Sherpa: 1D/2D modeling and fitting in Python

Brian L. Refsdal (brefsdal@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Stephen M. Doe (sdoe@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Dan T. Nguyen (dtn@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Aneta L. Siemiginowska (aneta@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Nina R. Bonaventura (nina@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Douglas Burke (dburke@cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Ian N. Evans (evans_i@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Janet D. Evans (janet@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Antonella Fruscione (antonell@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Elizabeth C. Galle (egalle@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
John C. Houck (houck@space.mit.edu) – *MIT Kavli Institute*, USA
Margarita Karovska (karovska@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Nicholas P. Lee (nlee@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Michael A. Nowak (mnowak@space.mit.edu) – *MIT Kavli Institute*, USA

Sherpa is a modern, general purpose fitting and modeling application available in Python. It contains a set of robust optimization methods that are critical to the forward fitting technique used in parametric data modeling. The Python implementation provides a powerful software package that is flexible and extensible with direct access to all internal data objects. Sherpa affords a highly proficient scientific working environment required by the challenges of modern data analysis. It is implemented as a set of Python modules with computationally-intensive portions written in C++/FORTRAN as extension modules using the Python C-API. It also provides a high level user interface with command-like functions in addition to the classes and functions at the API level. Sherpa is being developed by the Chandra X-ray Center (CXC) and is packaged with the Chandra data analysis software package (CIAO). Sherpa can also be built as a standalone application; it can be extended by the user, or embedded in other applications. It allows for analysis specific to astronomy, but also supports generic modeling and fitting tasks. The 'astro' module includes additional astronomy model functions, FITS image support, instrument models, and utilities. Sherpa's model library includes some commonly used 1D and 2D functions and most of the X-ray spectral models found in the High Energy Astrophysics Science Archive Research Center (HEASARC) XSPEC application. Sherpa also supports user-defined models written in Python, C++, and FORTRAN, allowing users to extend Sherpa with models not included in our model library. Sherpa has a set of optimization methods including LMDIF, implementations of Differential Evolution (Monte Carlo) and Nelder-Mead simplex. These functions minimize differences between data points and model values (as measured by a fit statistic such as the chi-squared, maximum likelihood, or a user-defined statistic). The generic I/O module includes back-end interfaces to read ASCII files using NumPy and astronomy image files (FITS) using PyFITS or CIAO Crates (CXC Data Model library in C++). Sherpa is general enough to fit and model data from a variety of astronomical observatories (e.g., Chandra, ROSAT, Hubble) and over many wavebands (e.g., X-ray, optical, radio). In fact, Sherpa can fit and model any data set that can be represented as collections of 1D or 2D arrays (and can be extended for data of higher dimensionality). Data sets can also be simulated with noise using any model. The visualization module also allows for multiple back-ends. An interface to Matplotlib and CIAO ChIPS (CXC plotting package layered on VTK in C++) are provided for line and histogram plotting. 2D visualization is supported by the Smithsonian Astrophysical Observatory (SAO) imager, DS9. The Sherpa command line uses a configured version of IPython to provide a high level shell with IPython 'magic' and readline support.

Introduction

Chandra is one of NASA's great observatories and astronomers from all over the world continue to use it for X-ray astronomy since its launch in 1999. Sherpa is one of many tools included in the Chandra Interactive Analysis of Observations (CIAO) [ciao] software package. Sherpa is a multi-dimensional, robust Python application that handles the task of modeling and fitting in Chandra data analysis. It is developed by a team of programmers and scientists in the Chandra X-ray Center (CXC) and many of the algorithms and numerical methods have been updated and optimized for the latest computing architectures. In past releases, Sherpa was comprised of a rigid YACC parser and much legacy C++ code and recently has been re-implemented into a cleaner, modular form.

Fitting and modeling

Fitting a model to data in Sherpa is done by modifying one or more model parameters until the differences are minimized between the predicted data points and the raw data. Scientists model data to find values that map to physical quantities, such as temperature, that cannot easily be measured from the data set alone. The analysis is not complete until the model expression contains the appropriate number of parameters, that the parameter values are known to some degree of confidence, and the probability of attaining such a fit by chance is acceptably low.

