gammapy-datasets/1.0`
- more information on gammapy here : <https://docs.gammapy.org/1.0/getting-started/index.html>

if difficulties, send a mail to [sami.caroff@lapp.in2p3.fr](mailto:sami.caroff@lapp.in2p3.fr)

## Lesson 1, source analysis :

The goal of this lesson is to download open data of HESS, and perform an analysis in order to extract the temporal variability of the source, as well as all the needed information to simulate and perform a LIV analysis of this source.

These data are from a flaring AGN, PKS 2155-304 during the night from July 29 to 30 2006. for more information see the [following article](#).

You should use the notebook contained in the LIVelihood folder :

- `conda activate gammapy-1.0`
- `go in LIVlib/python_scripts/`
- `jupyter notebook`
- `open light_curve_flare.ipynb`

Everything is explained in the notebook. After the extraction of the light curve, let's try to fit the data with a simple gaussian using minuit

```
In [20]: from iminuit import cost, Minuit

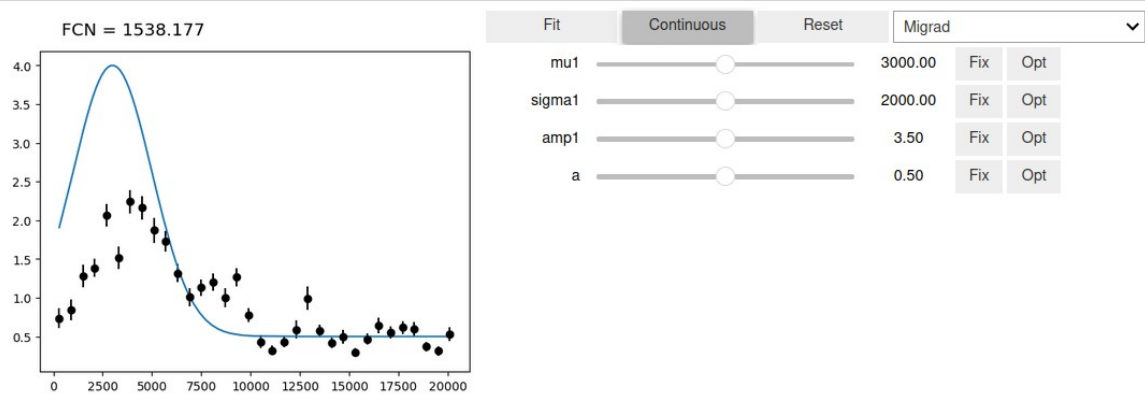
def gauss(x,mu,sigma):
    return np.exp(-np.square(x-mu)/(2*sigma**2))

def model(x,mu1,sigma1,amp1,a):
    return amp1*gauss(x,mu1,sigma1) + a

cost_function = cost.LeastSquares(time_table_seconds,1e10*flux_table[:,0],1e10*flux_error_table[:,0], model)

m = Minuit(cost_function, mu1=3000, sigma1=2000, amp1=3.5, a=0.5)
m.interactive()
```

Out[20]:



You can play with the buttons in order to fit by eye, you can as well uncomment the next cell to play with more complex models.

The amount of variable signal, baseline signal and background events is computed in the next cells from the fit.

```
In [27]: # the proportion of variable signal can be computed
proportion_gamma = intg/(intg_baseline + intg)

print("n_signal = ",proportion_gamma*Nexcess)
print("n_baseline = ",(1-proportion_gamma)*Nexcess)
print("n_background = ",N_hadrons)

N_baseline=(1-proportion_gamma)*Nexcess
N_signal=proportion_gamma*Nexcess

##### Total number #####
N_total=N_hadrons+N_baseline+N_signal

##### Proportions #####
props=N_signal/N_total
proph=N_hadrons/(N_hadrons+N_baseline)

print("proportion of signal = ",props)
print("proportion of background = ",proph)

n_signal = 613.6896745769895
n_baseline = 2085.1436587563435
n_background = 57.16666714350383
proportion of signal = 0.2226740473406424
proportion of background = 0.02668458740658489
```

The parameters that should be used to perform simulations in LIVelihood are computed in the last cell :

```
##### INPUT SIMULATIONS #####
print("Nevets total : ", N_total)
print("Tmin : ", time_table_seconds[0])
print("Tmax : ", time_table_seconds[-1])
print("Emin : 0.7 TeV")
print("Emax : 20 TeV")
print()
print('for double gaussian function :')
print('Normalisation : 1')
print('mu1 : {0} ; limits : [{1},{2}]'.format(np.format_float_scientific(m.params[0].value,precision=3),time_table_seconds[0],time_table_seconds[-1]))
print('sigma1 : {0} ; limits : [{1},{2}]'.format(np.format_float_scientific(m.params[1].value,precision=3),time_table_seconds[0],time_table_seconds[-1]))
print()
print('for Power law :')
print('Index : 3.4 ; limits : [1.5,7]')
print('Normalisation : 1')
print()
print('for Background')
print("Proportion signal : ",props)
print("Proportion background hadrons : ",proph)
print("Index hadrons : 2.7")
print("Index baseline : 3.4")
```

After this last step, you extracted all the ingredients needed for the LIVelihood simulations for the next steps.

## Lesson 2, source simulation and analysis with LIVelihood (without IRFs) :

LIVelihood can use the parameter you extracted in order to perform simulations of the source.

In this lesson, we will simulate the source based on the analysis made, and we will perform a LIV analysis supposing that the reconstruction of the instrument is perfect (no energy migration, flat effective area versus energy)

You can open the file LIVlib/src/LIVTutorialHandle\_1.cpp in a text editor. Everytime this file is modified, you need to recompile the code (sh make.sh). This file is already filled with the parameters obtained in the previous lessons, but you can change them if you want to play around with different parameters. This is the list of important parameters :

Energy range and time range

```
44 // Define energy range and time range
45 // 0s < t < 10000s
46 Tmin = 0;
47 Tmax = 10000;
48 // 700 GeV < E < 20 TeV
49 Emin = 0.7;
50 Emax = 20;
```

Light curve parameters (the parameters of the Gaussian you fitted previously)

```
// Initiate functions for pdf
LC = new TF1("LC",UtilityFunc::SingleGaussian_prop_LIV,Tmin-(Tmax-Tmin)*0.1,Tmax+(Tmax-Tmin)*0.1,4);
LC->SetParameter(1,4380); // mean of the Gaussian
LC->SetParLimits(1,0,10000); //Limits will define the fit limits if nuisance is used
LC->SetParameter(2,2596); // width of the Gaussian
LC->SetParLimits(2,100,10000);
LC->SetParameter(0,1); // normalisation
// delay
LC->SetParameter(3,0); // starting point of the fit for the delay
LC->SetParLimits(3,-20000,20000);
tau_param=LC->GetNumberFreeParameters()-1;
```

