



Introduction to Sherpa

Aneta Siemiginowska

Chandra X-ray Center

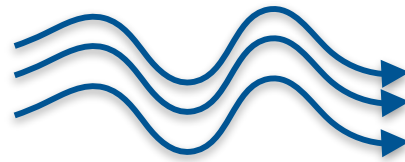
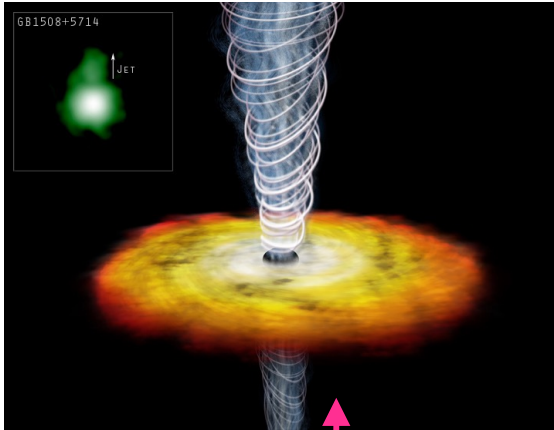
<http://cxc.harvard.edu/sherpa>

CIAO Workshop NCRA - Pune - October 2017

Observations and Data Collection

Detector collects photons, adds noise

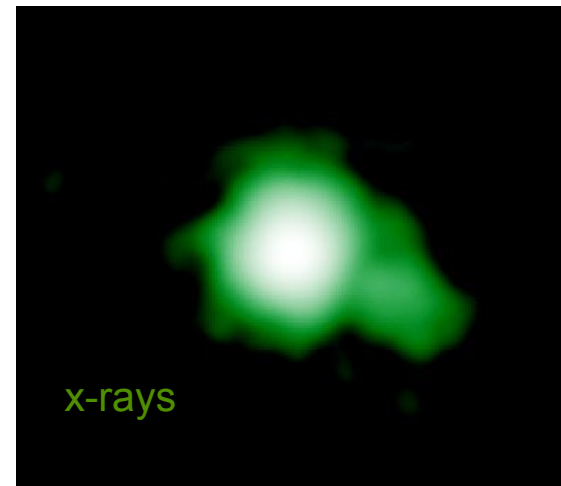
Astrophysical process



Random number of photons reach the detector



draw conclusion about the astrophysical source



Scientific Experiment

Observations

Data Preparation

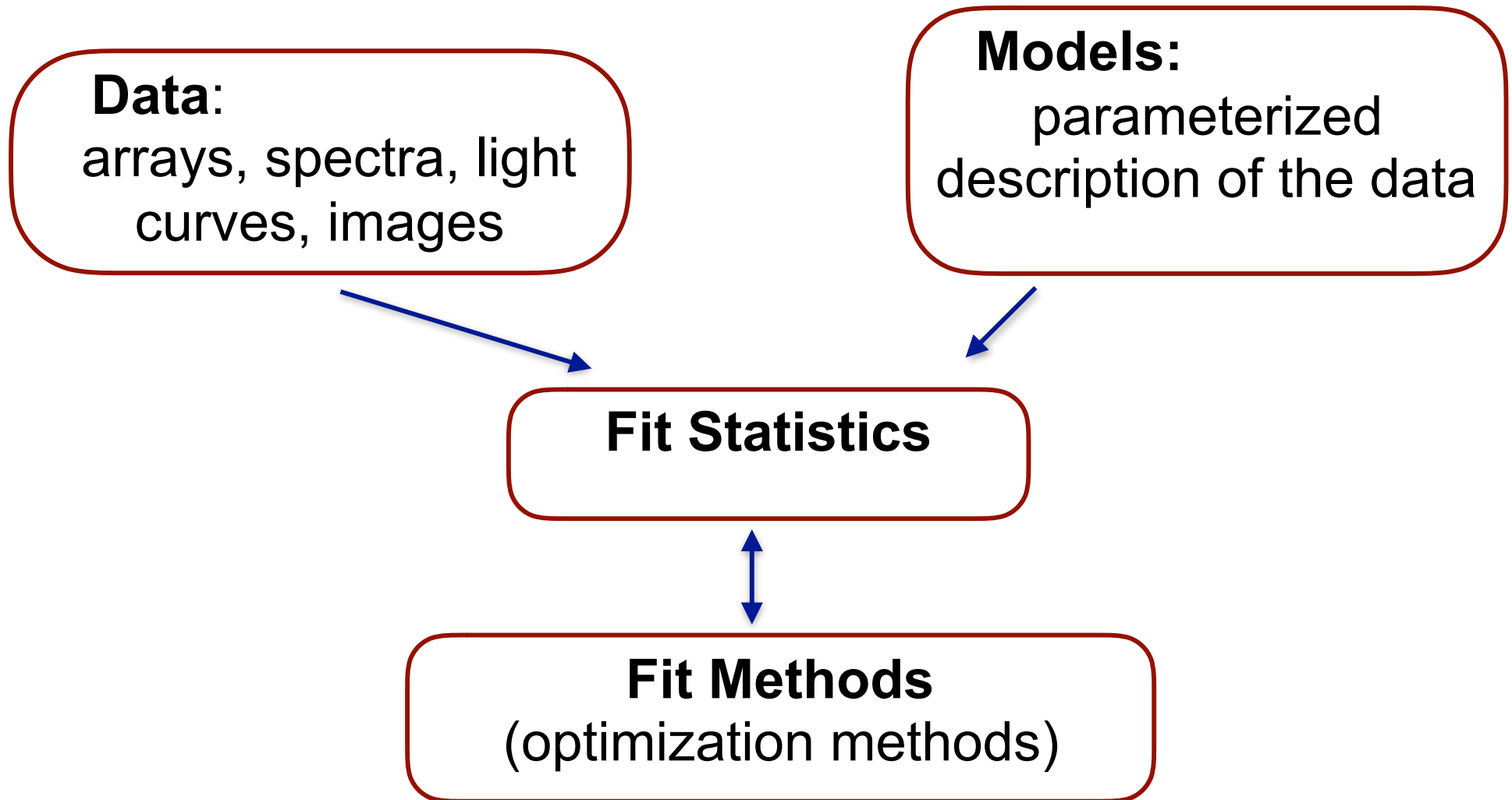
Instrument specific processing
software such as CIAO

Data Analysis:

source detections, source properties,
image analysis, features, spectra, physical properties of the
source, *apply models* to understand the source nature

Conclusions and Final Decision

Sherpa



Sherpa

Data Input/Output
Astropy.io
PyCrates

Models Library
Sherpa, XSPEC models,
user models, templates

Fit Statistics: Poisson and Gaussian likelihood

Fit Methods:
minimization and sampling

Visualization:
ChIPS, ds9, matplotlib

Final Evaluation
& Conclusions

Data in Sherpa

- **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

Data in Sherpa

- Load functions to input 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

- Multiple Datasets - data id

Default data id =1

load_data(2, "data2.dat", ncols=3)

Help file:

```
load_data( [id=1], filename, [options] )
```

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

Examples:

```
load_data("src", "data.txt", 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 the data

load_data expressions

notice/ignore commands in Sherpa

Examples:

```
notice(0.3,8)
```

```
notice2d("circle(275,275,50)")
```

Models in Sherpa

- Parameterized models: $f(x_i, p_k)$

absorption - N_H

photon index of a power law function - Γ

blackbody temperature kT

- Library of models

```
sherpa> list_models()
```

```
['absorptionedge',  
'absorptiongaussian',  
'atten',  
'bbody',  
'bbodyfreq',  
'beta1d',  
'beta2d',  
'blackbody',  
.....
```

- User Models can be added
- Model language to build compound model expressions

Building Models: Expressions

- Standard operations: $+$ $-$ $*$ $:$
- Linking parameters: `link()`
- **Convolution:**
 - **responses, arf & rmf files** via standard I/O
 - **PSF** - an image file or a Sherpa model
 - **load_conv()** - a generic kernel from a file or defined by a Sherpa model

Building Models: Examples

- Building composite models:

- models in the library: e.g. *powlaw1d*, *atten*
- give a **name** for a model component in the expression:

```
set_source(1, 'atten.abs1*atten.abs2*powlaw1d.p1')
```

```
set_source(2, 'abs1*abs2*powlaw1d.p2')
```

- Building a model expression with **convolved** and **unconvolved** components:

