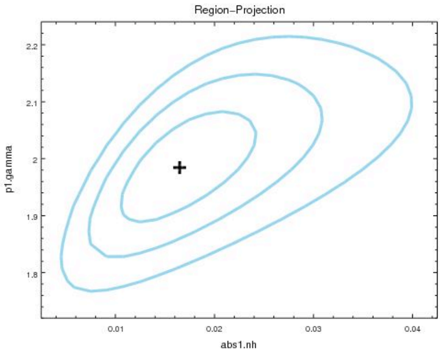
Modeling, Fitting and Statistics

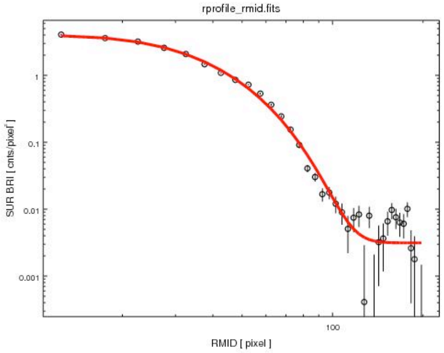
Aneta Siemiginowska
CXC Science Data System
http://cxc.harvard.edu/sherpa

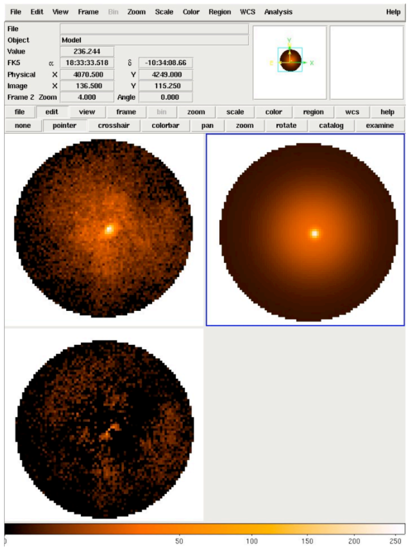
CXC DS Sherpa Team:
Stephen Doe, Dan Nguyan, Brian Refsdal

Sherpa

- Generalized fitting package with a powerful model language to fit 1D and 2D data
- Forward fitting technique - a model is evaluated, compared to the actual data, and then the parameters are changed to improve the match. This is repeated until convergence occurs.
- Python environment
- Modularized code
- Walkthrough with a few examples







File
Object Model
Value 236.244
FK5 α 18:33:33.518 δ -10:34:08.66
Physical X 4070.500 Y 4249.000
Image X 136.500 Y 115.250
Frame 2 Zoom 4.000 Angle 0.000

Modeling: what do we need to learn from our new observations?

- Data:
  - Write proposal, win and obtain new data
- Models:
  - model library that can describe the physical process in the source
  - typical functional forms or tables, derived more complex models - plasma emission models etc.
  - parameterized approach - models have parameters
- Optimization Methods:
  - to apply model to the data and adjust model parameters
  - obtain the model description of your data
  - constrain model parameters etc. search of the parameter space
- Statistics:
  - a measure of the model deviations from the data

Observations: Chandra Data and more…

- X-ray Spectra

  typically PHA files with the RMF/ARF calibration files
- X-ray Images

  FITS images, exposure maps, PSF files
- Lightcurves

  FITS tables, ASCII files
- Derived functional description of the source:
  - Radial profile
  - Temperatures of stars
  - Source fluxes
- Concepts of Source and Background data
- Any data array that needs to be fit with a model

Observations: Data I/O in Sherpa

- Load functions (PyCrates) to input the data:
  - data: load_data, load_pha, load_arrays, load_ascii
  - calibration: load_arf, load_rmf load_multi_arfs, load_multi_rmfs
  - background: load_bkg, load_bkg_arf , load_bkg_rmf
  - 2D image: load_image, load_psf
  - General type: load_table, load_table_model, load_user_model

Help file:

```
load_data( [id=1], filename, [options] )
load_image( [id=1], filename|IMAGECrate,[coord="logical"] )
```

Examples:

```
load_data("src", "data.txt", ncols=3)
```

- Multiple Datasets - data id

Default data id =1

```
load_data(2, “data2.dat”, ncols=3)
```

```
load_data("rprofile_mid.fits[cols RMID,SUR_BRI,SUR_BRI_ERR]")
load_data(“image.fits”)
load_image(“image.fits”, coord=“world”))
```

- Filtering of the data
  - load_data expressions
  - notice/ignore commands in Sherpa

Examples:

```
notice(0.3,8)
notice2d("circle(275,275,50)")
```

Modeling: Model Concept in Sherpa

- Parameterized models: $f(E,\Theta_i)$ or $f(x_i,\Theta_i)$

  absorption - N_H

  photon index of a power law function - Γ

  blackbody temperature kT

- Composite models:

  combined individual models in the library into a model that describes the observation

```
set_model("xsphabs.abs1*powlaw1d.p1")
set_model("const2d.c0+gauss2d.g2")
```

- Source models, Background models:

```
set_source(2,"bbody.bb+powlaw1d.pl+gauss1d.line1+gauss1d.line2")
set_bkg_model(2,"const1d.bkg2")
```

Modeling: Sherpa Models

- Model Library that includes XSPEC models

```
sherpa-11> list_models()
['atten',
 'bbody',
 'bbodyfreq',
 'beta1d',
 'beta2d',
 'box1d',...
```

- User Models:
  - Python Functions

    load_user_model, add_user_pars

  - Python interface to

    C/C++ or Fortran code/functions

Example Function myline:

```
def myline(pars, x):
    return pars[0] * x + pars[1]
```

In sherpa:

```
from myline import *

load_data(1, "foo.dat")
load_user_model(myline, "myl")
add_user_pars("myl", ["m","b"])
set_model(myl)
myl.m=30
myl.b=20
```

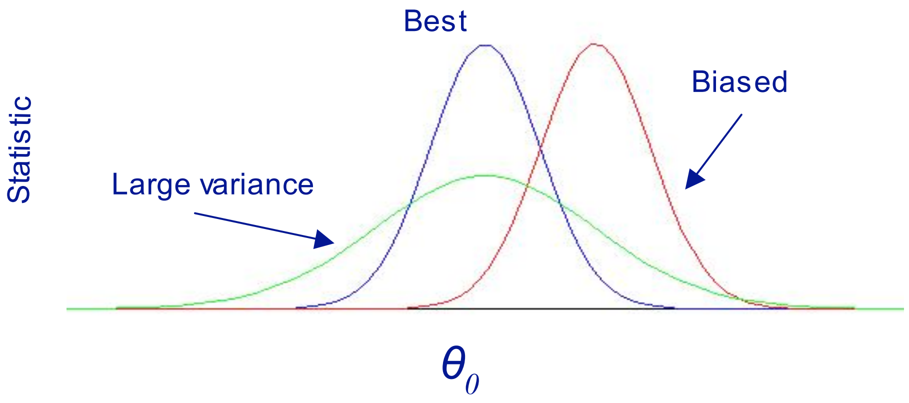
Modeling: Parameter Space

```
sherpa-21> set_model(xsphabs.abs1*xszphabs.zabs1*powlaw1d.p1)
sherpa-22> abs1.nH = 0.041
sherpa-23> freeze(abs1.nH)
sherpa-24> zabs1.redshift=0.312

sherpa-25> show_model()
Model: 1
apply_rmf(apply_arf((106080.244442 * ((xsphabs.abs1 * xszphabs.zabs1) * powlaw1d.p1))))
   Param        Type          Value          Min          Max      Units
   -----        ----          -----          ---          ---      -----
   abs1.nh         frozen        0.041        0       100000 10^22 atoms / cm^2
   zabs1.nh        thawed            1        0       100000 10^22 atoms / cm^2
   zabs1.redshift frozen         0.312        0           10
   p1.gamma      thawed            1      -10           10
   p1.ref          frozen            1     -3.40282e+38  3.40282e+38
   p1.ampl         thawed            1           0  3.40282e+38
```

Statistics in Sherpa

- χ^2 statistics with different weights
- Cash and Cstat based on Poisson likelihood

```
sherpa-12> list_stats()
['leastsq',
 'chi2constvar',
 'chi2modvar',
 'cash',
 'chi2gehrels',
 'chi2datavar',
 'chi2xspecvar',
 'cstat']
sherpa-13> set_stat(“chi2datavar”)
sherpa-14> set_stat(“cstat”)
```



Maximum Likelihood: Assessing the Quality of Fit

One can use the Poisson distribution to assess the probability of sampling data ***D_i*** given a predicted (convolved) model amplitude M_i. Thus to assess the quality of a fit, it is natural to maximize the product of Poisson probabilities in each data bin, *i.e.*, to maximize the Poisson likelihood:

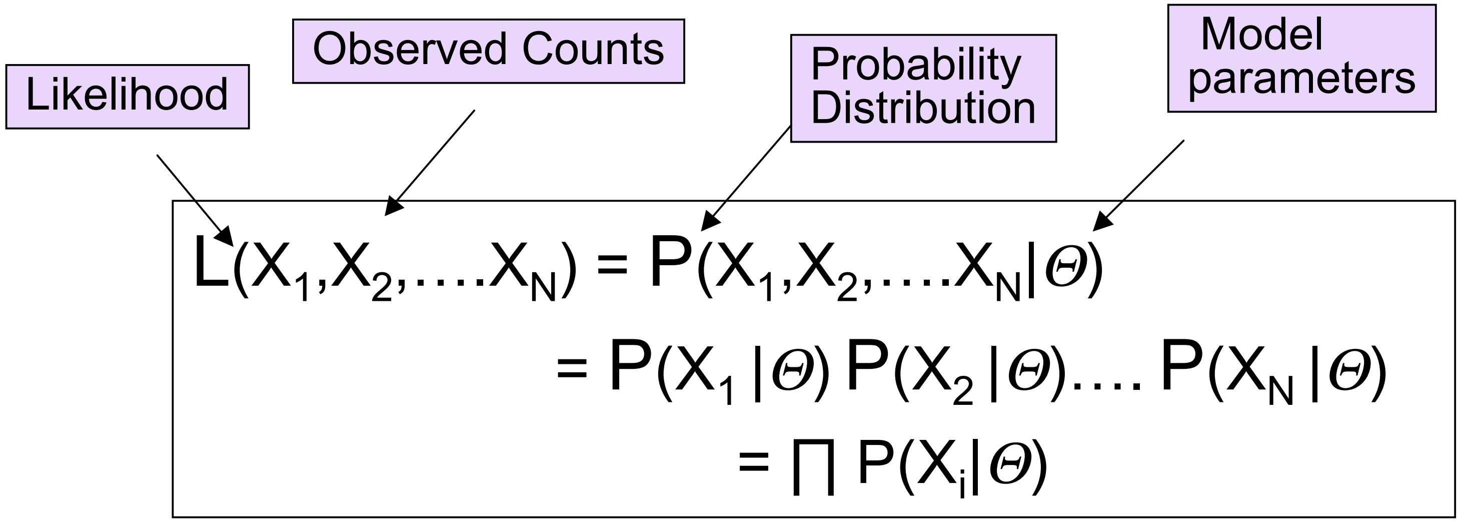
$$\mathrm{L} = \prod_i^N \mathrm{L}_i = \prod_i^N \frac{M_i^{D_i}}{D_i!} \exp(-M_i) = \prod_i^N p(D_i | M_i)$$

In practice, what is often maximized is the log-likelihood,

L = log$\mathcal{L}$. A well-known statistic in X-ray astronomy which is related to *L* is the so-called “Cash statistic”:

$$C \equiv 2\sum_i^N [M_i - D_i \log M_i] \propto -2L,$$

Likelihood Function



Likelihood

Observed Counts

Probability Distribution

Model parameters

$$
\begin{aligned}
L(X_1,X_2,\ldots X_N) &= P(X_1,X_2,\ldots X_N|\Theta) \\
&= P(X_1\,|\Theta)\,P(X_2\,|\Theta)\ldots P(X_N\,|\Theta) \\
&= \prod P(X_i|\Theta)
\end{aligned}
$$

P - Poisson Probability Distribution for X-ray data

$X_1,\ldots X_N$ - X-ray data - independent

Θ - model parameters

Likelihood Function: X-rays Example

- X-ray spectra modeled by a power law function:

$$f(E)= A * E^{-\Gamma}$$

E - energy; A, Γ - model parameters: a normalization and a slope

Predicted number of counts:

$$M_i = \int R(E,i)*A(E)\ AE^{-\Gamma}\ dE$$

For A = 0.001 ph/cm2/sec, Γ=2 then in channels i= (10, 100, 200)

