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

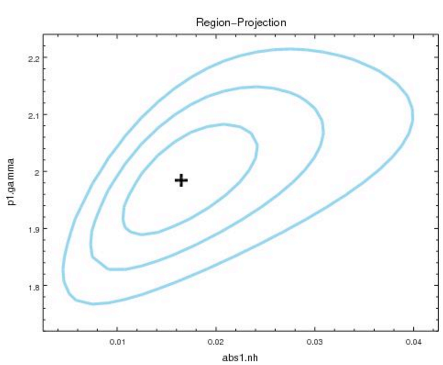
Outline

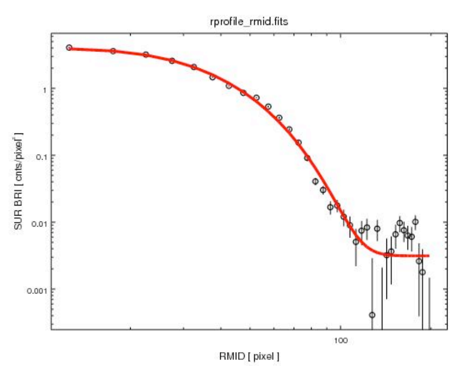
- What is Sherpa?
- Observations and Models
- Statistics
- Fitting, optimization and results
- Summary and Conclusion

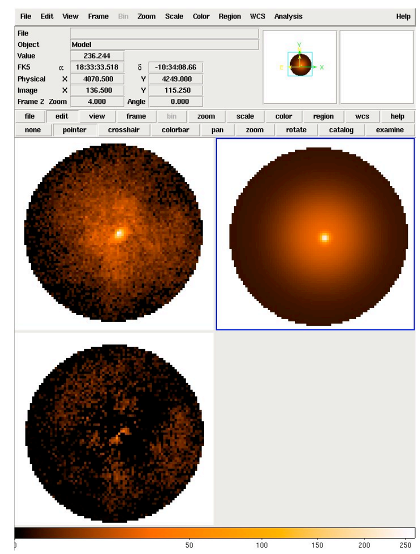
Sherpa β

CIAO's Modeling and Fitting package

- Generalized 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.
- Beta version in CIAO 4.0 with Python and S-lang scripting language
- New release planned for Dec.2008 in CIAO4.1
- Significant re-write to modularize the code for future improvements
- Walkthrough with a few examples







Modeling: what can I learn from 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

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)")

Observations: Data I/O in Sherpa

- Information about the data: show_data(), show_bkg()

```
sherpa> show_all()
Optimization Method: LevMar
Statistic:        Chi2Gehrels

Data Set: 1
name          = 3c273.pi
channel       = Int32[1024]
counts        = Int32[1024]
staterror     = None
syserror      = None
bin_lo        = None
bin_hi        = None
grouping      = Int16[1024]
quality       = Int16[1024]
exposure      = 38564.6089269
backscal      = 2.52643646989e-06
areascal      = 1.0
grouped       = True
subtracted    = False
units         = energy
response_ids  = [1]
background_ids = [1]

RMF Data Set: 1:1
name     = 3c273.rmf
detchans = 1024
```

```
sherpa> load_pha("3c273.pi")
statistical errors were found in file '3c273.pi'
but not used; to use them, re-read with use_errors=True
read ARF file 3c273.arf
read RMF file 3c273.rmf
statistical errors were found in file '3c273_bg.pi'
but not used; to use them, re-read with use_errors=True
read background file 3c273_bg.pi

sherpa> load_arf("3c273.arf")
sherpa> load_arf("3c273.rmf")
sherpa> load_bkg("3c273_bg.pi")
```

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 or Slang Functions

  load_user_model, add_user_pars

  - Python and Slang 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
```

Standard PHA based analysis in Sherpa:

- Source data:
  - can be modeled in energy/wavelength space.
  - multiple data sets can be modeled with the same or different models in one Sherpa session.
  - data can be filtered on the command line, or from filter file.
- Instrument responses (RMF/ARF):
  - are entered independently from the source data.
  - one set of instrument responses can be read once and applied to multiple data sets.
  - several instrument responses used in analysis of one source model or multiple data sets.
  - multiple response files can be used in one source model expression.
- Background files:
  - are entered independently from the source data.
  - multiple background files can be used for one data set,e.g. grating analysis
  - the same background can be applied to multiple data sets.
  - background can be modeled independently of the source data, and have its separate instrument responses.
  - background can be modeled simultaneously with the source data.
  - background can be subtracted from the source data (subtract/unsubtract).

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

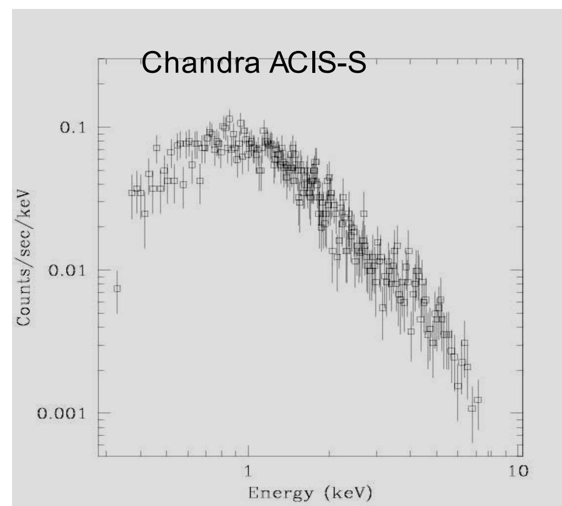
h- detector channels

E- Energy

θ- model parameters

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

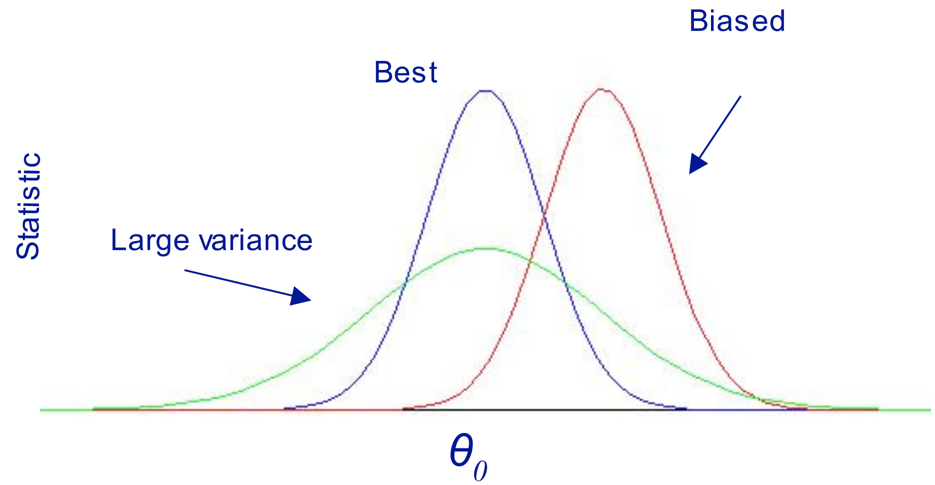
Need a Parameter Estimator - Statistics



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



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:

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

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":

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

(Non-) Use of the Poisson Likelihood

In model fits, the Poisson likelihood is not as commonly used as it should be. Some reasons why include:

- a historical aversion to computing factorials;
- the fact the likelihood cannot be used to fit "background subtracted" spectra;
- the fact that negative amplitudes are not allowed (not a bad thing physics abhors negative fluxes!);
- the fact that there is no "goodness of fit" criterion, i.e. there is no easy way to interpret $\mathcal{L}_{\mathbf{max}}$ (however, *cf*. the **CSTAT** statistic); and
- the fact that there is an alternative in the Gaussian limit: the χ^2 statistic.

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:

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

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.

Statistic 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")
```

