

Using A Pileup Model - Sherpa 4.1 (S-Lang)

Using A Pileup Model

Sherpa Threads (CIAO 4.1)

[S-Lang Syntax]

Table of Contents

- **Background Information**
 - ◆ Remove the acis detect afterglow Correction
- **Getting Started**
- **Reading in Data & Instrument Responses**
- **Defining the Models**
 - ◆ Multi-Component Source Model
 - ◆ Pileup Model
- **Fitting with the Neldermead and Monte Carlo Optimization Methods**
 - ◆ Fit using the Neldermead method
 - ◆ Fit using the Monte Carlo method
 - ◆ Calculate the parameter uncertainties
- **Examining the Pileup Fraction**
- **Saving the Fit Results**
- **Scripting It**
- **History**
- **Images**
 - ◆ Figure 1: Source Spectrum
 - ◆ Figure 2: Energy-filtered source spectrum
 - ◆ Figure 3: Plot of the fit and residuals

Using A Pileup Model

Sherpa Threads (CIAO 4.1)

[S-Lang Syntax]

Overview

Last Update: 29 Apr 2009 - new [script](#) command is available with CIAO 4.1.2

Synopsis:

This thread describes how to include a pileup model in the model expression when fitting data in *Sherpa*. The [Background Information](#) section contains information on how the data should be processed in order to run this thread.

Related Links:

- [The Chandra ABC Guide to Pileup](#) (PS, 23 pages)
- [Why Topics on pileup](#)
- [A comparison, for low pileup fractions, of the pileup models in Sherpa and Xspec with that in ISIS.](#)

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

Background Information

Experience with the [jdpileup](#) model has chiefly been for on-axis point sources, and in this thread, this condition is assumed. Even when analyzing piled data from a point-source, keep in mind that events in the core of the Point Spread Function (PSF) may be severely piled up while events in the PSF wings may be essentially unpiled, so generally, the fraction of the measuring aperture deemed to be piled up (f model parameter) is less than 1.

The standard procedure for [jdpileup](#) is:

1. **Remove the [acis_detect_afterglow](#) correction**, if applicable. [The next section](#) explains this step in more detail.
2. Reprocess the data with a [new bad pixel file](#) and apply the standard event filtering to create a level=2 event file; this is illustrated in the [Create a New Level=2 Event File](#) thread.
3. Extract a source spectrum from a circular region, usually about 2 arcsec in radius (e.g. by following the [psextract](#) thread).
4. Use the [pileup model in Sherpa](#) to fit a spectral model.

For a detailed discussion of the model, see [Davis \(2001\)](#).

Remove the `acis_detect_afterglow` Correction

An afterglow is the residual charge from the interaction of a cosmic ray in a CCD. [Standard data processing](#) (SDP, aka "the pipeline") uses the tool `acis_detect_afterglow` to flag possible cosmic ray events in the level 1 event file; these are then filtered out in the level 2 event file. It has been determined, however, that 3-5 % of the valid source photons may be rejected from diffracted spectra. These rejections, though a small fraction of the total events, are systematic and non-uniform.

In order to accurately model the pileup, it is necessary to remove this correction so that no source photons are eliminated from the event file. Read the [Pileup Talk](#) for more information, in particular [Data Preparation and Caveats](#). The [Remove the acis_detect_afterglow correction thread](#) shows how to undo the afterglow filtering.

Important Note

A new, more precise method for identifying afterglow events was introduced to SDP at version DS 7.4.0. If your data was processed with DS 7.4.0 or later, `acis_detect_afterglow` was not run in the pipeline. **The new hot pixel tools are more judicious with respect to throwing away piled source events. Users should be able to analyze the data without having to unfilter events first.**

The [Get Started section](#) of the [Remove the acis_detect_afterglow correction thread](#) shows how to check the processing version used for your data.

Getting Started

Sample ObsID used: 1618 (ACIS-S, NGC 4258)

The files used in this example were created by following these [CIAO threads](#):

- [Using psextract to Extract ACIS Spectra and Response Files for Point-like Sources](#)
- [Grouping a Grating Spectrum](#)

Here is a list of all the necessary files:

```
source_bin10.pi
background.pi
arf.fits
rmf.fits
```

The data files are available in [sherpa.tar.gz](#), as explained in the [Sherpa Getting Started thread](#).