The Power Law distribution of the events :

```
70 // define the energy distribution as a power law
71
72 PL = new TF1("PL",UtilityFunc::PowerLaw,Emin*EminFactor,Emax*EmaxFactor,2);
73 // Set parameters for spectrum
74 // Set spectral index to 3.4
75 PL->SetParameter(0,3.4);
76 PL->SetParLimits(0,1.5,7.);
77 PL->SetParameter(1,1); // normalisation
```

Proportion of signal and background :

```
80 // initiate the energy distribution for background
81 // Here simulate 65% of signal, with 6% of the background contaminated with hadrons
82 // Power law of hadron is supposed to be 2.7, power law of baseline is supposed to be 3.4 (same as signal)
83 SetBackground(0.65, 0.06, 2.7, 3.4,0.);
```

You can as well specify the redshift of the source here :

```

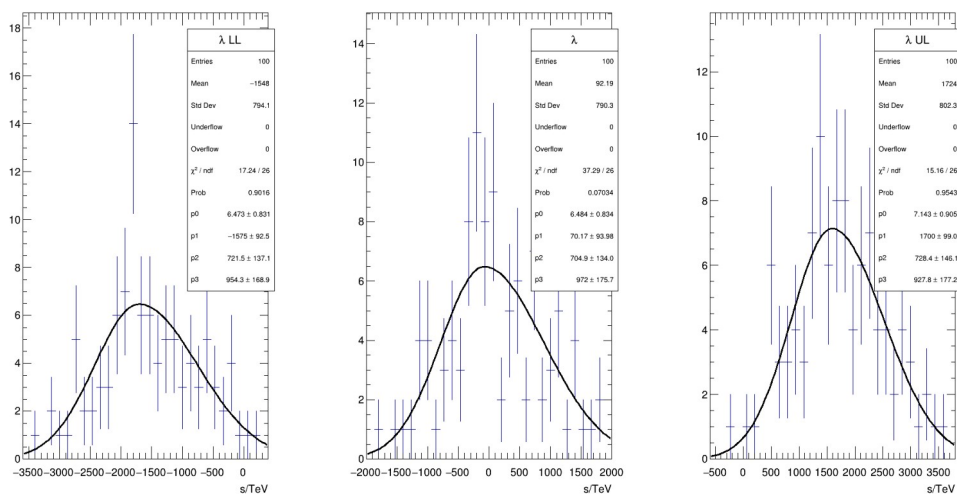
85 // Set Source distance
86 // This must be done for the kterm to be
87 // calculated properly.
88 // Set the redshift to 0.116
89 // and the source is not galactic.
90 double redshift = 0.116;
91 SetSourceDistance(redshift,false);
92 CreateTableDominguez(redshift);
93 //~ TF1_EBL->Draw();
94 SetRedshiftNuisance(0.001);

```

Verify that the different parameters are fine compared to what you computed on the data. After this verification, you can :

- sh make.sh
- root
- .L ../LIVlib-build/lib/libLIV.so
- .L scripts/Tutorial\_1.C
- test\_runner()

We will see after how the parameter of this test\_runner function can be tuned, but if nothing is specified, simple simulations are performed and analysis is performed in these simulations. This simulations simulate a delay of 0 s/TeV so you should reconstruct zero in mean. The results distribution is plotted, as well as the 95% confidence interval.



Results are as

well displayed in the terminal :

```

----- Simulations Results -----
----- Best Fit lambda -----
----- Mean Lambda : 70.1686+-93.9755
----- median      : 2.27374e-13
----- Sigma Lambda: 704.921+-133.99
----- Chi2       : 37.2924/26
----- Lower Limits lambda -----
----- Mean LL : -1574.67+-92.5171
----- median  : -1666.67
----- Sigma LL: 721.535+-137.086
----- Chi2   : 17.2427/26
----- Upper Limits lambda -----
----- Mean UL : 1700.38+-99.0276
----- median  : 1691.67
----- Sigma UL: 728.407+-146.135
----- Chi2   : 15.1629/26

```

```

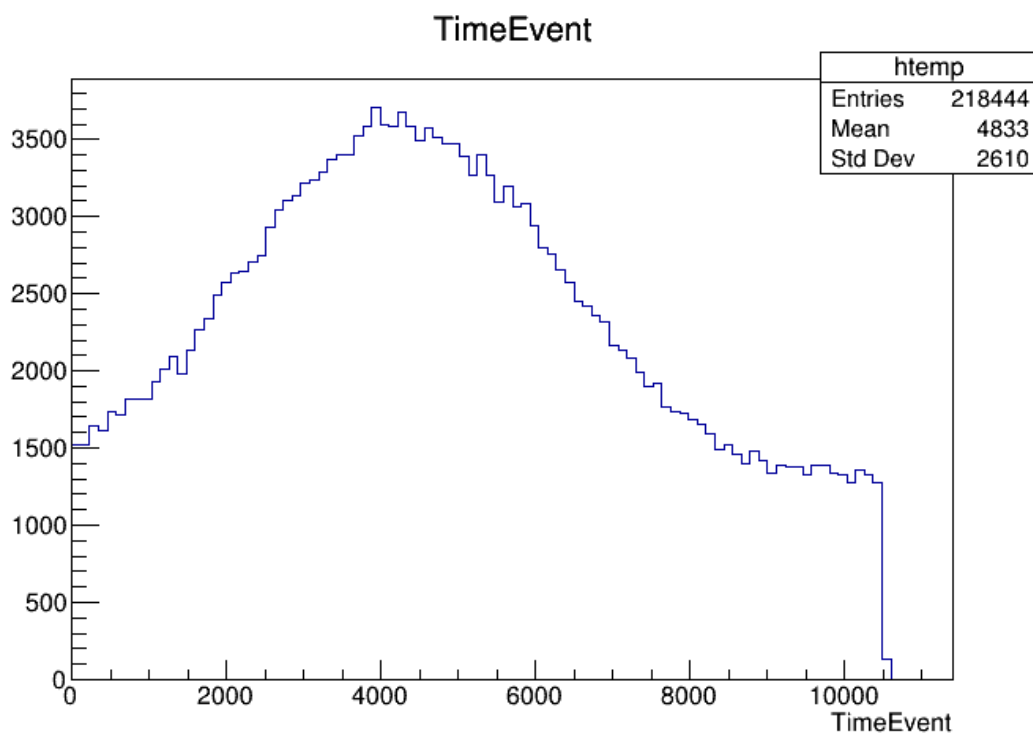
----- Quantum Gravity Energy Limit -----
-----Type   : Linear
-----Energy Limit: 2.46637e+14 TeV
-----Energy Limit: 0.246637 (e18 GeV)

```

Congrats, the limit on quantum gravity scale that we computed is 0.25e18 GeV, but this is of course an approximation since we supposed a perfect instrument.