Use

Sherpa is primarily used for Chandra X-ray data analysis. Many X-ray astronomers have used it for years to analyze their data, and have published the results based on Sherpa fitting and modeling. Sherpa is also used in the data pipeline of the Chandra Source Catalog (CSC) [csc] to estimate source sizes and spectral properties.

Design

To achieve an extensible, general purpose fitting and modeling application, we chose to implement a series of Python modules that we later joined to form the fitting application. At the object layer, many of these modules include C++ extensions for computationally intensive routines.

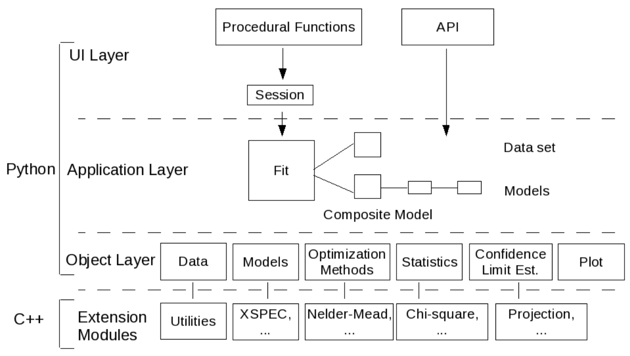

Figure 1. *Sherpa Module Design*

Sherpa can read data from a variety of astronomical observatories (e.g., Chandra, ROSAT, Hubble), as they all publish their data in the Flexible Image Transport System (FITS) format; Sherpa can deal with data in any waveband (X-ray, optical, or radio). Even other scientific or tabular data in ASCII format are supported. Data sets can even be simulated using a defined model and added noise.

Sherpa provides a set of 1D and 2D models, as well as an interface to the XSPEC model library (an X-ray spectral package published by the High Energy Astrophysics Science Archive Research Center (HEASARC) [xspec]. Users can also extend Sherpa by writing models of their own in Python, C++, or FORTRAN. An interface to a Fast Fourier Transform (FFT) library is available for convolutions to model the effects of point spread functions (PSFs), as well as some physical effects (e.g., spectral line broadening).

Sherpa provides robust optimization functions to handle the low signal-to-noise often found in X-ray data. They include Levenberg-Marquardt (LMDIF) [lm] and in-house implementations of Differential Evolution (Monte Carlo) [mc] and Nelder-Mead simplex [nm].

To complement the optimization functions, Sherpa includes native chi-squared and maximum likelihood fit statistics. The interface is generic enough that users can also define their own fit statistic functions.

Confidence limits for fitted parameter values can be calculated with the function, *projection*. To accurately estimate confidence limits for a given parameter, a multidimensional confidence region needs to be projected onto the parameter's associated axis in parameter space. For a given parameter, the projection function searches for the locations on that axis where the confidence region is projected. A new function, *confidence*, is a pure Python implementation that takes the same approach, but is much more efficient in searching parameter space and generally is both more robust and efficient than the projection function. Both methods compute confidence limits for each parameter independently and are currently computed in parallel on multi-core CPUs.

I/O interface

The I/O interface provides a middle layer where multiple back-ends can be used to support multiple file readers. Basic ASCII file reading is available using NumPy. More complicated astronomy file formats, like FITS, are standard. For example, Crates is a CXC FITS reader that supports the Chandra Data Model. Crates is the default I/O back-end and comes bundled with the CIAO software package. Alternatively, PyFITS [pyfits] can be used in a standalone installation of Sherpa. Each back-end (for Crates and PyFITS) interfaces to the same front-end that exposes I/O functions to other parts of Sherpa. Top-level I/O functions such as *load_data()* are written to use that front-end, so the choice of a particular back-end for I/O remains hidden from the rest of the application.

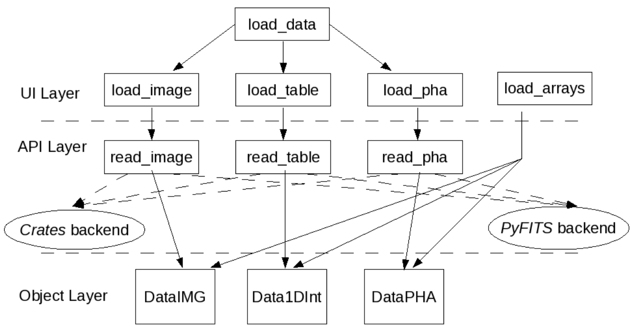

Figure 2. *Sherpa I/O Interface*

Visualization interface

Similarly, the visualization interface supports multiple back-ends for line and contour plotting and 2D imaging. ChIPS is a CIAO package available for line and contour plotting written in C++ on top of VTK.