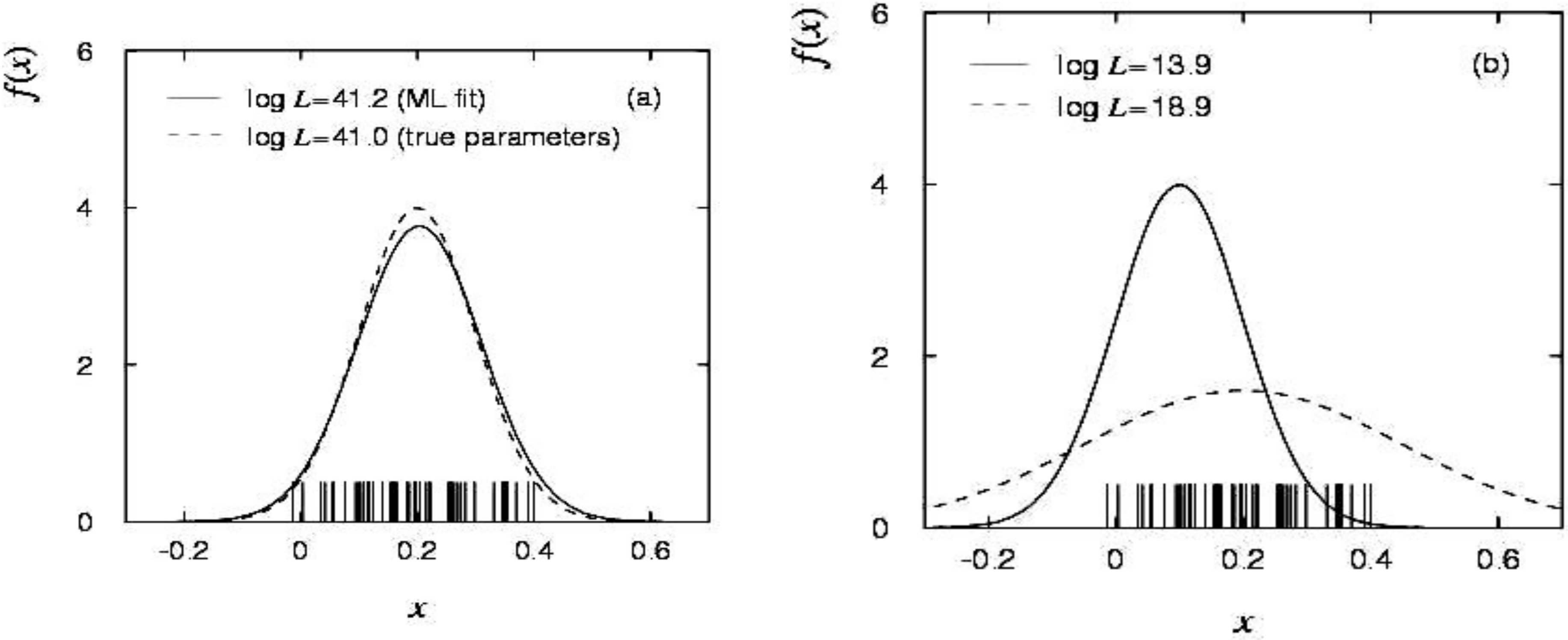
Predicted counts: M_i = (10.7, 508.9, 75.5)
Observed X_i = (15, 520, 74)
Calculate individual probabilities:
Use Incomplete Gamma Function
$\Gamma(X_i, M_i)$

$$\begin{aligned}\mathcal{L}(X_i) &= \prod^{N} \mathcal{P}(X_i \mid M_i(A,\Gamma)) \\ &= \mathcal{P}(15 \mid 10.7)\mathcal{P}(520 \mid 508.9)\mathcal{P}(74 \mid 75.5) \\ &= 0.116\end{aligned}$$

- Finding the maximum likelihood means finding the set of model parameters that maximize the likelihood function

Maximum Likelihood

If the hypothesized θ is close to the true value, then we expect a high probability to get data like that which we actually found.



χ^2–Statistic

Definition: $\chi^2 = \sum_i (D_i - M_i)^2 \; / \; M_i$

The χ^2 statistics is **minimized** in the fitting the data, varying the model parameters until the best-fit model parameters are found for the minimum value of the χ^2 statistic

Degrees-of-freedom = **k-1- N**

N – number of parameters
K – number of spectral bins

“Versions” of the χ^2 Statistic in Sherpa

The version of χ^2 derived above is called “data variance” χ^2 because of the presence of *D* in the denominator. Generally, the χ^2 statistic is written as:

$$\chi^2 \equiv \sum_i^N \frac{(D_i - M_i)^2}{\sigma_i^2} \quad ,$$

where σ_i^2 represents the (unknown!) variance of the Poisson distribution from which D_i is sampled.

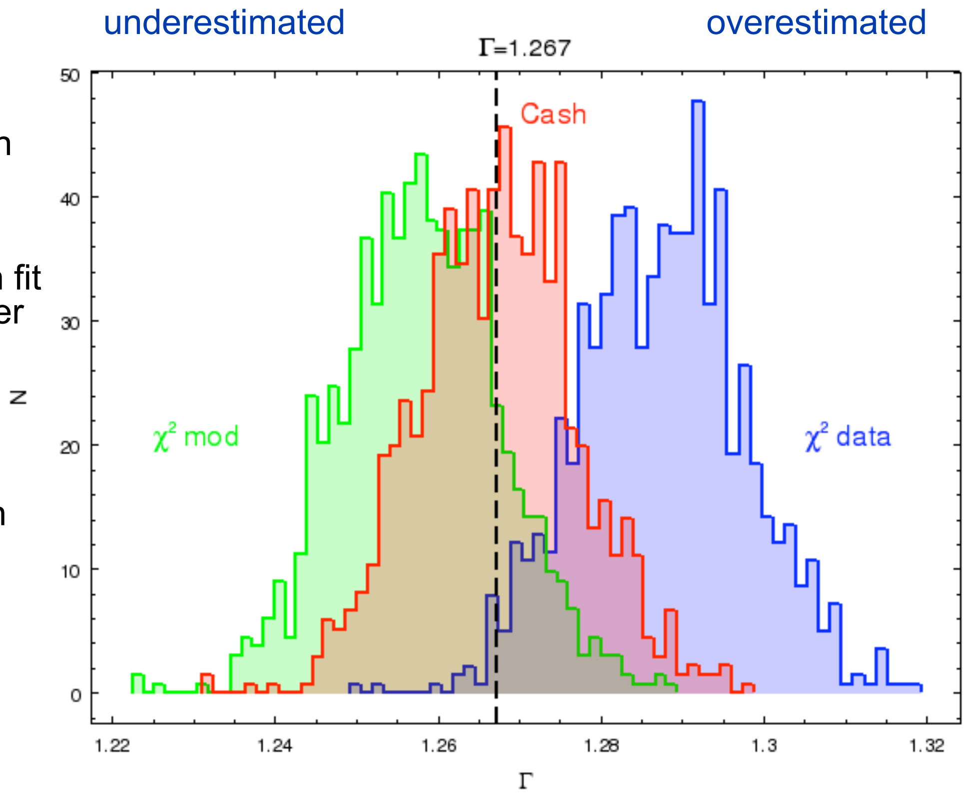
Sherpa name	**χ^2 Statistic**	σ_i^2
chi2datavar	Data Variance	D_i
chi2modvar	Model Variance	M_i
chi2gehrels	Gehrels	$[1+(D_i+0.75)^{1/2}]^2$
chi2constvar	“Parent”	$\frac{\sum_{i=1}^N D_i}{N}$
leastsq	Least Squares	1

Note that some X-ray data analysis routines may estimate σ_i during data reduction.
In PHA files, such estimates are recorded in the **STAT_ERR** column.

Statistics - Example of Bias

- The $\chi 2$ bias can affect the results of the X-ray spectral fitting
- Simulate Chandra spectrum given RMF/ARF and the Poisson noise - using fake_pha().
- The resulting simulated X-ray spectrum contains the model predicted counts with the Poisson noise. This spectrum is then fit with the absorbed power law model to get the best fit parameter value.
- Simulated 1000 spectra and fit each of them using different statistics: chi2datavar, chi2modvar and Cash.
- Plot the distribution of the photon index in the simulations with Γ=1.267.

Very High S/N data!



Fitting: Search in the Parameter Space

```
sherpa-28> fit()
Dataset              = 1
Method               = levmar
Statistic            = chi2datavar
Initial fit statistic = 644.136
Final fit statistic   = 632.106 at function evaluation 13
Data points          = 460
Degrees of freedom     = 457
Probability [Q-value] = 9.71144e-08
Reduced statistic      = 1.38316
Change in statistic    = 12.0305
   zabs1.nh       0.0960949
   p1.gamma       1.29086
   p1.ampl        0.000707365
```

```
sherpa-29> print get_fit_results()
datasets   = (1,)
methodname = levmar
statname   = chi2datavar
succeeded  = True
parnames   = ('zabs1.nh', 'p1.gamma', 'p1.ampl')
parvals    = (0.0960948525609, 1.29085977295, 0.000707365006941)
covarerr   = None
statval    = 632.10587995
istatval   = 644.136341045
dstatval   = 12.0304610958
numpoints  = 460
dof        = 457
qval       = 9.71144259004e-08
rstat      = 1.38316385109
message    = both actual and predicted relative reductions in the sum of
squares are at most
ftol=1.19209e-07
nfev       = 13
```

Fitting: Sherpa Optimization Methods

- Optimization - a minimization of a function:

> “A general function f(x) may have many isolated local minima, non-isolated minimum hypersurfaces, or even more complicated topologies. No finite minimization routine can guarantee to locate the unique, global, minimum of f(x) without being fed intimate knowledge about the function by the user.”

- Therefore:

1. Never accept the result using a single optimization run; always test the minimum using a different method.
2. Check that the result of the minimization does not have parameter values at the edges of the parameter space. If this happens, then the fit must be disregarded since the minimum lies outside the space that has been searched, or the minimization missed the minimum.
3. Get a feel for the range of values of the fit statistic, and the stability of the solution, by starting the minimization from several different parameter values.
4. Always check that the minimum "looks right" using a plotting tool.

Fitting: Optimization Methods in Sherpa

- “Single - shot” routines: Simplex and Levenberg-Marquardt

  start from a guessed set of parameters, and then try to improve the parameters in a continuous fashion:

  - Very Quick
  - Depend critically on the initial parameter values
  - Investigate a local behaviour of the statistics near the guessed parameters, and then make another guess at the best direction and distance to move to find a better minimum.
  - Continue until all directions result in increase of the statistics or a number of steps has been reached

- “Scatter-shot” routines: Monte Carlo

  try to look at parameters over the entire permitted parameter space to see if there are better minima than near the starting guessed set of parameters.

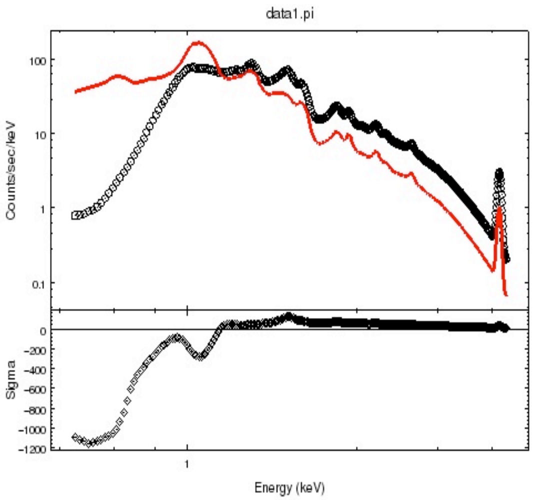
Optimization Methods: Comparison

Example: Spectral Fit with 3 methods

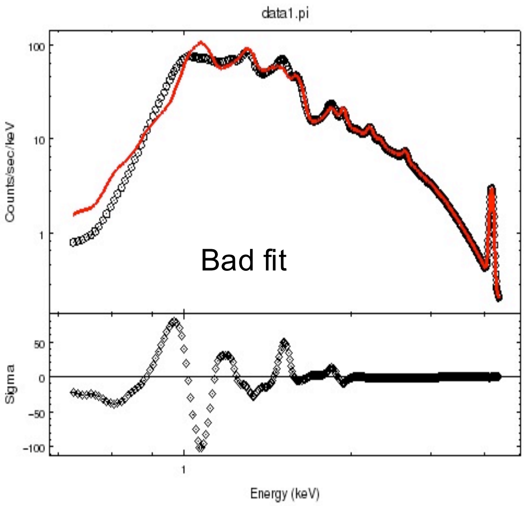
Data: high S/N simulated ACIS-S spectrum of the two temperature plasma
Model: photoelectric absorption plus two MEKAL components (correlated!)