```
set_full_model(1, 'psf(gauss2d.g2)+const2d.c1')
```

Building Models: Examples

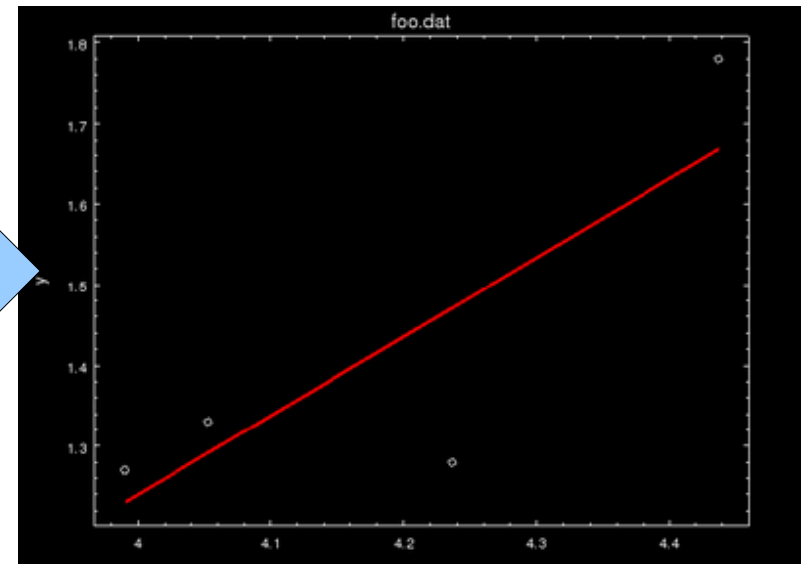
- Source and Background models:

```
set_source(2, 'xsphabs.abs1*(powlaw1d.p1+gauss1d.g1)')  
set_bkg_model(2, 'const1d.mybkg')
```

User Models: Python Function

- Adding a user model defined as a Python function is *Shockingly Simple!*

```
def myline(pars, x):  
    return pars[0] * x + pars[1]  
  
load_user_model(myline, "my1")  
add_user_pars("my1", ["m", "b"])  
set_source(my1)  
  
my1.m=30  
my1.b=20
```



User Models: Tables

- `load_table_model()`
- The file may be 1D data from a FITS table, ASCII table data, or 2D data from a FITS image file.
 - `modelname` - the name for the table model
 - `filename` - the name of the file which contains the data
 - `ncols` (table input) - number of columns to read;
 - `colkeys` (table input) - list of column names;
 - `dstype` (table input) - dataset type: *Data1D*, *Data1DInt*, *Data2D*, *Data2DInt*;
 - `coord` (image input) - the coordinate system: *logical*, *image*, *physical*, *world*, or *wcs*;
 - `method` - interpolation method in `sherpa.utils`:

linear_interp, *nearest_interp*, *neville*, *neville2d*;

```
sherpa> load_table_model("emap", "expmap.fits")
```

User Models: XSPEC Table Model

```
load_xstable_model("pul", "coplrefl.fits")
set_source(xsphabs.galabs * pul)
pul.xi = 1.5
pul.e_cut = 25
```

XSPEC table model `coplrefl.fits` is input as `pul` and used in the model expression.

```
show_source()
```

```
Model: 1
```

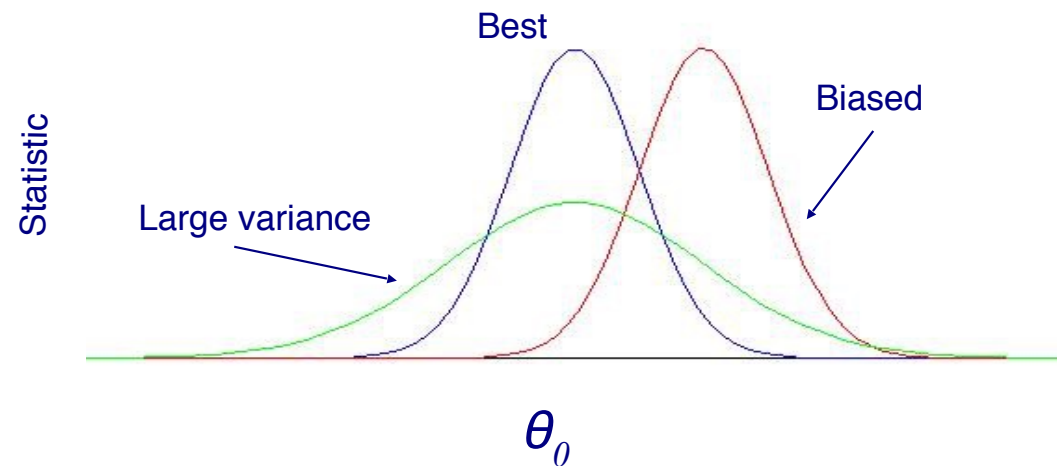
```
(xsphabs.galabs * xstablemodel.pul)
```

Param	Type	Value	Min	Max	Units
-----	----	-----	---	---	-----
galabs.nH	thawed	1	0	100000	10 ²² atoms / cm ²
pul.xi	thawed	1.5	1.477	3.977	
pul.gamma	thawed	1	0.5	1.5	
pul.e_cut	thawed	25	5	30	
pul.e_fold	thawed	15	5	30	
pul.redshift	frozen	0	0	5	
pul.norm	thawed	1	0	1e+24	

Fit Statistics in Sherpa

Fit statistics - math operation on data and model arrays

```
In [19]: list_stats()
Out[19]:
['cash',
 'chi2',
 'chi2constvar',
 'chi2datavar',
 'chi2gehrels',
 'chi2modvar',
 'chi2xspecvar',
 'cstat',
 'leastsq',
 'userstat',
 'wstat']
In [20]: set_stat('cash')
```

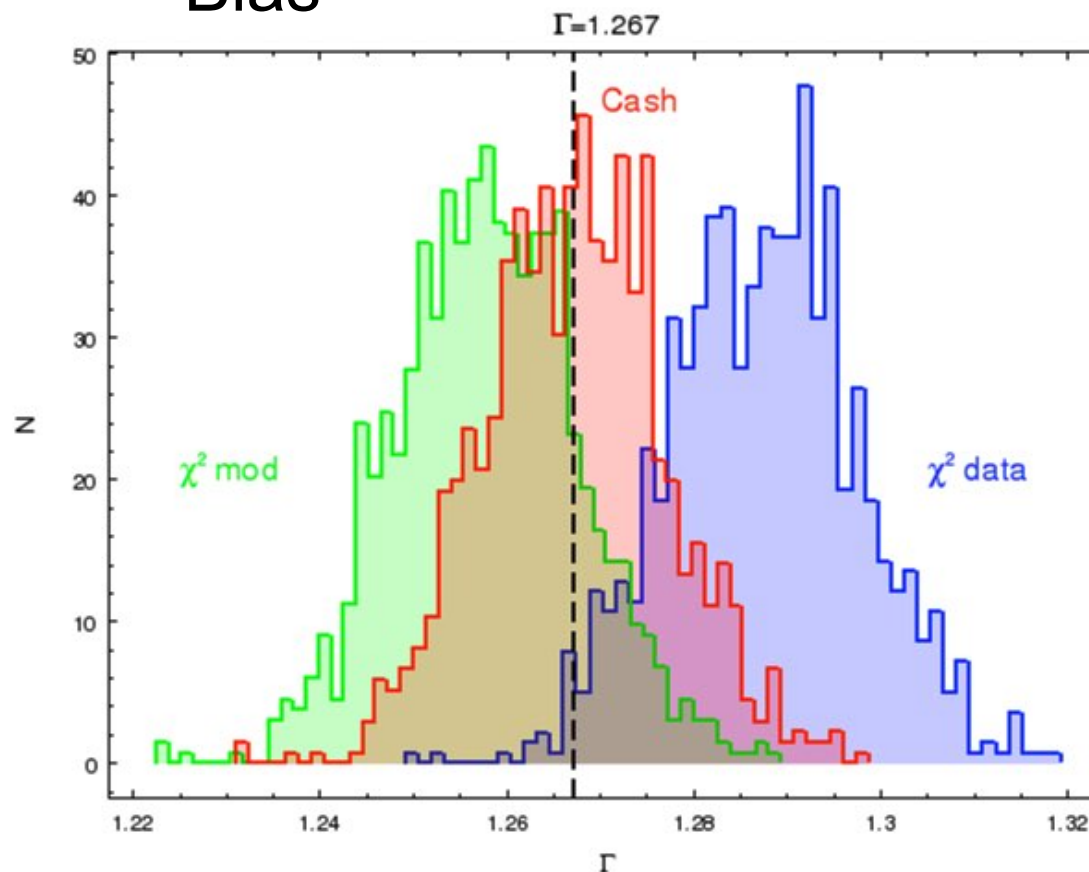


chi2 statistics as defined by different weights
and Poisson likelihood - cash/cstat/wstat

Fit Statistics in Sherpa

