LAT Analysis with Python

FSSC Workshops, Fall 2010

Introduction

- Python is a programming language similar to Tcl, Perl, Ruby, Scheme or Java
  - Very clear, readable syntax
  - Embeddable within apps as a scripting interface
  - http://wiki.python.org/moin/BeginnersGuide
- All of the Science Tools are included as python modules with some additional capabilities like
  - Upper limits
  - Direct access to results (and data)

Let's get started...

- You can access any of the gt* tools by using the gt_apps module

```
[user@localhost ~]$ python
Python 2.5.1 (r251:54863, Sep  6 2010, 12:08:53)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from gt_apps import *
>>>
```

- gtmaketime is called maketime, gtselect is called filter and gtbin is called evtbin.

- For example, if you wanted to make a counts map of the 3C 454 that we are looking at you would first set the options:

```
>>> evtbin['algorithm'] = 'CMAP'
>>> evtbin['evfile'] = '3C454_events_filtered.fits'
>>> evtbin['outfile'] = '3C454_cmap_python.fits'
>>> evtbin['scfile'] = 'NONE'
>>> evtbin['nxpix'] = 100
>>> evtbin['nypix'] = 100
>>> evtbin['binsz'] = 0.5
>>> evtbin['coordsys'] = 'CEL'
>>> evtbin['xref'] = 343.5
>>> evtbin['yref'] = 16.15
>>> evtbin['axisrot'] = 0.0
>>> evtbin['proj'] = 'STG'
```

The python method for calling gtbin is the evtbin object.

The options are named the same as in the command line tools. You can also read them back.

```
>>> my_file_name = evtbin['outfile']
>>> print my_file_name
test.cmap.fits
```

- If you want to see what will be executed:

```
>>> evtbin.command()
'time -p gtbin evfile=PG1553_filtered_gti.fits scfile=NONE outfile= algorithm="CMAP"
ebinalg="LOG" emin=30.0 emax=200000.0 ebinfile=NONE tbinalg="LIN" tbinfile=NONE
nxpix=100 nypix=100 binsz=0.5 coordsys="CEL" xref=238.929 yref=11.1901 axisrot=0.0
rafield="RA" decfield="DEC" proj="STG" evtable="EVENTS" sctable="SC_DATA"
efield="ENERGY" tfield="TIME" chatter=2 clobber=yes debug=no gui=no mode="ql"'
```

- Note that even hidden parameters are shown
- There are similar python objects for every command line tool.
- Now, execute the command:

```
>>> evtbin.run()
time -p /Users/jsperki1/Research/Fermi/ScienceTools/i386-apple-darwin10.4.0/bin/gtbin
evfile=PG1553_filtered_gti.fits scfile=NONE outfile=test.cmap.fits algorithm="CMAP"
ebinalg="LOG" emin=30.0 emax=200000.0 ebinfile=NONE tbinalg="LIN" tbinfile=NONE
nxpix=100 nypix=100 binsz=0.5 coordsys="CEL" xref=238.929 yref=11.1901 axisrot=0.0
rafield="RA" decfield="DEC" proj="STG" evtable="EVENTS" sctable="SC_DATA"
efield="ENERGY" tfield="TIME" chatter=2 clobber=yes debug=no gui=no mode="ql"
This is gtbin version ScienceTools-v9r17p0-fssc-20100906
real 0.33
user 0.22
sys 0.04
```

Likelihood Analysis

- You've already got a filtered event file and an XML file from the previous lessons so we'll just jump right into running the likelihood fit:

```
>>> import pyLikelihood
>>> from UnbinnedAnalysis import *
>>> obs = UnbinnedObs
('PG1553_filtered_gti.fits','SC.fits',expMap='expMap.fits',expCube='expCube.fits',irfs=
'P6_V3_DIFFUSE')
>>> like1 = UnbinnedAnalysis(obs,'PG1553_model.xml',optimizer='DRMNFB')
```

- There are lots of ways to inspect your objects like via the print command:

```
>> print obs
Event file(s): PG1553_filtered_gti.fits
Spacecraft file(s): SC.fits
Exposure map: expMap.fits
Exposure cube: expCube.fits
IRFs: P6_V3_DIFFUSE
```

```
>>> print like1
Event file(s): PG1553_filtered_gti.fits
Spacecraft file(s): SC.fits
Exposure map: expMap.fits
Exposure cube: expCube.fits
IRFs: P6_V3_DIFFUSE
Source model file: PG1553_model.xml
Optimizer: DRMNFB
```