Statistics - Example of Bias

• The $\chi 2$ bias can affect the results of the X-ray spectral fitting

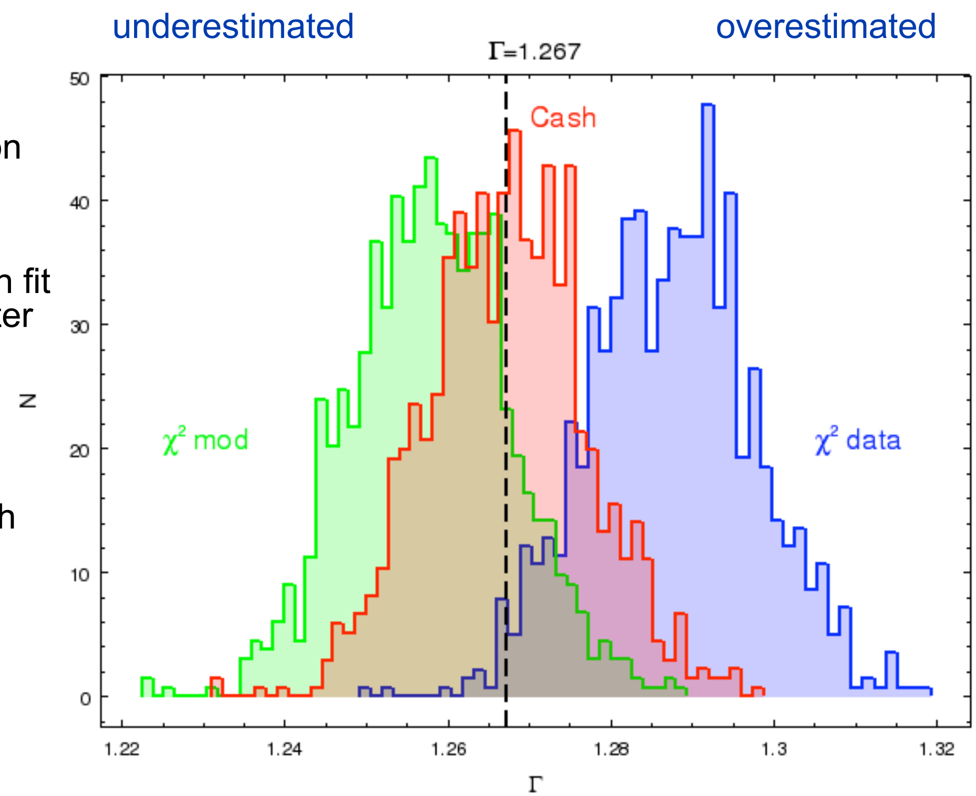
• Simulate Chandra spectrum given RMF/ARF and the Poisson noise - using fake_pha().

• 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.

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

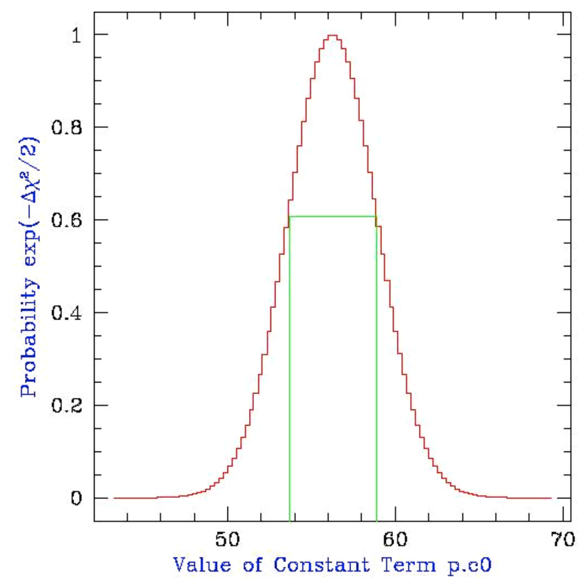
Essential issue = after the bets-fit parameters are found estimate the confidence limits for them. The region of confidence is given by (Avni 1976):

