SHERPA
The Modeling and Fitting Application of the CIAO Software System

The Sherpa Team:

Stephen Doe
Peter Freeman
Holly Jessop
Mike Noble
Aneta Siemiginowska

Bill Joye (1997)
Malin Ljungberg (1997-99)
Mark Birkinshaw (OPTIM)

Flexible Analysis in Sherpa

(There is no single, correct path to every final answer.)

- To analyze high-resolution spatial, spectral, and temporal data, one must overcome a slew of obstacles: for instance, fitting techniques based on the Gaussian distribution may not apply when the number of counts in each bin is low, or determining the best-fit of a complex model may not be easy.

- *Sherpa* thus provides many fit statistics and methods of optimization and parameter estimation that have different underlying assumptions and/or provide different strategies for dealing with difficult analysis problems.

- The cutting-edge nature of these analysis obstacles has led to an astrostatistics collaboration between the *Sherpa* team and members of Harvard University's Department of Statistics.

Selected Sherpa References

- As used to fit *Chandra* spectra of quasars, jets, and clusters:
  - *Cappi et al., ApJ, 2001*
  - *Marshall et al., ApJL, 2001*
- As used to fit *Chandra* images:
  - *Cagnoni et al., in preparation, 2001*
- As used to fit *ROSAT* images:
  - *Paolillo et al., in preparation, 2001*
- As used to fit multi-wavelength data:
  - The Large Bright Quasar Survey *(Forster et al., ApJS, 20 Nov 2000)*
  - The H/RCULES Project *(Forster et al., AAS, 2000)*

A Typical Sherpa Session

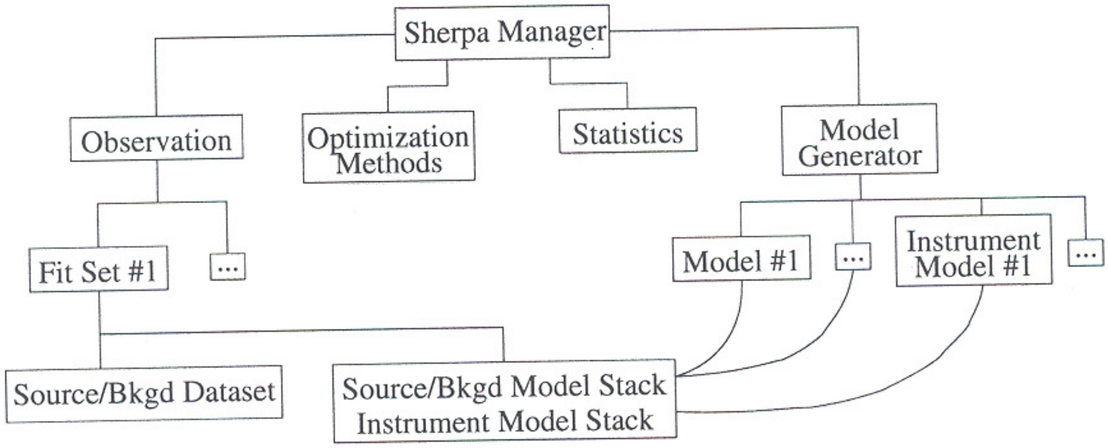
(The boiled-down version.)

The user:

- reads in data (and sets filters, *etc.*);
- builds model expressions;
- chooses a statistic;
- fits the model expressions to the data, one at a time;
- compares the results of the fits in order to select a best-fit model; and
- estimates the errors for the best-fit model parameters.

Sherpa: Under the Hood

- *Sherpa* is an object-oriented C++ application:



Simplified Relationship Diagram

(Not a Class Diagram!)

- A global pointer in `Sherpa Manager` allows communication between "widely separated" *Sherpa* objects.

Data Entry

- During a *Sherpa* session, the user may read in files containing:
  - source data
  - background data
  - errors on the source and/or background data
  - filters
  - statistical weights

  These data may be integer, float, or double precision and may be of arbitrary dimensionality.

