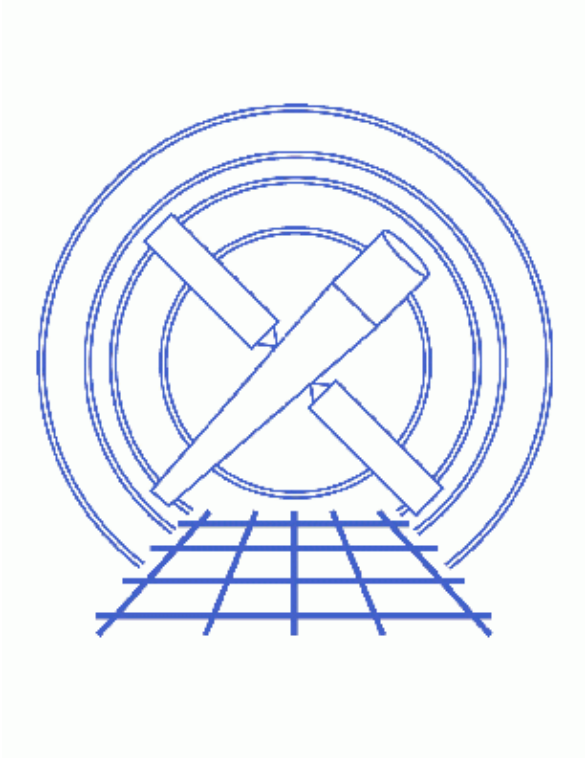
Accessing fit results using S−Lang



Sherpa Threads (CIAO 3.4)

Table of Contents

Accessing fit results using S−Lang

Sherpa Threads

Overview

Last Update: 1 Dec 2006 − reviewed for CIAO 3.4: no changes

Synopsis:

There are many S−Lang routines that can be used to access the fit results returned by *Sherpa*. Here we provide examples of some of the more common ones.

Read this thread if:

You want to access the fit results from *Sherpa*; perhaps to write to a file or use in a plot.

Related Links:

- The `sherpa-module` and `get` ahelp pages.
- The steps taken in this thread match those taken in the "Estimating Errors and Confidence Levels" and "Step−by−Step Guide to Estimating Errors and Confidence Levels " threads.

Proceed to the HTML or hardcopy (PDF: A4 | letter) version of the thread.

Getting Started

In this thread we repeat the steps taken in the two "Estimating Errors and Confidence Levels" threads − the version using the routines from `paramest.sl` and the version using the native *Sherpa* interface to the routines − and then show how the S−Lang functions in the *Sherpa* module can be used to access the fit results. Please see the above threads for a description of the various steps (here we use the "native" interface to the error calculations rather than the functions provided by `paramest.sl`).

The thread assumes that you are running the commands from the *Sherpa* shell; however, they will also work in a S−Lang script executed by `slsh` if you import *Sherpa* − i.e. `import("sherpa");` − before using any of the commands.

Find the best fit

Set up

Here we load in a dataset into *Sherpa*, set up the source model, and fit it (neglecting the the background component).

```
sherpa> erase all
sherpa> data source grouped pi.fits
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: statistical errors specified in the PHA file.
         These are currently IGNORED.  To use them, type:
         READ ERRORS "<filename>[cols CHANNEL,STAT_ERR]" fitsbin
RMF is being input from:
  /data/ciao/rmf.fits
ARF is being input from:
  /data/ciao/arf.fits
sherpa> ignore energy :0.5,8:
sherpa> show method
Optimization Method: Levenberg−Marquardt

      Name       Value         Min         Max                     Description
      ----       -----         ---         ---                     -----------
 1   iters        2000           1       10000    Maximum number of iterations
 2     eps       1e−03       1e−09           1               Absolute accuracy
 3   smplx           0           0           1  Refine fit with simplex (0=no)
 4 smplxep           1       1e−04        1000    Switch−to−simplex eps factor
 5 smplxit           3           1          20  Switch−to−simplex iters factor

sherpa> show statistic
Statistic:           Chi−Squared Gehrels
sherpa> source = xswabs[abs] * powlaw1d[p1]
abs.nH parameter value [0.1]
p1.gamma parameter value [1]
p1.ref parameter value [4]
p1.ampl parameter value [0.000149261]
sherpa> fit
 LVMQT: V2.0
 LVMQT: initial statistic value = 4583.05
 LVMQT: final statistic value = 83.2873 at iteration 6
           abs.nH  2.4061  10^22/cm^2
            p1.gamma  1.51851
            p1.ampl  0.000241434
```