```
In [19]: list_stats()
Out[19]:
['cash',
 'chi2',
 'chi2constvar',
 'chi2datavar',
 'chi2gehrels',
 'chi2modvar',
 'chi2xspecvar',
 'cstat',
 'leastsq',
 'userstat',
 'wstat']
In [20]: set_stat('cash')
```

Bias



“Handbook of X-ray Astronomy “
(2011), Arnaud, Smith, Siemiginowska

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 $f(x,p)$ 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 $f(x,p)$ 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 set of parameters, and then improve in a continuous fashion:
 - Very Quick
 - Depend critically on the initial parameter values
 - Investigate a local behaviour of the statistics near the initial 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: **moncar (differential evolution)**
search over the entire permitted parameter space for a better minima than near the starting initial set of parameters.
- Bayesian sampling methods: **Markov-Chain Monte Carlo**

Optimization Methods: Comparison

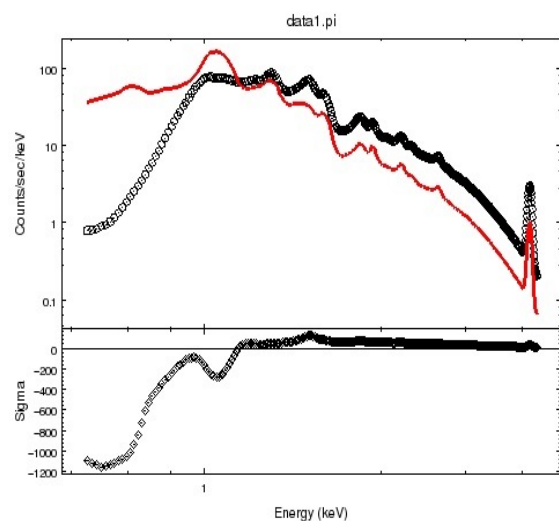
Example: Spectral Fit with 3 methods

Data: high S/N simulated ACIS-S spectrum of the two temperature plasma

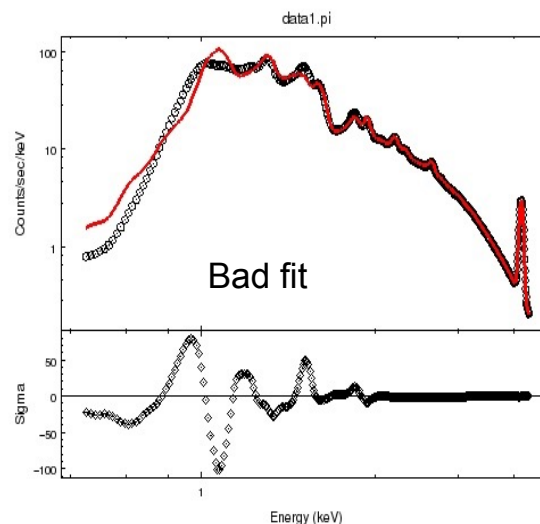
Model: photoelectric absorption plus two MEKAL components (correlated!)

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

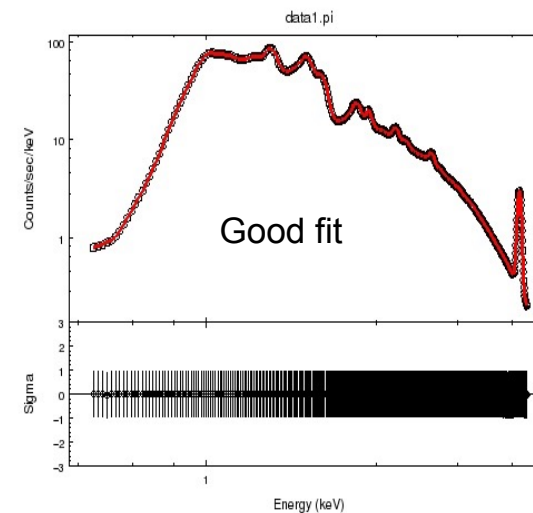
Method	Number of Iterations	Final Statistics
Levmar	31	1.55e5
Neldermead	1494	0.0542
Moncar	13045	0.0542



Data and Model with initial parameters



Levmar fit



Nelder-Mead and Moncar fit

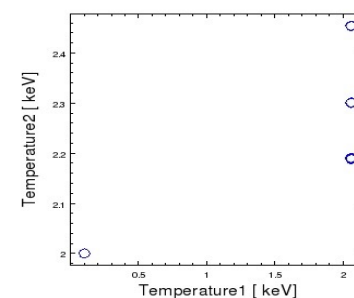
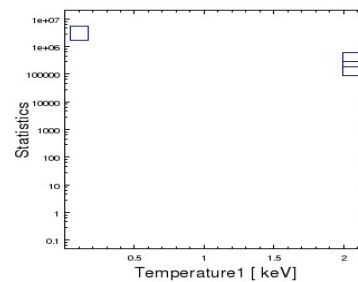
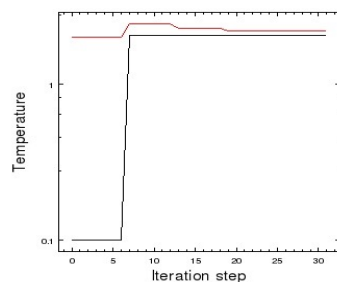
Optimization Methods: Probing Parameter Space

Temperature

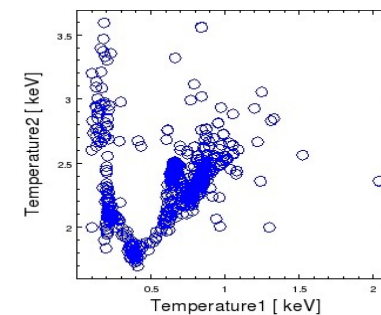
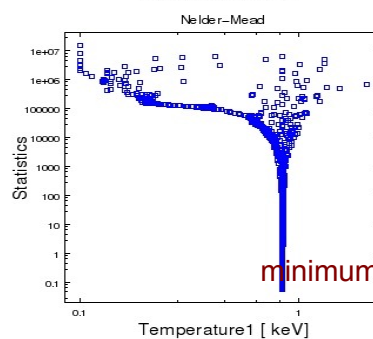
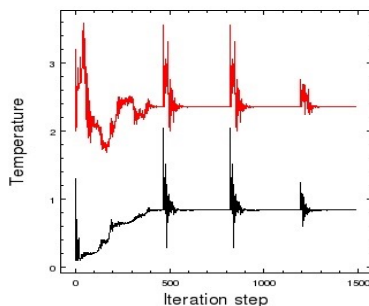
Statistics vs. Temperature

2D slice of Parameter Space probed by each method