- Currently supported file types include:
  - ASCII
  - FITS binary table
  - FITS image
  - PHA types I & II
  - IRAF imh
  - ROSAT qpoe

  For all types except ASCII, data entry is accomplished through the Data Model interface, so that, *e.g.*, one can use the Data Model filtering syntax within *Sherpa* data entry commands.

Building Model Expressions

- In *Sherpa*, one can build model expressions that represent the
  - source
  - background
  - instrument

  for each dataset.

  ⇒ Note, however, background and instrument model expressions are *not* required to carry out fits.

- Currently, nearly 40 one- and two-dimensional *Sherpa* models and 90 one-dimensional *XSPEC v. 11* models are available for building model expressions.

Building Model Expressions

- The *Sherpa model language* resolves ambiguity by allowing the user to give a unique name or alias to each instance of a model.
- Example:
  - If two datasets are entered, and each is to be fit with a different Gaussian model:

    ```
    sherpa> gauss1d[g1]
    sherpa> gauss1d[g2]
    sherpa> source 1 = g1
    sherpa> source 2 = g2
    ```

  - If, on the other hand, they are to be fit with the same Gaussian model:

    ```
    sherpa> gauss1d[g1]
    sherpa> source 1:2 = g1
    ```

- Note the similarity to object-oriented programming:
  - `gauss1d` can be considered the blueprint "class."
  - `g1` and `g2` can be considered "instantiated objects" of "class" `gauss1d`.

Building Model Expressions

- Model parameters can be *linked* to other parameters.
- Example:
  - A particular atomic line is observed by detectors with different resolutions. One can model this line with two Gaussian functions whose centroids (but not amplitudes or widths) are linked:

```
sherpa> source 1 = gauss1d[g1]
sherpa> source 2 = gauss1d[g2]
sherpa> g1.pos => g2.pos
```

- Model parameters can also be linked to other models.
- Example:
  - One can model emission from an accretion disk using a blackbody function whose temperature is a function of radius:

```
sherpa> Temperature = POLY
sherpa> BB.kT => Temperature
```

Building Model Expressions

- A model can be *nested* within another model.
- Example:
  - Transform an input dataspace to log-space, and evaluate a blackbody in that space:

```
sherpa> logenergy = shlog
sherpa> source = bb{logenergy}
```

- Different models can be defined along each axis of a multi-dimensional dataset.
- Example:
  - Model two-dimensional data that have spectral information along one axis and spatial information (*e.g.* radius) along the other:

```
sherpa> data image.fits
sherpa> lorentz[Spatial]
sherpa> pow[Spec]
sherpa> source = Spatial{x1}*Spec{x2}
```

Instrument Models

- Instrument models describe the mapping from photon space (where source and background models are evaluated) to counts space (where fit statistics are computed) for a particular detector.

  ⇒ *The instrument model class, by hiding detector-dependent details, allows Sherpa to be a mission-independent fitting application.*

- Currently *Sherpa* offers three instrument model types:
  - `RSP`, in which the evaluated one-dimensional model is multiplied by an ancillary response (*i.e.* effective area) and then folded through a response matrix;
  - `PSFFromTCD`, in which the evaluated one- or two-dimensional model is convolved with an analytic kernel (*e.g.* Gaussian) defined in *CIAO*'s `TCD` library;
  - and `PSFFromFile`, in which the evaluated two-dimensional model is convolved with a numeric kernel.

Optimization in Sherpa

Optimization is the action of minimizing χ^2 or $-\log\mathcal{L}$ by varying the thawed parameters of the model. The user may choose between several optimization methods in *Sherpa*, including ones which:

- Find the local minimum.
  - `POWELL`
  - `SIMPLEX`
  - `LEVENBERG-MARQUARDT`

  These algorithms are not computationally expensive, but they are also not appropriate for finding the global minimum of a complex statistical surface when starting from a random point.

