Advanced Sherpa

Tom Aldcroft
CXC Operations Science Support

CIAO Workshop
August 6, 2011

Opening new vistas with

Advanced Sherpa

Tom Aldcroft

CXC Operations Science Support

CIAO Workshop

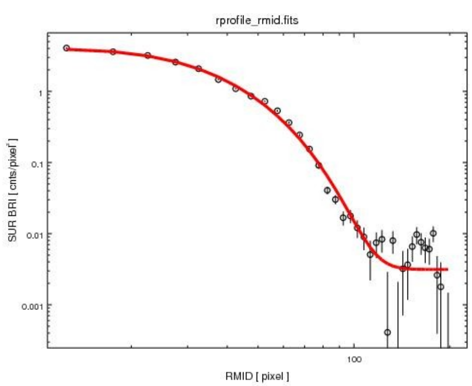
August 6, 2011

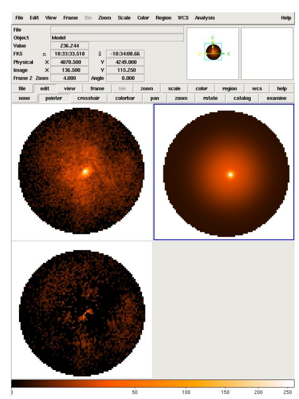
Sherpa

Generalized fitting package with a powerful model language to fit 1D and 2D data

Basic Sherpa

- Interactive (command line) usage
- Scripting using command syntax
- Data access with `show_*` and `print`





```
sherpa> load_pha('acis_pha3.fits')
sherpa> set_source(xsphabs.abs1 * powlaw1d.p1)
sherpa> subtract()
sherpa> fit()
sherpa> show_fit()

Optimization Method: LevMar
name     = levmar
ftol     = 1.19209289551e-07
xtol     = 1.19209289551e-07
gtol     = 1.19209289551e-07
maxfev   = None
epsfcn   = 1.19209289551e-07
factor   = 100.0
verbose = 0

Statistic: Chi2Gehrels

Fit:Dataset               = 1
Method                 = levmar
Statistic              = chi2gehrels
Initial fit statistic = 6.83386e+10
Final fit statistic   = 37.9079 at evaluation 22
Data points            = 44
Degrees of freedom     = 42
Probability [Q-value] = 0.651155
Reduced statistic      = 0.902569
Change in statistic    = 6.83386e+10
   p1.gamma       2.15852
   p1.ampl        0.00022484
```

This works very well much of the time, but ...

Doing more with Sherpa

Python Inside: Sherpa user interface and high-level functions are Python

- Sherpa provides an interface to let users:
  - Access the internal objects used within Sherpa
  - Easily define user model and user statistic functions
  - Use Sherpa as an imported library in your Python program
- Paradigm change – CIAO/Sherpa is not the environment, it is a powerful library tool in your Python analysis environment[1].
- Move beyond short *scripts* to full-blown *programs*[2].
- Real life examples:
  - Fit X-ray models to hundreds of Chandra L3 sources (including faint ones), put results in a database, and import to a google web app to browse the results.
  - Fit data where the errors are dominated by quantization (i.e. data are integerized)
  - Generate complex thermal models requiring parallel fitting using > 30 CPUs and a GUI fitting application

[1]Sherpa can even run outside of CIAO! See http://cxc.cfa.harvard.edu/contrib/sherpa/

[2]See http://www.astropython.org and http://python4astronomers.github.com for more about Python and astronomy

Topics

Take another swig of coffee and get ready for some code

- Getting data into Sherpa
- Digging into Sherpa: getting at the objects underneath
- Creating user models and user statistics functions
  - Using functions
  - Using classes (you too can write an OOP)
- Goodness of fit for low-count X-ray spectra
- Parallelization with MPI

Sit back and relax

- Deproject: a Sherpa extension module
- Keeping Chandra cool: a Sherpa success story

Getting data into Sherpa

- Sherpa has many ways of loading data and other things:

```
sherpa-17> load_<TAB>
load_arf              load_bkg_arf     load_filter      load_pha          load_state        load_user_model
load_arrays           load_bkg_rmf     load_grouping    load_preferences  load_staterror    load_user_stat
load_ascii            load_colormap    load_image       load_psf          load_syserror
load_ascii_transform  load_conv        load_multi_arfs  load_quality      load_table
load_bkg              load_data        load_multi_rmfs  load_rmf          load_table_model
```