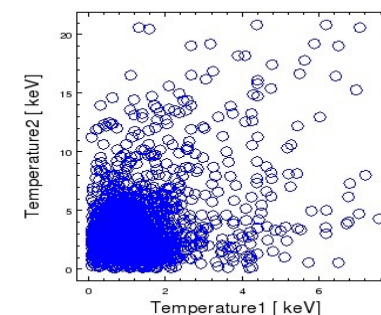
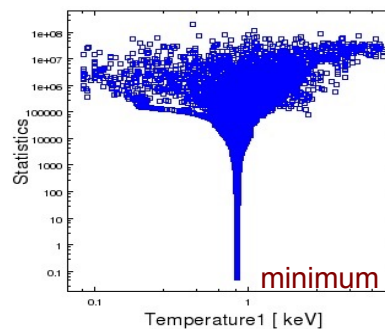
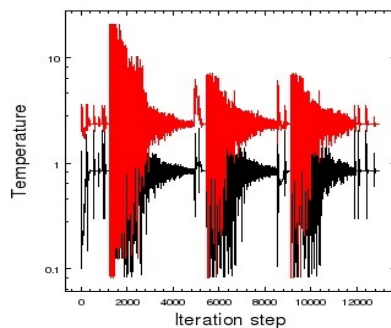
levmar



simplex



moncar



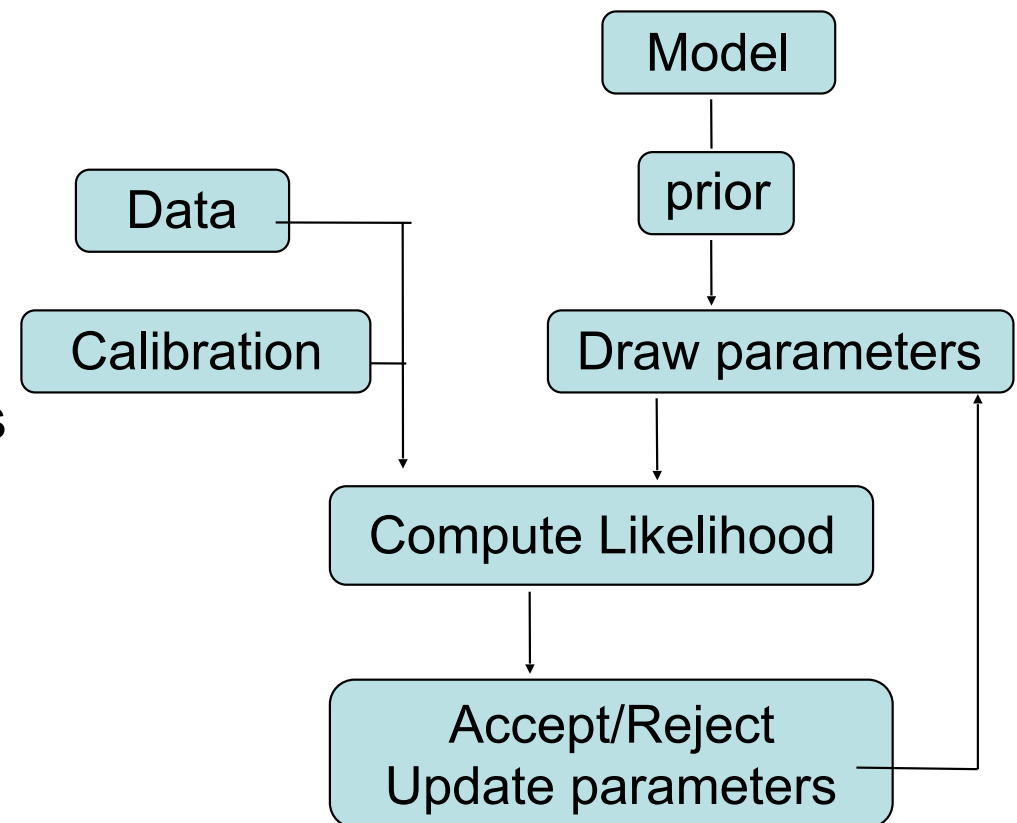
Sherpa, MCMC and Bayesian Analysis

MCMC samplers in Sherpa:

Metropolis and Metropolis-Hastings algorithms

Support for the Bayesian analysis with priors.

- Explores parameter space and summarizes the full posterior or profile posterior distributions.
- Computed parameter uncertainties can include systematic or calibration errors.
- Simulates replicate data from the posterior predictive distributions.



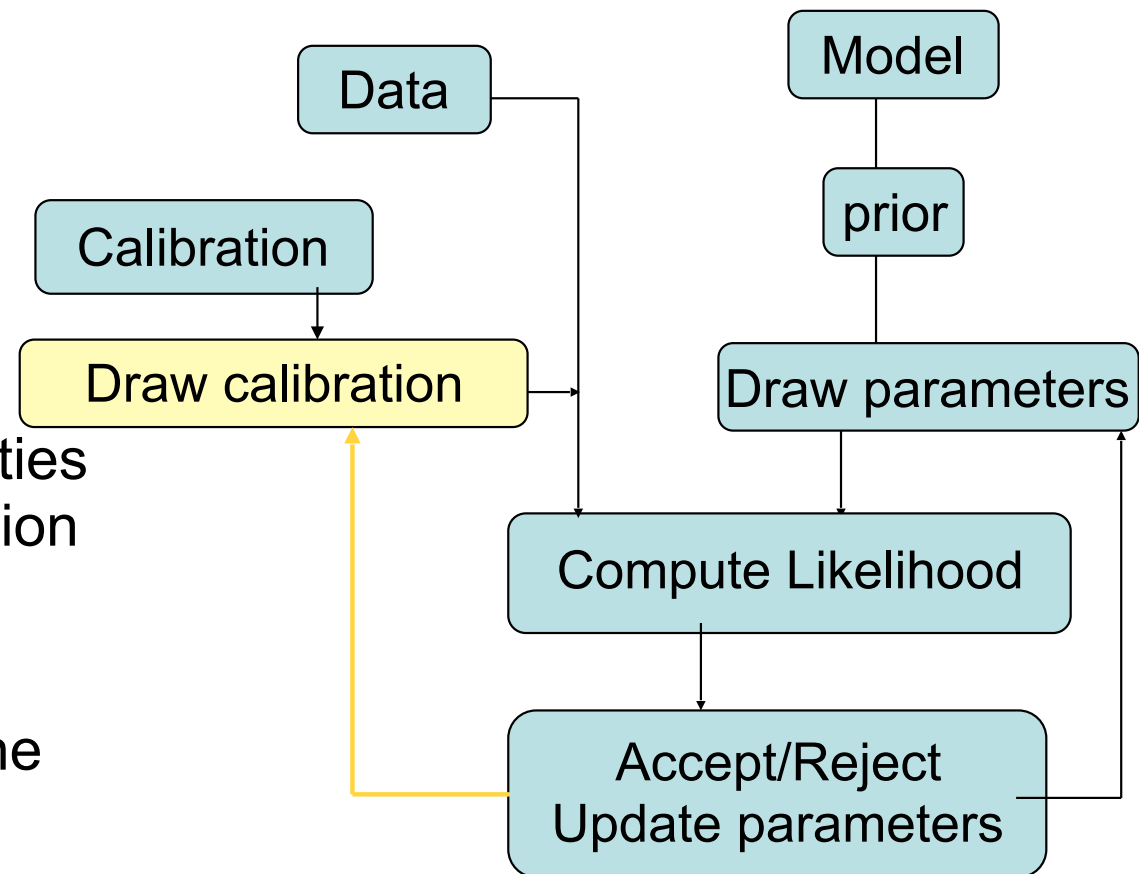
Sherpa, MCMC and Bayesian Analysis

MCMC samplers:

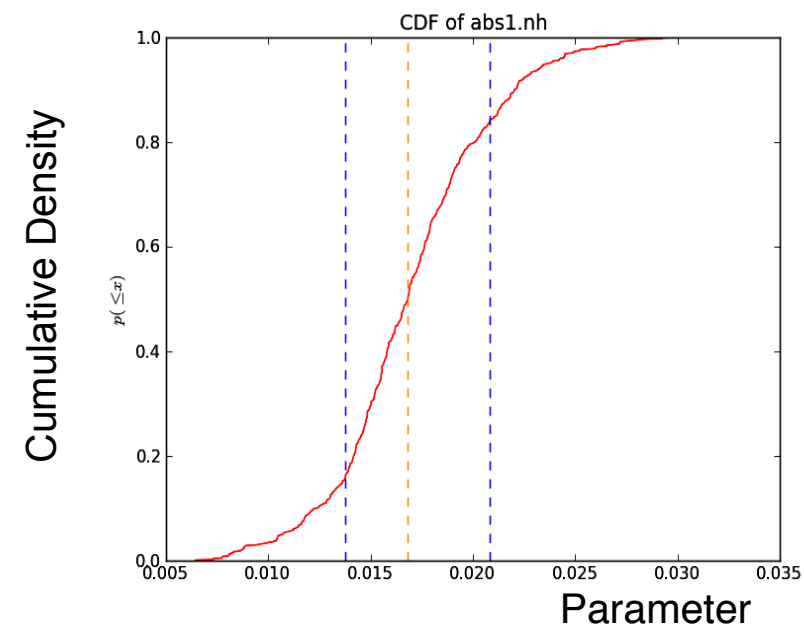
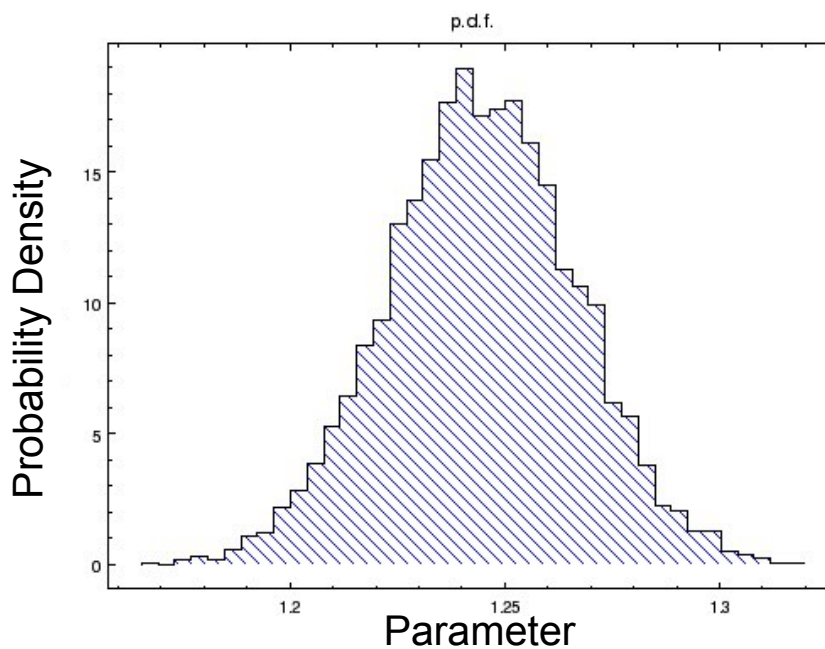
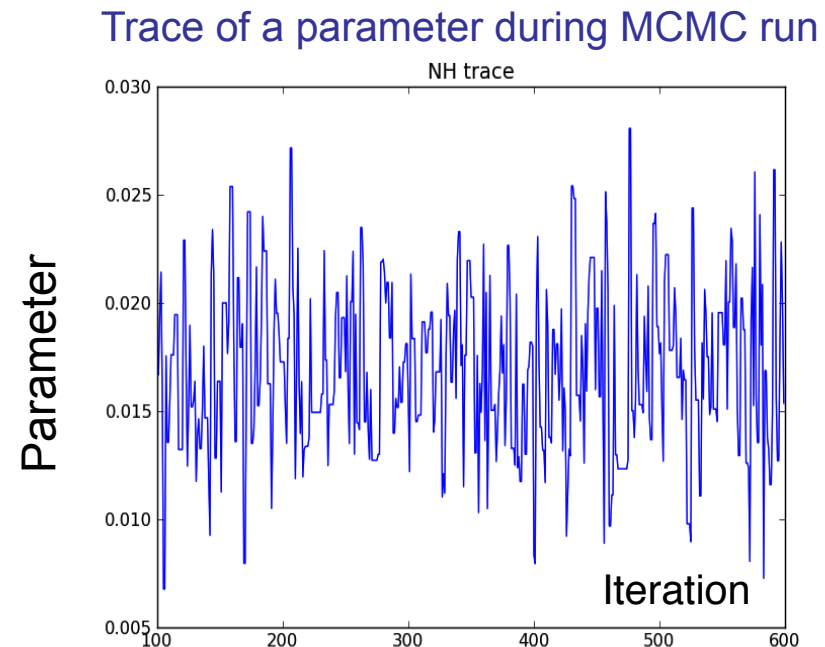
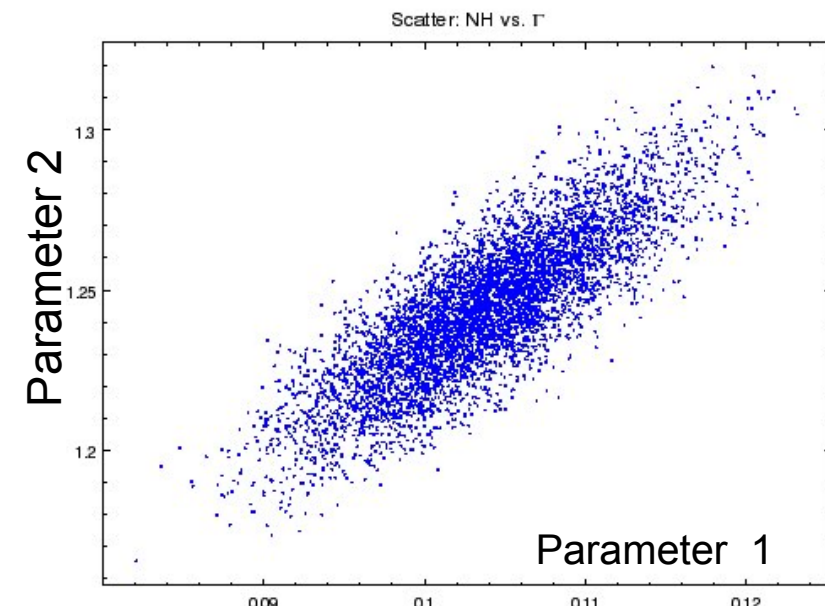
Metropolis and Metropolis-Hastings algorithms

Support for Bayesian analysis with priors.

- Explores parameter space and summarizes the full posterior or profile posterior distributions.
- Computed parameter uncertainties can include systematic or calibration errors.
- Simulates replicate data from the posterior predictive distributions.



Visualization of the MCMC Results



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 best-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

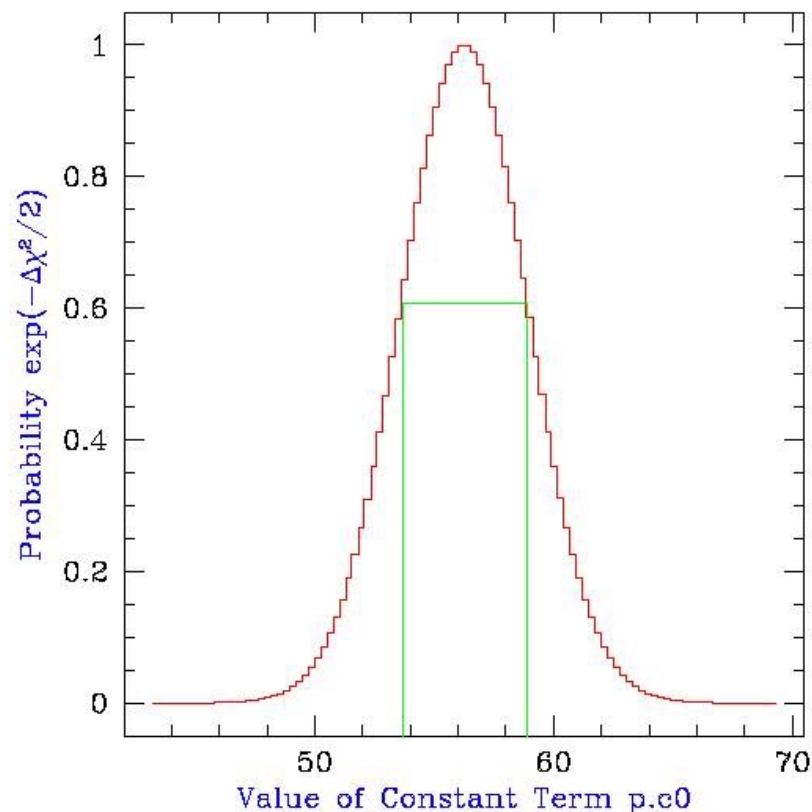
α - level

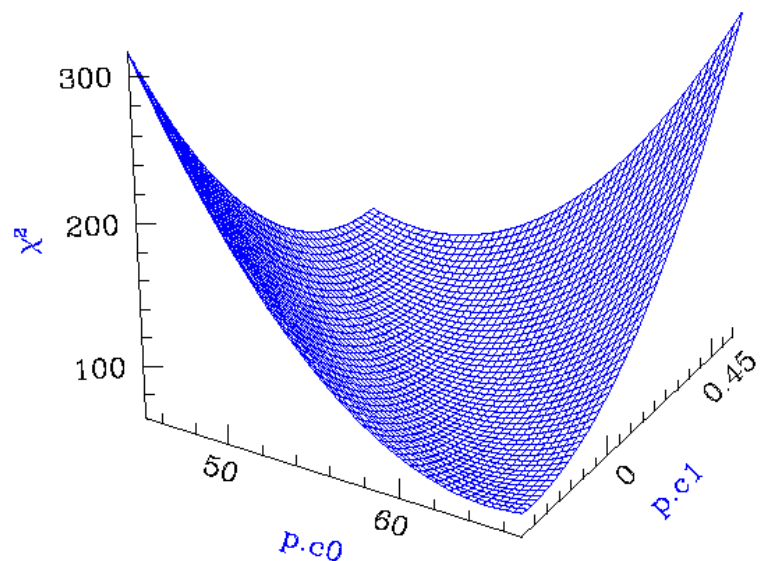
χ^2_{\min} - minimum

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

TABLE 1
CONSTANTS FOR CALCULATING CONFIDENCE REGIONS

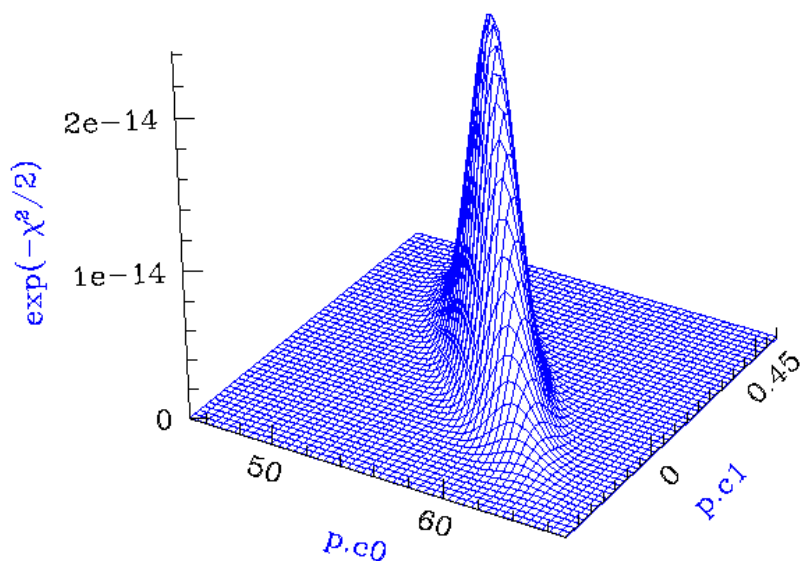
α (%)	q (NO. OF INTERESTING PARAMETERS)		
	1	2	3
68.....	1.00	2.30	3.50
90.....	2.71	4.61	6.25
99.....	6.63	9.21	11.30





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*).



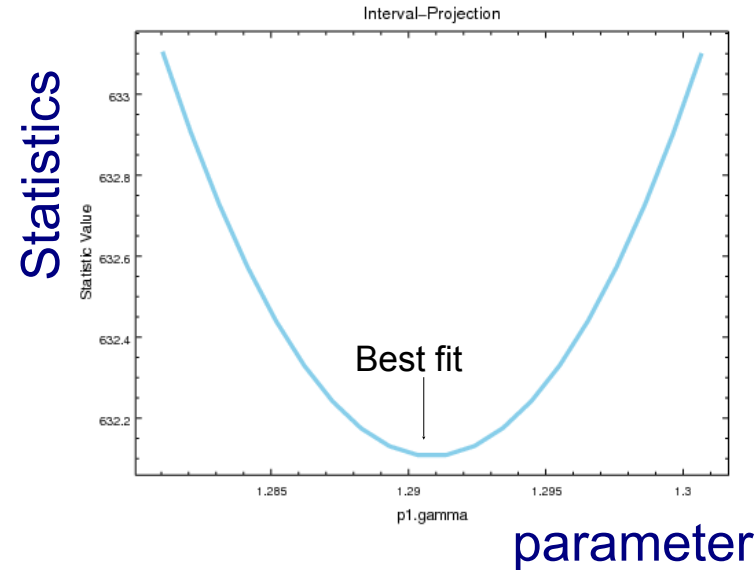
Confidence Intervals