- You can also see all of the available attributes and functions of an object via the 'dir' function.

```
>>> dir(like1)
['NpredValue', 'Ts', 'Ts_old', '_Nobs', '__call__', '__class__', '__delattr__',
'__dict__', '__doc__', '__format__', '__getattribute__', '__getitem__', '__hash__',
'__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', '_errors', '_importPlotter', '_inputs', '_isDiffuseOrNearby',
'_minosIndexError', '_npredValues', '_plotData', '_plotResiduals', '_plotSource',
'_renorm', '_separation', '_setSourceAttributes', '_srcCnts', '_srcDialog', '_xrange',
'addSource', 'covar_is_current', 'covariance', 'deleteSource', 'disp', 'e_vals',
'energies', 'energyFlux', 'energyFluxError', 'fit', 'flux', 'fluxError', 'freePars',
'freeze', 'getExtraSourceAttributes', 'logLike', 'maxdist', 'minosError', 'model',
'nobs', 'normPar', 'observation', 'oplot', 'optObject', 'optimize', 'optimizer',
'par_index', 'params', 'plot', 'plotSource', 'reset_ebounds', 'resids',
'restoreBestFit', 'saveCurrentFit', 'setFitTolType', 'setFreeFlag', 'setPlotter',
'setSpectrum', 'sourceNames', 'srcModel', 'state', 'syncSrcParams', 'thaw', 'tol',
'tolType', 'total_nobs', 'writeCountsSpectra', 'writeXml']
```

- You should play with these and see what they do

- Let's do the fit now...

```
>>> like1.tol
0.001
>>> like1.fit(verbosity=0)
273356.93879166018
>>> like1.model['_1FGLJ1555.7+1111']
_1FGLJ1555.7+1111
Spectrum: PowerLaw2
44 Integral: 5.013e-01 3.039e-02 1.000e-04 1.000e+04 ( 1.000e-07)
45 Index: 1.674e+00 3.340e-02 0.000e+00 5.000e+00 (-1.000e+00)
46 LowerLimit: 2.000e+02 0.000e+00 3.000e+01 5.000e+05 ( 1.000e+00) fixed
47 UpperLimit: 4.000e+05 0.000e+00 3.000e+01 5.000e+05 ( 1.000e+00) fixed
```

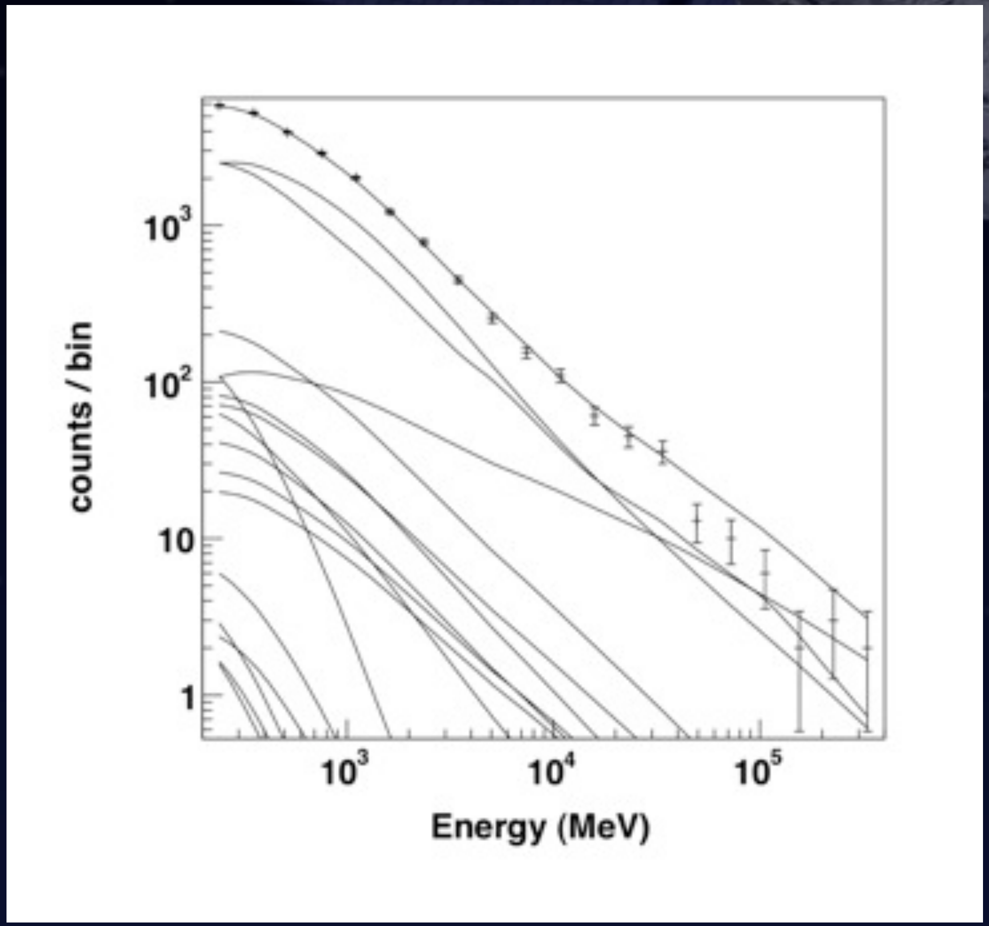
Check the tolerance.

