Chandra X-ray Center

URL: http://cxc.harvard.edu/ciao3.4/paramest.html
Last modified: December 2006

AHELP for CIAO 3.4

paramest

Context: sherpa

Jump to: Description Examples Parameters NOTES CHANGES Bugs See Also

Synopsis

An interactive interface to the parameter estimation routines in Sherpa.

Syntax

```
rproj( parlist [, option1, option2, ... ] )
runc( parlist [, option1, option2, ... ] )
iproj( par [, option1, option2, ... ] )
iunc( par [, option1, option2, ... ] )
proj( parlist [, option1, option2, ... ] )
unc( parlist [, option1, option2, ... ] )
cov( parlist [, option1, option2, ... ] )
```

Description

The routines presented here provide an interactive interface to Sherpa's parameter−estimation routines. This means that Sherpa will ask you for the values it needs − in a manner similar to the way CIAO tools prompt for parameters − rather than relying on you setting them from the Sherpa prompt prior to calling the routine. There are interactive versions of: PROJECTION, UNCERTAINTY, COVARIANCE, INTERVAL−PROJECTION, INTERVAL−UNCERTAINTY, REGION−PROJECTION, and REGION−UNCERTAINTY. If the CIAO contributed software package is installed, as described in the NOTES section below, then the routines can be made available by calling

```
require("paramest");
```

in Sherpa.

The available routines are:

Routine	Arguments	Sherpa command
rproj()	parlist [,options]	REGION−PROJECTION
runc()	parlist [,options]	REGION−UNCERTAINTY
iproj()	par [,options]	INTERVAL−PROJECTION
iunc()	par [,options]	INTERVAL−UNCERTAINTY
proj()	parlist [,options]	PROJECTION
unc()	parlist [,options]	UNCERTAINTY
covar()	parlist [,options]	COVARIANCE

Overview

When one of the routines is called, it first checks to see if the parameters in question are not frozen and then displays their current best−fit values. The routine will then asks you for the other values to use. It will only ask you those values it needs − so if you choose to automatically calculate the limits in rproj() it will not ask you for the minimum and maximum values as well. It also takes advantage of the parameter library to set minimum and maximum values appropriate to the hard limits for each parameter − e.g. when manually setting the limits for rproj(). The parameter names are given as either the parlist or par parameters in the table above:

- parlist means a space− or comma−separated list of parameter names, such as "clus.kt gal.nh". For rproj() and runc() there must be two parameter names in the list, for the other routines there can be one or more or, if "", all thawed parameters will be used.
- par means a string containing the parameter name, such as "clus.kt".

Using a command

Once the routines have been installed (see the CIAO scripts page for instructions), the routines can be used instead of the corresponding Sherpa command in the table above. For instance, after fitting a spectrum with a model defined as:

```
source = xsmekal[clus] * xsphabs[gal]
```

then you can PROJECTION on the temperature parameter of the XSMEKAL model by saying:

```
  sherpa> require("paramest")
  sherpa> proj("clus.kt")
  Parameter name:        Curr Val        Hard Min/Max
  clus.kt                 8.52597       0.008         64

  Report error at this number of sigma from the best−fit value (0:) (1): 1.6
  Projection complete for parameter: clus.kT

  Computed for projection.sigma = 1.6
          --------------------------------------------------------
          Parameter Name      Best−Fit Lower Bound     Upper Bound
          --------------------------------------------------------
            clus.kT            8.52597  −1.70505        +2.73444


  The PROJECTION run has finished for clus.kt.
  The get_proj() command can be used to access the data
  of this run.
```

or REGION−UNCERTAINTY on the temperature and nH parameters by saying:

```
  sherpa> runc("clus.kt gal.nh")
  Parameter name:        Curr Val        Hard Min/Max
  clus.kt                 6.57998       0.008         64
  gal.nh                0.0904323       1e−07         10

  Should the limits be calculated automatically? (yes):
  Multiply estimated grid limits by this factor (3): 2
  Should the X−axis be evaluated using logarithmic spacing? (no):
  Number of points to evaluate along the X axis (1:) (10):
  Should the Y−axis be evaluated using logarithmic spacing? (no):
  Number of points to evaluate along the Y axis (1:) (10):
  Comma−separated list of contour levels in units of sigma (1,2,3):
  Region−Uncertainty: computing grid size...done.
                      outer grid loop 20% done...
                      outer grid loop 40% done...
                      outer grid loop 60% done...
                      outer grid loop 80% done...
  Minimum: 65.5876
  Levels are: 67.8836 71.7686 77.4176

  The REGION−UNCERTAINTY run has finished for clus.kt and gal.nh.
  The get_regunc() command can be used to access the surface data
  of this run.
```

The numerical values reported above will be different for your dataset. Note that the

```
require("paramest")
```

line is only required once per session; it can be added to your ~/.sherparc file so that it is available to every Sherpa session you start.