```
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()
```

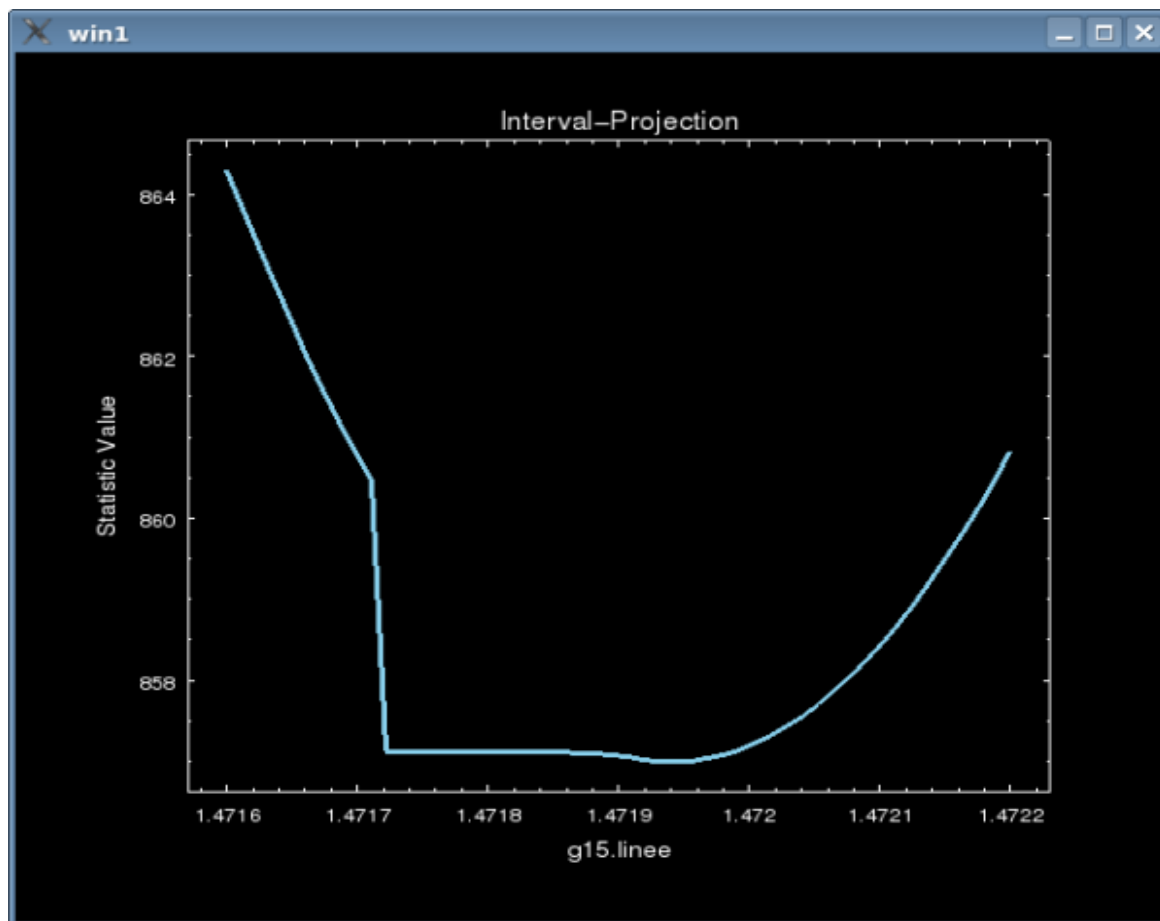
```
mek1.kT lower bound: -0.00113811
mek1.kT upper bound:  0.0011439
mek2.kT lower bound: -0.00365452
mek2.kT upper bound:  0.00364805
mek1.norm lower bound: -0.00377224
mek2.norm lower bound: -0.00164417
mek2.norm upper bound:  0.00164816
abs1.nH lower bound: -0.00147622
mek1.norm upper bound:  0.00376011
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, -0.001138111192153346,
-0.0037722356859711814, -0.0036545192286010497, -0.0016441656050858455)
parmaxes = (0.001472679745547989, 0.0011439029752089436,
0.0037601110158367312, 0.003648045819133916, 0.001648162229710648)
nfits = 103
```

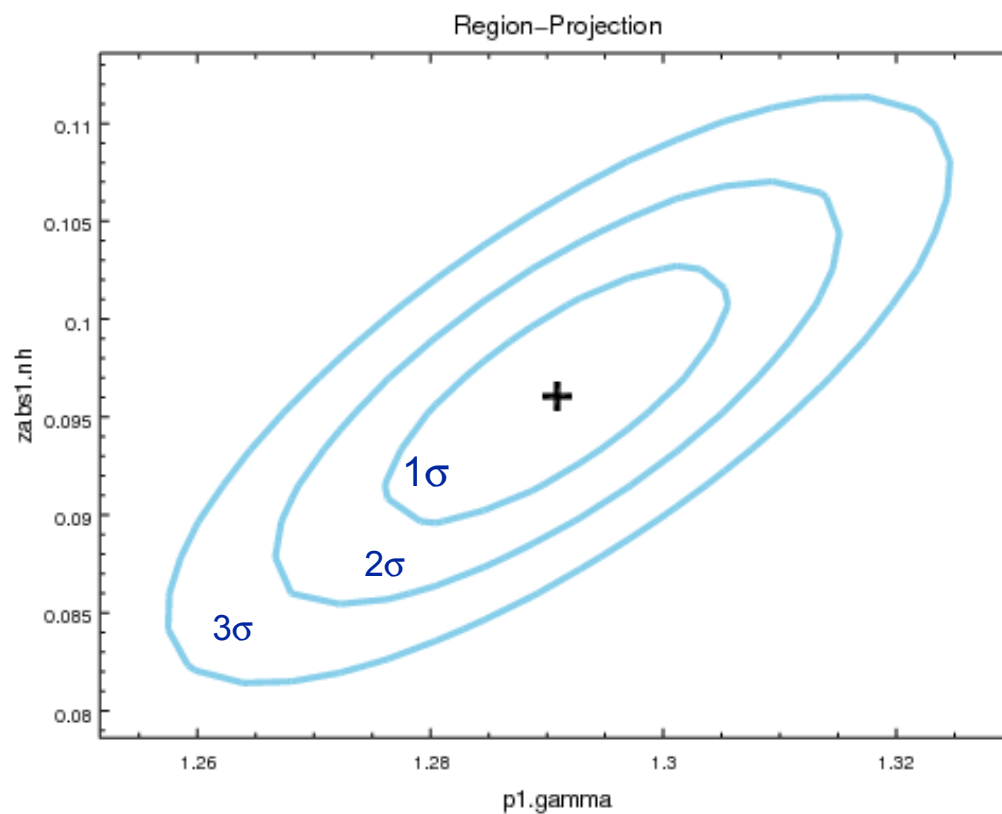
Not well-behaved Surface



Non-Gaussian Shape

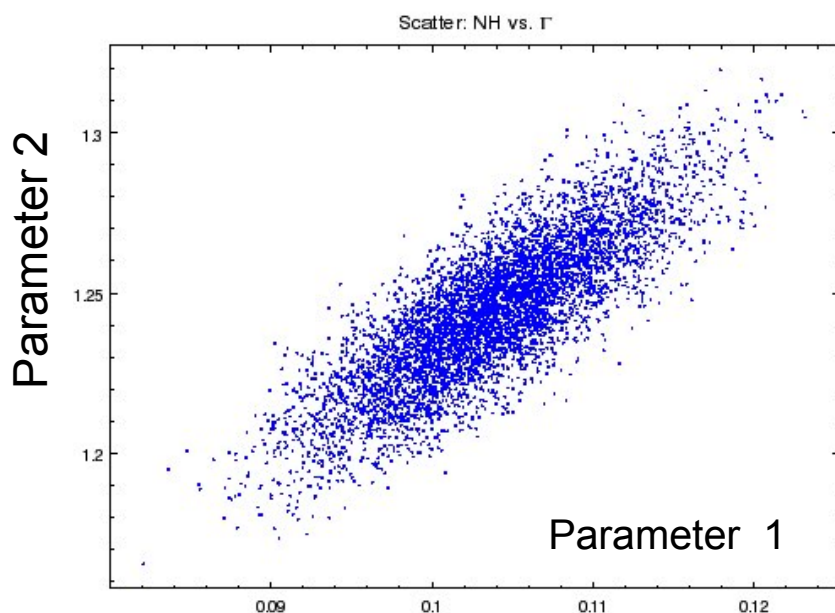
Confidence Regions