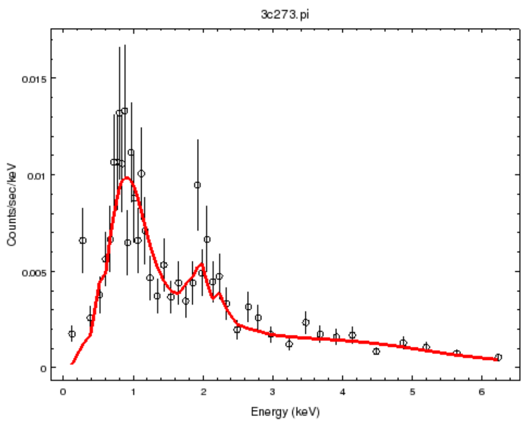

Figure 3. *ChIPS line plot of X-ray data with fitted model*

Sherpa includes a back-end to matplotlib [mpl] as an alternative for standalone installations. DS9 [ds9], the SAO imager, is the primary back-end for image visualization.

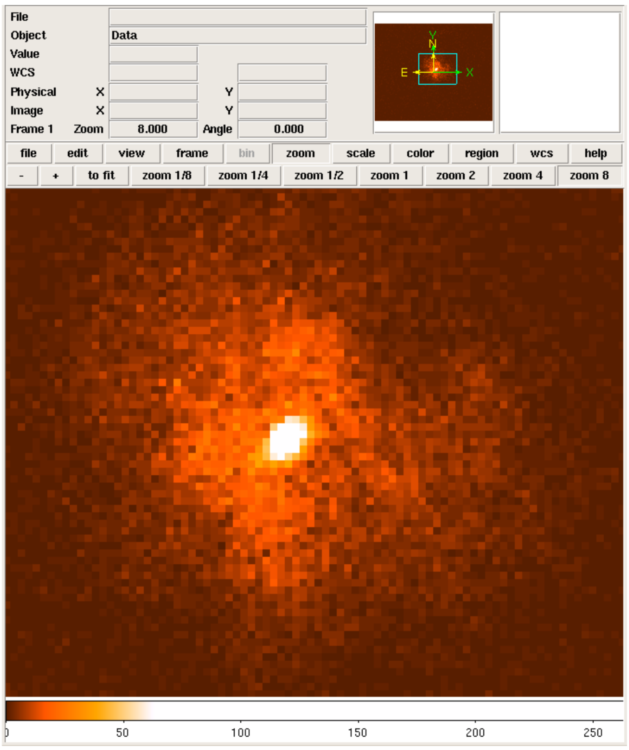

Figure 4. *DS9 image of X-ray source*

With generic interfaces, Sherpa offers users the freedom to add their own back-ends as needed.

API

The Sherpa UI that developers can build upon comes in two levels, the basic API and a set of procedural functions. The API includes all the additional functionality for X-ray analysis, imported like a normal

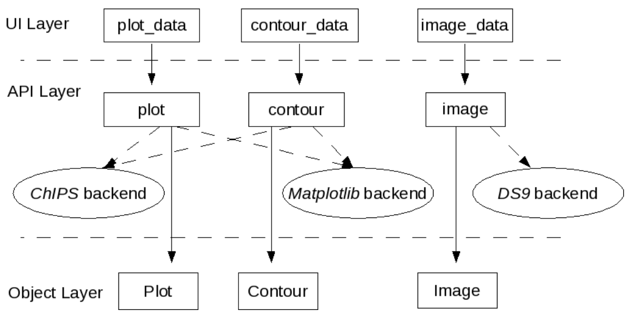

Figure 5. *Sherpa Visualization Interface*

Python package. A typical example of an X-ray spectral fit, such as modeling the redshifted photo-electric absorption of a quasar, is shown below.