Method	Number of Iterations	Final Statistics
Levmar	31	1.55e5
Neldermead	1494	0.0542
Moncar	13045	0.0542

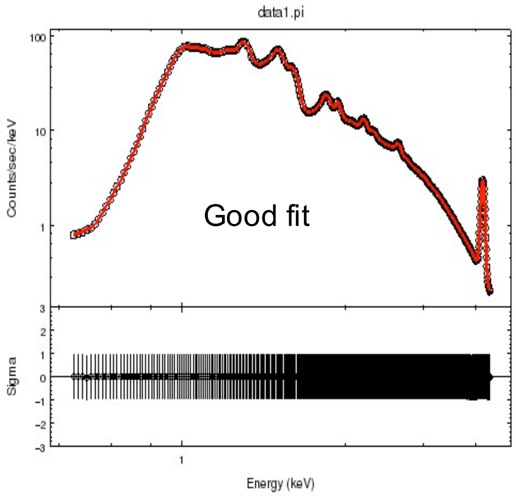
Start fit from the same initial parameters
Figures and Table compares the efficiency and final results



Data and Model with initial parameters



Levmar fit



Nelder-Mead and Moncar fit

Optimization Methods: Probing Parameter Space

Statistics vs iteration

Temperature

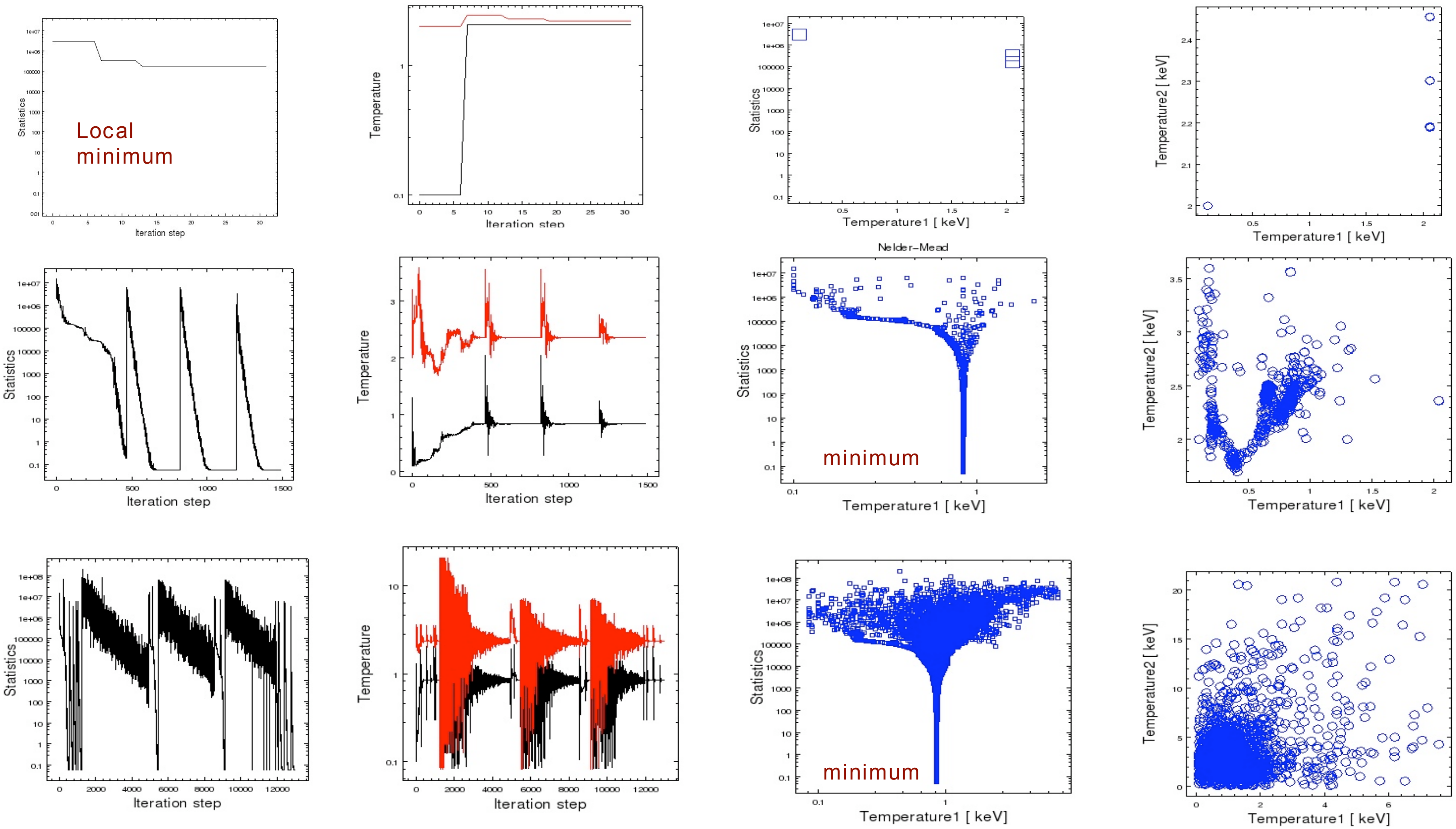
Statistics vs. Temperature

2D slice of Parameter Space probed by each method

levmar

neldermead

moncar



Optimization Methods: Summary

- "levmar" method is fast, very sensitive to initial parameters, performs well for simple models, e.g. power law, one temperature models, but fails to converge in complex models.
- "neldermead" and "moncar" are both very robust and converge to global minimum in complex model case.
- "neldermead" is more efficient than "moncar", but "moncar" probes larger part of the parameter space
- "moncar" or "neldermead" should be used in complex models with correlated parameters

Final Analysis Steps

- How well are the model parameters constrained by the data?
- Is this a correct model?
- Is this the only model?
- Do we have definite results?
- What have we learned, discovered?
- How our source compares to the other sources?
- Do we need to obtain a new observation?

Confidence Limits

Essential issue = after the bets-fit parameters are found estimate the confidence limits for them. The region of confidence is given by (Avni 1976):

$$\chi^2_\alpha = \chi^2_{min} + \Delta(\nu,\alpha)$$

ν - degrees of freedom
α - level
χ^2_{min} - minimum

Δ depends only on the number of parameters involved not on goodness of fit

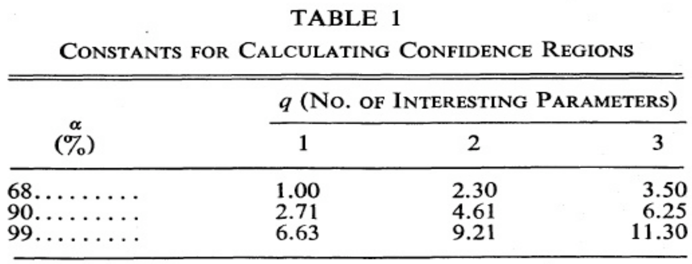
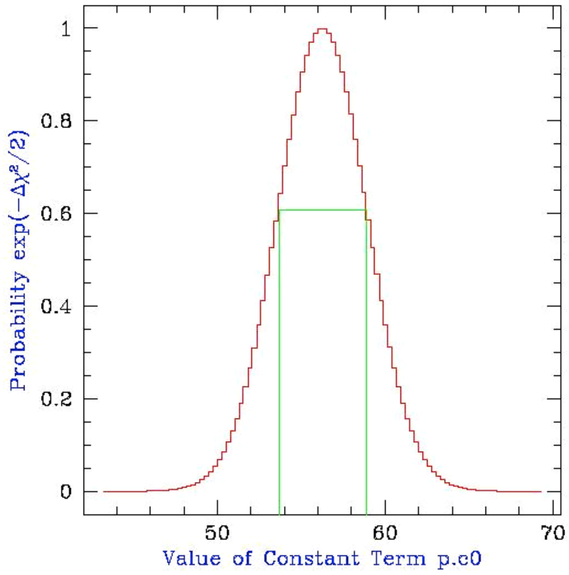


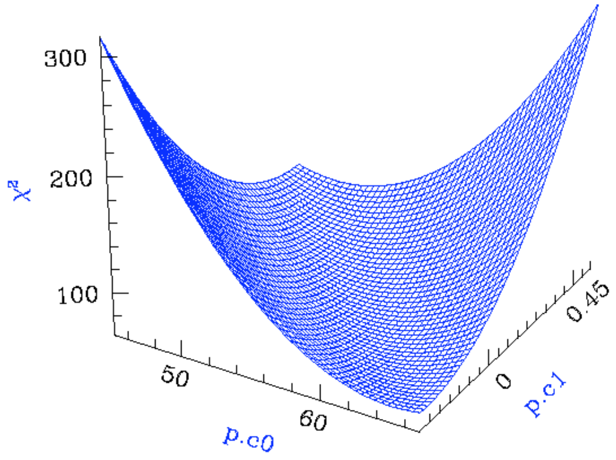
TABLE 1

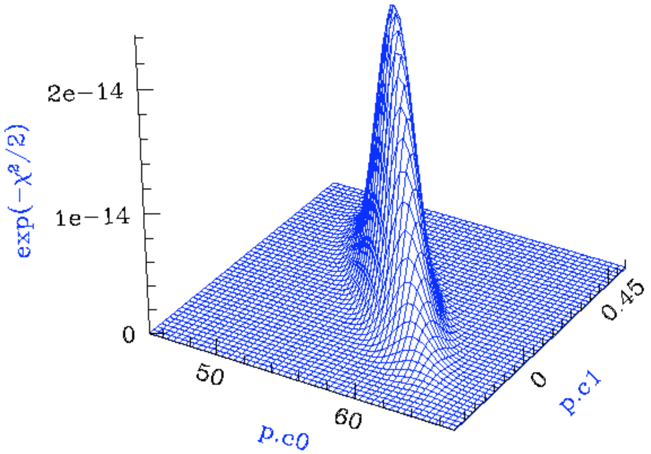
CONSTANTS FOR CALCULATING CONFIDENCE REGIONS

α (%)	q (No. of Interesting Parameters) 1	2	3
68.........	1.00	2.30	3.50
90.........	2.71	4.61	6.25
99.........	6.63	9.21	11.30



Calculating Confidence Limits means Exploring the Parameter Space - Statistical Surface

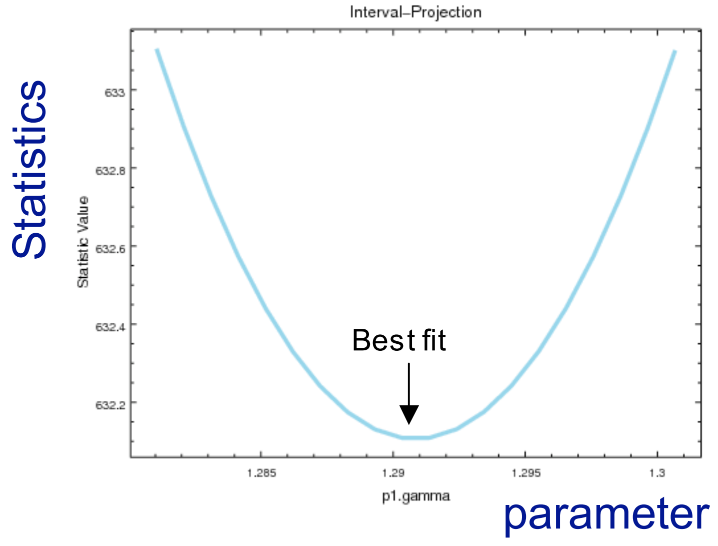




Example of a "well-behaved" statistical surface in parameter space, viewed as a multi-dimensional paraboloid (χ^2, *top*), and as a multi-dimensional Gaussian ($\exp(-\chi^2/2) \approx \mathcal{L}$, *bottom*).