Since we are using a chi−square statistic we can get an idea of how well the model fits using the `GOODNESS` command:

```
sherpa> goodness
Goodness: computed with Chi−Squared Gehrels

DataSet 1: 131 data points −− 128 degrees of freedom.
 Statistic value       = 83.2873
 Probability [Q−value] = 0.999225
 Reduced statistic     = 0.650682
```

Accessing the FIT results

The best−fit parameter values are displayed at the end of the fit − as shown above − but can also be displayed using the `list par` command:

```
sherpa> list_par
```

```
  #                 Name Type       Value Lnk Frz         Min         Max       Delta
  1              abs.nH  src      2.4061   0   0       1e−07          10          −1
  2            p1.gamma  src      1.5185   0   0         −10          10          −1
  3              p1.ref  src           4   0   1           1      7.4971          −1
  4             p1.ampl  src  0.00024143   0   0  1.4926e−06    0.014926          −1
  5 AutoReadResponse.rmf inst"/data/ciao/rmf.fits"
  6 AutoReadResponse.arf inst"/data/ciao/arf.fits"
```

The fit information can also be obtained using the `get_par()` function to get the parameter values.

```
sherpa> par = get_par("p1.gamma")
sherpa> print(par)
name             =  p1.gamma
model            =  powlaw
type             =  src
value            =  1.51851
min              =  −10
max              =  10
delta            =  −1
units            =  NULL
frozen           =  0
linked           =  0
linkexpr         =  NULL
```

The main fields of instrument are highlighted in bold above − the parameter name and its best−fit value. The other fields provide other information about the parameter − such as whether it is part of a source or instrument model and whether it is frozen or thawed − as described in the `get_par()` and `CREATE` ahelp pages.

Above we used `get_par()` to find information on one parameter ("`p1.gamma`"). If called with no argument it returns the results for all the defined parameters − an array of structures, one structure per parameter, in the order that `list_par()` uses:

```
sherpa> pars = get_par
sherpa> pars
Struct_Type[6]
sherpa> print(pars[0])
name             =  abs.nH
model            =  xswabs
type             =  src
value            =  2.4061
min              =  1e−07
max              =  10
delta            =  −1
units            =  10^22/cm^2
frozen           =  0
linked           =  0
linkexpr         =  NULL
sherpa> print(pars[1])
name             =  p1.gamma
model            =  powlaw
type             =  src
value            =  1.51851
min              =  −10
max              =  10
delta            =  −1
units            =  NULL
frozen           =  0
linked           =  0
linkexpr         =  NULL
sherpa> print(pars[2])
name             =  p1.ref
model            =  powlaw
type             =  src
value            =  4
```

```
min             =  1
max             =  7.4971
delta           =  −1
units           =  NULL
frozen          =  1
linked          =  0
linkexpr        =  NULL
```

The `get_fit()` routine returns a structure which contains information of the quality of the fit.

```
sherpa> good = get_fit()
sherpa> print(good)
dataset         =  1
datatype        =  source
stat            =  83.2873
numbins         =  131
dof             =  128
rstat           =  0.650682
qval            =  0.999225
```

These values can also be obtained individually:

```
sherpa> stat = get_statistic
sherpa> print(stat)
83.2873
sherpa> nbins = length(get_data)
sherpa> print(nbins)
131
sherpa> ndof = nbins − get_num_par_thawed
sherpa> print(ndof)
128
sherpa> get_qvalue(ndof,stat)
0.999225
```

Note that `get_num_par_thawed()` returns the *total number* of thawed parameters, whether or not they are part of the source expression. This means that adding extra models will change the value returned by `get_num_par_thawed()` −

```
sherpa> get_num_par_thawed
3
sherpa> paramprompt off
Model parameter prompting is off
sherpa> xsmekal[plasma]
sherpa> get_num_par_thawed
5
```

− and so care should be taken when calculating the number of degrees of freedom in a fit using the approach above.

Accessing other useful information about the fit

A number of other S−Lang may be of interest; here we show some of them in action with little commentary:

```
sherpa> get_method_expr
levenberg−marquardt
sherpa> get_stat_expr
chi gehrels
sherpa> get_source_expr
(abs * p1)
sherpa> cpts = strtok( get_source_expr, " ()*+−/" )
sherpa> print(cpts)
abs
p1
sherpa> print(get_defined_models)
```

```
abs
p1
plasma
sherpa> get_exptime
7854.47
sherpa> filters = get_filter_expr
sherpa> vmessage( "The first filter was '%s'", filters[0] )
The first filter was 'ignore source 1 energy : 0.5 , 8 : '
```

The "is" routines (see "ahelp is") may also prove useful.