- One of my favorites doesn't appear in any Sherpa thread[1] : `load_arrays()`
- This provides a generic way to load *memory arrays* as Sherpa datasets
- Example: ASCII file format not understood by Sherpa. Instead use asciitable[2]

```
sherpa> load_data('csc.rdb[cols ra, dec]')
IOErr: opening file  has failed with ERROR - Failed to open 'csc.rdb[cols ra, dec]'.

sherpa> import asciitable
sherpa> dat = asciitable.read('csc.rdb', Reader=asciitable.RdbReader)
sherpa> load_arrays(1, dat['ra'], dat['dec'], Data1D)
```

[1] From the google search “sherpa load_arrays”
[2] http://cxc.harvard.edu/contrib/asciitable

Sherpa

Getting data into Sherpa

- Didn't we just replace one line with three? But now we **own** the data!

```
sherpa> dat = asciitable.read('csc.rdb')
sherpa> ra = dat['ra']
sherpa> dec = dat'dec']
Sherpa> dist = calc_dist(ra, dec, ra.mean(), dec.mean())
sherpa> load_arrays(1, dist, dat['mag'], Data1D)
```

- `load_arrays()` works for 2-D and PHA data as well

HINT: get help on CIAO functions by googling “ahelp <function>” or “ciao ahelp <function>”. This doesn't seem to work with bing.

Digging into Sherpa: getting the good bits

- Sherpa also has many ways of showing the current analysis state:

```
sherpa> show_<TAB>
show_all show_bkg_model  show_conf  show_data   show_fit    show_method show_proj show_source
show_bkg show_bkg_source show_covar show_filter show_kernel show_model  show_psf  show_stat
```

```
sherpa> load_pha('acis_pha3.fits')
sherpa> set_source(xsphabs.abs1 * powlaw1d.p1)
sherpa> subtract()
sherpa> fit()
Sherpa> show_fit()
Optimization Method: LevMar
name    = levmar
ftol    = 1.19209289551e-07
xtol    = 1.19209289551e-07
gtol    = 1.19209289551e-07
maxfev  = None
epsfcn  = 1.19209289551e-07
factor  = 100.0
verbose = 0

Statistic: Chi2Gehrels
Chi Squared with Gehrels variance

Fit:Dataset               = 1
Method                 = levmar
Statistic              = chi2gehrels
Initial fit statistic = 31.5124
Final fit statistic    = 31.5124 at evaluation 5
Data points            = 1024
Degrees of freedom     = 1021
Probability [Q-value] = 1
Reduced statistic      = 0.0308642
Change in statistic    = 8.81487e-07
   abs1.nH        0.112892
   p1.gamma       3.07627
   p1.ampl        0.00011212
```

- Great for interactive analysis but what about *using* the results?
- OLD school
  - Run as a script and pipe output to a file
  - Write a separate script (perl?) to parse and store in a new table
  - Writing code to reliably parse all these tidbits is a very fun and interesting way to spend your day. NOT.
- Nice shiny way
  - Run as a python script
  - Directly access results and store in desired format .. or use python twitter API to immediately tweet the results.