Reading in Data & Instrument Responses

The spectra that will be used in this session have already been binned by a factor of 10. The data set is input to *Sherpa* with the `load_pha` command:

```
sherpa> load_pha("source_bin10.pi");
statistical errors were found in file 'source_bin10.pi'
but not used; to use them, re-read with use_errors=True
read ARF file arf.fits
```

Using A Pileup Model - Sherpa 4.1 (S-Lang)

```
read RMF file rmf.fits
statistical errors were found in file 'background.pi'
but not used; to use them, re-read with use_errors=True
read background file background.pi
```

Since the response files are defined in the header of `source_pha.fits`, the instrument response is automatically established for use:

```
sherpa> show_all();
Data Set: 1
Filter: 0.0737-14.9212 Energy (keV)
Noticed Channels: 1.0-1024.0
name           = source_bin10.pi
channel        = Float64[1024]
counts        = Float64[1024]
staterror     = None
syserror      = None
bin_lo        = None
bin_hi        = None
grouping      = Int16[1024]
quality       = Int16[1024]
exposure      = 20651.1490345
backscal      = 8.22673825187e-07
areascal      = 1.0
grouped       = True
subtracted    = False
units         = energy
response_ids  = [1]
background_ids = [1]

RMF Data Set: 1:1
name          = rmf.fits
detchans     = 1024
energ_lo     = Float64[1077]
energ_hi     = Float64[1077]
n_grp       = Int16[1077]
f_chan      = UInt32[1500]
n_chan      = UInt32[1500]
matrix      = Float64[440721]
offset      = 1
e_min       = Float64[1024]
e_max       = Float64[1024]

ARF Data Set: 1:1
name         = arf.fits
energ_lo    = Float64[1077]
energ_hi    = Float64[1077]
specresp    = Float64[1077]
bin_lo      = None
bin_hi      = None
exposure    = 20650.7059988

Background Data Set: 1:1
name         = background.pi
channel      = Float64[1024]
counts      = Float64[1024]
staterror   = None
syserror    = None
bin_lo      = None
bin_hi      = None
grouping    = Int16[1024]
quality     = Int16[1024]
exposure    = 20651.1490345
backscal    = 1.69973935689e-05
areascal    = 1.0
grouped     = True
```

Using A Pileup Model - Sherpa 4.1 (S-Lang)

```
subtracted      = False
units           = energy
response_ids    = [1]
background_ids  = []

Background RMF Data Set: 1:1
name            = rmf.fits
detchans       = 1024
energ_lo       = Float64[1077]
energ_hi       = Float64[1077]
n_grp          = Int16[1077]
f_chan         = UInt32[1500]
n_chan         = UInt32[1500]
matrix         = Float64[440721]
offset         = 1
e_min          = Float64[1024]
e_max          = Float64[1024]

Background ARF Data Set: 1:1
name           = arf.fits
energ_lo       = Float64[1077]
energ_hi       = Float64[1077]
specresp      = Float64[1077]
bin_lo        = None
bin_hi        = None
exposure      = 20650.7059988
```

If this is not the case for your data, you will need to manually set the instrument response:

```
sherpa> load_arf("arf.fits");
sherpa> load_rmf("rmf.fits");
```

The background is subtracted from the data, rather than fit simultaneously:

```
sherpa> subtract();
```

If the background file is not referenced in the header of `source_pha.fits`, you can manually load the background with `load_bkg`.

The input data set may be plotted:

```
sherpa> plot_data();
```

The data are plotted in energy space since the instrument response provides the information necessary for the computation of the number of predicted counts in each detector bin; [Figure 1](#) shows the resulting plot.