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

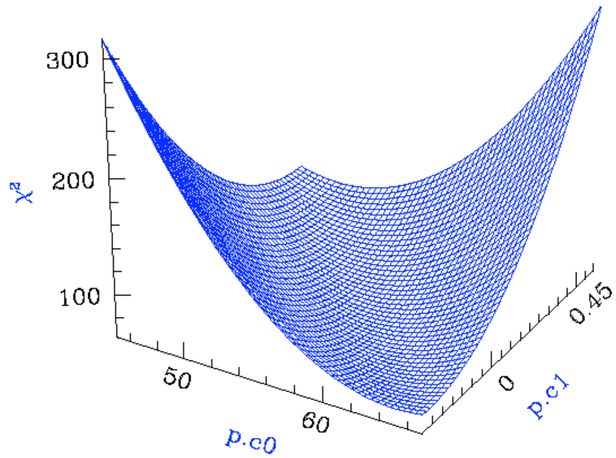
ν - degrees of freedom
α - significance
χ^2_{min} - minimum

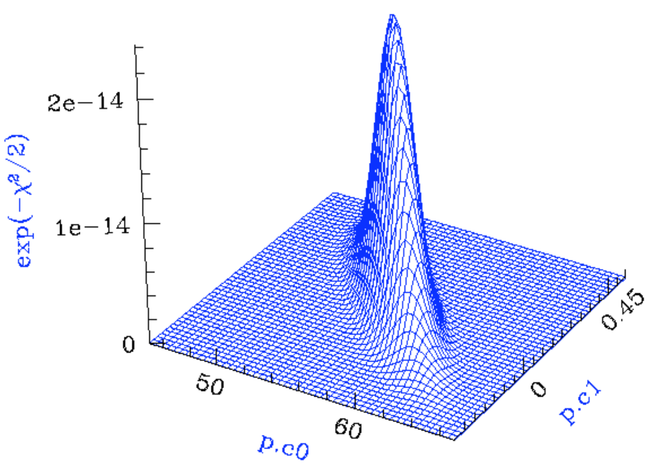
Δ depends only on the number of parameters involved not on goodness of fit

Significance α	Number of parameters 1	2	3
0.68	**1.00**	**2.30**	**3.50**
0.90	**2.71**	**4.61**	**6.25**
0.99	**6.63**	**9.21**	**11.30**



Calculating Confidence Limits
means
Exploring the Parameter Space - Statistical Surface



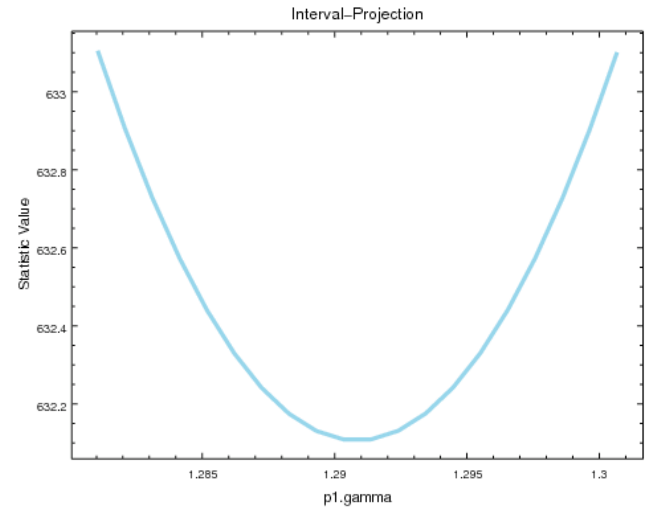


Example of a "well-behaved" statistical surface in parameter space, viewed as a multi-dimensional paraboloid (χ^2, *top*), and as a multi-dimensional Gaussian ($\exp(-\chi^2/2) \approx \mathcal{L}$, *bottom*).

Confidence Intervals

```
sherpa-39> covariance()
Dataset            = 1
Confidence Method     = covariance
Fitting Method       = levmar
Statistic           = chi2datavar
covariance 1-sigma (68.2689%) bounds:
  Param          Best-Fit  Lower Bound  Upper Bound
  -----          --------  -----------  -----------
  zabs1.nh       0.0960949  -0.00436915   0.00436915
  p1.gamma        1.29086  -0.00981129   0.00981129
  p1.ampl      0.000707365 -6.70421e-06  6.70421e-06
```

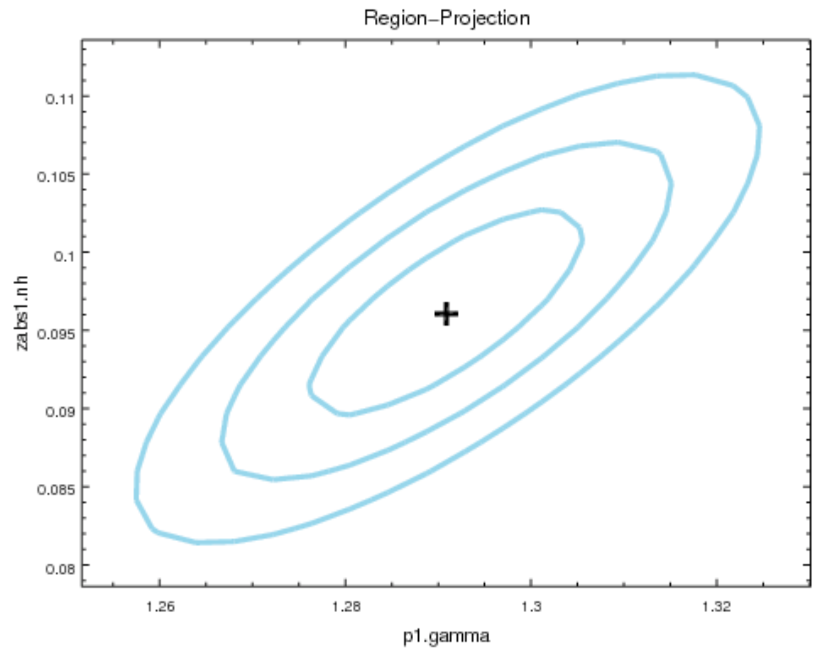
```
sherpa-40> projection()
Dataset            = 1
Confidence Method     = projection
Fitting Method       = levmar
Statistic           = chi2datavar
projection 1-sigma (68.2689%) bounds:
  Param          Best-Fit  Lower Bound  Upper Bound
  -----          --------  -----------  -----------
  zabs1.nh       0.0960949  -0.00435835   0.00439259
  p1.gamma        1.29086  -0.00981461   0.00983253
  p1.ampl      0.000707365 -6.68862e-06   6.7351e-06
```