Digging into Sherpa: getting the good bits

- Sherpa lets you `get_*` what you need:

```
sherpa> get_<TAB>
Display all 188 possibilities? (y or n)
get_analysis             get_contour_levels       get_fit_plot             get_model_plot           get_psf_plot
get_areascal             get_contour_range        get_fit_results          get_model_plot_prefs     get_pyType
get_arf                  get_contour_visible      get_frame                get_model_type           get_quality
get_arf_plot             get_contour_xrange       get_frame_border_visible get_num_par              get_rate
get_attribute            get_contour_yrange       get_frame_scale          get_num_par_frozen       get_ratio_contour
get_axes                 get_contour_zrange       get_frame_visible        get_num_par_thawed       get_ratio_image
get_axis                 get_coord                get_functions            get_number_cols          get_ratio_plot
get_axis_range           get_counts               get_grouping             get_number_rows          get_reg_proj
get_axis_scale           get_covar                get_histogram            get_object_coordinfo     get_reg_unc
get_axis_text            get_covar_opt            get_histogram_range      get_object_count         get_region
get_axis_transform       get_covar_results        get_histogram_xrange     get_order_plot           get_region_visible
get_axis_visible         get_covariance_results   get_histogram_yrange     get_par                  get_resid_contour
get_backscal             get_crate_item_type      get_image                get_photon_flux_hist     get_resid_image
get_bkg                  get_crate_type           get_image_range          get_pick                 get_resid_plot
get_bkg_arf              get_curve                get_image_visible        get_pileup_model         get_rmf
get_bkg_chisqr_plot      get_curve_range          get_image_xrange         get_piximg               get_server_id
get_bkg_delchi_plot      get_curve_visible        get_image_yrange         get_piximg_shape         get_source
get_bkg_fit_plot         get_curve_xrange         get_indep                get_piximgvals           get_source_contour
get_bkg_model            get_curve_yrange         get_int_proj             get_plot                 get_source_image
get_bkg_model_plot       get_data                 get_int_unc              get_plot_aspect_height   get_source_plot
get_bkg_plot             get_data_aspect_ratio    get_kernel_contour       get_plot_aspect_ratio    get_specresp
get_bkg_ratio_plot       get_data_contour         get_kernel_image         get_plot_aspect_width    get_split_plot
get_bkg_resid_plot       get_data_contour_prefs   get_kernel_plot          get_plot_range           get_stat
get_bkg_rmf              get_data_image           get_key                  get_plot_title           get_stat_name
get_bkg_source           get_data_plot            get_key_names            get_plot_visible         get_staterror
get_bkg_source_plot      get_data_plot_prefs      get_keyval               get_plot_xrange          get_syserror
get_chisqr_plot          get_default_depth        get_label                get_plot_yrange          get_transform
get_col                  get_default_id           get_label_text           get_point
get_transform_matrix
get_col_names            get_delchi_plot          get_line                 get_point_visible        get_transform_type
get_colorbar             get_dep                  get_method               get_preference           get_window
get_colorbar_border_visible get_dims              get_method_name          get_preferences          get_window_title
get_colorbar_visible     get_dmType               get_method_opt           get_proj                 get_xaxis
get_colvals              get_dmType_str           get_model                get_proj_opt             get_xsabund
get_conf                 get_energy_flux_hist     get_model_autoassign_func get_proj_results        get_xscosmo
get_conf_opt             get_error                get_model_contour        get_projection_results   get_xsxsect
get_conf_results         get_exposure             get_model_contour_prefs  get_psf                  get_yaxis
get_confidence_results   get_filter               get_model_image          get_psf_contour
get_contour              get_fit_contour          get_model_pars           get_psf_image
```

Digging into Sherpa: getting the good bits

- Most everything you get_*() will be a python object and that's the prize
  - Internally Sherpa uses hierarchical objects for most things
  - You can find and examine internal object attributes by <TAB> digging

```
sherpa> load_pha(1, 'acis_pha3.fits')
sherpa> dataset = get_data(1)
sherpa> dataset
        <DataPHA data set instance 'acis_pha3.fits'>
sherpa> dataset.<TAB>
Display all 159 possibilities? (y or n) n

sherpa> dataset.get_<TAB>
...
sherpa> counts = dataset.counts
sherpa> b = numpy.where(dataset.counts > 3)
sherpa> b
        (array([ 14,  16,  30,  45,  97, 118]),)
sherpa> c = dataset.channel[b]
sherpa> e = dataset._channel_to_energy(c)
array([ 0.2117,  0.2409,  0.4453,  0.6643,  1.4235, 1.7
```

numpy: core python numerical library
where(): return indices where expr is True

Return selected elements from channel array

I found this function just by <TAB> digging.
The _ in front means it wasn't intended for external use but why not live dangerously.
Documentation? Who needs it.