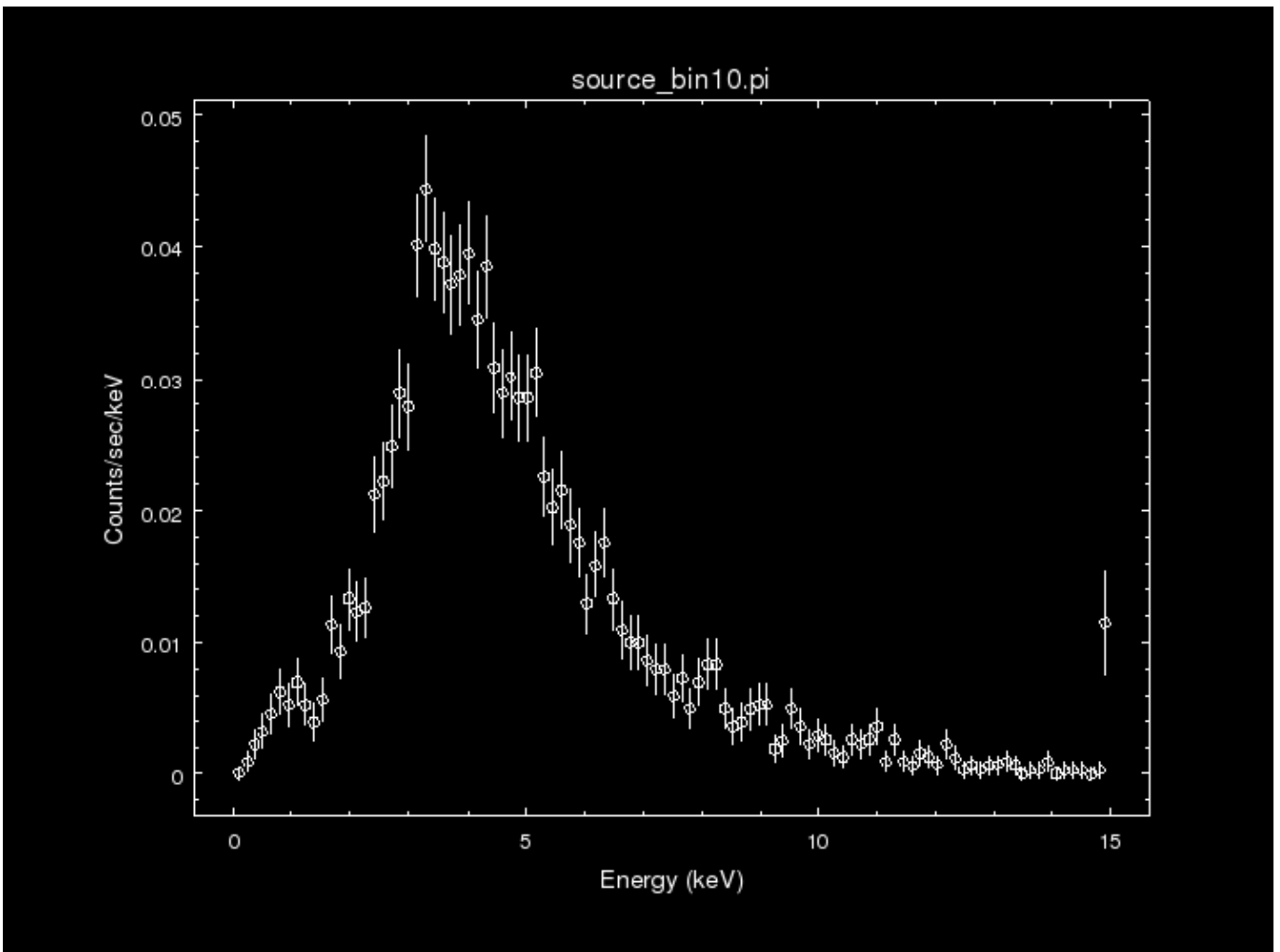


Figure 1: Source Spectrum

Plot of background-subtracted source spectrum

A filter may be applied to the data before proceeding:

```
sherpa> ignore();  
sherpa> notice(1,7);  
sherpa> plot_data();
```

Note that the pileup correction will include the entire available energy information **regardless of the specified filter**. The fit statistics, however, are calculated only on the specified bins.

Figure 2 shows the filtered data, which has an energy range of 1 - 7 keV. See the Choosing an Energy Filter why topic for help in picking a range.