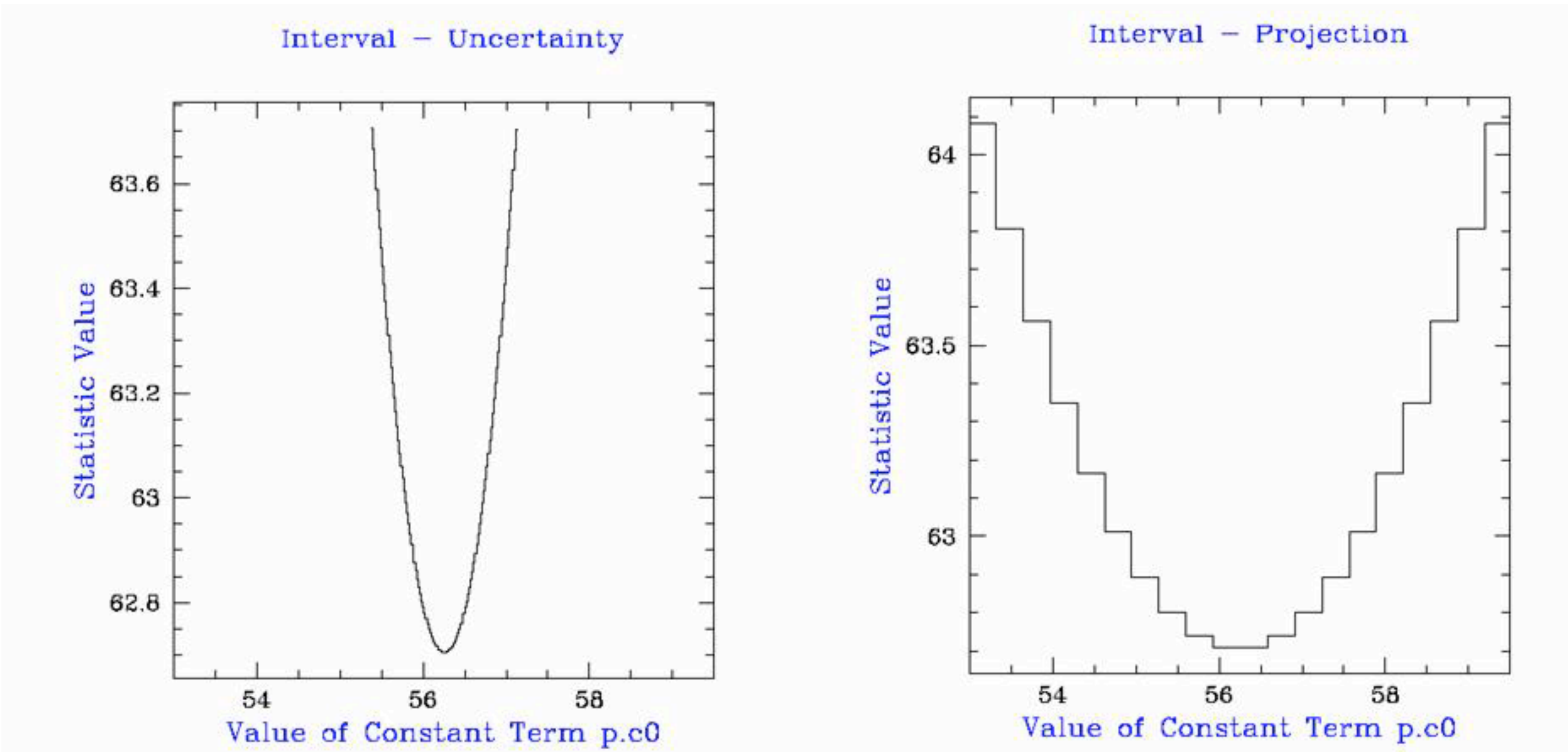

```
sherpa-48> print get_proj_results()
datasets   = (1,)
methodname = projection
fitname    = levmar
statname   = chi2datavar
sigma      = 1
percent    = 68.2689492137
parnames   = ('zabs1.nh', 'p1.gamma', 'p1.ampl')
parvals    = (0.0960948525609, 1.29085977295, 0.000707365006941)
parmins    = (-0.00435834667074, -0.00981460960484, -6.68861977704e-06)
parmaxes   = (0.0043925901652, 0.00983253275984, 6.73510303179e-06)
nfits      = 46
```