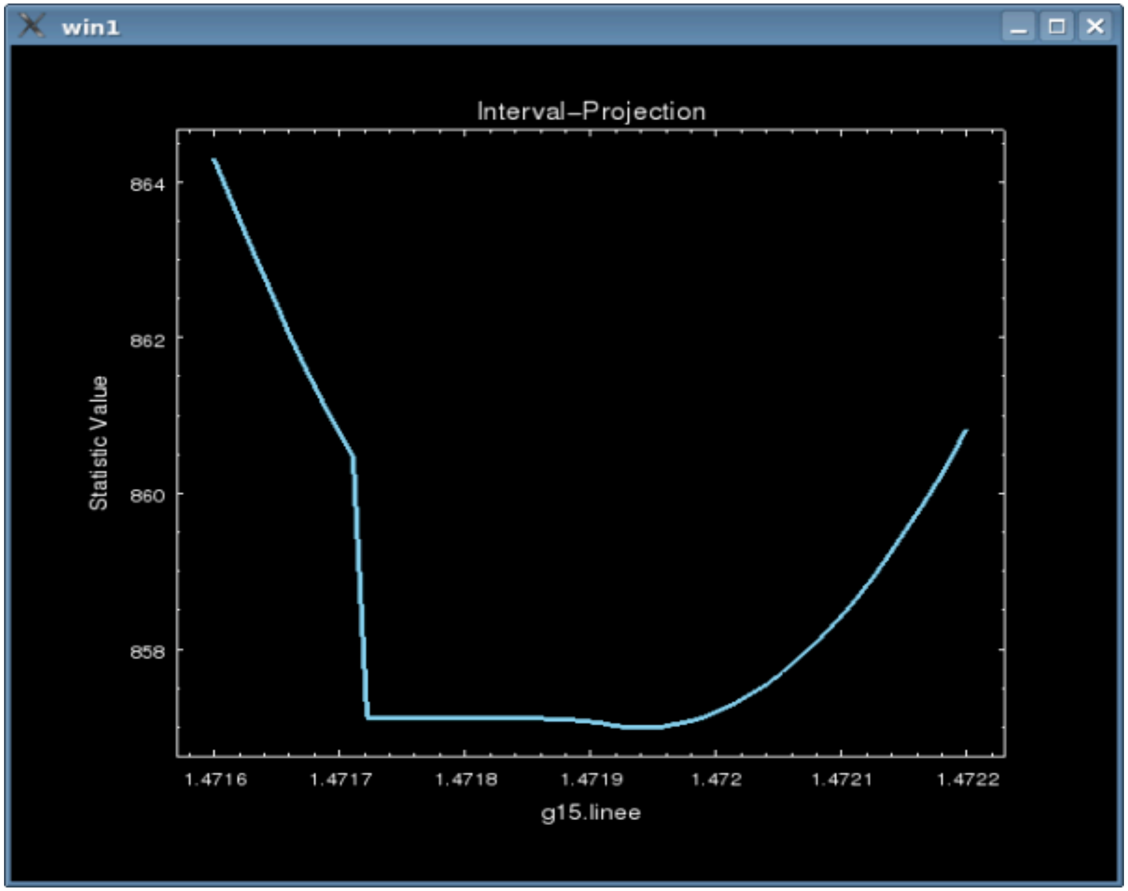
Confidence Intervals

```
sherpa-40> covariance()
Dataset             = 1
Confidence Method   = covariance
Fitting Method      = neldermead
Statistic           = chi2datavar
covariance 1-sigma (68.2689%) bounds:
   Param            Best-Fit  Lower Bound  Upper Bound
   -----            --------  -----------  -----------
   abs1.nH            1.1015  -0.00153623   0.00153623
   mek1.kT          0.841024  -0.00115618   0.00115618
   mek1.norm        0.699764  -0.00395776   0.00395776
   mek2.kT           2.35844  -0.00371253   0.00371253
   mek2.norm         1.03725  -0.00172503   0.00172503
sherpa-42> conf()
mek1.kT lower bound:    -0.00113811
mek1.kT upper bound:    0.0011439
mek2.kT lower bound:    -0.00365452
mek2.kT upper bound:    0.00364805
mek1.norm lower bound:  -0.00377224
mek2.norm lower bound:  -0.00164417
mek2.norm upper bound:  0.00164816
abs1.nH lower bound:    -0.00147622
mek1.norm upper bound:  0.00376011
abs1.nH upper bound:    0.00147268
Dataset             = 1
Confidence Method   = confidence
Fitting Method      = neldermead
Statistic           = chi2datavar
confidence 1-sigma (68.2689%) bounds:
   Param            Best-Fit  Lower Bound  Upper Bound
   -----            --------  -----------  -----------
   abs1.nH            1.1015  -0.00147622   0.00147268
   mek1.kT          0.841024  -0.00113811    0.0011439
   mek1.norm        0.699764  -0.00377224   0.00376011
   mek2.kT           2.35844  -0.00365452   0.00364805
   mek2.norm         1.03725  -0.00164417   0.00164816
```



```
sherpa-42 > print get_conf_results()
----------> print(get_conf_results())
datasets    = (1,)
methodname = confidence
fitname     = neldermead
statname    = chi2datavar
sigma       = 1
percent     = 68.2689492137
parnames    = ('abs1.nH', 'mek1.kT', 'mek1.norm', 'mek2.kT', 'mek2.norm')
parvals     = (1.1015003421601872, 0.84102381214069499, 0.69976355976410642,
2.3584395600380756, 1.0372453037692799)
parmins     = (-0.0014762187156509565, -0.0011381111192153346, -0.0037722356859711814, -
0.0036545192286010497, -0.0016441656050858455)
parmaxes    = (0.0014726797455 47989, 0.0011439029752089436, 0.0037601110158367312,
0.0036480458191339 16, 0.0016481622297 10648)
nfits       = 103
```

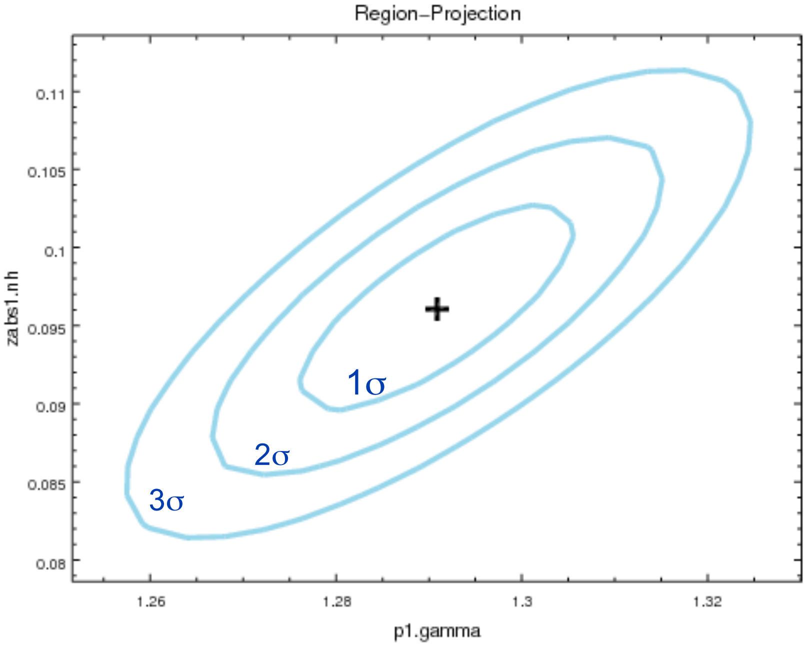
Not well-behaved Surface



Non-Gaussian Shape

Confidence Regions

```
sherpa-61> reg_proj(p1.gamma,zabs1.nh,nloop=[20,20])
sherpa-62> print get_reg_proj()
min     = [ 1.2516146   0.07861824]
max     = [ 1.33010494  0.11357147]
nloop   = [20, 20]
fac     = 4
delv    = None
log     = [False False]
sigma   = (1, 2, 3)
parval0 = 1.29085977295
parval1 = 0.0960948525609
levels  = [ 634.40162888  638.28595426  643.93503803]
```



,

Distributions of Flux and Parameters

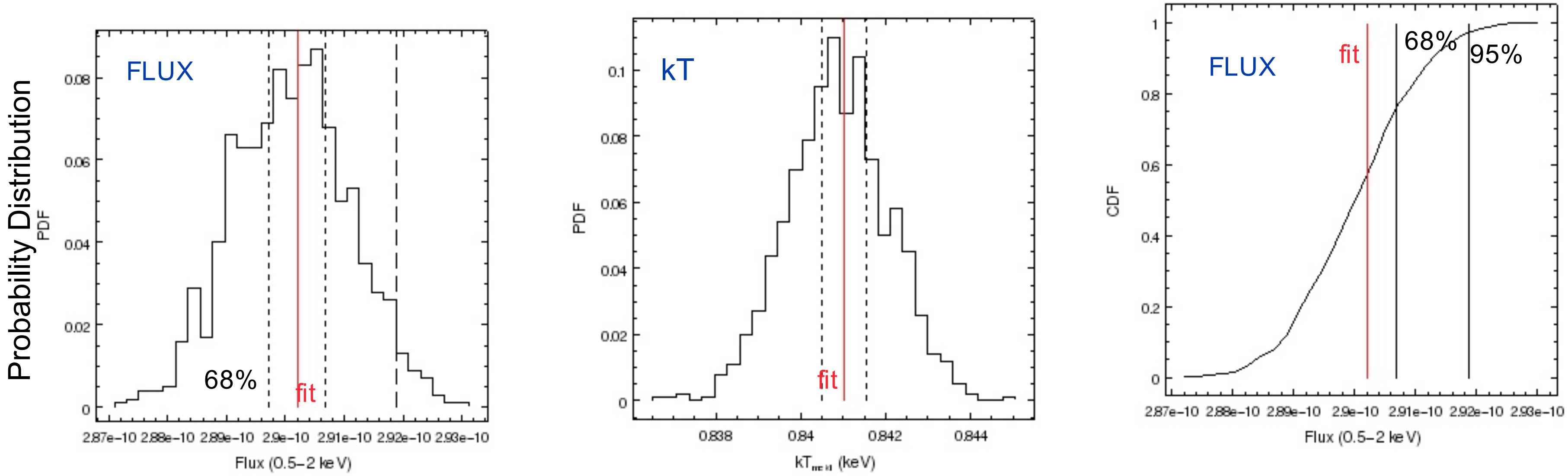
Function: sample_energy_flux

Monte Carlo Simulations of parameters assuming Gaussian distributions for all the parameters
Characterized by the covariance matrix, includes correlations between parameters.

```
sherpa-19> flux100=sample_energy_flux(0.5,2.,num=100)
sherpa-20> print flux100
---------> print(flux100)
[[  2.88873592e-10   1.10331438e+00   8.40356670e-01   6.97503733e-01
    2.35411369e+00   1.03580042e+00]
 [  2.90279483e-10   1.10243140e+00   8.41174148e-01   7.01009661e-01
 sherpa-26> plot_energy_flux(0.5,2,num=1000)
```

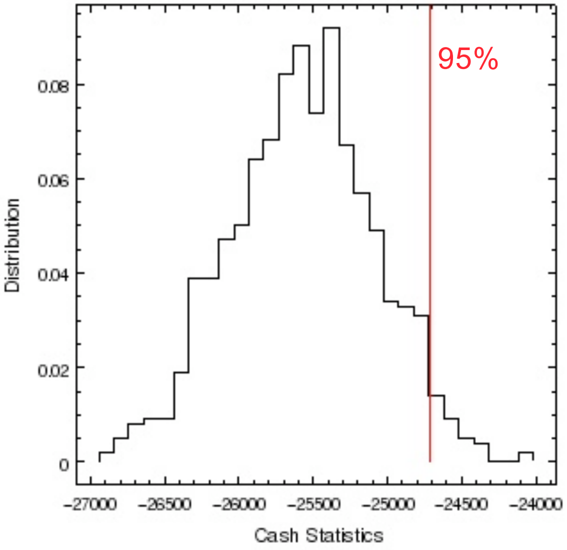
* Characterize distributions: plot PDF and CDF and obtain Quatiles of 68% and 95%

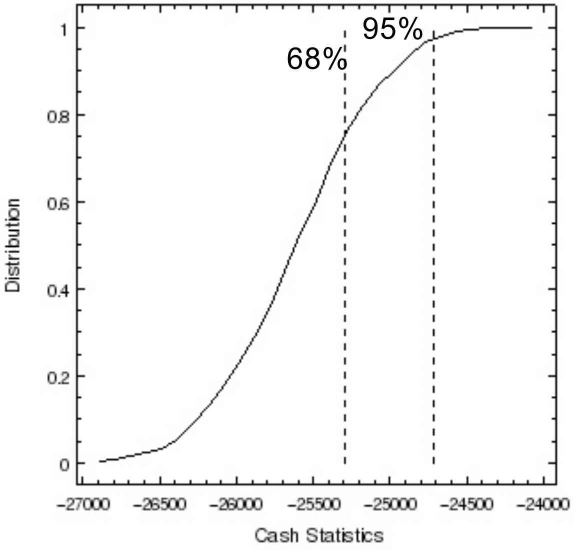
```
sherpa-30> fluxes=numpy.sort(flux1000[:,0])
sherpa-31> a95=fluxes(0.95*len(flux1000[:,0])-1)
sherpa-32> a68=fluxes(0.68*len(flux1000[:,0])-1)
```



Goodness of Fit

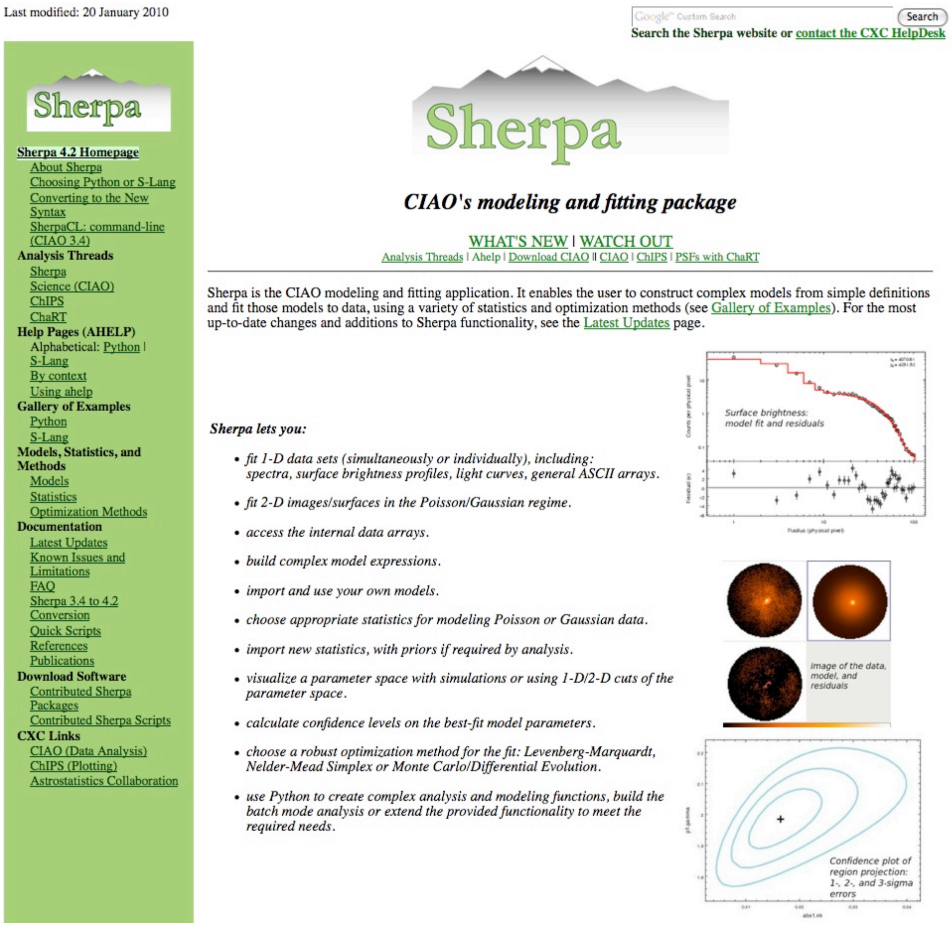
Need simulations for the fit with Cash likelihood statististics to obtain the shape of the distribution.