```
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]
```

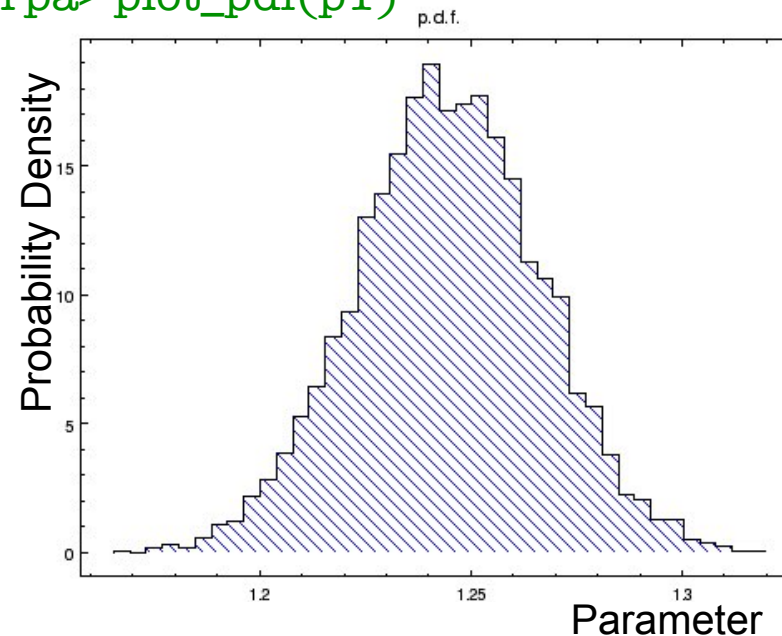


MCMC Results: Probability Distributions

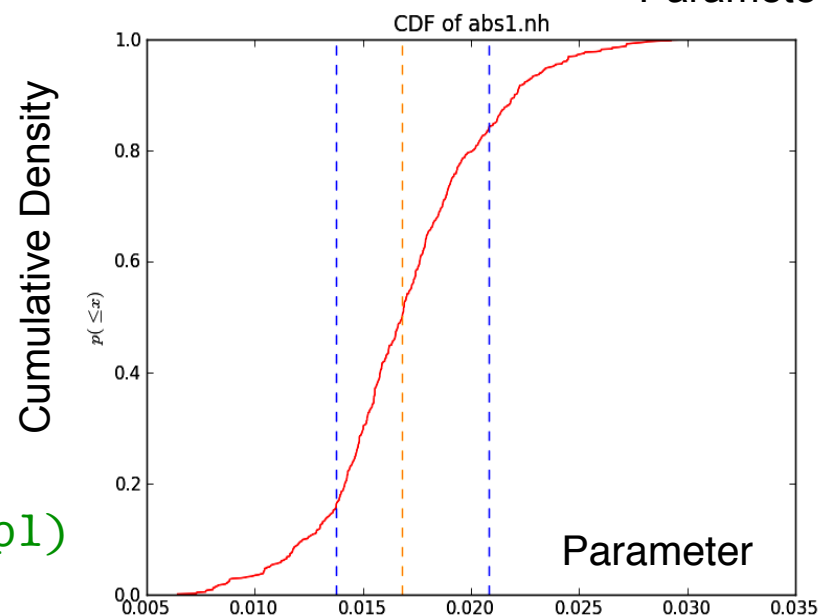
```
sherpa> plot_scatter(p1,p2)
```