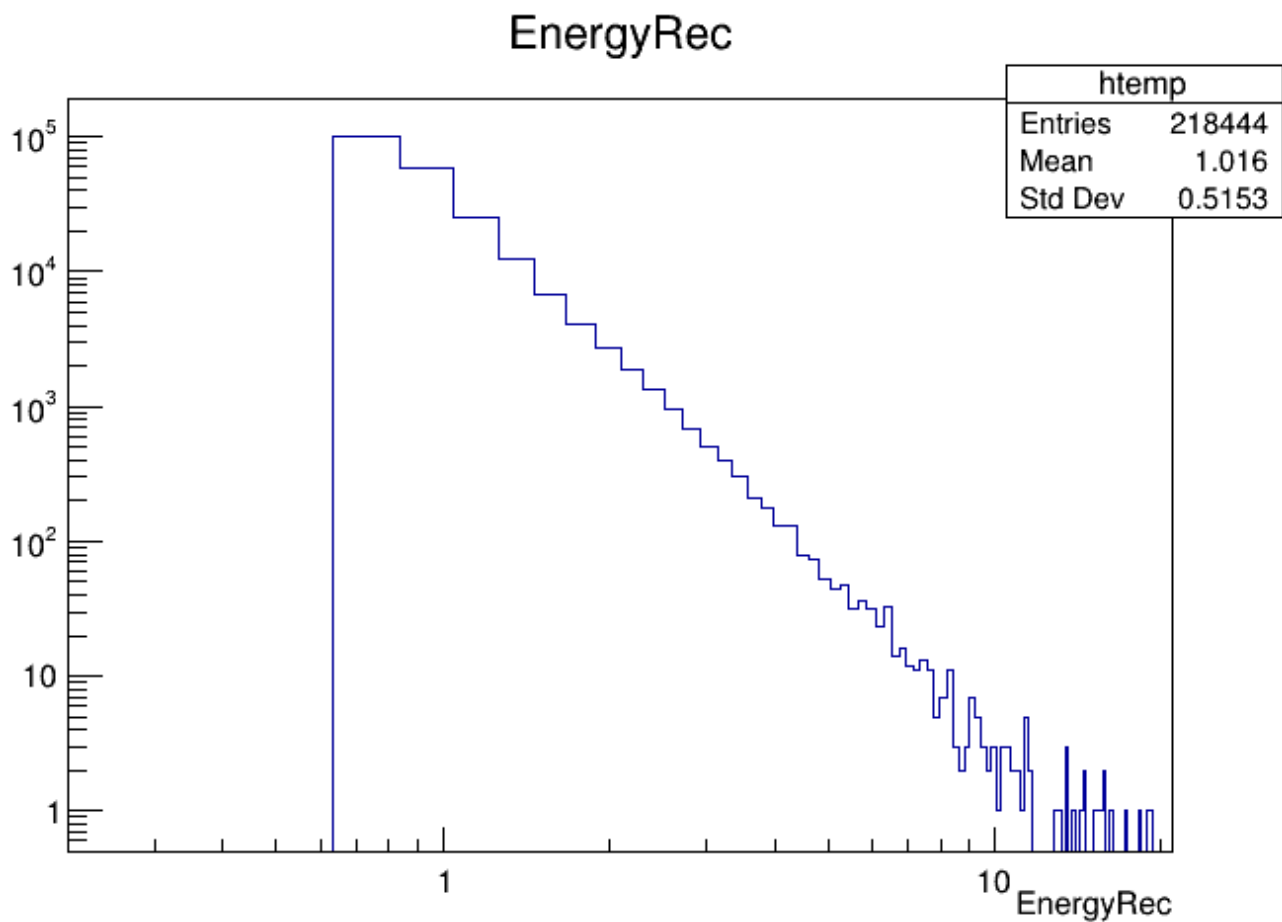
Simulations can be directly accessed and plotted by root, they are contained in LIVlib/results/SimulationTreeToyDataExample.root.

- Go in LIVlib results (cd LIVlib/results)
- root SimulationTreeToyDataExample.root
- treeSimu->Draw("TimeEvent") (to Draw the time distribution)



You can as well plot the energy distribution :

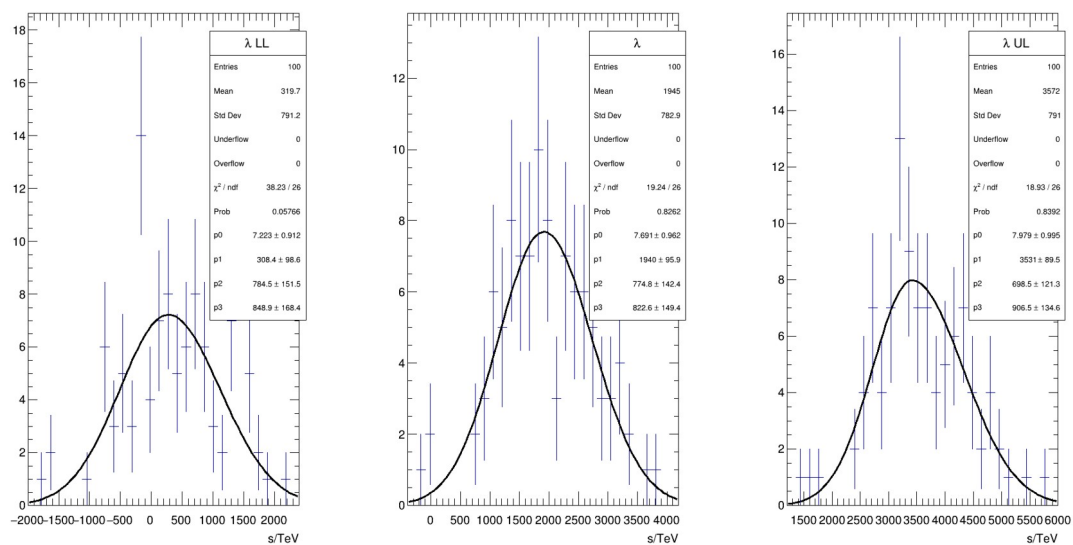
- treeSimu->Draw("EnergyRec")