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]
```



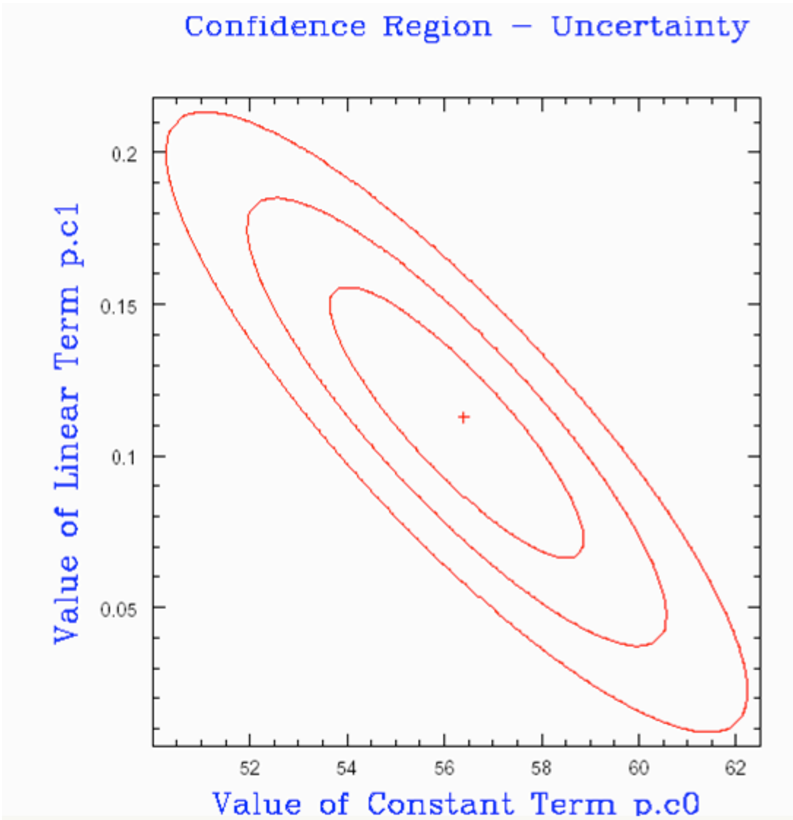
Behaviour of Statistics for One Parameter

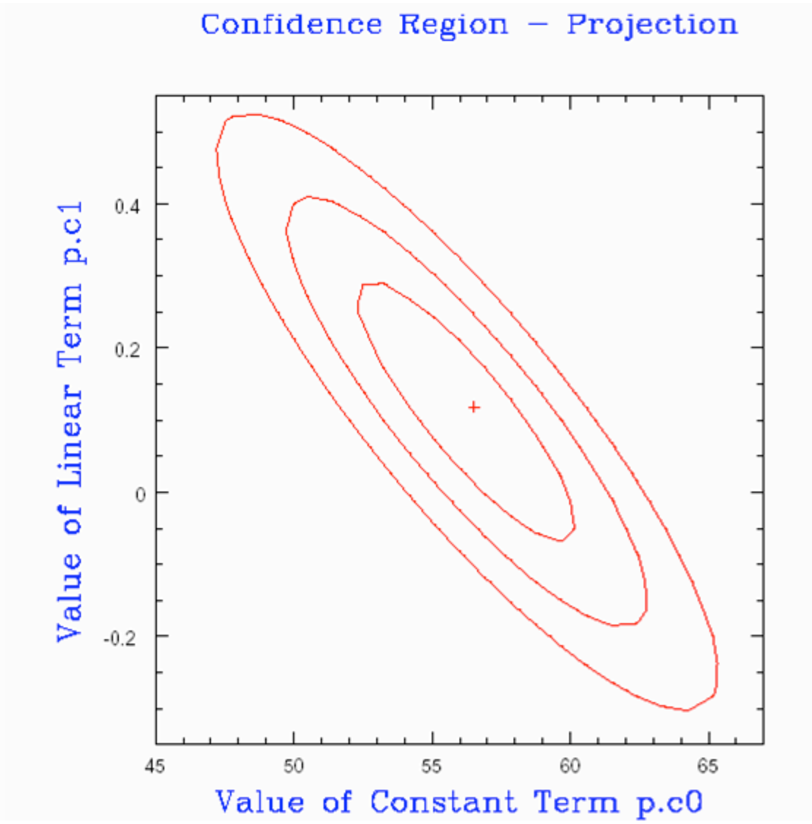


Comparison of Two methods in Sherpa

Confidence Limits for Two Parameters

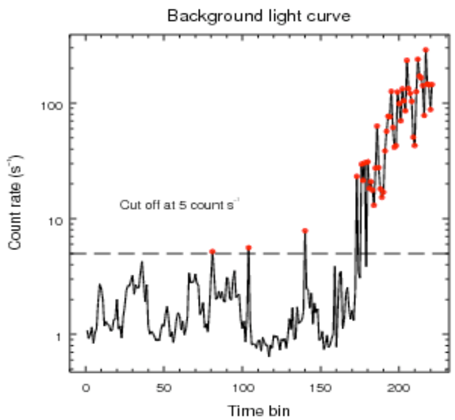
+ *Best fit parameters*
  1σ, 2σ, 3σ contours





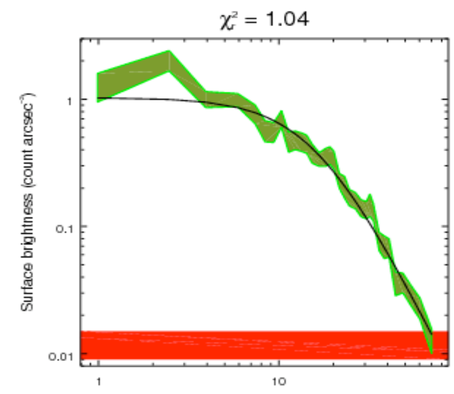
Comparison of Two methods in Sherpa

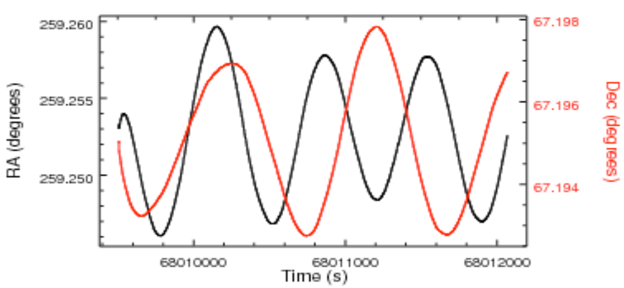
ChIPS Chandra Plotting Package

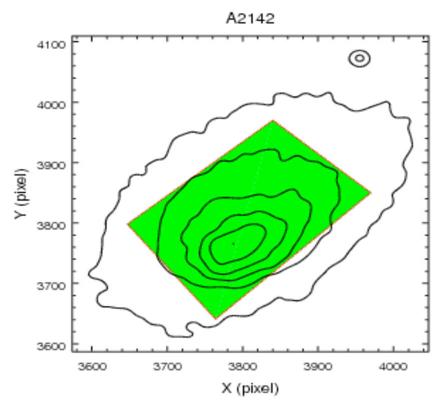


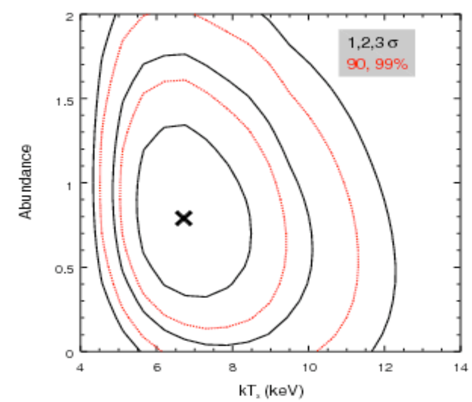
CIAO4 Infrastructure changed

- SM replaced with the VTK
- Modern graphics package
- Publication quality plots









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

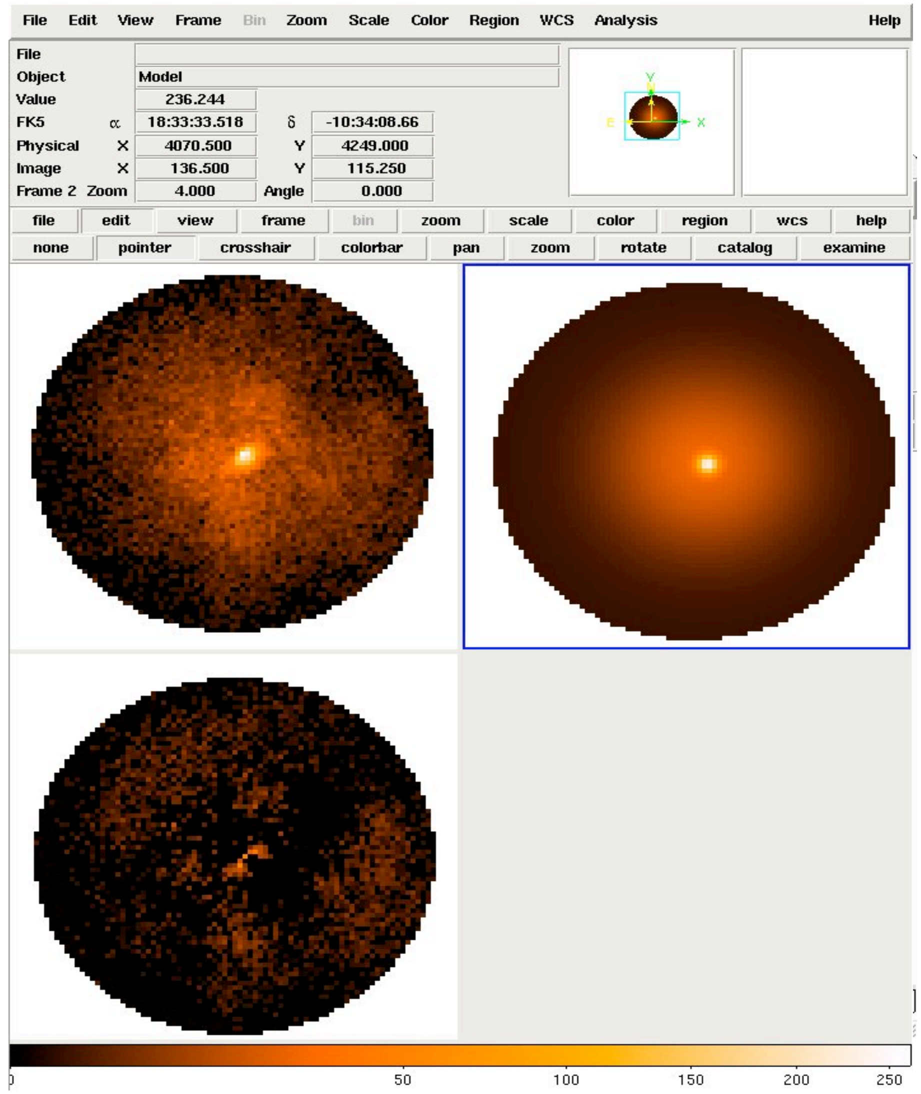
List of Sherpa Commands

Read Image data and Display in ds9	`load_image("image2.fits")` `image_data()` `set_coord("physical")` `notice2d("circle(4065.5,4250.5,78.8)")` `image_data()`
Set Statistics and Optimization Method	`set_stat("cash")` `set_method("neldermead")`
Define Model and Set Model parameters	`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, Display Get Confidence Range	`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()
```