Learn more on Sherpa Web Pages

http://cxc.harvard.edu/sherpa/index.html



Last modified: 20 January 2010

Search the Sherpa website or **contact the CXC HelpDesk**

Sherpa 4.2 Homepage
- About Sherpa
- Choosing Python or S-Lang
- Converting to the New Syntax
- SherpaCL: command-line (CIAO 3.4)

Analysis Threads
- Sherpa
- Science (CIAO)
- ChIPS
- ChaRT

Help Pages (AHELP)
- Alphabetical: Python | S-Lang
- By context
- Using ahelp

Gallery of Examples
- Python
- S-Lang

Models, Statistics, and Methods
- Models
- Statistics
- Optimization Methods

Documentation
- Latest Updates
- Known Issues and Limitations
- FAQ
- Sherpa 3.4 to 4.2 Conversion
- Quick Scripts
- References
- Publications

Download Software
- Contributed Sherpa Packages
- Contributed Sherpa Scripts

CXC Links
- CIAO (Data Analysis)
- ChIPS (Plotting)
- Astrostatistics Collaboration

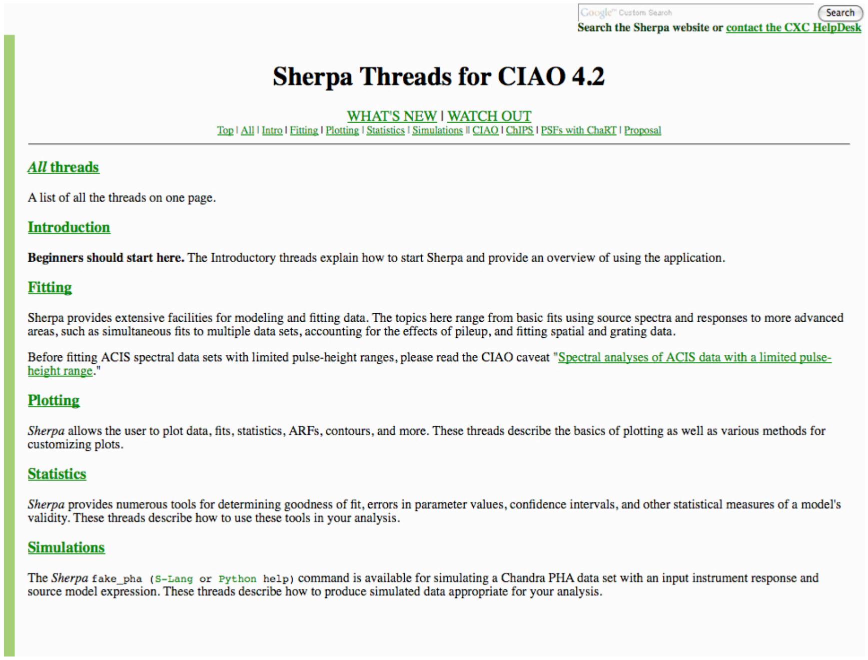
CIAO's modeling and fitting package

WHAT'S NEW | WATCH OUT
Analysis Threads | Ahelp | Download CIAO || CIAO | ChIPS | PSFs with ChaRT

Sherpa is the CIAO modeling and fitting application. It enables the user to construct complex models from simple definitions and fit those models to data, using a variety of statistics and optimization methods (see Gallery of Examples). For the most up-to-date changes and additions to Sherpa functionality, see the Latest Updates page.

Sherpa lets you:

- *fit 1-D data sets (simultaneously or individually), including: spectra, surface brightness profiles, light curves, general ASCII arrays.*
- *fit 2-D images/surfaces in the Poisson/Gaussian regime.*
- *access the internal data arrays.*
- *build complex model expressions.*
- *import and use your own models.*
- *choose appropriate statistics for modeling Poisson or Gaussian data.*
- *import new statistics, with priors if required by analysis.*
- *visualize a parameter space with simulations or using 1-D/2-D cuts of the parameter space.*
- *calculate confidence levels on the best-fit model parameters.*
- *choose a robust optimization method for the fit: Levenberg-Marquardt, Nelder-Mead Simplex or Monte Carlo/Differential Evolution.*
- *use Python to create complex analysis and modeling functions, build the batch mode analysis or extend the provided functionality to meet the required needs.*



Search the Sherpa website or **contact the CXC HelpDesk**

Sherpa Threads for CIAO 4.2

WHAT'S NEW | WATCH OUT
Top | All | Intro | Fitting | Plotting | Statistics | Simulations || CIAO | ChIPS | PSFs with ChaRT | Proposal

All threads

A list of all the threads on one page.

Introduction

Beginners should start here. The Introductory threads explain how to start Sherpa and provide an overview of using the application.

Fitting

Sherpa provides extensive facilities for modeling and fitting data. The topics here range from basic fits using source spectra and responses to more advanced areas, such as simultaneous fits to multiple data sets, accounting for the effects of pileup, and fitting spatial and grating data.

Before fitting ACIS spectral data sets with limited pulse-height ranges, please read the CIAO caveat "Spectral analyses of ACIS data with a limited pulse-height range."

Plotting

Sherpa allows the user to plot data, fits, statistics, ARFs, contours, and more. These threads describe the basics of plotting as well as various methods for customizing plots.

Statistics

Sherpa provides numerous tools for determining goodness of fit, errors in parameter values, confidence intervals, and other statistical measures of a model's validity. These threads describe how to use these tools in your analysis.

Simulations

The *Sherpa* `fake_pha (S-Lang or Python help)` command is available for simulating a Chandra PHA data set with an input instrument response and source model expression. These threads describe how to produce simulated data appropriate for your analysis.

Freeman, P., Doe, S., & Siemiginowska, A.\ 2001, SPIE 4477, 76

Doe, S., et al. 2007, Astronomical Data Analysis Software and Systems XVI, 376, 543

Refsdal et al. 2009 - Sherpa: 1D/2D modeling and fitting in Python in Proceedings of the 8th Python in Science conference (SciPy 2009), G Varoquaux, S van der Walt, J Millman (Eds.), pp. 51-57

A Simple Problem

Fit Chandra 2D Image data in Sherpa using Command Line Interface in Python

- Read the data
- Choose statistics and optimization method
- Define the model
- Minimize to find the best fit parameters for the model
- Evaluate the best fit - display model, residuals, calculate uncertainties