Import the base and astronomy modules

```
>>> from sherpa.all import *
>>> from sherpa.astro.all import *
```

Read in a data set from file and setup a filter from 0.3-7.5 keV.

```
>>> pha = read_pha('q1127_src1.pi')
>>> pha.notice(0.3, 7.5)
```

Instantiate individual model classes and setup initial parameter values, freezing and thawing parameters as necessary.

```
>>> abs1 = XSphabs('abs1')
>>> abs1.nH = 0.041
>>> abs1.nH.freeze()
>>> zabs1 = XSzphabs('zabs1')
>>> zabs1.redshift=0.312
>>> p1 = PowLaw1D('p1')
```

Inspection of the source data set can provide clues to the initial parameter values. Some simple Sherpa models include functions to estimate the initial parameter values, based on the data. The algorithms for such “guess” functions are basic (e.g., maximum value, average, determining full-width half max) and do not necessarily perform well for composite models.

```
>>> p1.guess(*pha.to_guess(), limits=True, values=True)
```

A scripting language like Python allows users to define their composite models at run time. Here, the composite model is defined as a product of the three and convolved with the instrument response.

```
>>> model = standard_fold(pha, abs1*zabs1*p1)
```

The Fit object is initialized with class instances of data, model, statistic, optimization method, and confidence limit method.

```
>>> f = Fit(pha, model, Chi2DataVar(), LevMar(),
            Projection())
>>> results = f.fit()
```

The fit results object includes the fitted parameter values and the additional information calculated during the fit. They include the initial, final, and change in statistic values, the number of function evaluations used by the optimization method, the number of data points, the degrees of freedom, the null hypothesis probability (Q-value), and the reduced statistic value.

```
>>> print(results.format())
Method                = levmar
Statistic             = chi2datavar
Initial fit statistic = 17917.4
Final fit statistic   = 686.013 at function evaluation 22
Data points           = 494
Degrees of freedom    = 491
Probability [Q-value] = 1.27275e-08
Reduced statistic     = 1.39717
Change in statistic   = 17231.4
   zabs1.nH       0.094812
   p1.gamma       1.28615
   p1.ampl        0.000705228
```

In determining if the model suitably represents the data, the maximum likelihood ratio (MLR) can be computed. Given two models, the MLR can determine which model better describes a particular data set. The more complex model should be picked when the ratio is less than 0.05. Once a fit has been run and the model selected that best describes the data set, Sherpa can estimate the confidence limits using the projection algorithm.

Estimate 1 sigma confidence limits using projection. The projection results object displays the upper and lower parameter bounds with the best-fit values.

```
>>> results = f.est_errors()
>>> print(results.format())
Confidence Method     = projection
Fitting Method        = levmar
Statistic             = chi2datavar
projection 1-sigma (68.2689%) bounds:
   Param            Best-Fit  Lower Bound  Upper Bound
   -----            --------  -----------  -----------
   zabs1.nH         0.094812  -0.00432843   0.00436733
   p1.gamma          1.28615  -0.00968215   0.00970885
   p1.ampl       0.000705228 -6.63203e-06  6.68319e-06
```

Procedural UI

The same script can be run in a more compact form using the procedural UI. Users can perform common operations --such as reading files, defining models, and fitting --by calling predefined fuctions, without having to write their own code in Python.

Import all astronomy procedural functions

```
>>> from sherpa.astro.ui import *
```

Read data set from file and setup a filter

```
>>> load_data('q1127_src1.pi')
>>> notice(0.3, 7.5)
```

Model instantiation uses a unique syntax of the form 'modeltype.identifier' to create a model and label it in a single line. xsphabs is a model class with abs1 as the identifier and the expression xsphabs.abs1 returns an instance of the class with rich comparison methods. Model parameter values are initialized similarly to the API.

```
>>> set_model(xsphabs.abs1*xszphabs.zabs1*powlaw1d.p1)
>>> abs1.nH = 0.041
>>> freeze(abs1.nH)
>>> zabs1.redshift=0.312
>>> guess(p1)
```

The statistic can be set as an instance of a Sherpa statistic class or a string identifier to native Sherpa statistics.

```
>>> set_stat('chi2datavar')
```

Execute the fit and display the results using a logger.

```
>>> fit()
...<fit output>
```

Compute the confidence limits and display the results using a logger.

```
>>> proj()
...<confidence limit output>
```

Confidence

Projection estimates the confidence limits for each parameter independently (currently this can be done in parallel). Projection searches for the N-sigma limits, where N-sigma corresponds to a given change in the fit statistic from the best-fit value.