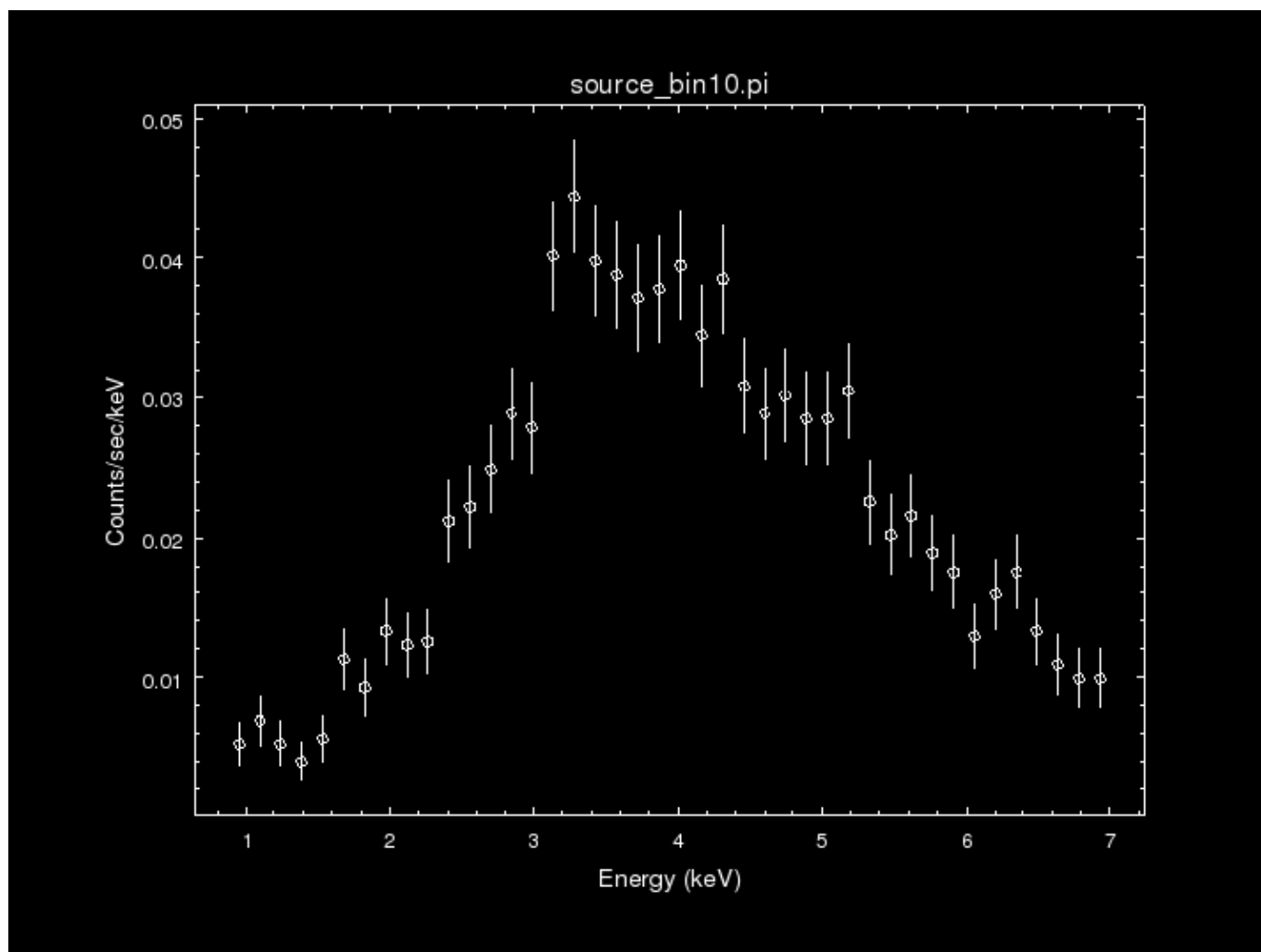


Figure 2: Energy-filtered source spectrum

Plot of background-subtracted, energy-filtered source spectrum

Defining the Models

Multi-Component Source Model

Since we have background-subtracted the data (rather than fitting it simultaneously), it is only necessary to create a source model expression. We model this source with a power law (`xspowerlaw`) absorbed by the interstellar medium (`xswabs`).