A Simple Problem

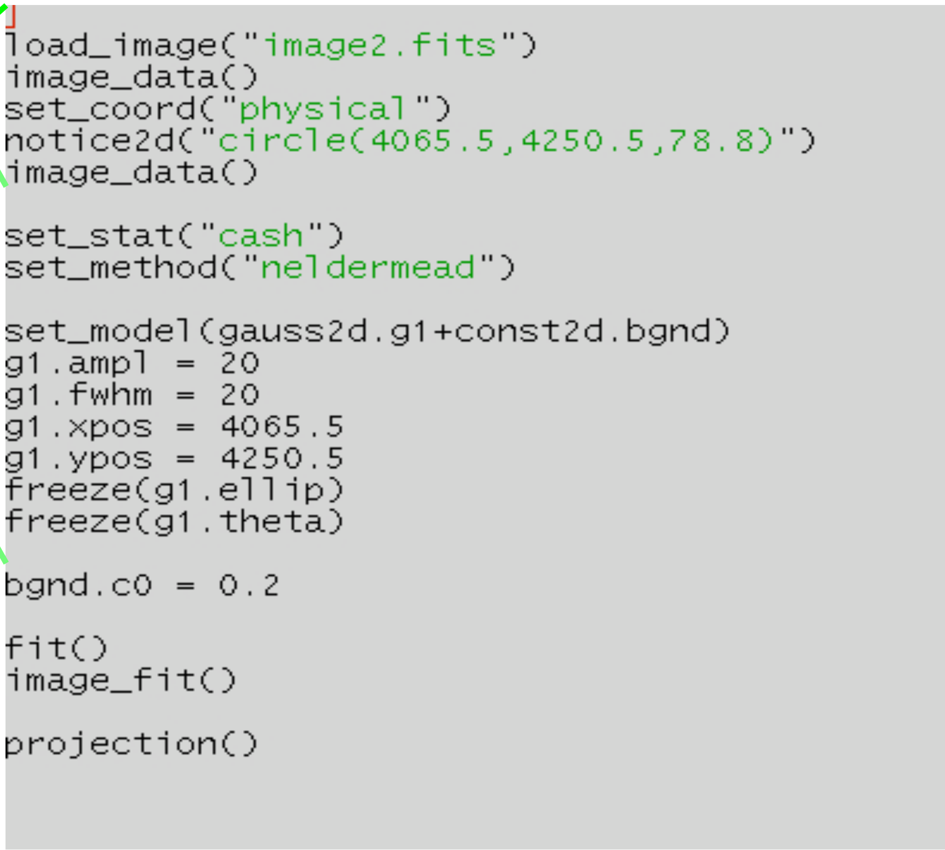
Read Image data and Display in ds9

Set Statistics and Optimization Method

Define Model and Set Model parameters

Fit, Display
Get Confidence Range

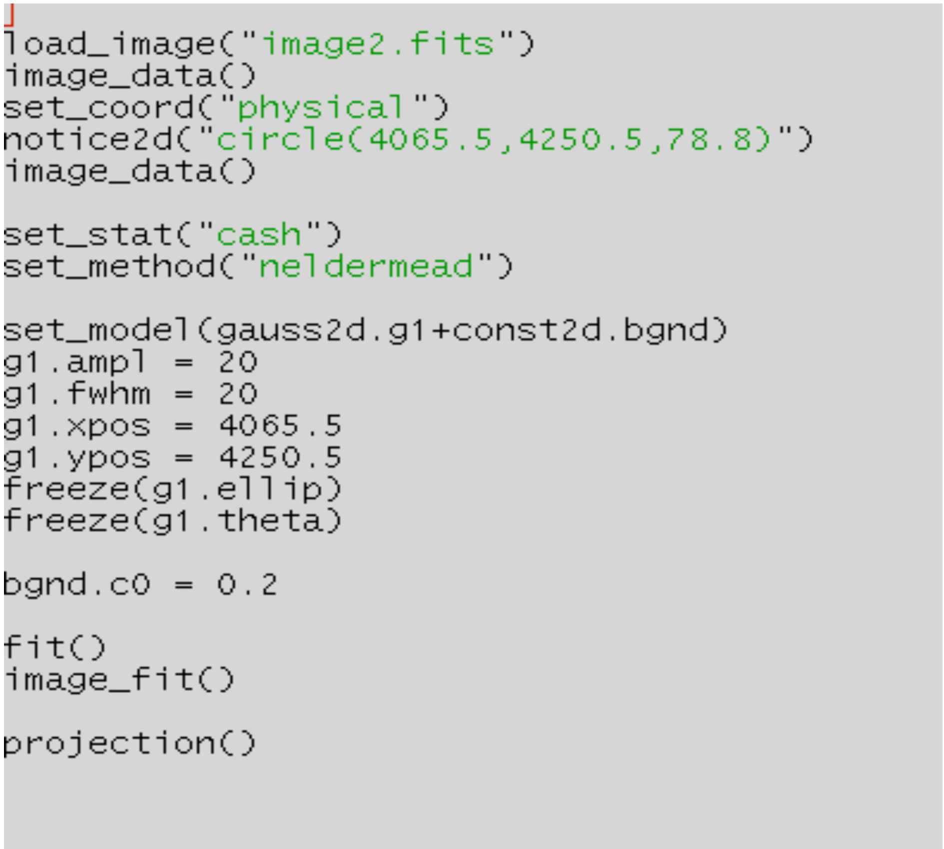
List of Sherpa Commands



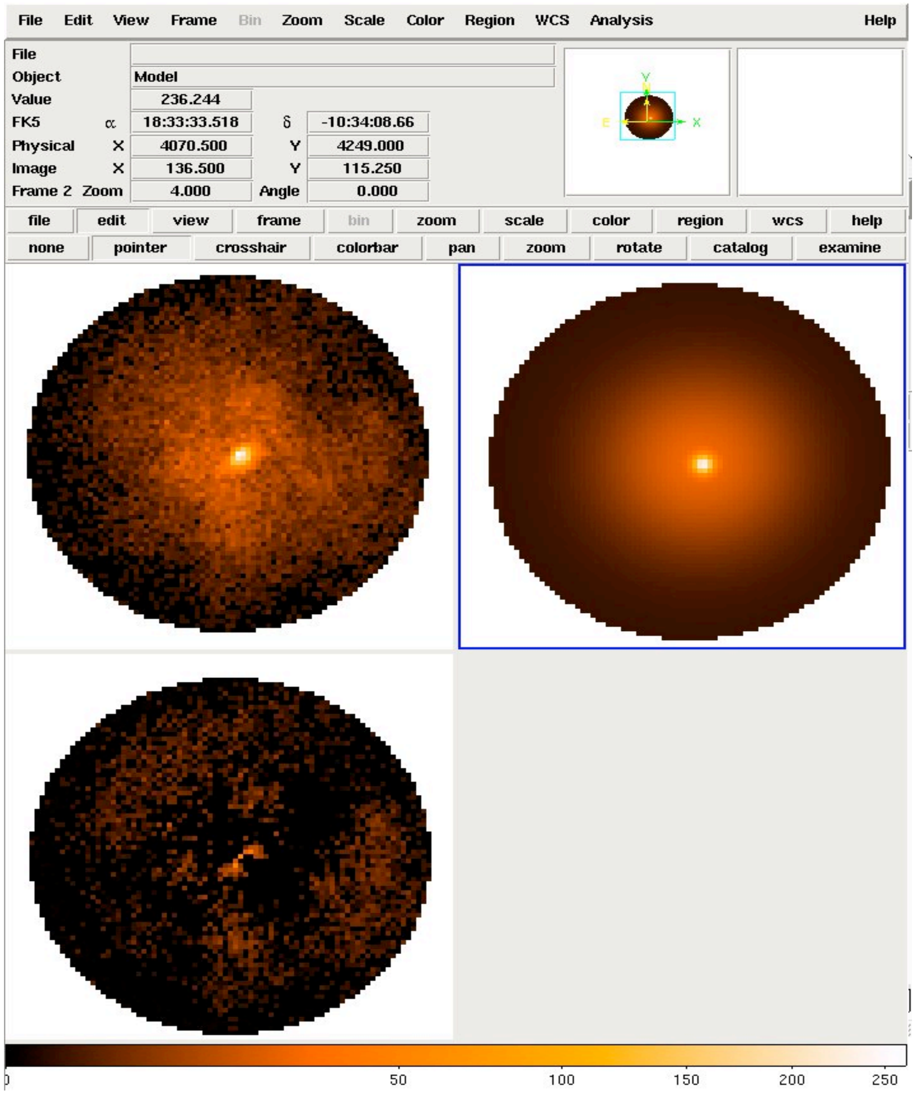
```
load_image("image2.fits")
image_data()
set_coord("physical")
notice2d("circle(4065.5,4250.5,78.8)")
image_data()

set_stat("cash")
set_method("neldermead")

set_model(gauss2d.g1+const2d.bgnd)
g1.ampl  = 20
g1.fwhm  = 20
g1.xpos  = 4065.5
g1.ypos  = 4250.5
freeze(g1.ellip)
freeze(g1.theta)

bgnd.c0 = 0.2

fit()
image_fit()

projection()
```

A Simple Problem

List of Sherpa Commands



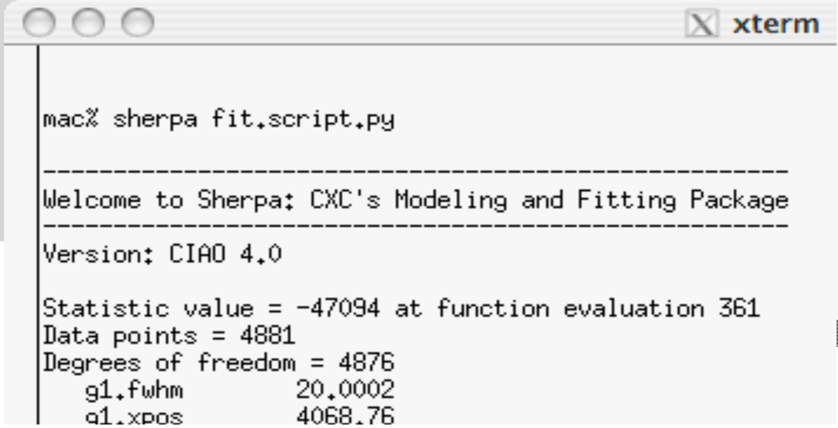
```
load_image("image2.fits")
image_data()
set_coord("physical")
notice2d("circle(4065.5,4250.5,78.8)")
image_data()

set_stat("cash")
set_method("neldermead")

set_model(gauss2d.g1+const2d.bgnd)
g1.ampl  = 20
g1.fwhm  = 20
g1.xpos  = 4065.5
g1.ypos  = 4250.5
freeze(g1.ellip)
freeze(g1.theta)

bgnd.c0  = 0.2

fit()
image_fit()

projection()
```



File
Object: Model
Value: 236.244
FK5 α 18:33:33.518 δ -10:34:08.66
Physical X 4070.500 Y 4249.000
Image X 136.500 Y 115.250
Frame 2 Zoom 4.000 Angle 0.000

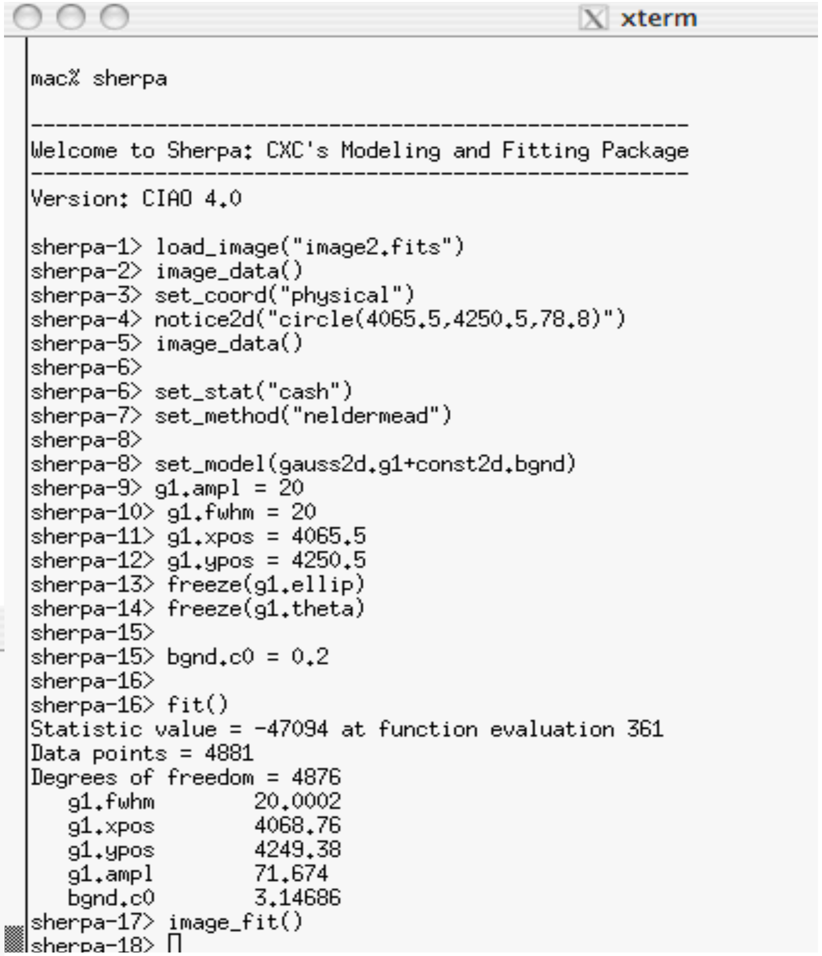
List of Sherpa Commands

```
load_image("image2.fits")
image_data()
set_coord("physical")
notice2d("circle(4065.5,4250.5,78.8)")
image_data()

set_stat("cash")
set_method("neldermead")

set_model(gauss2d.g1+const2d.bgnd)
g1.ampl = 20
g1.fwhm = 20
g1.xpos = 4065.5
g1.ypos = 4250.5
freeze(g1.ellip)
freeze(g1.theta)

bgnd.c0 = 0.2

fit()
image_fit()

projection()
```



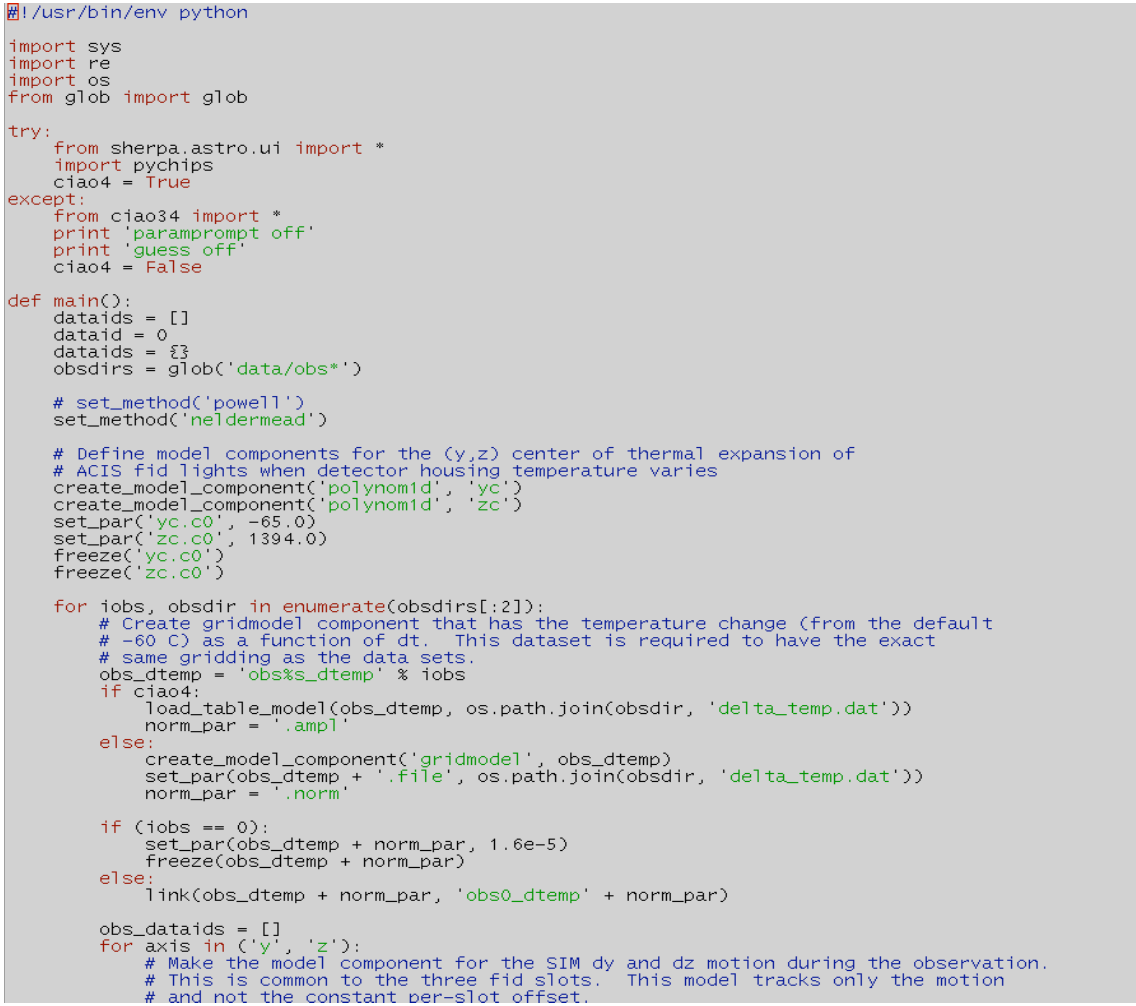
```
mac% sherpa fit.script.py

-----------------------------------------------------
Welcome to Sherpa: CXC's Modeling and Fitting Package
-----------------------------------------------------
Version: CIAO 4.0

Statistic value = -47094 at function evaluation 361
Data points = 4881
Degrees of freedom = 4876
   g1.fwhm         20.0002
   g1.xpos         4068.76
```