Setting values and optional parameters

The default values from the prompts are taken from the approproate fields in the state object; for instance sherpa.cov.sigma for the sigma parameter when the function cov() is called. It is also possible to set the parameter values when you call the function using a similar approach to that of command−line tools (i.e. "ahelp parameter"). The following rproj() call sets the auto parameter to no and the sigma parameter to "1,1.6" (without prompting for them):

```
  sherpa> rproj("clus.kt gal.nh","auto−","sigma=1,1.6")
```

A 'plist paramest' will list these parameters and indicate how they relate to the Sherpa state object, but this should not be needed by most users since the routines prompt you for the required values. The parameter section of this document ("ahelp −b PARAM paramest") also gives this information.

It is expected that this functionality will not be used much as it is somewhat clumsy. It does allow the use of hidden parameters − discussed next − and the use of "mode=h" − e.g.

```
  sherpa> rproj("clus.kt gal.nh","auto−","mode=h")
```

− to stop the prompting of parameter values (useful if you want to re−run a function with the same settings as the last run).

Hidden parameters

Two classes of routines have hidden parameters − i.e. they can be explicitly set but will not be prompted for when the function is run. The routines are:

The unc() routine

The hidden parameters are tolsig and deltasig (see the parameter list below for more information).

Projection routines

The rproj(), iproj(), and proj() routines have a hidden parameter called "fastopt" (see the parameter list below for more information).

These parameters were chosen to be hidden since it is likely that most users will not want to change them. If you do want to change them then you have two choices: provide them as part of the optional parameter list − e.g.

```
  sherpa> rproj("clus.kt gal.nh","fastopt−")
```

− or set the value in the corresponding state object before calling the routine − e.g.

```
  sherpa> sherpa.regproj.fast = 0
  sherpa> rproj("clus.kt gal.nh")
```

More examples can be found in the Examples section and the "Estimating Errors and Confidence Levels" Sherpa thread.

Example 1

```
sherpa> rproj()
Usage: rproj( parlist [, arg1, ..., argN ] )

where parlist are the 2 parameter names (e.g. "clus.kt gal.nh").
```

All the functions will display a brief usage message if called with no arguments. It is best to include the "()" when doing this to make sure that the command is interpreted as a S−Lang function and not as an abbreviation of a Sherpa command.

Example 2

```
sherpa> rproj("clus.kt gal.nh")
```

This function runs Sherpa's REGION−PROJECTION command for the parameters clus.kt and gal.nh.

The user will be prompted for all the values needed by the command, except for the "fastopt" parameter, whose value is taken from the sherpa.regproj.fast variable.

Example 3

```
sherpa> rproj("clus.kt gal.nh","auto−")
```

This is similar to the previous example except that the auto parameter is set to no and will not be asked for.

Example 4

```
sherpa> sherpa.regproj.arange = 0
sherpa> rproj("clus.kt gal.nh")
```

This is the same as the previous example except that the auto parameter will still be prompted for − although its default value will be set to no (i.e. 0).

Example 5

```
sherpa> rproj("clus.kt gal.nh","auto−","fast+","sigma=1,1.6")
```

Here we set the auto parameter to no, fastopt to yes, and sigma to "1,1.6". The same syntax for setting parameters will work for all the routines (although the set of parameters relevant to each routine is not the same).

Example 6

```
sherpa> runc("clus.kt gal.nh")
```

This function runs Sherpa's REGION−UNCERTAINTY command for the parameters clus.kt and gal.nh.

Example 7

```
sherpa> iproj("clus.kt")
```

This function runs Sherpa's INTERVAL−PROJECTION command for the parameter clus.kt.

Example 8

```
sherpa> iunc("clus.kt")
```

This function runs Sherpa's INTERVAL−UNCERTAINTY command for the parameter clus.kt.

Example 9

```
sherpa> proj("clus.kt")
```

This function runs Sherpa's PROJECTION command for the parameter clus.kt.

Example 10

```
sherpa> proj("clus.kt gal.nh")
```

This function runs Sherpa's PROJECTION command for the parameters clus.kt and gal.nh.

Example 11

```
sherpa> proj("")
```

This function runs Sherpa's PROJECTION command for all thawed parameters.