Errors on individual parameters (projection)

Set up

We will use the projection method to estimate 1 sigma errors on the gamma parameter of the powerlaw component.

```
sherpa> restore_proj
sherpa> projection p1.gamma
Projection complete for parameter: p1.gamma

Computed for projection.sigma = 1
        ------------------------------------------------------------
        Parameter Name      Best−Fit Lower Bound     Upper Bound
        ------------------------------------------------------------
            p1.gamma         1.51851  −0.105572       +0.107951
```

Accessing the PROJECTION results

The results of the run can be accessed using the get_proj() routine, as shown below. This routine always returns an array of structures, even if there is only one element in the array.

```
sherpa> res = get_proj()
sherpa> print(res[0])
name             =  p1.gamma
val              =  1.51851
vlo              =  1.41294
vhi              =  1.62646
sigma            =  1
```

Here we re−run to get the 90% confidence limits (corresponding to 1.6 sigma) on both the power−law slope (gamma) and the column density of absorbing material (nH):

```
sherpa> sherpa.proj.sigma = 1.6
sherpa> projection p1.gamma abs.nh
Projection complete for parameter: abs.nH
Projection complete for parameter: p1.gamma

Computed for projection.sigma = 1.6
        ------------------------------------------------------------
        Parameter Name      Best−Fit Lower Bound     Upper Bound
        ------------------------------------------------------------
           abs.nH             2.4061  −0.240423       +0.260944
            p1.gamma         1.51851  −0.167618       +0.174267
```

Note that when we use get_proj() the order of the returned values need not match that specified at the prompt (the values above also have the abs.nH reported before the p1.gamma parameter):

```
sherpa> errs = get_proj
sherpa> print(errs[0])
name             =  abs.nH
val              =  2.4061
vlo              =  2.16568
vhi              =  2.66704
sigma            =  1.6
sherpa> print(errs[1])
name             =  p1.gamma
val              =  1.51851
vlo              =  1.35089
vhi              =  1.69278
sigma            =  1.6
```

Similar behaviour is seen if you use the UNCERTAINTY and COVARIANCE methods, although the name of the routines used to access the error estimates changes to `get_unc()` and `get_cov()` respectively.

How does the fit surface vary for a parameter (interval−projection)

Set up

Here we use INTERVAL−PROJECTION to see how the fit statistic varies with the `gamma` parameter of the power law component. Since we already know that the 90% errors are approximately `+− 0.2` we choose to set the axis range manually:

```
sherpa> restore_intproj
sherpa> sherpa.intproj.arange = 0
sherpa> sherpa.intproj.min = 1
sherpa> sherpa.intproj.max = 2
sherpa> intproj p1.gamma
Interval−Projection: grid size set by user.
                     outer grid loop 20% done...
                     outer grid loop 40% done...
                     outer grid loop 60% done...
                     outer grid loop 80% done...

sherpa> ticks maj y 10
sherpa> ticks min y 5
sherpa> redraw
```

The resulting plot looks like this (the calls to the `TICKS` command are to add extra numeric labels to the Y axis since the default settings for this plot are not too helpful). The "confidence intervals" table in "`ahelp projection`" list a range of common confidence levels and the corresponding change in chi−square values (i.e. the statistic value on the Y axis in this plot).

Accessing the INTERVAL−PROJECTION results

The data from this plot can be read into S−Lang variables using the `get_intproj()` routine and used to convert the y axis into "delta chi squared":

```
sherpa> iplot = get_intproj
sherpa> print(iplot)
x0               =  Double_Type[20]
y                =  Double_Type[20]
name             =  p1.gamma
bfit             =  1.51851
config           =  sherpa_VisParEst_State
```

```
sherpa> clear
sherpa> curve(iplot.x0,iplot.y-get_fit().stat)
0
sherpa> simpleline
sherpa> symbol none
sherpa> line 0 2.71 3 2.71
sherpa> ln 1 red
sherpa> xlabel "\gamma"
sherpa> ylabel "\Delta \Chi^2"
sherpa> xlabel size 1.4
sherpa> ylabel size 1.4
sherpa> tickvals size 1.4
sherpa> redraw
```

The resulting plot looks like this  . It is essentially identical to the interval−projection plot  except that the best−fit value is drawn at `y=0` (since we subtracted off `get_fit().stat` from the `iplot.y` values) and the points are connected by a straight line rather than drawn as a histogram.

How are two parameters correlated (region−projection)?