Command Line View



```
mac% sherpa

-----------------------------------------------------
Welcome to Sherpa: CXC's Modeling and Fitting Package
-----------------------------------------------------
Version: CIAO 4.0

sherpa-1> load_image("image2.fits")
sherpa-2> image_data()
sherpa-3> set_coord("physical")
sherpa-4> notice2d("circle(4065.5,4250.5,78.8)")
sherpa-5> image_data()
sherpa-6>
sherpa-6> set_stat("cash")
sherpa-7> set_method("neldermead")
sherpa-8>
sherpa-8> set_model(gauss2d.g1+const2d.bgnd)
sherpa-9> g1.ampl = 20
sherpa-10> g1.fwhm = 20
sherpa-11> g1.xpos = 4065.5
sherpa-12> g1.ypos = 4250.5
sherpa-13> freeze(g1.ellip)
sherpa-14> freeze(g1.theta)
sherpa-15>
sherpa-15> bgnd.c0 = 0.2
sherpa-16>
sherpa-16> fit()
Statistic value = -47094 at function evaluation 361
Data points = 4881
Degrees of freedom = 4876
   g1.fwhm         20.0002
   g1.xpos         4068.76
   g1.ypos         4249.38
   g1.ampl         71.674
   bgnd.c0         3.14686
sherpa-17> image_fit()
sherpa-18> ▯
```

Sherpa Scripts



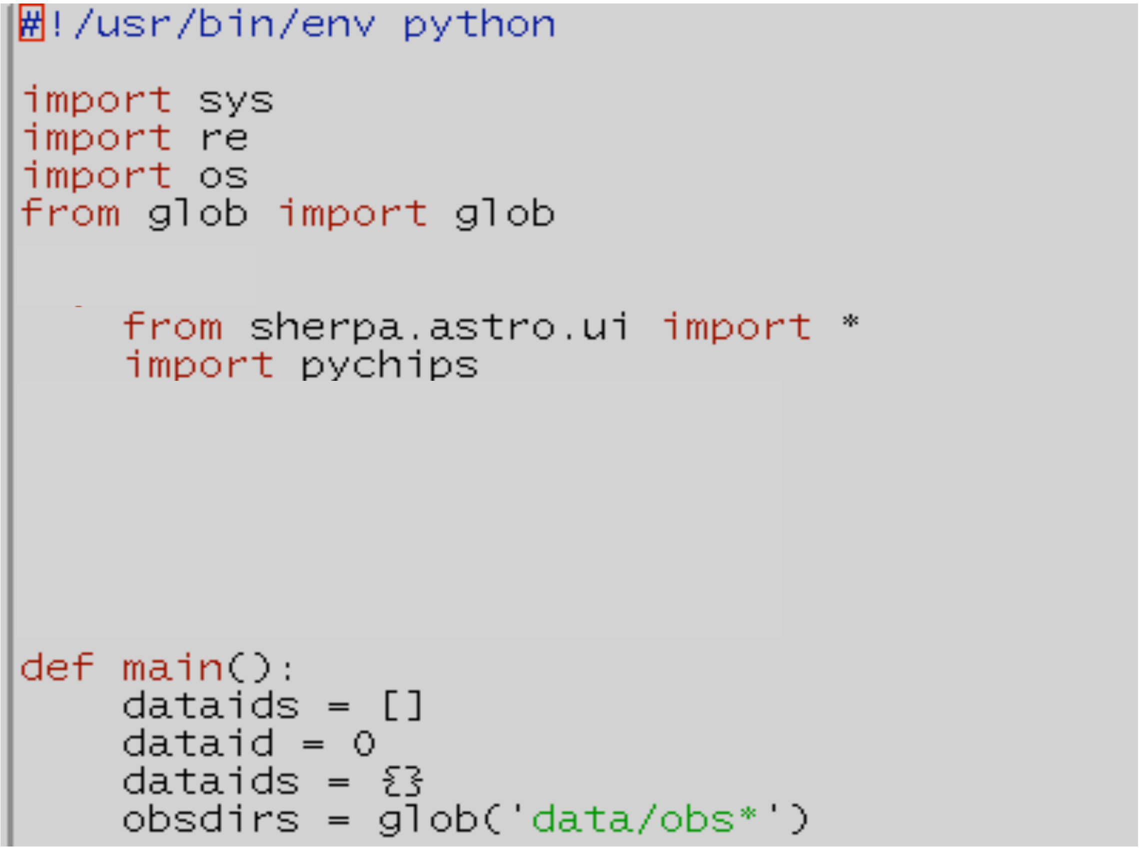
```
#!/usr/bin/env python

import sys
import re
import os
from glob import glob

try:
    from sherpa.astro.ui import *
    import pychips
    ciao4 = True
except:
    from ciao34 import *
    print 'paramprompt off'
    print 'guess off'
    ciao4 = False

def main():
    dataids = []
    dataid = 0
    dataids = {}
    obsdirs = glob('data/obs*')

    # set_method('powell')
    set_method('neldermead')

    # Define model components for the (y,z) center of thermal expansion of
    # ACIS fid lights when detector housing temperature varies
    create_model_component('polynom1d', 'yc')
    create_model_component('polynom1d', 'zc')
    set_par('yc.c0', -65.0)
    set_par('zc.c0', 1394.0)
    freeze('yc.c0')
    freeze('zc.c0')

    for iobs, obsdir in enumerate(obsdirs[:2]):
        # Create gridmodel component that has the temperature change (from the default
        # -60 C) as a function of dt.  This dataset is required to have the exact
        # same gridding as the data sets.
        obs_dtemp = 'obs%s_dtemp' % iobs
        if ciao4:
            load_table_model(obs_dtemp, os.path.join(obsdir, 'delta_temp.dat'))
            norm_par = '.ampl'
        else:
            create_model_component('gridmodel', obs_dtemp)
            set_par(obs_dtemp + '.file', os.path.join(obsdir, 'delta_temp.dat'))
            norm_par = '.norm'

        if (iobs == 0):
            set_par(obs_dtemp + norm_par, 1.6e-5)
            freeze(obs_dtemp + norm_par)
        else:
            link(obs_dtemp + norm_par, 'obs0_dtemp' + norm_par)

        obs_dataids = []
        for axis in ('y', 'z'):
            # Make the model component for the SIM dy and dz motion during the observation.
            # This is common to the three fid slots.  This model tracks only the motion
            # and not the constant per-slot offset.
```

Setup Environment

Set the System

Import Sherpa and Chips

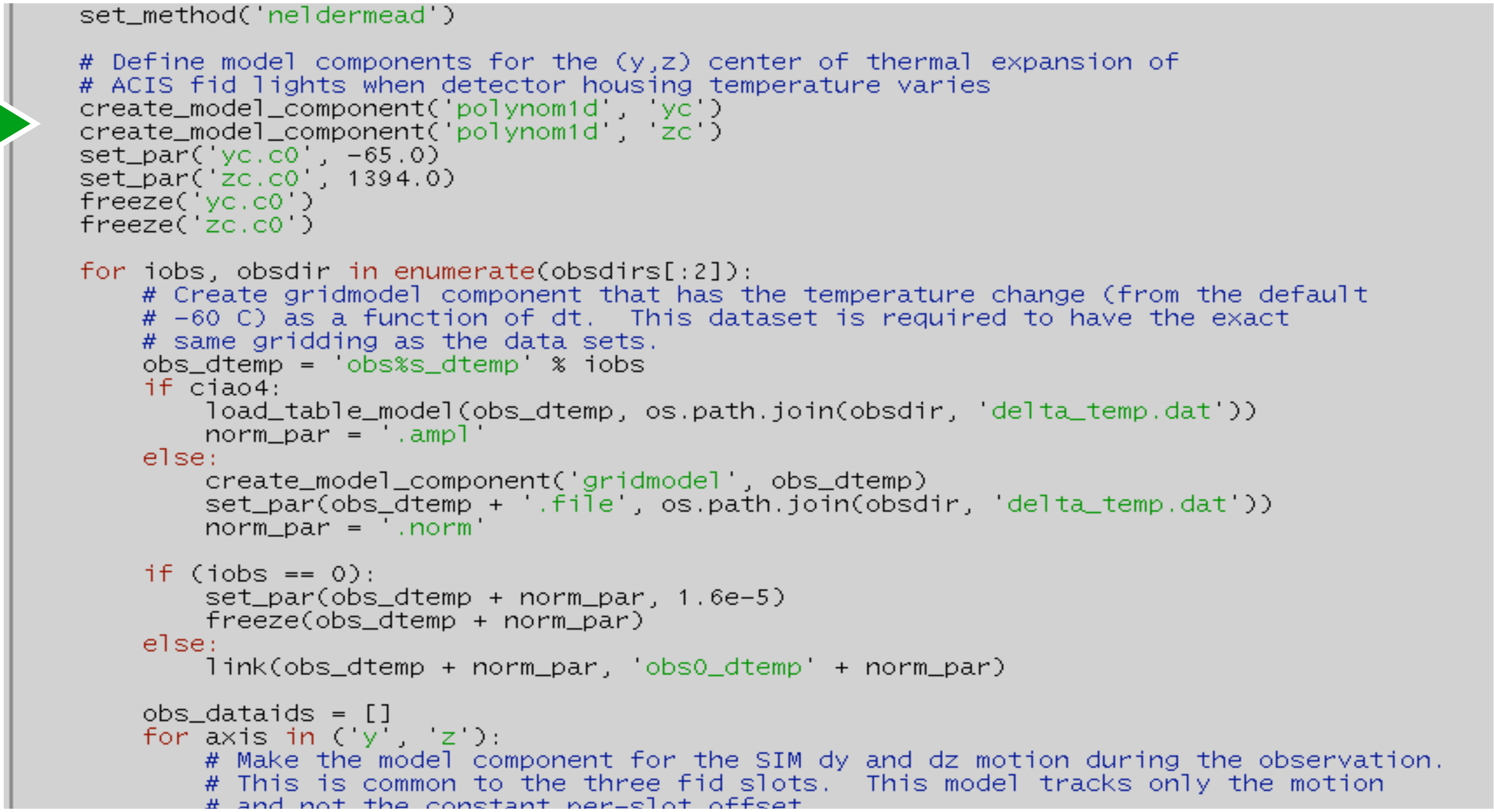
Define directories



```
#!/usr/bin/env python

import sys
import re
import os
from glob import glob


    from sherpa.astro.ui import *
    import pychips







def main():
    dataids = []
    dataid = 0
    dataids = {}
    obsdirs = glob('data/obs*')
```

Model Parameters

Loops



```
set_method('neldermead')

# Define model components for the (y,z) center of thermal expansion of
# ACIS fid lights when detector housing temperature varies
create_model_component('polynom1d', 'yc')
create_model_component('polynom1d', 'zc')
set_par('yc.c0', -65.0)
set_par('zc.c0', 1394.0)
freeze('yc.c0')
freeze('zc.c0')

for iobs, obsdir in enumerate(obsdirs[:2]):
    # Create gridmodel component that has the temperature change (from the default
    # -60 C) as a function of dt.  This dataset is required to have the exact
    # same gridding as the data sets.
    obs_dtemp = 'obs%s_dtemp' % iobs
    if ciao4:
        load_table_model(obs_dtemp, os.path.join(obsdir, 'delta_temp.dat'))
        norm_par = '.ampl'
    else:
        create_model_component('gridmodel', obs_dtemp)
        set_par(obs_dtemp + '.file', os.path.join(obsdir, 'delta_temp.dat'))
        norm_par = '.norm'

    if (iobs == 0):
        set_par(obs_dtemp + norm_par, 1.6e-5)
        freeze(obs_dtemp + norm_par)
    else:
        link(obs_dtemp + norm_par, 'obs0_dtemp' + norm_par)

    obs_dataids = []
    for axis in ('y', 'z'):
        # Make the model component for the SIM dy and dz motion during the observation.
        # This is common to the three fid slots.  This model tracks only the motion
        # and not the constant per-slot offset
```