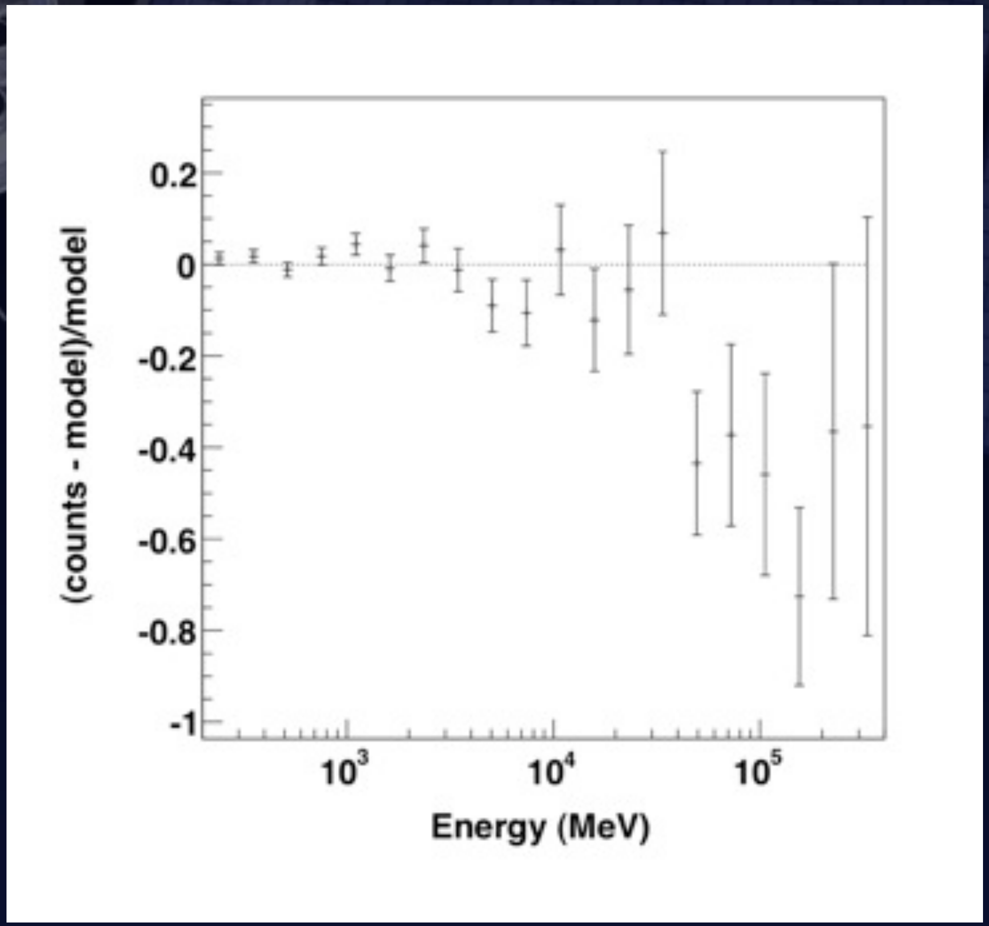
You can get the details for any source (including extended ones). The name must match one of the ones in the XML file.