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()
```

xterm

```
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

xterm

```
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>
```

Old CIAO

```
paramprompt off
guess off
method powell
polynom1d[yc]
yc integrate off
polynom1d[zc]
zc integrate off
yc.c0 = -65.0
zc.c0 = 1394.0
freeze yc.c0
freeze zc.c0
gridmodel[obs0_dtemp]
obs0_dtemp integrate off
obs0_dtemp.file = data/obs9593/delta_temp.dat
obs0_dtemp.norm = 1.6e-05
freeze obs0_dtemp.norm
polynom1d[obs0_dy]
obs0_dy integrate off
obs0_dy.c0 = 0.0
freeze obs0_dy.c0
thaw obs0_dy.c1
thaw obs0_dy.c2
data 1 "data/obs9593/mag_centroid_bin_0.fits[cols dt,yag]"
errors 1 = 0.020000
polynom1d[obs0_y0]
obs0_y0 integrate off
source 1 = obs0_dtemp * (obs0_y0 - yc) + obs0_dy + obs0_y0
data 2 "data/obs9593/mag_centroid_bin_1.fits[cols dt,yag]"
errors 2 = 0.020000
polynom1d[obs0_y1]
obs0_y1 integrate off
source 2 = obs0_dtemp * (obs0_y1 - yc) + obs0_dy + obs0_y1
data 3 "data/obs9593/mag_centroid_bin_2.fits[cols dt,yag]"
errors 3 = 0.020000
polynom1d[obs0_y2]
obs0_y2 integrate off
source 3 = obs0_dtemp * (obs0_y2 - yc) + obs0_dy + obs0_y2
fit 1,2,3
freeze obs0_dy
polynom1d[obs0_dz]
obs0_dz integrate off
obs0_dz.c0 = 0.0
freeze obs0_dz.c0
thaw obs0_dz.c1
thaw obs0_dz.c2
data 4 "data/obs9593/mag_centroid_bin_0.fits[cols dt,zag]"
errors 4 = 0.020000
polynom1d[obs0_z0]
obs0_z0 integrate off
source 4 = obs0_dtemp * (obs0_z0 - zc) + obs0_dz + obs0_z0
data 5 "data/obs9593/mag_centroid_bin_1.fits[cols dt,zag]"
errors 5 = 0.020000
polynom1d[obs0_z1]
obs0_z1 integrate off
source 5 = obs0_dtemp * (obs0_z1 - zc) + obs0_dz + obs0_z1
data 6 "data/obs9593/mag_centroid_bin_2.fits[cols dt,zag]"
errors 6 = 0.020000
polynom1d[obs0_z2]
obs0_z2 integrate off
```