For the chi-squared fit statistic, the relation between sigma and chi-squared is $\sigma = \sqrt{\Delta\chi^2}$. For our maximum likelihood fit statistics, the relation has the form $\sigma = \sqrt{2 * \Delta \log L}$. Projection has the added feature that if a new minimum is found in the boundary search, the process will restart using the new found best-fit values. The accuracy of the confidence limits using the projection and confidence methods is based on the assumption that the parameter space is quadratic, where in, the fit statistic function for a given parameter can be expanded using a Taylor series. Also, Sherpa assumes that the best-fit point is sufficiently far ($\approx 3\sigma$) from the parameter space boundaries. Cases where these assumptions do not hold, users should use an alternative approach such as Markov Chain Monte Carlo (MCMC) to map the parameter space using specific criteria. MCMC support within Sherpa is currently in research and development and should be available in future releases.

Fit Statistics

Sherpa has a number of χ^2 statistics with different variances. The χ^2 fit statistic is represented as

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

where D_i represents the observed data, M_i represents the predicted model counts, and σ_i^2 represents the variance of the sampling distribution for D_i.

Sherpa defines many flavors of the χ^2 statistic with different variances. The native flavors of the variance include leastsq

$$\sigma^2 \equiv 1,$$

chi2constvar

$$\sigma^2 \equiv \frac{\sum_{i=1}^{N} D_i}{N},$$

chi2modvar

$$\sigma^2 \equiv M_i,$$

chi2gehrels

$$\sigma^2 \equiv [1 + \sqrt{D_i + 0.75}]^2,$$

chi2datavar

$$\sigma^2 \equiv D_i \qquad \text{for } D_i > 0,$$

and chi2xspecvar

$$\sigma^2 \equiv \begin{cases} D_i & \text{if } D_i > 0 \\ 1 & \text{if } D_i = 0 \end{cases}.$$

Sherpa likelihood statistics include Cash

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

and the C-statistic

$$C \equiv 2 \sum_i [M_i - D_i + D_i(\log D_i - \log M_i)],$$

where D_i represents the observed data, M_i represents the predicted model counts, and L, the log-likelihood

$$L \equiv \prod_i \frac{M_i^{D_i}}{D_i!} \exp(-M_i).$$

Visualization

Visualizing the parameter space can help determine if the assumptions of projection hold or not. 1D line plots of the statistic against parameter values are available with the interval projection function. For a given parameter, the function steps away from the best-fit value, and refits to find the statistic value at a number of points in parameter space. Then a curve showing how the statistic varies with parameter value is drawn.

```
>>> int_proj(p1.gamma)
```

2D confidence contours can be drawn with the region projection function. For any two parameters, a grid is constructed, such that the function refits at each point to find the fit statistic. This maps out parameter space around the best-fit parameter values.
Confidence contours can then be drawn (corresponding to $1\sigma, 2\sigma, 3\sigma$ confidence by default. The plot boundaries are set to be 4σ by default (assuming that parameter space boundaries are no closer than $\approx 4\sigma$ from the best-fit values).

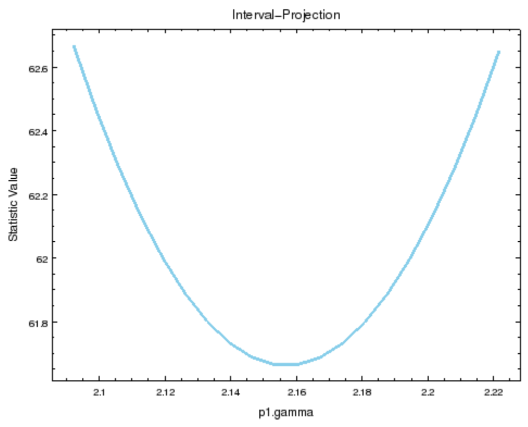

Figure 6. *Interval projection line plots typically look parabolic in a well behaved parameter space.*