Set up

In this section we use the `REGION−PROJECTION` command of *Sherpa* to see whether the `gamma` and `nH` parameters are correlated. We "cheat" and go straight to the best set up obtained in the "Confidence limits" threads:

```
sherpa> restore regproj
sherpa> sherpa.regproj.arange = 0
sherpa> sherpa.regproj.min = [1.2,2]
sherpa> sherpa.regproj.max = [1.9,2.8]
sherpa> sherpa.regproj.nloop = [21,21]
sherpa> sherpa.regproj.sigma = [1,1.6]
sherpa> chips.mingridsize = 100
sherpa> regproj p1.gamma abs.nh
Region-Projection: computing grid size with covariance...done.
                   outer grid loop 20% done...
                   outer grid loop 40% done...
                   outer grid loop 60% done...
                   outer grid loop 80% done...
Minimum: 83.2873
Levels are: 85.5833 87.7093

(messages omitted)
```

Accessing the REGION−PROJECTION results

As with the other commands, the results are accessible from S−Lang, this time by the `get_regproj()` function. It returns a structure which contains the axes values − as 1D arrays in the fields `x0` and `x1` − the statistic value at each point − in the `y` field − and an array of values giving the contour levels in the `levels` field:

```
sherpa> conf = get_regproj
sherpa> print(conf)
x0              =  Double_Type[441]
x1              =  Double_Type[441]
y               =  Double_Type[441]
levels          =  Double_Type[2]
```

```
name             =  String_Type[2]
bfit             =  Double_Type[2]
config           =  sherpa_VisParEst_State
sherpa> print(conf.levels)
85.5833
87.7093
sherpa> writeascii(stdout,conf.x0[[0:4]],conf.x1[[0:4]],conf.y[[0:4]])
1.2     2       92.8386
1.2     2.04    93.1307
1.2     2.08    94.0888
1.2     2.12    95.662
1.2     2.16    97.8023
```

where we have used the `writeascii()` routine to print out the first 5 rows of the `x0`, `x1`, and `y` fields.

History

14 Jan 2005	reviewed for CIAO 3.2: no changes
21 Dec 2005	reviewed for CIAO 3.3: no changes
01 Dec 2006	reviewed for CIAO 3.4: no changes

URL: http://cxc.harvard.edu/sherpa/threads/get_fit_results/ Last modified: 1 Dec 2006