New CIAO

```
#!/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('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
```

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

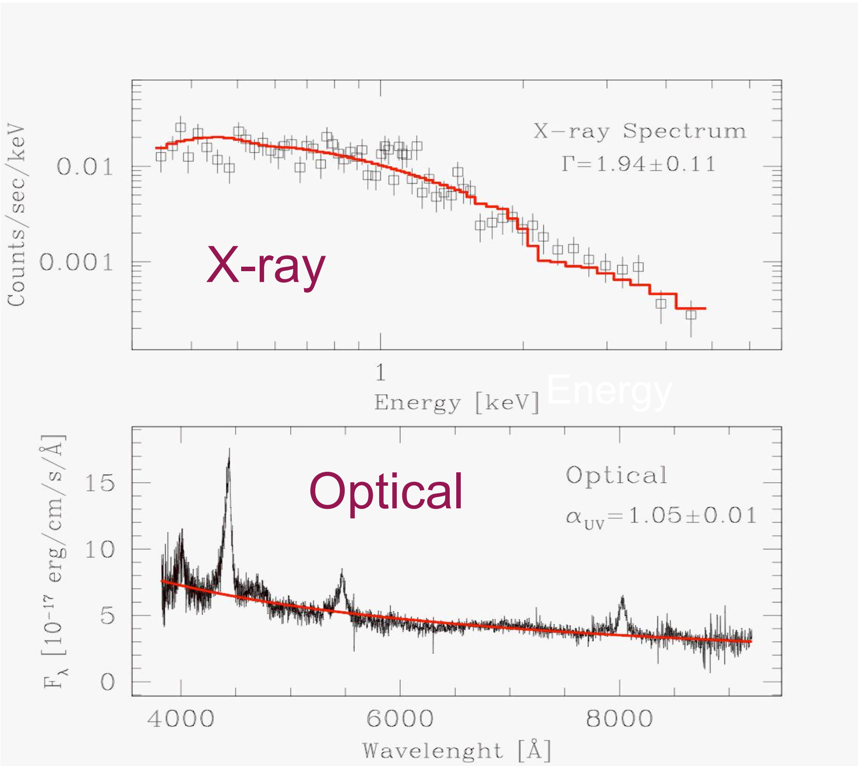
Units Conversion

```
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

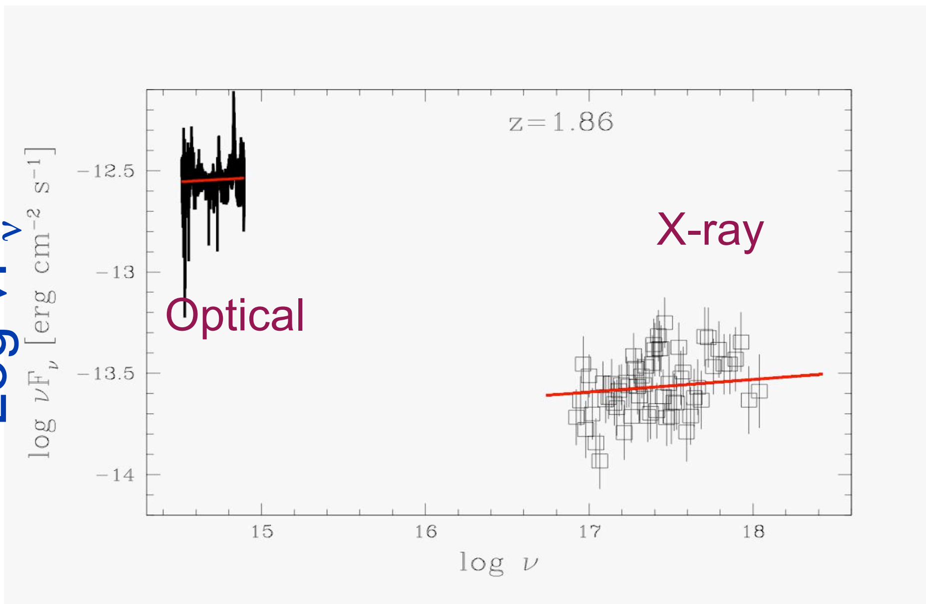
Flux



Wavelength

Quasar SED

Log νF_ν



Log ν

Learn more on Sherpa Web Pages

Last modified: 15 May 2008

Sherpa$_\beta$

Sherpa 3.4 Homepage
Sherpa 4.0 Beta Homepage
Choosing Python or S-Lang
About the Sherpa Beta Release
Analysis Threads
Sherpa
Science (CIAO)
ChIPS
ChaRT
Help Pages (AHELP)
Alphabetical
By context
Using ahelp
Models, Statistics, and Methods
Models
Statistics
Documentation
Bug List
CXC Links
CIAO (Data Analysis)
ChIPS (Plotting)
Astrostatistics Collaboration

News
Previous Items

15 May 2008 NEW

The Fitting FITS Image Data thread (S-Lang or Python) has been updated for Sherpa Beta.

Sherpa$_\beta$

CIAO's modeling and fitting package

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

The CIAO 4 release features an experimental (beta) version of the new Sherpa, the CIAO modeling and fitting package. Sherpa enables the user to construct complex models from simple definitions and fit those models to data, using a variety of statistics and optimization methods.

Sherpa is designed for use in a variety of modes: as a user-interactive application and in batch mode. Sherpa is an importable module for scripting languages (Python or S-Lang) and is available as a C/C++ library for software developers. In addition, users may write their own Python and S-Lang scripts for use in Sherpa. Refer to the Should I Use Sherpa in Python or S-Lang? page for help in deciding which language to use.

Since this is the initial phase of the Sherpa redesign, not all of the functionality of the CIAO 3.4 version is implemented yet. The About the Sherpa Beta Release page outlines new features and provides a summary of missing items. Please send feedback and questions on Sherpa Beta to the Helpdesk.

Citing *Sherpa* in a Publication

If you are writing a paper and would like to cite *Sherpa*, we recommend the following paper:

Sherpa: a mission-independent data analysis application (ADS)
P. E. Freeman, S. Doe, A. Siemiginowska
SPIE Proceedings, Vol. 4477, p.76, 2001