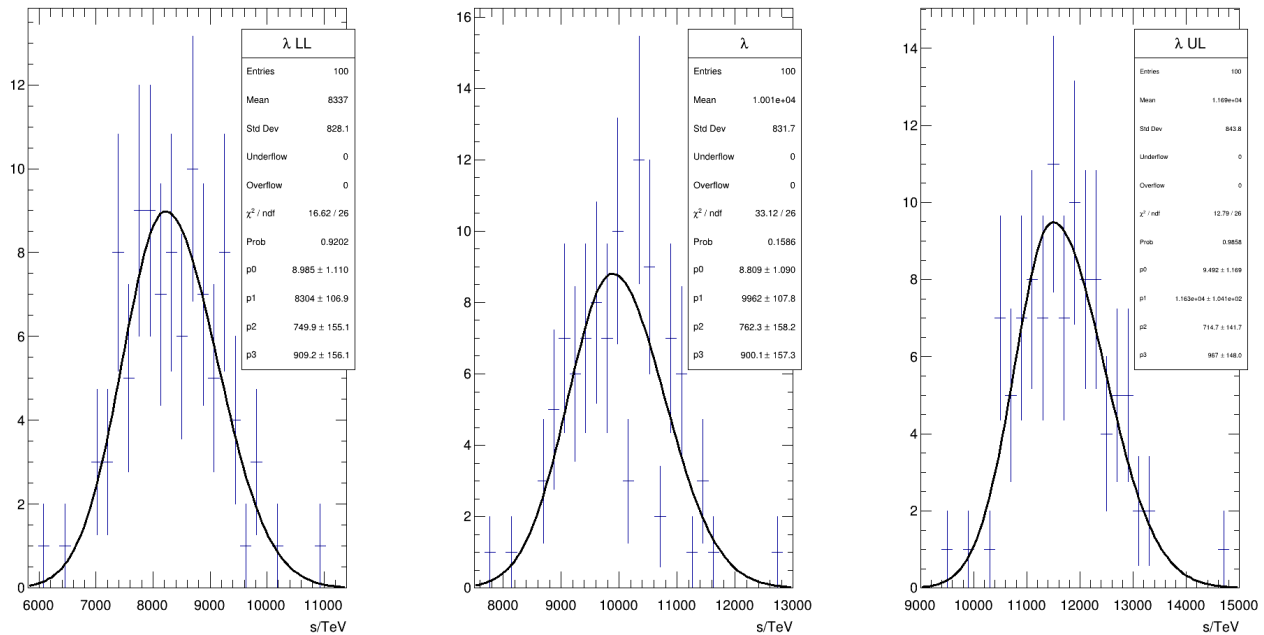
It is as well possible to inject a fake LIV effect in the simulations in order to check that it is well reconstructed. It is mandatory to perform this kind of check to be sure that the software is able to reconstruct well LIV effect.

let's try, redo :

- root
- .L ../LIVlib-build/lib/libLIV.so
- .L scripts/Tutorial\_1.C
- test\_runner(2000) (injection of 2000 s/TeV)



Mean reconstructed lag is indeed 2000 s/TeV as what we injected in the simulations. Such a lag would be a 2 sigma effect, so it can be considered as a detection. Let's inject a stronger one, 10000 s/TeV



We got here a clear detection, incompatible with 0. In the next lesson we will see if it is still true if we take into account systematics.

Some exercises to go further :

- Change the different parameter to see how the limit evolve
  - spectral index
  - proportion of signal
  - Width of light curve Gaussian
  - redshift

### Lesson 3, A bit of uncertainty :

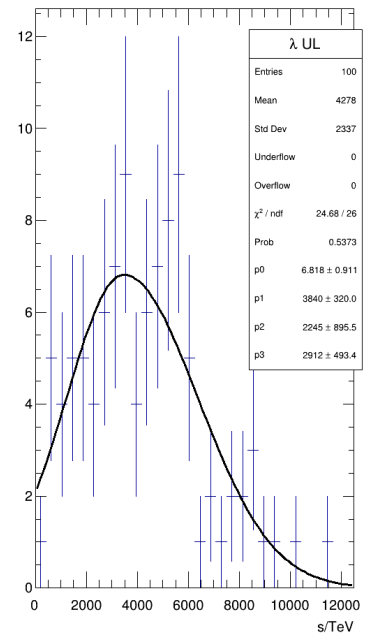
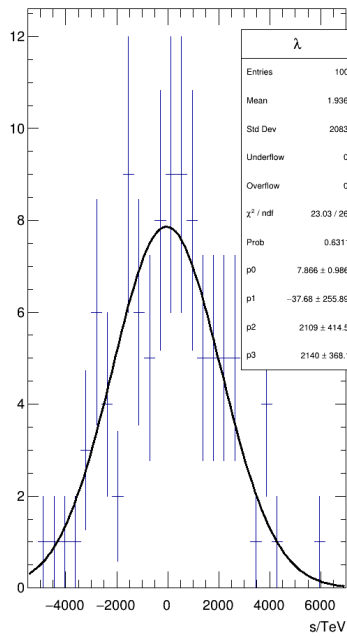
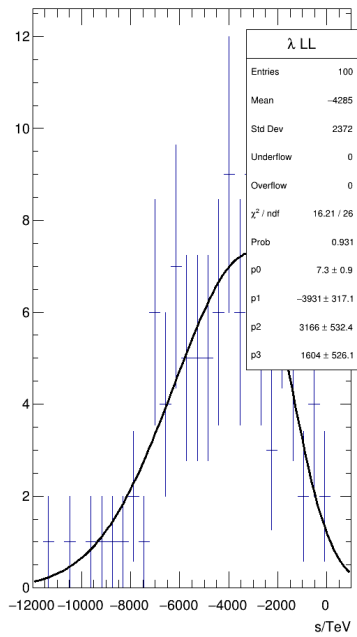
It is very important to take into account the systematical uncertainty of the apparatus you are using, in the previous example this is all the assumptions we made :

- the energy is always perfectly reconstructed for all events
- the amount of background is perfectly known
- there is no uncertainty on the temporal distribution of the events
- no uncertainty on the spectral distribution of events
- no uncertainty on the redshift

Let's try to go further, the systematics of the HESS observatory are already contained in the code, let's see how the analysis evolve if we take them into account.

- root
- `.L ../LIVlib-build/lib/libLIV.so`
- `.L scripts/Tutorial_1.C`
- `test_runner(0,1,1)` (injection of 0 s/TeV, 1 to perform simulations, 1 to say that we are doing the work with the nuisance parameters)

It will take more time to give a results since more free parameters are added to the fits, let's be patient... It should take around 5 minutes.



As expected, the distribution of the results is wider.