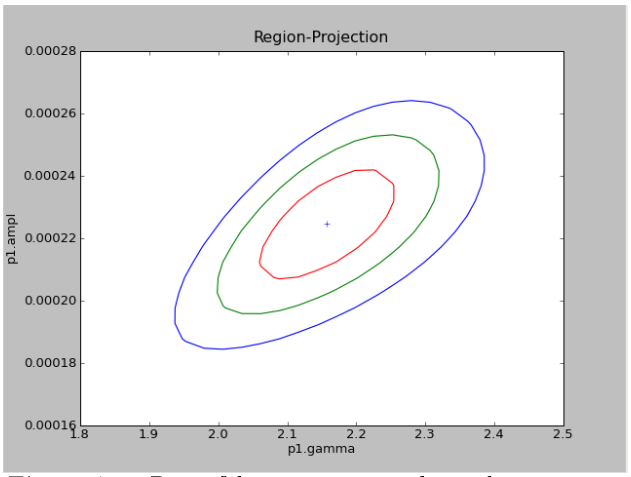

Figure 7. *2D confidence contours, shown here using matplotlib, are a typical product of a Sherpa session. Contours provide the parameter value expectation to some degree of confidence. The smaller the contour, the more constrained the best-fit values are.*

```
>>> reg_proj(p1.gamma, p1.ampl)
```

Once parameter space has been mapped out, then the line or contour plots can provide users with a visual of parameter space around the best-fit parameter values. For 1D line and contour plotting the Sherpa high-level UI includes many convenience functions that hide much of the API boiler-plate in the plot module.

```
>>> plot_fit_delchi()
```

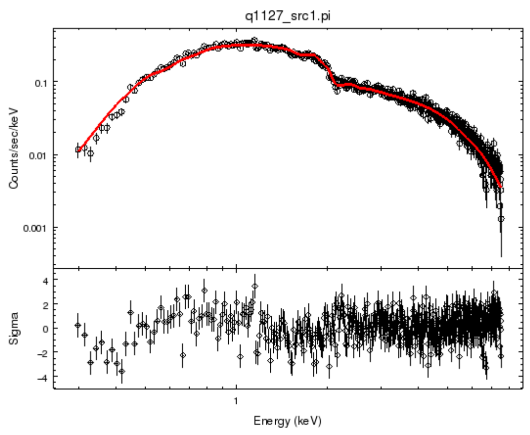

Figure 8. *ChIPS line plot of source data set and error bars, fitted model, and delta chi-squared residuals*

Convenience functions are also available for 2D imaging using an interface to DS9 with the XPA messaging system [xpa].

```
>>> image_fit()
```