- Attempt to find the global minimum.
  - `GRID` and `GRID-POWELL`
  - `MONTE` and `MONTE-POWELL`
  - `SIMULATED ANNEALING`

  These are computationally intensive algorithms which are useful for searching complex statistical surfaces, starting from a random point.

Parameter Estimation in Sherpa

Currently available parameter estimation methods:

- `UNCERTAINTY`
- `PROJECTION` (or the profile likelihood)
- `COVARIANCE`

Parameter estimation methods that will eventually become available to *Sherpa* users:

- Data simulation and fitting
- The Laplace Approximation
- Numerical integration via brute force
- Numerical integration via `BAYESPACK`
- Markov-Chain Monte Carlo

χ^2-Based Statistics

The χ^2 statistic is

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

where

- D_i represents the observed datum in bin i;
- M_i represents the predicted model counts in bin i; and
- σ_i^2 represents the variance of the sampling distribution for D_i.

χ^2 **Statistic**	σ_i^2
`GEHRELS`	$[1 + \sqrt{D_i + 0.75}]^2$
`DVAR`	D_i
`MVAR`	M_i
`PARENT`	$\frac{\sum_{i=1}^{N} D_i}{N}$
`PRIMINI`	M_i from previous best-fit

Likelihood-Based Statistics

The **CASH** statistic is

$$C \equiv 2\sum_i [M_i - D_i \log M_i] \propto -2 \log\mathcal{L}\,,$$

where

- D_i represents the observed datum in bin i;
- M_i represents the predicted model counts in bin i; and
- $\mathcal{L} = \Pi_i \frac{M_i^{D_i}}{D_i!} \exp(-M_i)$.

```
#
#  Sherpa: Modeling 2D image data, X-ray cluster
#
#################################################

paramprompt off

data image_small.fits
image data

# load the region file to ds9
# Region: Load src3.reg

ignore image


# inspect the filter

image filter

image data


source = beta2d[b2]


# b2.r0 parameter value [350]
# b2.alpha parameter value [1]
# b2.xpos parameter value [217.5]
# b2.ypos parameter value [204.5]
# b2.ellip parameter value [0]
# b2.theta parameter value [0]
# b2.ampl parameter value [12]


# freeze b2.xpos
# freeze b2.ypos

statistics cash

# fit

# CTRL-C

#sherpa> fit
# LVMQT: V2.0
# LVMQT: initial function at value = 7.33261e+06
# LVMQT: iteration limit reached
# LVMQT: final function value = 120186 at iteration 100
#             b2.r0  32.663
#             b2.alpha  1.23249
#             b2.ampl  2.61009


# Include ellipticity and theta
thaw b2.ellip
thaw b2.theta

fit
```

```
# ctrl-C
# read saved parameter values

use session.1

image fit

# tile frames, show data, model, errors in three frames
#
# create new frame and display residuals there
#

image residuals
# write res.fits

# run csmooth - adaptive smoothing with Gaussian kernel
#       and look at the results
#
#
# load res_smth4.fits to ds9, zoom
```

```
#
# Example of Sherpa session in CIAO 2.0  for image analysis
# using a PSffromFile as a model
#
#############################


data center_img.fits

# to display

image data

# to see current setup

show

parampromt off

source = psffromfile[mypsf]

show mypsf

#PSFfromFile[mypsf]
#    Param   Type      Value        Min        Max                 Units
#    -----   ----      -----        ---        ---                 -----
# 1numCuts frozen          1          1          1
# 2convTyp frozen          1          1          2
# 3   file string: "good.psf.float.fits "
# 4  xsize frozen         32          1       1024
# 5  ysize frozen         32          1       1024
# 6   xoff frozen          0       -512        512
# 7   yoff frozen          0       -512        512
# 8   xpos thawed        512          1       1024
# 9   ypos thawed        512          1       1024
#10   norm frozen          1          0       1000


# Parameters of PSFfromFile are listed above and you can set
# parameters for fitting. Also you can display
# your psf


image psf


statistics Cash
thaw mypsf.norm

fit

# image fit creates images in 3 frames

image fit

write residuals
```