```
sherpa> plot_pdf(p1)
```



```
sherpa> plot_cdf(p1)
```



Distributions of Flux and Parameters

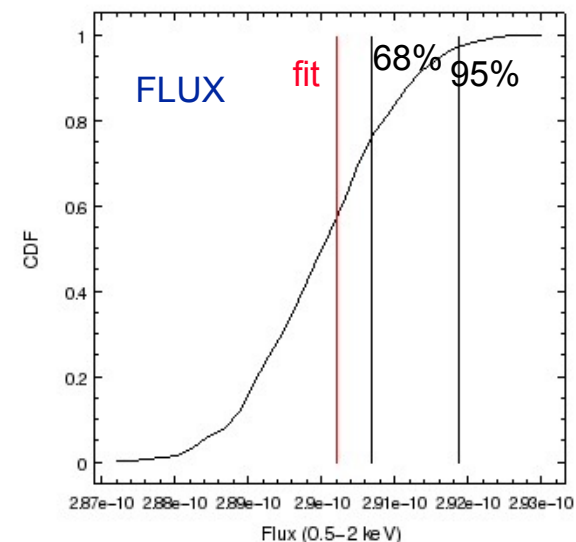
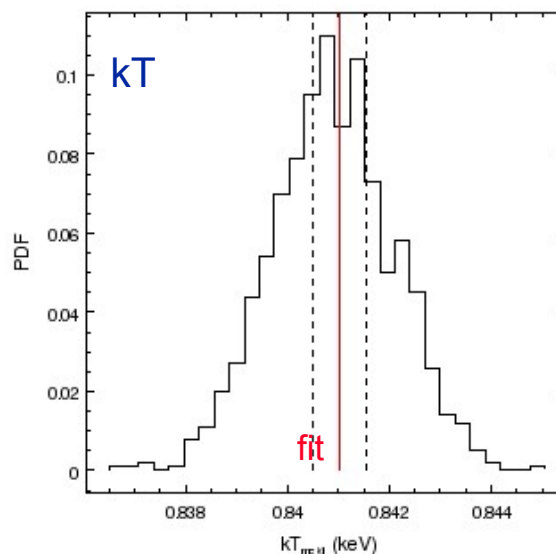
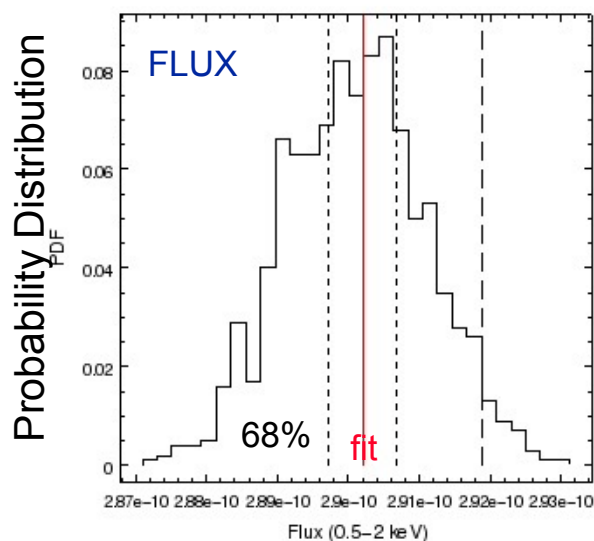
Functions: `sample_energy_flux`, `sample_flux`

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)
```