Do the fit. The number that's there is the total TS of the model

- You can plot the models and the residuals, access the TS of any source and save the results

```
>>> like.plot()
```






```
>>> like2.Ts('_1FGLJ1555.7+1111')
2409.9824593166122
>>> like1.logLike.writeXml('PG1553_fit1.xml')
```

Likelihood Looping

- Now you are prepared to create likelihood based light curves and likelihood based differential spectral points.
- Algorithm:
  - Use evtbin/maketime/expCube/expMap to divide up your data into energy and/or time bins.
  - Use UnbinnedAnalysis to fit your model
  - Read out the details in python like

```
obs = UnbinnedObs('LC_PG1553_filtered_gti_bin'+str(this_bin)
   +'.fits','SC.fits',expMap='LC_expMap_bin'+str(this_bin)
   +'.fits',expCube='LC_expCube_bin'
   +str(this_bin)+'.fits',irfs='P6_V3_DIFFUSE')
like = UnbinnedAnalysis(obs,'PG1553_fit2.xml',optimizer='NewMinuit')
like.fit(verbosity=0,covar=True)
IntFlux.append(like.model['3C454'].funcs['Spectrum'].getParam('Integral').value())
Gamma.append(like.model['3C454'].funcs['Spectrum'].getParam('Index').value())
IntFluxErr.append(math.sqrt(like.covariance[0][0]))
GammaErr.append(math.sqrt(like.covariance[1][1]))
TS.append(like.Ts('3C454'))
```

Upper Limits?

- You can compute an upper limit on any source in your model file. Let's do it for one of the secondary (background) sources.

```
>>> like.Ts('_1FGLJ1555.7+1111')
27.386814499412139
>>> from UpperLimits import UpperLimits
>>> ul=UpperLimits(like)
>>> ul['_1FGLJ1555.7+1111'].compute()
0 0.516659997048 -8.19436536403e-08 1.06703428967e-07
1 0.608104330636 0.0752282547901 1.32473024814e-07
2 0.699548664224 0.282170619224 1.59817568172e-07
3 0.790992997813 0.600532181406 1.88566967931e-07
4 0.882437331401 1.01528523902 2.18871678037e-07
5 0.958180659232 1.42342619883 2.45210426096e-07
(2.4079464677188238e-07, 0.9454820364677855)
>>> ul['_1FGLJ1555.7+1111'].results
[2.41e-07 ph/cm^2/s for emin=100.0, emax=300000.0, delta(logLike)=1.35]
```

- You can also compute the upper limit over arbitrary energy intervals.

```
>>> ul['_1FGLJ1555.7+1111'].compute(emin=200,emax=4e5)
0 0.516659997048 -8.19436536403e-08 5.1823634147e-08
1 0.608104330636 0.0752282547901 6.10078032374e-08
2 0.699548664224 0.282170619224 7.01943874827e-08
3 0.790992997813 0.600532181406 7.93830543117e-08
4 0.882437331401 1.01528523902 8.85740045624e-08
5 0.958180659232 1.42342619883 9.61886521108e-08
(9.491203094687220 4e-08, 0.94548203664677855)
>>> ul['_1FGLJ1555.7+1111'].results
[2.41e-07 ph/cm^2/s for emin=100.0, emax=300000.0, delta(logLike)=1.35, 9.49e-08 ph/
cm^2/s for emin=200.0, emax=400000.0, delta(logLike)=1.35]
```

Tip of the Iceberg

- There are lots of other tools in the python kit:
  - Plotting routines (matplotlib and rootplot)
  - Access to the fits files (pyfit)
  - Direct IRF access
- Check out the user contributed tools for examples (and useful tools)
- I encourage you to explore and see what’s available.

Bonus: likeSED

- User tool to generate differential flux bins
- Takes a likelihood object as input

```
>>> from likeSED import *
This is likeSED version 08.
>>> inputs = likeInput(like2,'_1FGLJ1555.7+1111',nbins=9, model='PG1553_fit2.xml')
>>> centers = [0.2767, 0.5917, 1.265, 2.706, 5.787, 12.37, 26.46, 86.60]
>>> inputs.plotBins()
>>> inputs.fullFit(CoVar=True)
Full energy range model for _1FGLJ1555.7+1111:
_1FGLJ1555.7+1111
Spectrum: PowerLaw2
44 Integral: 4.944e-01 2.995e-02 1.000e-04 1.000e+04 ( 1.000e-07)
45 Index: 1.672e+00 3.337e-02 0.000e+00 5.000e+00 (-1.000e+00)
46 LowerLimit: 2.000e+02 0.000e+00 3.000e+01 5.000e+05 ( 1.000e+00) fixed
47 UpperLimit: 4.000e+05 0.000e+00 3.000e+01 5.000e+05 ( 1.000e+00) fixed
>>> sed = likeSED(inputs)
>>> sed.getECent()
```

- Run the fit and plot the results

```
>>> sed.fitBands()
-Running Likelihood for band0-
-Running Likelihood for band1-
-Running Likelihood for band2-
-Running Likelihood for band3-
-Running Likelihood for band4-
-Running Likelihood for band5-
-Running Likelihood for band6-
-Running Likelihood for band7-
>>> sed.Plot()
```