- With some care you can manipulate the internal object attributes

```
sherpa> dat = asciitable.read('csc.rdb', Reader=asciitable.RdbReader)
sherpa> load_arrays(1, dat['ra'], dat['dec'], Data1D)
sherpa> dataset = get_data()
sherpa> dataset.y = dataset.x**2
sherpa> dataset.staterror = dataset.y / 20
```

Digging into Sherpa: source and fit results

- Now something more useful: examine source model parameters and fit results

```
sherpa> source = get_source()
sherpa> source.parts
(<XSphabs model instance 'xsphabs.abs1'>,
 <PowLaw1D model instance 'powlaw1d.p1'>)

sherpa> for par in source.pars:
             print par.fullname, par.val, par.min, par.max, par.frozen

abs1.nH 0.112891604641 0.0 100000.0 False
p1.gamma 3.0762703235 -10.0 10.0 False
p1.ref 1.0 -3.40282346639e+38 3.40282346639e+38 True
p1.ampl 0.000112120110443 0.0 3.40282346639e+38 False

sherpa> fit = get_fit_results()
sherpa> print [x for x in dir(fit) if not x.startswith('_')]
['covarerr', 'datasets', 'dof', 'dstatval', 'extra_output', 'format', 'istatval',
'message', 'methodname',
 'modelvals', 'nfev', 'numpoints', 'parnames', 'parvals', 'qval', 'rstat', 'statname',
'statval', 'succeeded']
```

Digging into Sherpa: source and fit results

- Don't just examine. ***Organize*** and tabulate!

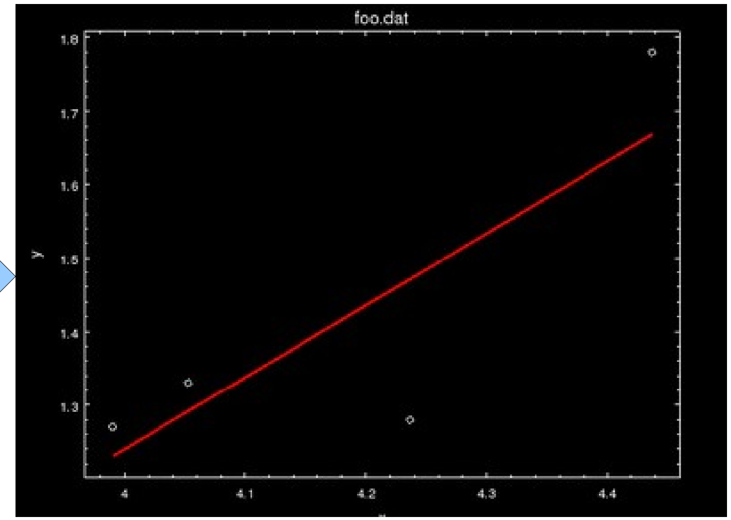
```
import sqlite3
conn = sqlite3.connect('csc_fits.db')
c = conn.cursor()
c.execute("""create table fit_pars
 (source_name text, par_name text, par_val real)""")

for parname, parval in zip(fit.parnames, fit.parvals):
    c.execute("insert into fit_pars values (?, ?, ?)",
              (source.name, parname, parval))

conn.commit()
c.close()
```

```
ccosmos% sqlite3 csc_fits.db
SQLite version 3.3.6
Enter ".help" for instructions
sqlite> select * from fit_pars;
(xsphabs.abs1 * powlaw1d.p1)|abs1.nH|0.112891604640818
(xsphabs.abs1 * powlaw1d.p1)|p1.gamma|3.07627032349591
(xsphabs.abs1 * powlaw1d.p1)|p1.ampl|0.000112120110443 43
```

User models with python functions

- Adding a user model defined with a python function is *shockingly simple!*

```
def myline(pars, x):
    return pars[0] * x + pars[1]

load_user_model(myline, "myl")
add_user_pars("myl", ["m","b"])
set_source(myl)

myl.m=30
myl.b=20
```



Sure, but any real model has to be written in C or Fortran, right? Not necessarily.

- Numerical processing with **NumPy** is in C so any vectorized calculations are fast.
- The **SciPy** library provides a large selection of optimized numerical algorithms using well known fortran and C numerical libraries.
- Prototype the user model in Python. If it's too slow then profile the code and convert the hot spots to C or C++.

But what about my existing C / Fortran model code? Google "sherpa user models".

Sherpa

User models with Python classes

- Frequently a user model function requires associated metadata (atomic data, table file names, non-fitted parameters, etc)
- This is a typical problem in fitting (remember Fortran COMMON blocks?)
- Python provides a very clean solution: **classes**

Ever wonder what's the deal with "object oriented programming"? Here it is. The object stores metadata.