Example 12

```
sherpa> unc("clus.kt clus.abund gal.nh")
```

This function runs Sherpa's UNCERTAINTY command for the parameters clus.kt, clus.abund, and gal.nh.

Example 13

```
sherpa> cov("clus.kt")
```

This function runs Sherpa's COVARIANCE command for the parameter clus.kt.

Parameters

name	type	def	min
auto	boolean	yes	
scale	real	3	
xmin	real	0	
xmax	real	0	
xlog	boolean	no	
xnum	integer	10	1
ymin	real	0	
ymax	real	0	
ylog	boolean	no	
ynum	integer	10	1
sigma	real	1	1
contours	string	1,2,3	
tolsig	real	0.01	0
deltasig	real	0.01	0
fastopt	boolean	yes	

Detailed Parameter Descriptions

Parameter=auto `(boolean default=yes)`

Should the limits be calculated automatically?

Used by rproj(), runc(), iproj(), and iunc() to determine whether the limits should be calculated automatically (method depends on the routine) or specified by the user as the xmin and xmax parameters (and ymin/max for the rproj() and runc() functions).

Corresponds to the arange field of the sherpa.regproj, sherpa.regunc, sherpa.intproj, and sherpa.intunc state objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=scale `(real default=3)`

Multiply estimated grid limits by this factor

Used by rproj(), runc(), and iproj() if auto is true to calculate the grid limits. estimated.

Corresponds to the expfac field of the sherpa.regproj, sherpa.regunc, sherpa.intproj, and sherpa.intunc state objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=xmin `(real default=0)`

Minimum value for X−axis

Used by rproj(), runc(), iproj(), and iunc() if auto is false to find the minimum value of the parameter for the grid. The minimum and maximum values for this value will be set to match the hard limits for the model parameter and the prompt will be changed to include the parameter name.

Corresponds to the min[0] field of the sherpa.regproj and sherpa.regunc objects and the min field of the sherpa.intproj and sherpa.intunc state objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=xmax `(real default=0)`

Maximum value for X−axis

Used by rproj(), runc(), iproj(), and iunc() if auto is false to find the maximum value of the parameter for the grid. The minimum and maximum values for this value will be set to match the hard limits for the model parameter and the prompt will be changed to include the parameter name.

Corresponds to the max[0] field of the sherpa.regproj and sherpa.regunc objects and the max field of the sherpa.intproj and sherpa.intunc state objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=xlog `(boolean default=no)`

Should the X−axis be evaluated using logarithmic spacing?

Used by rproj(), runc(), iproj(), and iunc() to find out whether the X−axis should be evaluated using logarithmic, rather than linear, spacing between the points.

Corresponds to the log[0] field of the sherpa.regproj and sherpa.regunc objects and the log field of the sherpa.intproj and sherpa.intunc state objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=xnum `(integer default=10 min=1)`

Number of points to evaluate along the X axis

Used by rproj(), runc(), iproj(), and iunc() to find out the number of grid points to use along the X−axis.

Corresponds to the nloop[0] field of the sherpa.regproj and sherpa.regunc objects and the nloop field of the sherpa.intproj and sherpa.intunc state objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=ymin `(real default=0)`

Minimum value for Y−axis

Used by rproj() and runc() if auto is false to find the minimum value of the parameter for the grid. The minimum and maximum values for this value will be set to match the hard limits for the model parameter and the prompt will be changed to include the parameter name.

Corresponds to the min[1] field of the sherpa.regproj and sherpa.regunc objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=ymax `(real default=0)`

Maximum value for Y−axis

Used by rproj() and runc() if auto is false to find the maximum value of the parameter for the grid. The minimum and maximum values for this value will be set to match the hard limits for the model parameter and the prompt will be changed to include the parameter name.

Corresponds to the max[1] field of the sherpa.regproj and sherpa.regunc objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=ylog `(boolean default=no)`

Should the Y−axis be evaluated using logarithmic spacing?

Used by rproj() and runc() to find out whether the Y−axis should be evaluated using logarithmic, rather than linear, spacing between the points.

Corresponds to the log[1] field of the sherpa.regproj and sherpa.regunc objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=ynum `(integer default=10 min=1)`

Number of points to evaluate along the Y axis

Used by rproj() and runc() to find out the number of grid points to use along the Y−axis.

Corresponds to the nloop[1] field of the sherpa.regproj and sherpa.regunc objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=sigma `(real default=1 min=1)`

Report error at this number of sigma from the best−fit value

Used by iproj(), iunc(), proj(), unc(), and cov() in the computation of confidence intervals.