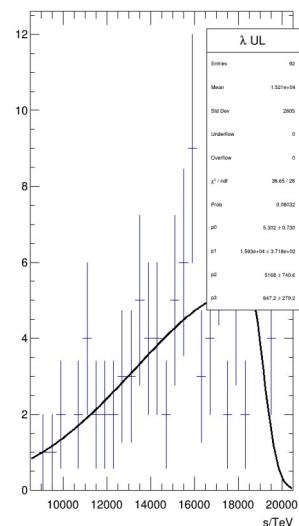
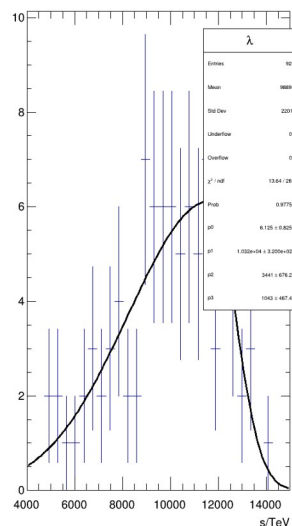
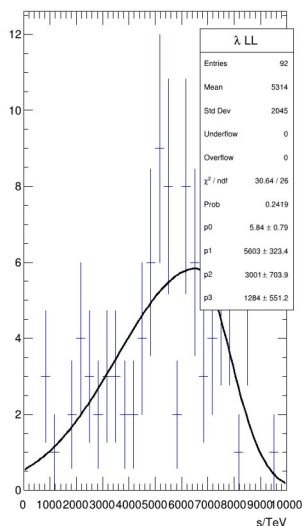
----- Quantum Gravity Energy Limit -----  
 -----Type : Linear-----  
 -----Energy Limit: 1.14845e+14 TeV-----  
 -----Energy Limit: 0.114845 (e18 GeV)-----

And Quantum gravity limit is less constraining. Let's see if we still get a detection with 10000 s/TeV injected ?

- root
- .L ../LIVlib-build/lib/libLIV.so
- .L scripts/Tutorial\_1.C
- test\_runner(10000,1,1)

(injection of 10000 s/TeV, 1 to perform simulations, 1 to say that we are doing the work with the nuisance parameters)

And be patient, again...





```

----- Best Fit lambda -----
----- Mean Lambda : 10323+-319.965
----- median      : 10050
----- Sigma Lambda: 3441.28+-676.164
----- Chi2        : 13.6381/26

```

Ouch, we now have : 10323 +/- 3441 so we only have ~3 sigmas... this is no more a detection if we take into account systematics.

Exercise to go further :

- how much lag injected do you need to have a detection taking into account the systematics of the experiment ?

#### Lesson 4, Interlude, from the DL3 data to the ON OFF DL3 :

DL3 data contain all the gamma-like event in the field of view. We are only interested in the amount of event in the direction of the source, within a certain angular region that we used for the high level analysis. This step will permit as well to extract the energy distribution of the background.

- Open the jupyter notebook LIVlib/python\_scripts/Produce\_ON\_OFF\_DL3.ipynb
- Nothing to do, simply run it, it will produce files the ON OFF DL3 data that LIVelihood can use

a source position need to be specified for source selection :

```

target_position = SkyCoord(
    329.71693826 * u.deg, -30.2255890 * u.deg, frame="icrs"
)
selection = dict(
    type="sky_circle",
    frame="icrs",
    lon=target_position.ra,
    lat=target_position.dec,
    radius=2 * u.deg,
)

```


as well as the position and size of the ON region (you should use the same you used for the analysis of the source) :


```

# define the on region
target_position = SkyCoord(ra=329.72, dec=-30.23, unit="deg", frame="icrs")
on_region_radius = Angle("0.11 deg")
on_region = CircleSkyRegion(center=target_position, radius=on_region_radius)

```

It should produce files named like :

 obs\_id\_33788\_unbinned\_spectrum.fits

 obs\_id\_33789\_unbinned\_spectrum.fits

This is the file that will be used by LIVelihood to access to the IRFs and the events. You can copy paste them in an other directory for the rest of the tutorial.

## Lesson 5, Analysis and simulation with the IRFs (effective area only):

LIVelihood is able to use the IRFs contained in the DL3 files in order to perform a more realistic simulation. The Likelihood in this case is precomputed in tables, in order to not perform the same computation for each event added to the global likelihood.

Please go in the scripts/Tutorial\_1.C

It contains the following line of code :

```
SourceHandle* source = (SourceHandle*) new  
TutorialHandle_1(powLIV,mod,shift,UseMC,1,8,UseNuisance,UseTable,UseAcceptance  
,UseMigMatrix,"/home/livelihood/LIVlib/data_hess/");
```

**"/home/livelihood/LIVlib/data\_hess/"** is the place where the data are stored, you can change it if you chose another name to store the data previously produced.

This line of code defines the binning of the table (there is one table per run) :

```
source->CreateTables3("Example",50,20,50,-20000,20000);
```

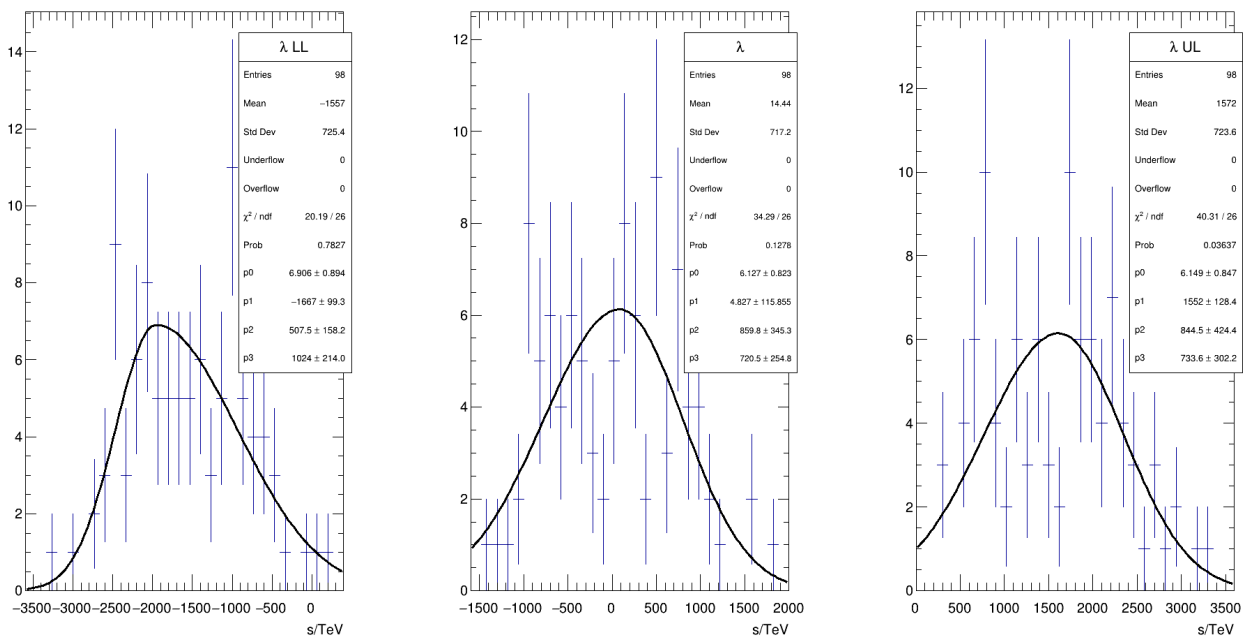
and creates table called "Example\_100" with 100 bins in energy, 20 bins in time, 100 bins in delay sampled from -20000 s/TeV to +20000 s/TeV.