Image 1: Plot of interval−projection results

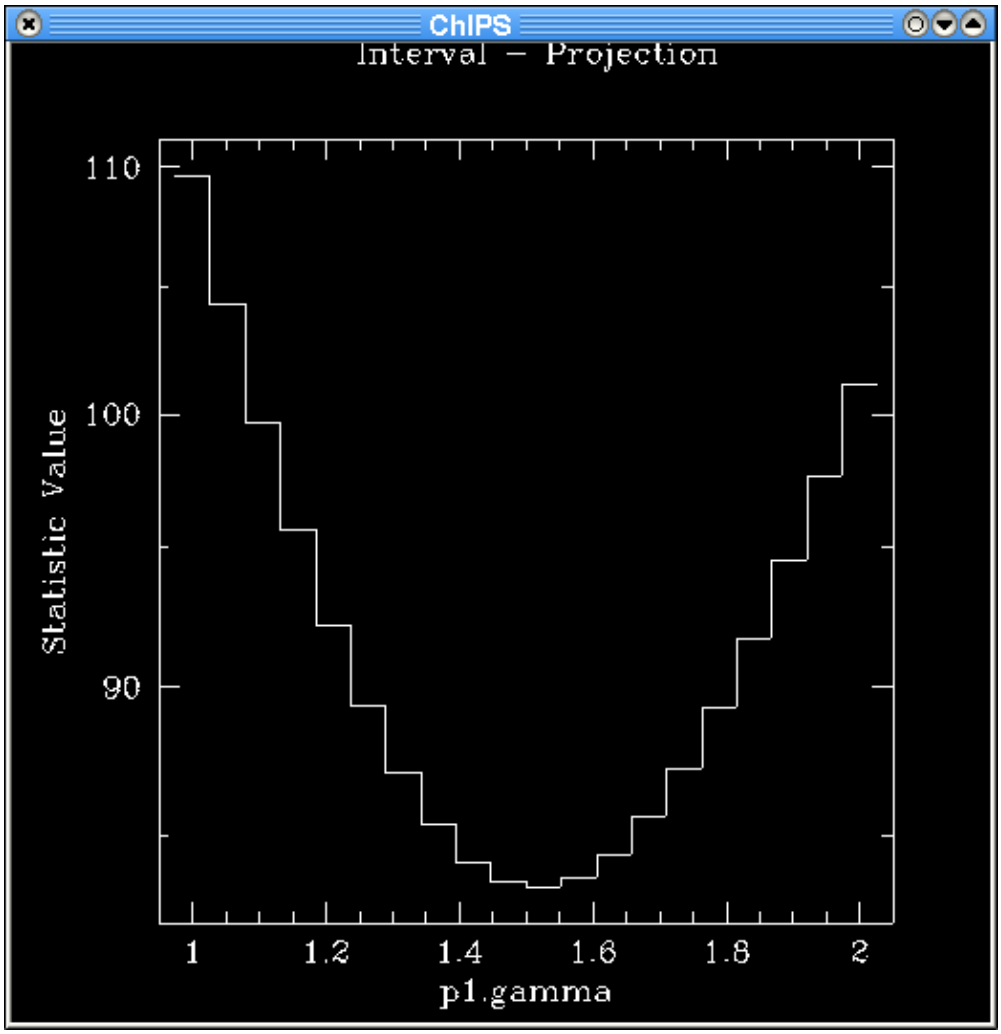
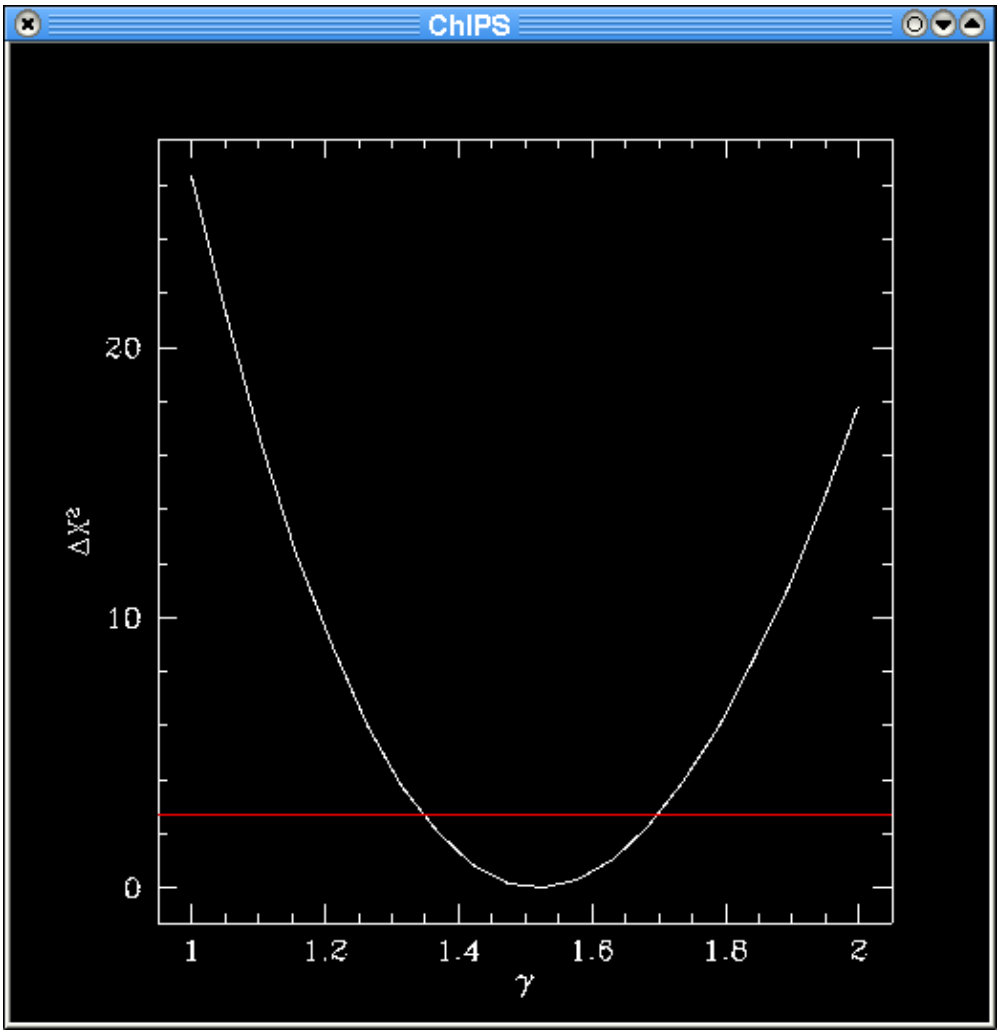


Image 2: Interval−projection results converted to a delta chi squared plot



The red line is drawn at a delta chi−squared value of 2.71, which corresponds to the 90% confidence level. See the "confidence intervals" table in "ahelp projection" for a range of common confidence levels and the corresponding changes in statistic value.