Sherpa - Summary

- Modeling and fitting application for Python.
- User Interface and high level functions written in Python.
- Modeling 1D/2D (N-D) data: arrays, spectra, images.
- Powerful language for building complex expressions.
- Provides a variety of statistics and optimization methods (including Bayesian analysis) .
- Support for wcs, responses, psf, convolution.
- Extensible to include user models, statistics and optimization methods.
- Included in several software packages.
- Source code on GitHub <https://github.com/sherpa/sherpa>
- Open development with continuous integration via Travis

Core Team:

Omar Laurino, Doug Burke, Warren McLaughlin, Dan Nguyen, Aneta Siemiginowska

Code contributions:

Tom Aldcroft, Jamie Budynkiewicz, Christoph Deil, Brigitta Sipocz

The screenshot shows the GitHub repository page for `sherpa/sherpa`. The browser's address bar shows the URL `https://github.com/sherpa/sherpa`. The repository name is `sherpa / sherpa`. The page displays various statistics: 1,177 commits, 32 branches, 8 releases, 9 contributors, and GPL-3.0 license. There are buttons for 'Unwatch' (11), 'Unstar' (24), and 'Fork' (21). The navigation bar includes 'Code', 'Issues 79', 'Pull requests 15', 'Projects 0', 'Wiki', 'Pulse', and 'Graphs'. The main content area shows a merge commit by `olaurino` titled 'Merge #315 - Release/ciao4.9' with the latest commit `1ebebb7` from 12 days ago. Below the merge commit, there are three folders: `docs` (Remove S-Lang files, 2 years ago), `extern` (why are these files still in the way?, 4 months ago), and `helpers` (Update copyright year to include 2016, 4 months ago).

sherpa / sherpa

Unwatch 11 Unstar 24 Fork 21

Code Issues 79 Pull requests 15 Projects 0 Wiki Pulse Graphs

Sherpa is a modeling and fitting application for Python <http://cxc.cfa.harvard.edu/contrib/sh...>

1,177 commits 32 branches 8 releases 9 contributors GPL-3.0

Branch: master New pull request Create new file Upload files Find file Clone or download

olaurino Merge #315 - Release/ciao4.9 Latest commit 1ebebb7 12 days ago

docs	Remove S-Lang files.	2 years ago
extern	why are these files still in the way?	4 months ago
helpers	Update copyright year to include 2016.	4 months ago

Open Development

sherpa / sherpa

Unwatch 11 Unstar 24 Fork 21

Code Issues 79 Pull requests 15 Projects 0 Wiki Pulse Graphs

Filters is:pr is:open Labels Milestones [New pull request](#)

<input type="checkbox"/> 15 Open ✓ 168 Closed	Author	Labels	Milestones	Assignee	Sort
<input type="checkbox"/> Skip test if no plotting backend is installed ✓ 3 - Working area:tests type:bug 14					
<input type="checkbox"/> Use relative imports for XSPEC module ✓ 3 - Working area:code dep:xspec					
<input type="checkbox"/> remove requires_data decorator from make_data_path fixture ✓ 3 - Working area:tests 2					

Code contribution

Add more commits by pushing to the `bug-fix-tests-when-no-matplotlib` branch on `DougBurke/sherpa`.

Travis
- continuous integration



All checks have passed [Hide all checks](#)
1 successful check

continuous-integration/travis-ci/pr — The Travis CI... [Details](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

[Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Using Sherpa in Astronomy Software

- IRIS - GUI for data exploration and SED fitting
http://www.usvao.org/index.html%3Fpage_id=357.html
- BAX - Bayesian X-ray Analysis
<https://github.com/JohannesBuchner/BXA>
- XMM-Newton Source Catalog:
<http://xmm-catalog.irap.omp.eu/docs/spectral-fitting>
 - web interface to spectral fitting of the sources in 3XMM-DR6 catalog
- **Astropy^a** Affiliated packages: <http://www.astropy.org/affiliated/index.html>
 - GammaPy <https://gammapy.readthedocs.io/en/latest/>
 - Naima <https://naima.readthedocs.io/en/latest>
- **Saba - Sherpa-Astropy Bridge** <https://saba.readthedocs.io/en/latest/>
 - Google funded a summer student (through GSOC program) to develop the code and documentation.
 - pending application for Astropy affiliated package.

a) From astropy.org web page:

Astropy is a community-driven package intended to contain much of the core functionality and some common tools needed for performing astronomy and astrophysics with Python.

Summary

- Sherpa is a Python package.
- It provides models, fit statistics and optimization methods for variety of problems in astronomy.
- It is flexible and extensible as it accepts new models, statistics or optimization.
- Sherpa can also be included in a modeling software in Python.

Learn more on Sherpa Web Pages

<http://cxc.harvard.edu/sherpa>

Sherpa
CIAO's modeling and fitting package

WHAT'S NEW | WATCH OUT
Analysis Threads | Ahelp | Download CIAO II CIAO I ChIPS | PSF Central

Latest Version
Sherpa version for CIAO 4.9 was released on December 16, 2016. *Sherpa* now runs under Python 2.7 and Python 3.5. The major updates are: the update of the XSPEC models to version 12.9.0d; the addition of the `vsstat` statistic, for including a background dataset as a model component (it is based on the XSPEC `version`); support for background data sets in user statistics created with `load_user_stat`; a major upgrade to the Python docstrings (the help pages are still available in this release); and it is the first CIAO release based on the `develop` branch of *Sherpa*. Please consider contributing to *Sherpa* development, whether by adding code, fixing bugs, or changing documentation.

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;
- perform Bayesian analysis with Poisson Likelihood and priors, using Metropolis or Metropolis-Hastings algorithm in the MCMC (Markov-Chain Monte Carlo);
- and use Python to create complex analysis and modeling functions, build the batch mode analysis or extend the provided functionality to meet the required needs.

CHANDRA X-RAY OBSERVATORY

CXC HOME | PROPOSER | ARCHIVE | DATA ANALYSIS | INSTRUMENTS & CALIBRATION | FOR THE PUBLIC

Search <http://cxc.harvard.edu/sherpa/>
Google Custom Search Search

Contact the CXC HelpDesk

Sherpa Threads

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

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.

Scripts
To quickly access the scripts used in each of the Sherpa threads, visit the [Sherpa Quick Scripts page](#).

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

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

There are also Sherpa Jupyter Notebooks available on Sherpa GitHub page:

<https://github.com/sherpa/sherpa/wiki>

Example of Image Fitting:

<http://nbviewer.jupyter.org/github/anetasie/SherpaNotebooks/blob/master/ImageFitting.ipynb>

SABA: Sherpa - Astropy Bridge

- `astropy.modeling`
 - uses `scipy.optimize`, weighted least square, does not calculate uncertainties
- **SABA** <https://saba.readthedocs.io/en/latest/>
 - bridge between the model definition language in `astropy.modeling` and the powerful fitting capabilities of Sherpa.
- **Sherpa**
 - has a selection of robust optimization algorithms coupled with configurable fit statistics.
 - can estimate parameter confidence intervals, with methods that allow for coupled non-gaussian errors.
 - has an MCMC sampler that can be used to explore the posterior probability distribution.