Let's try it, do :

- root
- .L ../LIVlib-build/lib/libLIV.so
- .L scripts/Tutorial\_1.C
- test\_runner(0,1,0,1,1,0)

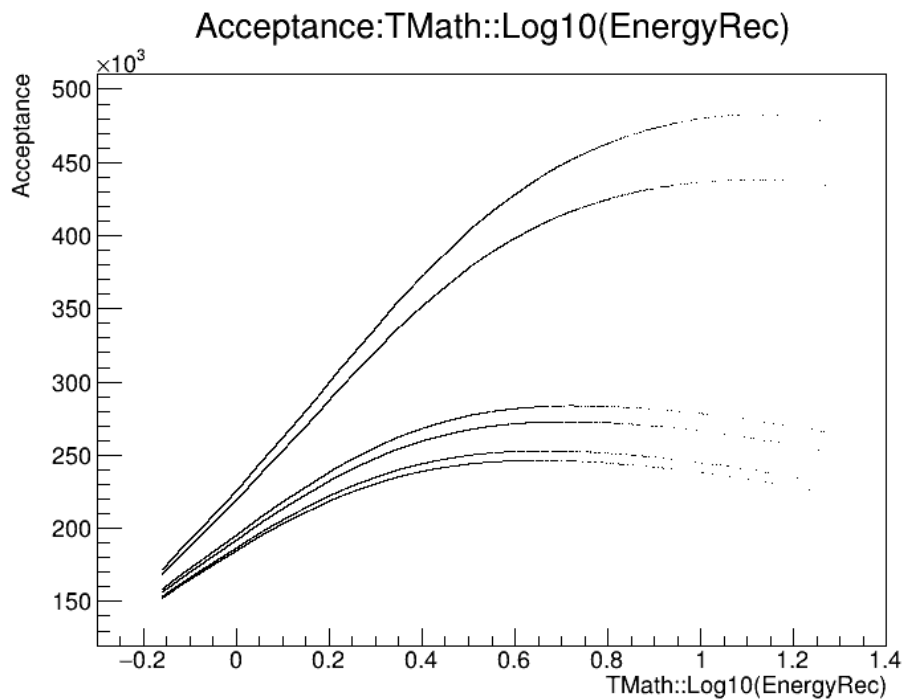
(0 lag injected, 1 Monte carlo produced , 0 no systematics, 1 use table, 1 use effective area, 0 do not use migration matrix).

With this setup we are simulating only the effect of the effective area versus the energy. This is the results we get :



Looks good. We can go further, you can have a look at Monte Carlo simulations

- root results/SimulationTreeToyDataExample.root
- treeSimu->Draw("Acceptance:TMath::Log10(EnergyRec)")



It permits to plot the acceptance versus the log of energy for all the events, you can see that different runs have different acceptance (this is mostly due to the variability of the zenith angle during the night).

Let's have a look at the tables, it contains the likelihood of the events :

- `root ../LIVlib-build/IRF/IRFtables_Example_100.root`
- `.ls`

you should obtain the list of the histograms contained in the file :

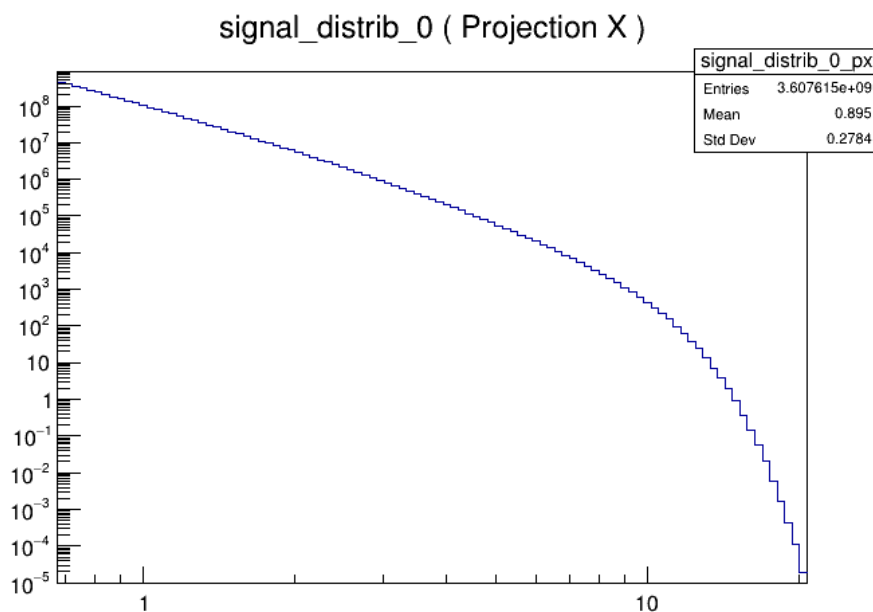
**KEY: TH3F signal\_distrib\_0**

is a 3D histogram of the first run, it contain the 3D PDF

**KEY: TH1F signal\_norm\_0**

is the normalisation of the pdf, let's plot them :