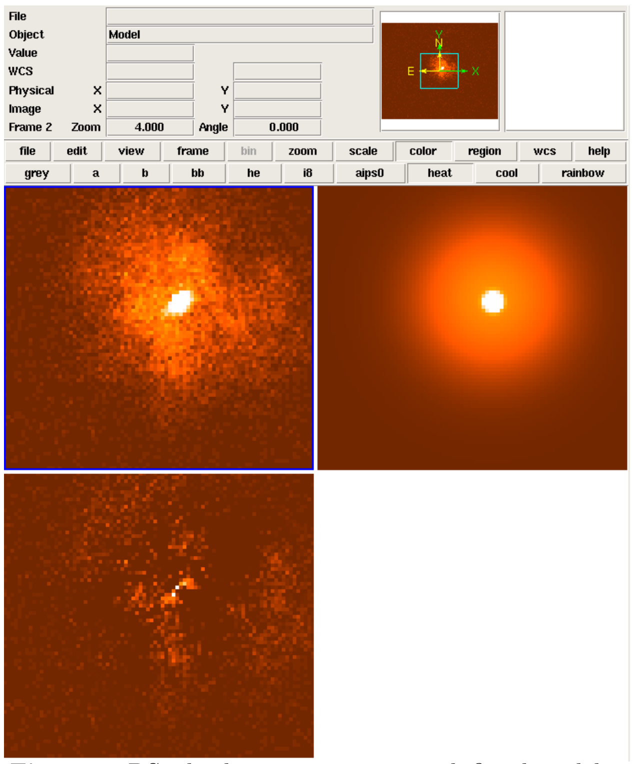

Figure 9. *DS9 displays source image with fitted model and residuals.*

User Models

Users who wish to use their own models in Sherpa can follow the user model interface. User defined models can be written in Python, C++ or FORTRAN. An example of a user-defined model in Python using the Sherpa high-level UI is shown below.

```
>>> def calc(pars, x, **kwargs):
        """
        y = m*x + b
        """
        return pars[0]*x + b

>>> load_user_model(calc, "mymodel")
>>> add_user_pars("mymodel", ["m", "b"], [-3, 5])
```

In the function signature for calc, pars is a list of user defined parameter values and x is a NumPy array representing the model's grid.

Conclusion

Sherpa is a mature, robust fitting and modeling application, with continuing development. The Python code is modular and extensible with plug-in back-ends, and is flexible enough for general use.

Users can download a source tarball and install Sherpa standalone [alone] or download it with the many tools included in the CIAO software package [bundle]. Documentation is available with Python help, CIAO ahelp files, and web pages that detail each function [ahelp] and that show scripting threads [threads].

Users creating standalone installations using distutils are required to install dependencies like Python, NumPy, and Fastest Fourier Transform in the West (FFTW) [fftw].

Future plans for Sherpa include an implementation of MCMC for complicated parameter spaces (provided by the CHASC astro-statistics group [chasc]); speed and performance improvements using parallel techniques for model evaluation; improved documentation (possibly using Sphinx [sphinx]); and a web-based version control system that allows users to download the latest stable version of Sherpa.

Support of the development of Sherpa is provided by National Aeronautics and Space Administration through the Chandra X-ray Center, which is operated by the Smithsonian Astrophysical Observatory for and on behalf of the National Aeronautics and Space Administration contract NAS8-03060.

References

[ahelp] http://cxc.harvard.edu/sherpa/ahelp/index_python.html

[alone] http://hea-www.harvard.edu/uinfra/sherpa/Documentation/download/index.html

[bundle] http://cxc.harvard.edu/ciao/download

[chasc] http://hea-www.harvard.edu/AstroStat/

[ciao] http://cxc.harvard.edu/ciao

[csc] http://cxc.harvard.edu/csc

[ds9] http://hea-www.harvard.edu/RD/ds9/

[fftw] M. Frigo and S.G. Johnson, *The Design and Implementation of FFTW3*, Proceedings of the IEEE 93 (2), 216–231 (2005). Special Issue on Program Generation, Optimization, and Platform Adaptation. http://www.fftw.org/

[lm] Lecture Notes in Mathematics 630: Numerical Analysis, G.A. Watson (Ed.), Springer-Verlag: Berlin, 1978, pp. 105-116

[mc] R. Storn, and K. Price, *Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*, J. Global Optimization 11, 1997, pp. 341-359 http://www.icsi.berkeley.edu/~storn/code.html

[mpl] J.D. Hunter, *Matplotlib: A 2D graphics environment.* Computing in Science and Engineering. 9: 90-95 (2007). http://matplotlib.sourceforge.net.

[nm] J.A. Nelder and R. Mead, Computer Journal, 1965, vol 7, pp. 308-313. Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, Paul E. Wright *Convergence Properties of the Nelder-Mead Simplex Algorithm in Low Dimensions*, SIAM Journal on Optimization, Vol. 9, No. 1 (1998), pp. 112-147. http://citeseer.ist.psu.edu/3996.html. Wright, M. H. (1996) *Direct Search Methods: Once Scorned, Now Respectable* in Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis) (D.F. Griffiths and G.A. Watson, eds.), 191-208, Addison Wesley Longman, Harlow, United Kingdom. http://citeseer.ist.psu.edu/155516.html

[pyfits] http://www.stsci.edu/resources/software_hardware/pyfits

[sphinx] G. Brandl, *Sphinx, Python documentation generator*, http://sphinx.pocoo.org/

[threads] http://cxc.harvard.edu/sherpa/threads/all.html

[xpa] http://hea-www.harvard.edu/saord/xpa/

[xspec] http://heasarc.gsfc.nasa.gov/docs/xanadu/xspec/