Corresponds to the sigma field of the sherpa.intproj, sherpa.intunc, sherpa.proj, sherpa.unc, and sherpa.cov objects. See 'ahelp sherpa.intproj' (or the relevant object) for more details.

Parameter=contours `(string default=1,2,3)`

Comma−separated list of contour levels in units of sigma

Used by rproj() and runc() to determine at what level to draw the confidence contours.

Corresponds to the sigma field of the sherpa.regproj and sherpa.regunc objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

Parameter=tolsig `(real default=0.01 min=0)`

Tolerance for sigma (smaller means more accurate errorbars)

Used by unc() and is a hidden parameter, in that it will not be prompted for by the routine. To set its value call unc() with it as a parameter or set sherpa.unc.eps beforehand.

Corresponds to the eps field of the sherpa.unc object. See 'ahelp sherpa.unc' for more details.

Parameter=deltasig `(real default=0.01 min=0)`

Redo best fit if find sigma value > deltasig below old best fit

Used by unc() and is a hidden parameter, in that it will not be prompted for by the routine. To set its value call unc() with it as a parameter or set sherpa.unc.remin beforehand.

Corresponds to the remin field of the sherpa.unc object. See 'ahelp sherpa.unc' for more details.

Parameter=fastopt `(boolean default=yes)`

Use a fast optimisation method (LM or SIMPLEX) rather than METHOD setting?

Used by rproj(), iproj(), and proj() to determine whether or not to switch to a fast optimisation method during the run. In general you will want to leave this as yes, unless the fit surface is complicated (at which point you need to be careful in interpreting error estimates whatever method used).

It is a hidden parameter, in that it will not be prompted for by the routine. To set its value call the function with it as a parameter or set its value − e.g. sherpa.regproj.fast − beforehand.

Corresponds to the fast field of the sherpa.regproj, sherpa.intproj, and sherpa.proj objects. See 'ahelp sherpa.regproj' (or the relevant object) for more details.

NOTES

This script is not an official part of the CIAO release but is made available as "contributed" software via the CIAO scripts page. Please see the installation instructions page for help on installing the package.

Parameter values and the Sherpa state object

The Sherpa commands called be the functions described here can be configured using fields in the Sherpa state object (also called a configuration variable). The routines described here hide this interface from the user, instead using the parameter library to provide a more interactive, tool−like, interface. However, they will pick up the current values from the state object and use them as the default values in the parameter prompts, as shown in the example below:

```
sherpa> sherpa.cov.sigma = 1.6
sherpa> cov("p1.gamma")
Parameter name:        Curr Val           Limits
p1.gamma                1.51866         −10         10

Report error at this number of sigma from the best−fit value (0:) (1.6):

Computed for covariance.sigma = 1.6
...
```

The names of the parameters of the functions do not exactly match the fields of the state objects (e.g. sherpa.regproj.nloop is represented by the xnum and ynum parameters here). See the parameter description above for the relationship between the two sets of names.

CHANGES

Version 1.12 (CIAO 3.2)

The script has been updated to account for fixes made to run_regproj(), run_regunc(), run_intproj(), and run_intunc() in CIAO 3.2. The only user−visible change is that iproj() and iunc() will no longer produce the following warning message:

```
  Invalid Parameter: struct has no field named config
```

The script can now be loaded by saying

```
require("paramest");
```

rather than having to use

```
() = evalfile("paramest.sl");
```

The old method will still work.

Bugs

Parameter values appear in the Sherpa history buffer

Once a routine has finished, and responses that were made to the parameter prompts will be added to the Sherpa history buffer. This means that if you use the "up arrow" to repeat any commands you will first have to scroll through all the responses you made.

See Also

chandra
: guide

sherpa
: bye, calc_kcorr, dataspace, dcounts, dollarsign, echo, eflux, eqwidth, erase, flux, get, get_dcounts_sum, get_dir, get_eflux, get_eqwidth, get_filename, get_flux2d, get_flux_str, get_lfactorial, get_mcounts_sum, get_pflux, get_source_components, get_verbose, groupbycounts, guess, is, journal, list, list_par, mcounts, numbersign, plot_eprof, plot_rprof, prompt, reset, run, set, set_analysis, set_axes, set_coord, set_dataspace, set_dir, set_verbose, setplot, sherpa−module, sherpa_plotfns, sherpa_utils, show, simspec, use, version

URL:
http://cxc.harvard.edu/ciao3.4/paramest.html
Last modified: December 2006