The absorption model will be referred to as "abs1", and the broken power law will be "power1"; the product of the two is assigned as the source model for the data set:

```
sherpa> set_model(xswabs.abs1*xspowerlaw.power1);

sherpa> show_model();
Model: 1
apply_rmf(apply_arf((20651.1490345 * (xswabs.abs1 * xspowerlaw.power1))))
Param      Type      Value      Min      Max      Units
```

Using A Pileup Model - Sherpa 4.1 (S-Lang)

```
-----
abs1.nh      thawed      1      0      100000 10^22 atoms / cm^2
power1.phoindex thawed      1      -2      9
power1.norm  thawed      1      0      1e+24

sherpa> guess(get_model());
WARNING: No guess found for apply_rmf(apply_arf((20651.1490345 * (xswabs.abs1 * xspowerlaw.power1))))

sherpa> guess(power1);

sherpa> show_model();
Model: 1
apply_rmf(apply_arf((20651.1490345 * (xswabs.abs1 * xspowerlaw.power1))))
Param      Type      Value      Min      Max      Units
-----
abs1.nh      thawed      1      0      100000 10^22 atoms / cm^2
power1.phoindex thawed      1      -2      9
power1.norm  thawed  0.000116019  1.16019e-07  0.116019
```

Note that since the model contains multiple components, the *Sherpa* `guess()` command cannot be used to estimate the initial parameter values for the full model expression based on the data; each model component must be considered separately.

If you know the Galactic column density for your source, you may set `abs1.nh` and freeze it:

```
sherpa> abs1.nh = <value> ;
sherpa> freeze(abs1.nh);
```

In this thread, however, we allow it to vary during the fit.

```
sherpa> thaw(abs1.nh);

sherpa> show_model();
Model: 1
apply_rmf(apply_arf((20651.1490345 * (xswabs.abs1 * xspowerlaw.power1))))
Param      Type      Value      Min      Max      Units
-----
abs1.nh      thawed      1      0      100000 10^22 atoms / cm^2
power1.phoindex thawed      1      -2      9
power1.norm  thawed  0.000116019  1.16019e-07  0.116019
```

Pileup Model

It is also necessary to define the pileup model (`jdpileup`) immediately after defining the source model:

```
sherpa> set_pileup_model(jdpileup.jdp);

sherpa> print(get_pileup_model());
jdpileup.jdp
Param      Type      Value      Min      Max      Units
-----
jdp.alpha  thawed      0.5      0      1
jdp.g0     frozen      1  1.17549e-38  1
jdp.f      thawed      0.95     0.9      1
jdp.n      frozen      1  1.17549e-38  100
jdp.ftime  frozen      3.241   1.17549e-38  5      sec
jdp.fracexp frozen      0.987   0      1
jdp.nterms frozen      30      1      100
```

Read carefully the following information on how to set the pileup model parameters:

- **alpha**

Using A Pileup Model - Sherpa 4.1 (S-Lang)

α parameterizes "grade migration" in the detector, and represents the probability, per photon count greater than one, that the piled event is not rejected by the spacecraft software as a "bad event". Specifically, if n photons are piled together in a single frame, the probability of them being retained (as a single photon event with their summed energy) is given by $\alpha^{(n-1)}$.

α is the parameter most likely to be allowed to vary in a fit.

- **g_0 :**

Grade correction for single photon detection, i.e. a fraction g_0 of single photon events will be retained as good grades.

In practice, this should be frozen to unity (the default) in any fit.

- **f :**

The fraction of events in the source extraction region to which pileup will be applied. A typical value is approximately 0.95, and it should always be > 0.85 . f is the second most likely parameter, after α , to be allowed to vary in a fit.

It is recommended that the lower limit of this parameter be set to 0.85 before fitting (the default is 0.9):

```
sherpa> jdp.f.min=0.85;
```

- **n :**

Divide the model counts among n regions, to which the pileup model will be applied independently. This should be approximately the number of 3x3 pixel islands in the extracted spectra; for point sources, it is unity (the default).

This should remain a frozen parameter in any fit.

- **$ftime$**

The frame time parameter ($ftime$) should be set to the good exposure time per frame, which is recorded in the header keyword `EXPTIME` of the **event file**. (Note that `EXPTIME` is used instead of `TIMDEL` because the latter includes the transfer time, which $ftime$ should not.)