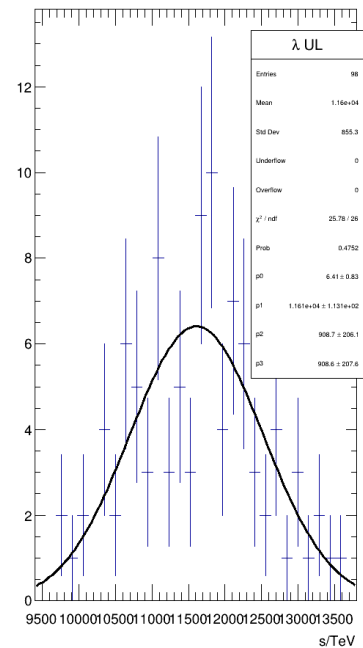
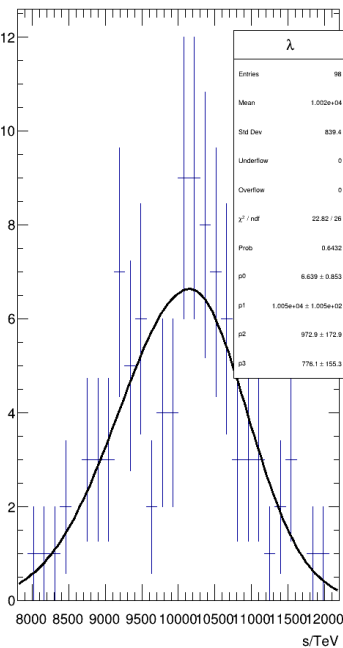
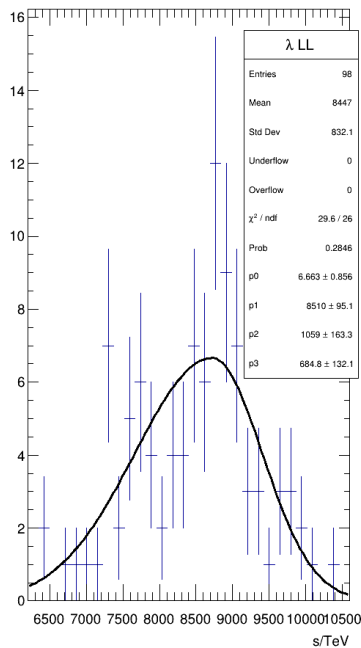
- `signal_distrib_0->ProjectionX()->Draw()`



This is the projection of the energy distribution of the PDF, this is not really useful or meaningful, but can be used to spot bugs sometimes.

Let's perform the same work with an injected lag :

- root
- .L ../LIVlib-build/lib/libLIV.so
- .L scripts/Tutorial\_1.C
- test\_runner(10000,1,0,1,1,0)



Impressive, the lag is well reconstructed (Mean Lambda :  $10048.8 \pm 100.52$ ) , the tables are fine.

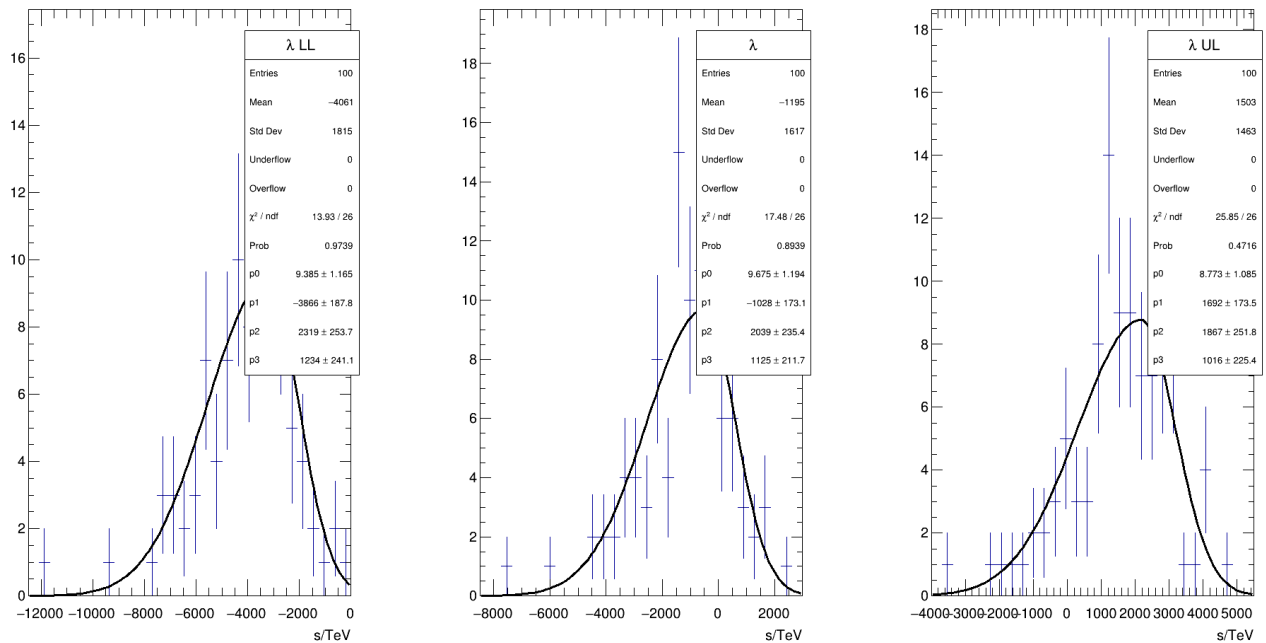
## Lesson 6, Analysis with IRFs (energy migration) :

We will take into account that the energy migration is slow for both table and simulations. In simulations, it implies another step that consists to generate reconstructed energy thanks to the migration matrix.

To add energy migration effect, simply do :

- root
- .L ../LIVlib-build/lib/libLIV.so
- .L scripts/Tutorial\_1.C
- test\_runner(0,1,0,1,1,1)

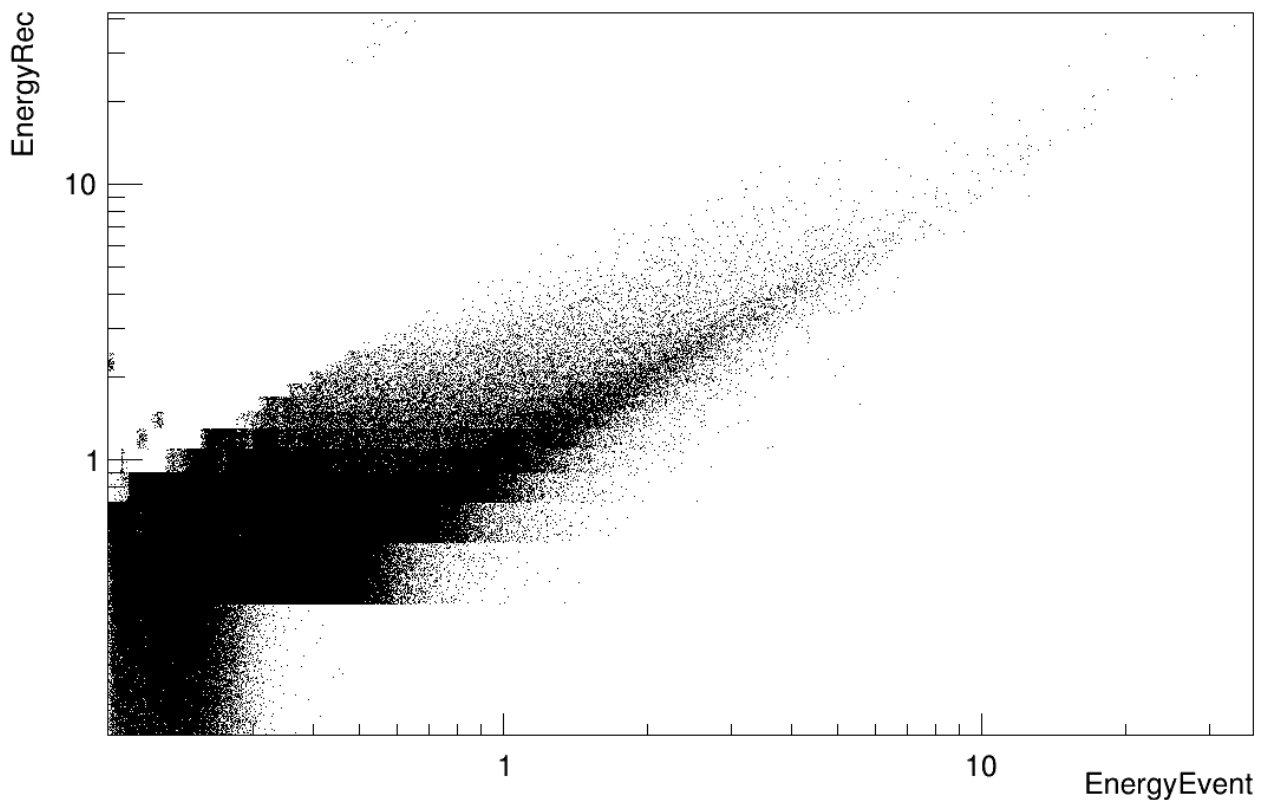
Simulation and table computing will take time...