```
\bibitem[Freeman et al.(2001)]{2001SPIE.4477...76F} Freeman, P., Doe, S.,
\& Siemiginowska, A.\ 2001, \procspie, 4477, 76
```

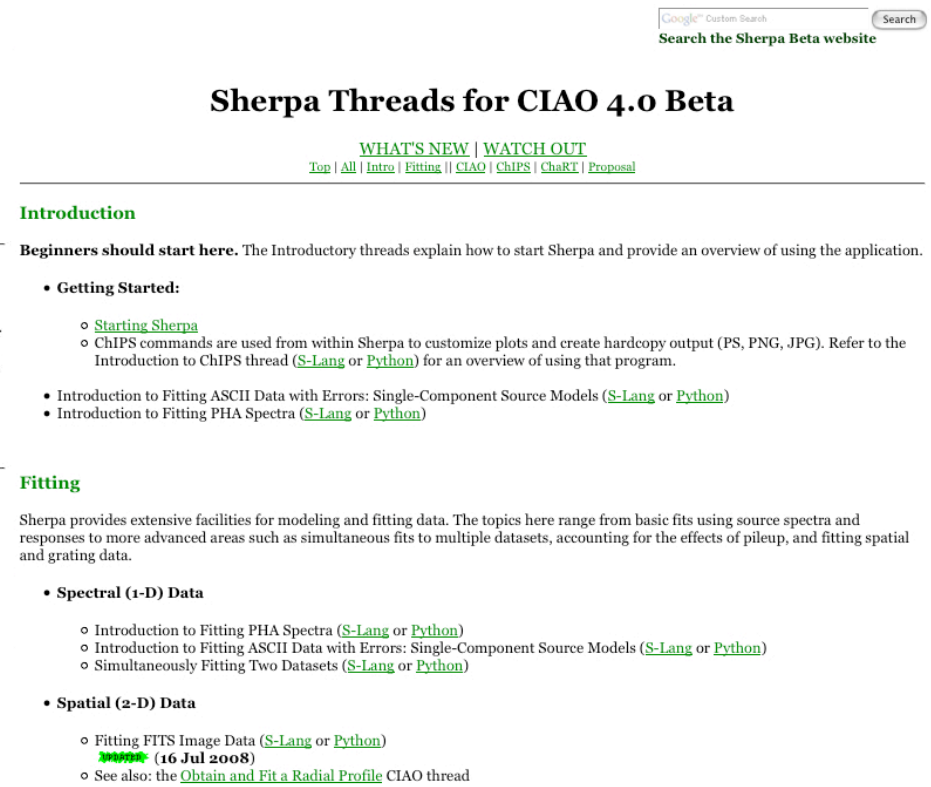
The specific version of CIAO and CALDB (if applicable) used for the analysis should be mentioned as well.

A reference for the Python interface to *Sherpa* is also available:

Developing Sherpa with Python (ADS)
S. Doe, et al.
Astronomical Data Analysis Software and Systems XVI, 376, 543

```
\bibitem[Doe et al.(2007)]{2007ASPC..376..543D} Doe, S., et al.\ 2007,
Astronomical Data Analysis Software and Systems XVI, 376, 543
```

Further guidelines are available from the Acknowledgement of Use of Chandra Resources.



Google Custom Search | Search
Search the Sherpa Beta website

Sherpa Threads for CIAO 4.0 Beta

WHAT'S NEW | WATCH OUT
Top | All | Intro | Fitting || CIAO | ChIPS | ChaRT | Proposal

Introduction

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

- **Getting Started:**
  - Starting Sherpa
  - ChIPS commands are used from within Sherpa to customize plots and create hardcopy output (PS, PNG, JPG). Refer to the Introduction to ChIPS thread (S-Lang or Python) for an overview of using that program.
- Introduction to Fitting ASCII Data with Errors: Single-Component Source Models (S-Lang or Python)
- Introduction to Fitting PHA Spectra (S-Lang or Python)

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 datasets, accounting for the effects of pileup, and fitting spatial and grating data.

- **Spectral (1-D) Data**
  - Introduction to Fitting PHA Spectra (S-Lang or Python)
  - Introduction to Fitting ASCII Data with Errors: Single-Component Source Models (S-Lang or Python)
  - Simultaneously Fitting Two Datasets (S-Lang or Python)
- **Spatial (2-D) Data**
  - Fitting FITS Image Data (S-Lang or Python) UPDATED **(16 Jul 2008)**
  - See also: the Obtain and Fit a Radial Profile CIAO thread

Freeman, P., Doe, S., & Siemiginowska, A.\ 2001, SPIE 4477, 76
Doe, S., et al. 2007, Astronomical Data Analysis Software and Systems XVI, 376, 543