The CIAO command may be executed from within *Sherpa*, as long as it is escaped with an exclamation point (!):

```
sherpa> !dmkeypar acisf01618_evt2.fits EXPTIME echo+
3.2
sherpa> jdp.ftime=3.2;
```

- **$fracexp$**

This parameter is a fraction ≤ 1 that multiplies the frame exposure time to create a shorter, effective frame exposure time.

Users should set and freeze $fracexp$ to unity, and set the frame exposure time via the $ftime$ parameter only:

```
sherpa> jdp.fracexp=1;
```

Information on using this parameter to model novel deadtime effects is available in the [The Chandra ABC Guide to Pileup](#).

- **$nterms$:**

Using A Pileup Model - Sherpa 4.1 (S-Lang)

The `nterms` parameter represents the maximum number of photons considered for pileup in a single frame.

This should be left frozen at a value of 5. In other words, the expansion of the model will include terms corresponding to 0, 1, 2, 3, 4, and 5 photon events landing in the same extraction region during the same frame time.

```
sherpa> jdp.nterms=5;
```

The pileup model parameters are now set as shown:

```
sherpa> print(get_pileup_model());
jdpileup.jdp
```

Param	Type	Value	Min	Max	Units
jdp.alpha	thawed	0.5	0	1	
jdp.g0	frozen	1	1.17549e-38	1	
jdp.f	thawed	0.95	0.85	1	
jdp.n	frozen	1	1.17549e-38	100	
jdp.ftime	frozen	3.2	1.17549e-38	5	sec
jdp.fracexp	frozen	1	0	1	
jdp.nterms	frozen	5	1	100	

Fitting with the Neldermead and Monte Carlo Optimization Methods

To find a global minimum for the fit to the data, we first choose to use one of the simpler fit optimization methods: Neldermead. This will allow for a quick fit, but may require some "hands-on" work afterwards.

Fit using the Neldermead method

We set the optimization method to 'neldermead', the fit statistic to 'chi2datavar', and start the fit:

```
sherpa> set_method("neldermead");
sherpa> set_stat("chi2datavar");

sherpa> fit();
Solar Abundance Vector set to angr: Anders E. & Grevesse N. Geochimica et Cosmochimica Acta 53
Cross Section Table set to bcmc: Balucinska-Church and McCammon, 1998
Dataset = 1
Method = neldermead
Statistic = chi2datavar
Initial fit statistic = 1727.08
Final fit statistic = 77.7473 at function evaluation 828
Data points = 42
Degrees of freedom = 37
Probability [Q-value] = 0.000101442
Reduced statistic = 2.10128
Change in statistic = 1649.34
jdp.alpha = 0.432791
jdp.f = 0.928049
abs1.nh = 5.94924
power1.phoindex = 1.35462
power1.norm = 0.00195233
```

The parameter space of models which include a pileup model component is very complex and presents a challenge to the search and fitting algorithm. By using the Neldermead method, we are settling for finding a *local* minimum; now one should examine the solution, check the error bars, find a new minimum, refit, etc.

Another option is to use a different optimization method, one which will start from several initial parameter settings and explore in a more consistent way the entire parameter space; the "moncar" method that uses random starting locations can be used in *Sherpa*.

Fit using the Monte Carlo method

Now we change the method to Monte Carlo and re-run the fit to see how the results change. Note that we start at the results obtained by Neldermead, but you could reset the model parameters before continuing with the `reset()` command, if desired.

```
sherpa> set_method('moncar');
sherpa> fit();
Dataset                = 1
Method                 = moncar
Statistic              = chi2datavar
Initial fit statistic  = 77.7473
Final fit statistic    = 77.7473 at function evaluation 9237
Data points           = 42
Degrees of freedom     = 37
Probability [Q-value] = 0.000101442
Reduced statistic      = 2.10128
Change in statistic    = 1.44529e-06
  jdp.alpha           0.432506
  jdp.f               0.92814
  abs1.nh             5.94892
  power1.phoindex     1.35457
  power1.norm         0.00195322
```

This table compares the results from the two runs with different optimization methods:

parameter	Neldermead	Monte Carlo
abs1.nh	5.94924	5.94892
power1.phoindex	1.35462	1.35457
power1.norm	0.00195233	0.00195322
jdp.alpha	0.432791	0.432506
jdp.f	0.928049	0.92814