This is the results we got. A bias can be observed, this is generally due to the precision of the tables. It takes a lot of time to improve the precision of table, so I suggest to not focus on that. Since the bias is less important than the statistical uncertainty it is OK.

You can as well look at the Monte Carlo simulations :

- root results/SimulationTreeToyDataExample.root
- treeSimu->Draw("EnergyRec:EnergyEvent")



it is interesting to see that the binning is visible in the data, another way of improvement of this measurement can be to increase the matrix binning (cf lesson 4).

## Lesson 7, Different types of simulations; simulations based on real data :

LIVelihood can handle 3 different types of simulations :

- Full simulations (the standard way, used in all the tutorial before)
  - Events are simulated from the spectra and light curve and the IRFs
  - caveat : we have to suppose a spectra for the background (2.7) and suppose that they follow the same IRFs than gamma-like
- OFF simulations :
  - the signal is simulated thanks to full simulation
  - the background is simulated from the distribution of OFF event in the DL3
  - Should be more realistic but depends of the OFF event stat in the DL3
  - It is completely useless for sources where the background is negligible (this one for example)
- ON OFF simulations :
  - the signal and background are simulated from ON and OFF region
  - Permits to resimulate real data, the results obtained from this simulation can be considered as real results

Let's try with the third one, go in the file src/LIVTutorialHandle\_1.cpp

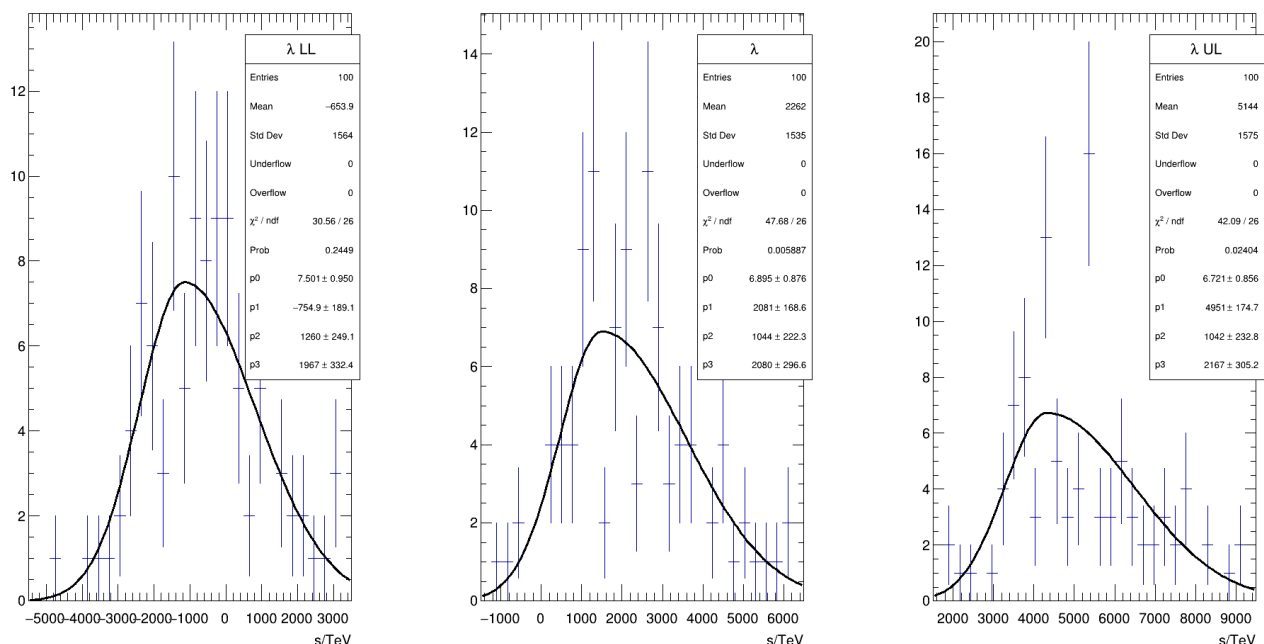
in this line :

```
if(UseToyMC) GenerateOnTheFlightSimulations(Nevent, Nsimu, lambda_i, UseAcceptance,
UseMigMatrix, "ToyDataExample",0,0,300);
```

change to :

```
if(UseToyMC) GenerateOnTheFlightSimulations(Nevent, Nsimu, lambda_i, UseAcceptance,
UseMigMatrix, "ToyDataExample",0,2,300);
```

This time the limit will use real data, let's see what we get :



And the 95% limit is :

```
----- Quantum Gravity Energy Limit -----
-----Type : Linear
-----Energy Limit: 6.21044e+13 TeV
-----Energy Limit: 0.0621044 (e18 GeV)
```

So the results show a positive ~2 sigma lag, not incompatible with a statistical fluctuation.

## Lesson 8, going further :

There is many ways to go further here

- The gaussian shape of the light curve does not reflect the full complexity of the light curve, then another more realistic model can be used
- You can as well play in the simulation side, what would have been the results with a flare at a redshift of 0.5 ? of 0.01 ? Pay attention to the fact that signal of the source decrease as  $z^2$ ...
- You can extract quadratic LIV limits, as well as DSR instead of LIV.
- Try to simulate any other source you may know and/or would be interested
- Any idea ? Go ahead !