A Complex Example

Fit Chandra and HST Spectra with Python script

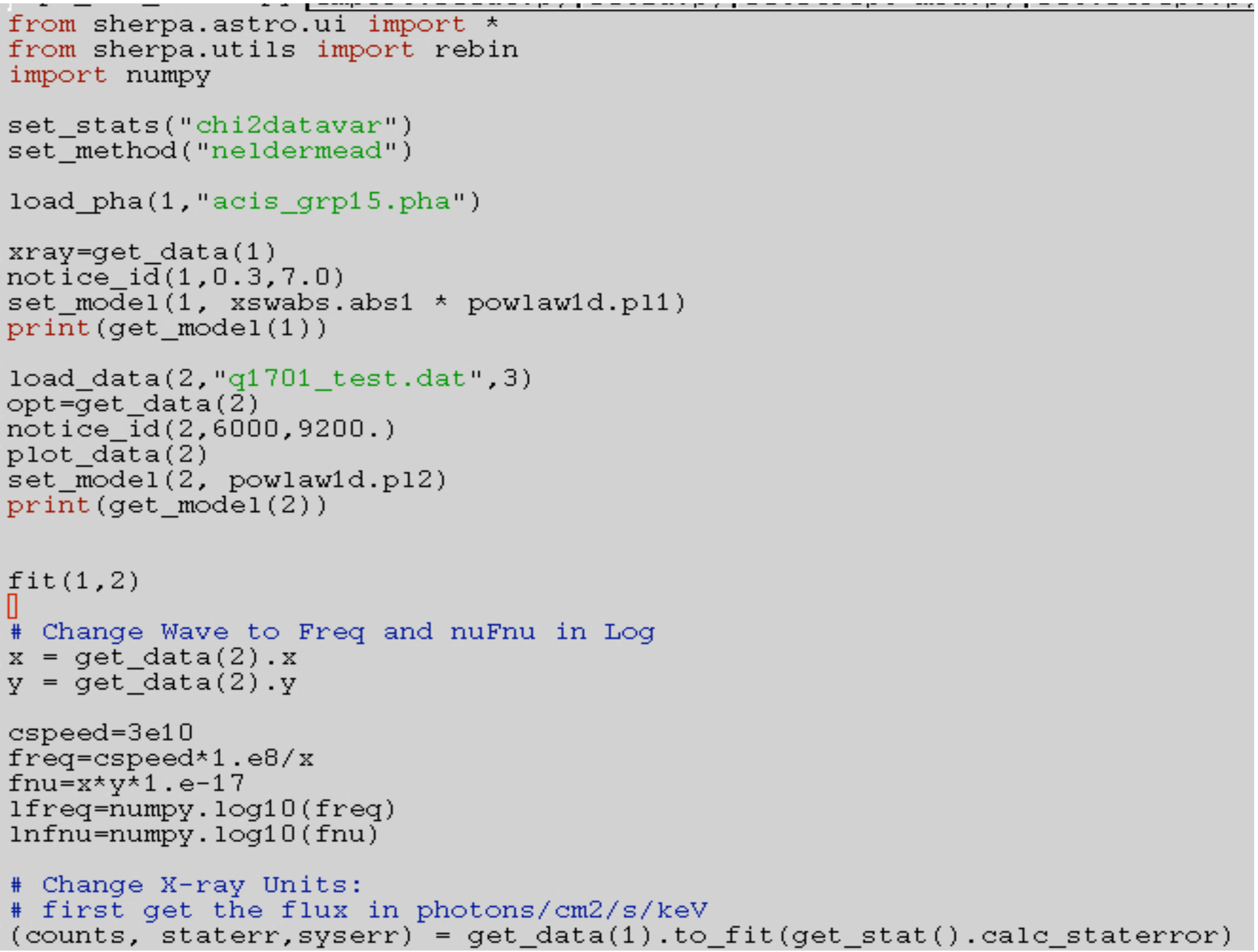
- Setup the environment
- Define functions
- Run script and save results in nice format.
- Evaluate results - do plots, check uncertainties, derive data and do analysis of the derived data.

Setup

X-ray spectra

Optical spectra

Units Conversion

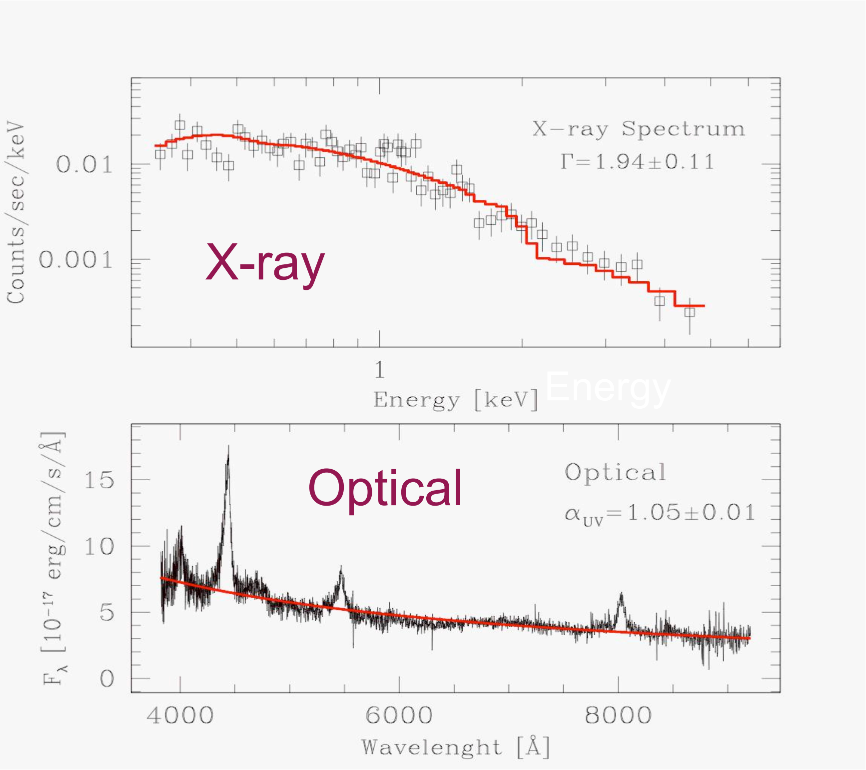


```
from sherpa.astro.ui import *
from sherpa.utils import rebin
import numpy

set_stats("chi2datavar")
set_method("neldermead")

load_pha(1,"acis_grp15.pha")

xray=get_data(1)
notice_id(1,0.3,7.0)
set_model(1, xswabs.abs1 * powlaw1d.pl1)
print(get_model(1))

load_data(2,"q1701_test.dat",3)
opt=get_data(2)
notice_id(2,6000,9200.)
plot_data(2)
set_model(2, powlaw1d.pl2)
print(get_model(2))


fit(1,2)

# Change Wave to Freq and nuFnu in Log
x = get_data(2).x
y = get_data(2).y

cspeed=3e10
freq=cspeed*1.e8/x
fnu=x*y*1.e-17
lfreq=numpy.log10(freq)
lnfnu=numpy.log10(fnu)

# Change X-ray Units:
# first get the flux in photons/cm2/s/keV
(counts, staterr,syserr) = get_data(1).to_fit(get_stat().calc_staterror)
```

Fit Results

X-ray data with RMF/ARF and Optical Spectra in ASCII

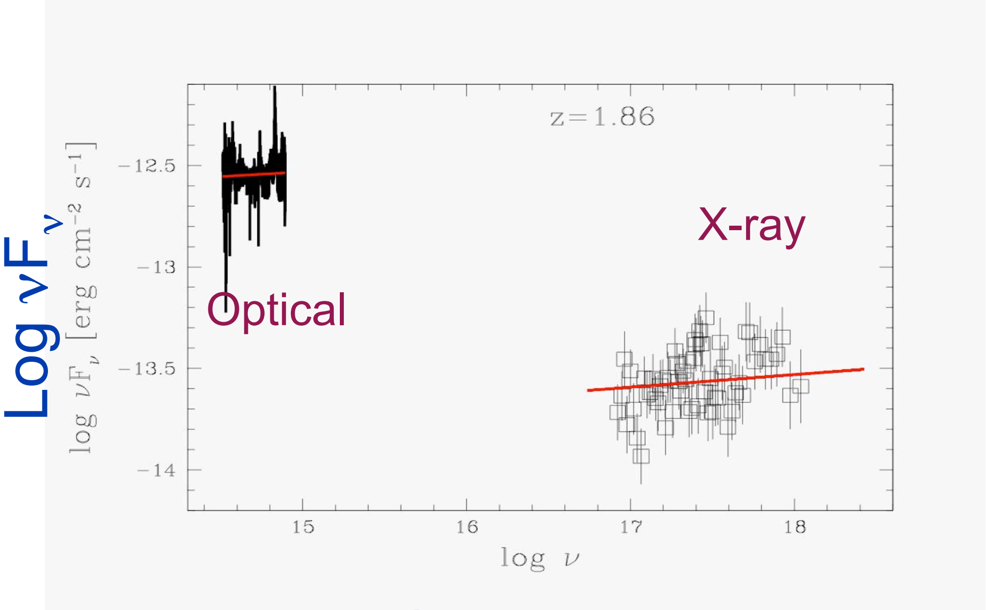
Flux



Wavelength

Quasar SED

Log νF_ν



Log ν

What do we really do?

Example:

I've observed my source, reduce the data and finally got my X-ray spectrum – what do I do now? How can I find out what does the spectrum tell me about the physics of my source?

Run *Sherpa*! But what does this program really do?

Fit the data => $C(h)=\int R(E,h)\ A(E)\ M(E,\theta)dE$

Counts

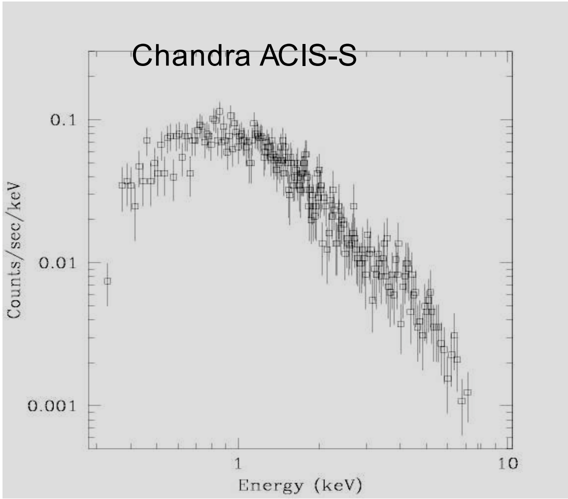
Response

Effective Area

Model

h- detector channels
E- Energy
θ- model parameters

Assume a model and look for the best model parameters which describes the observed spectrum.

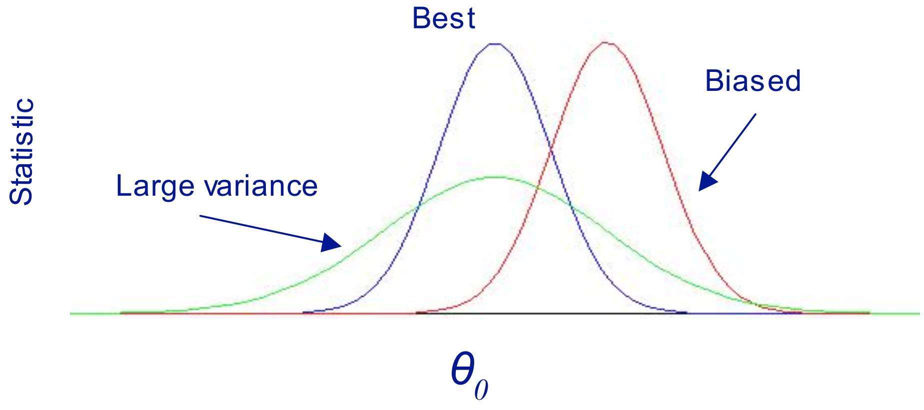


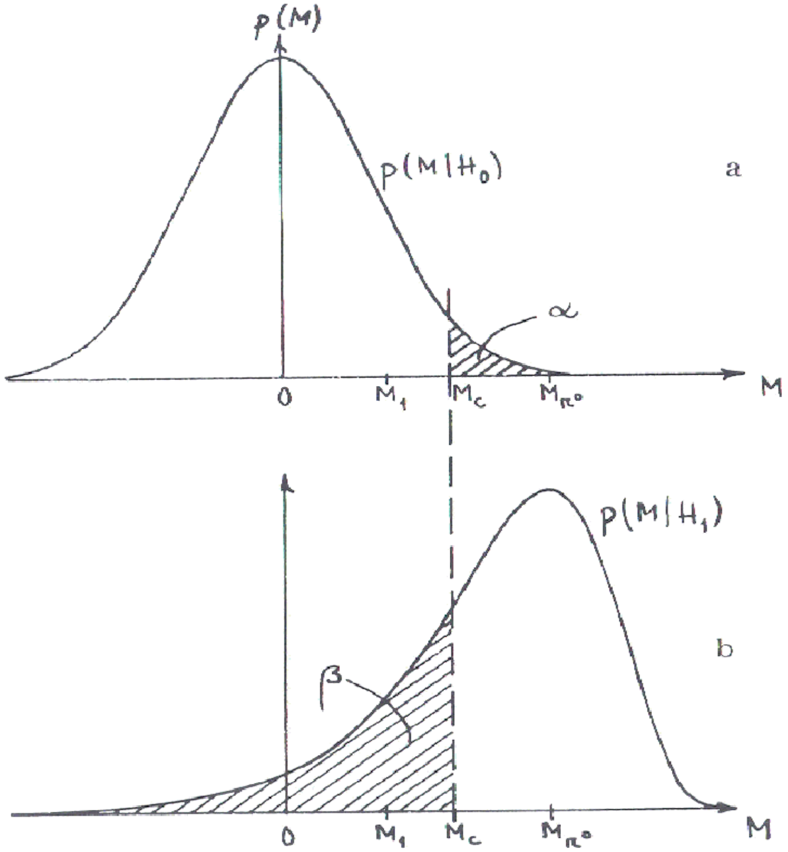
Need a Parameter Estimator - Statistics

Parameter Estimators: Statistics

Requirements on Statistics:

- **Unbiased**
  - converge to true value with repeated measurements
- **Robust**
  - less affected by outliers
- **Consistent**
  - true value for a large sample size (Example: rms and Gaussian distribution)
- **Closeness**
  - smallest variations from the truth




p(M)
p(M|H₀)
a
α
0
M₁
M_c
M_R°
M
p(M|H₁)
b
β
0
M₁
M_c
M_R°
M