Calculate the parameter uncertainties

At this point, we continue with the first set of fit results obtained by using the Neldermead optimization method. For the best-fit values to really mean something, we need to find the uncertainties on the parameters. The `covariance` and `projection` commands can be used to estimate confidence intervals for the thawed parameters:

```
sherpa> set_method("neldermead");
sherpa> set_proj_opt("fast", False);
sherpa> proj();
WARNING: hard maximum hit for parameter jdp.alpha
Dataset                = 1
Confidence Method      = projection
Fitting Method         = neldermead
Statistic              = chi2datavar
projection 1-sigma (68.2689%) bounds:
  Param                Best-Fit  Lower Bound  Upper Bound
  -----              -
  jdp.alpha            0.432506  0.432506     0.432506
  jdp.f                0.92814   0.92814      0.92814
  abs1.nh              5.94892   5.94892      5.94892
  power1.phoindex     1.35457   1.35457      1.35457
  power1.norm         0.00195322 0.00195322 0.00195322
```

Using A Pileup Model - Sherpa 4.1 (S-Lang)

jdp.alpha	0.432791	-0.289777	-----
jdp.f	0.928049	-0.303487	0.0316072
abs1.nh	5.94924	-0.56033	0.588849
power1.phoindex	1.35462	-0.234803	0.239429
power1.norm	0.00195233	-0.00113728	0.0159352

The table lists the best-fit values for the model parameters, as well as the upper and lower bounds on each of them; 'jdp.alpha' does not have an upper bound, so it is unconstrained towards the hard maximum. Notice that the `proj()` command must be preceded by `'set_proj_opt('fast', False)'` to use the current optimization method (Neldermead) instead of LevMar, the default method used by `projection`.

Finally, plot the fit and the residuals of the fit:

```
sherpa> plot_fit_delchi();
```

The final plot is show in [Figure 3](#).

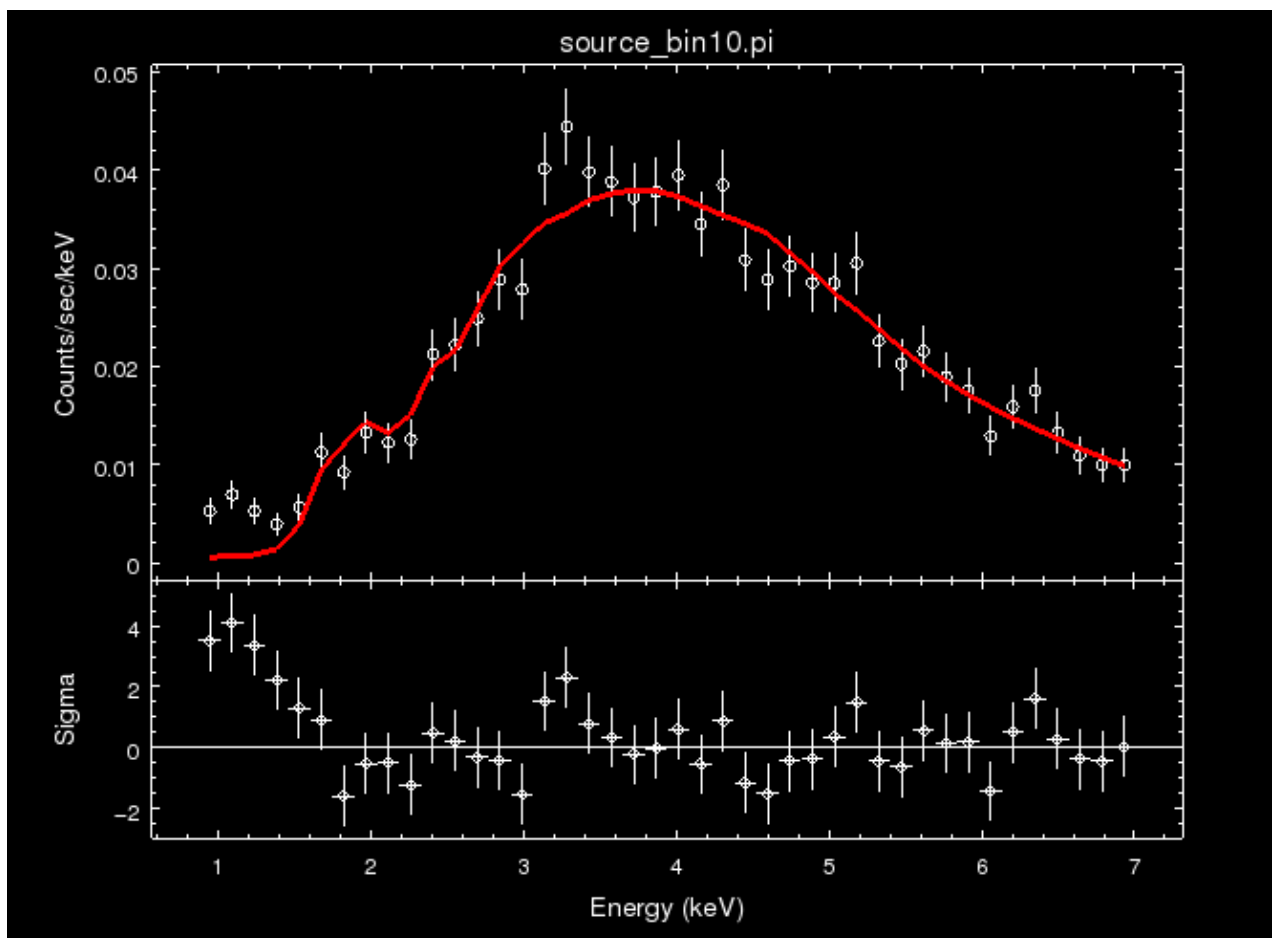


Figure 3: Plot of the fit and residuals

Plot of the model fit to the source spectrum, with residuals.

Examining the Pileup Fraction

It is now possible to see what fraction of the events are calculated as piled:

```
sherpa> print(get_pileup_model());
jdpileup.jdp
Param          Type          Value          Min          Max          Units
-----
jdp.alpha      thawed      0.432791          0          1
jdp.g0         frozen          1  1.17549e-38          1
jdp.f          thawed      0.928049          0.9          1
jdp.n          frozen          1  1.17549e-38          100
jdp.ftime      frozen          3.2  1.17549e-38          5          sec
jdp.fracexp    frozen          1          0          1
jdp.nterms     frozen          5          1          100

1: 0.360958  0.833503
2: 0.147551  0.147458
3: 0.04021   0.0173916
4: 0.00821843 0.00153841
5: 0.0013438 0.000108867
*** pileup fraction: 0.166497
```

This command prints the pileup fractions from the most recent fit. From left to right, the columns report the number of piled photons, percentage of that number of photons in the pileup region, and percentage of that number of piled photons in the total events. The fraction of pileup in the observation is printed after the breakdown. The next paragraph analyzes the above results, which should clarify this explanation.

In this example, the first row indicates that ~35% of the frames contained a single photon in the pileup region and ~83% of the events were single-photon events. The second row shows that ~23% of the frames contained 2 photons in the pileup region and ~15% of the events were due to 2 photons, and so forth. For this observation, the total pileup fraction is ~17%.

Saving the Fit Results

Before exiting *Sherpa*, save the fit results to a file:

```
sherpa> save("pileup_fit.save");
```

These results may then be restored in a later session with the restore command.

Scripting It

The file fit.sl is an S-Lang script which performs the primary commands used above; it can be executed by typing `execfile("fit.sl")` on the Sherpa command line.

The Sherpa script command may be used to save *everything* typed on the command line in a Sherpa session:

```
sherpa> script(filename="sherpa.log", clobber=False);
```

The CXC is committed to helping Sherpa users transition to new syntax as smoothly as possible. If you have existing Sherpa scripts or save files, submit them to us via the CXC Helpdesk and we will provide the CIAO/Sherpa 4.1 syntax to you.

History

01 Dec 2008 updated for CIAO 4.1

29 Apr 2009 new script command is available with CIAO 4.1.2

URL: <http://cxc.harvard.edu/sherpa/threads/pileup/index.sl.html>

Last modified: 29 April 2009