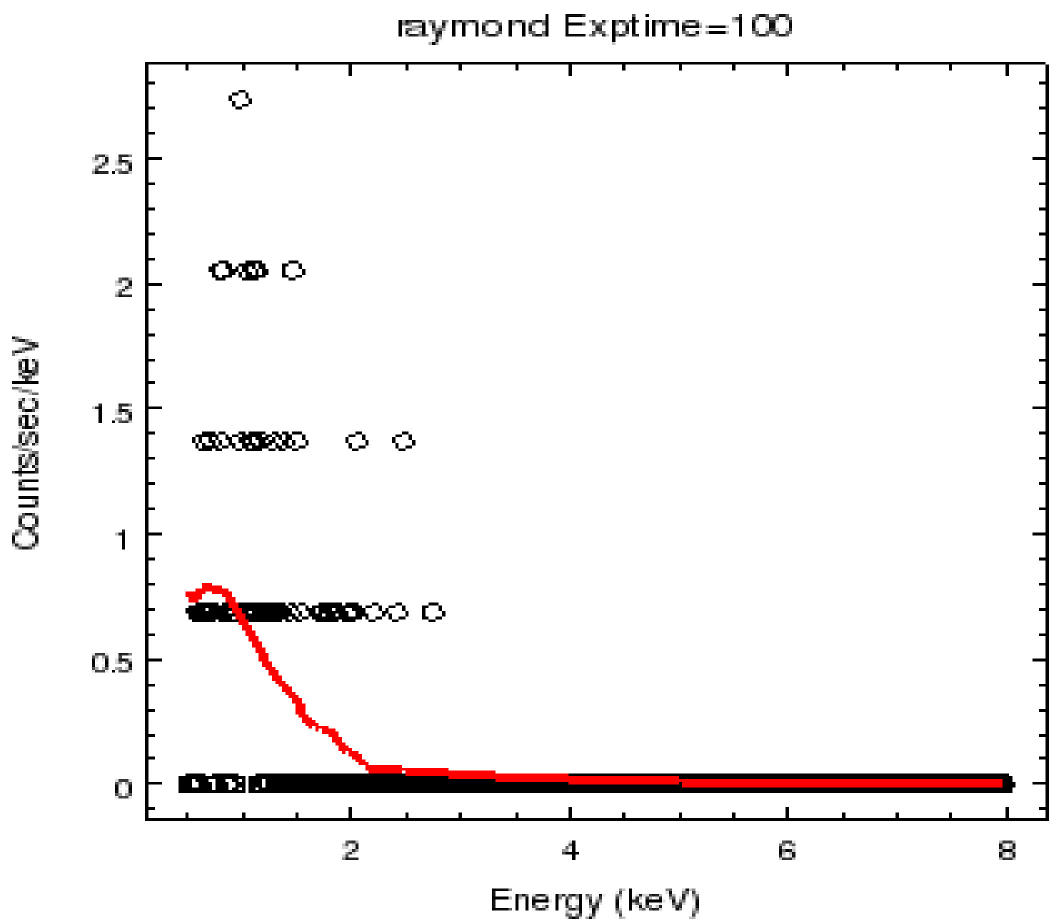
```
class FITS_TableModel(object):
    """Simplest possible FITS table model.  Table has two col
        kT       : temperature
        spectrum : corresponding spectrum in an N-element array
       In this model the spectrum nearest in temperature is returned.
       The energy bins of the fitted spectrum is ignored here.
    """
    def __init__(self, filename):
        hdus = pyfits.open(filename)
        self.kT = hdus[1].data.field('kT')
        self.spectra = hdus[1].data.field('spectrum')
        hdus.close()

    def __call__(self, pars, x):
        kT = pars[0]
        i = numpy.searchsorted(self.kT, [kT])[0]
        if (kT - self.kT[i-1]) < (self.kT[i] - kT):
            i -= 1
        return self.spectra[i]

user_model_func = FITS_TableModel('plasma_spectra.fits')
load_user_model(user_model_func,"myspec")
add_user_pars("myspec", ["kT"])
set_source(myspec)
```

Object initialization with "filename" via __init__. Read the FITS data and store within the object.

Here's the magic: the object created by the class can be called directly as a function and it will run the special __call__ method.

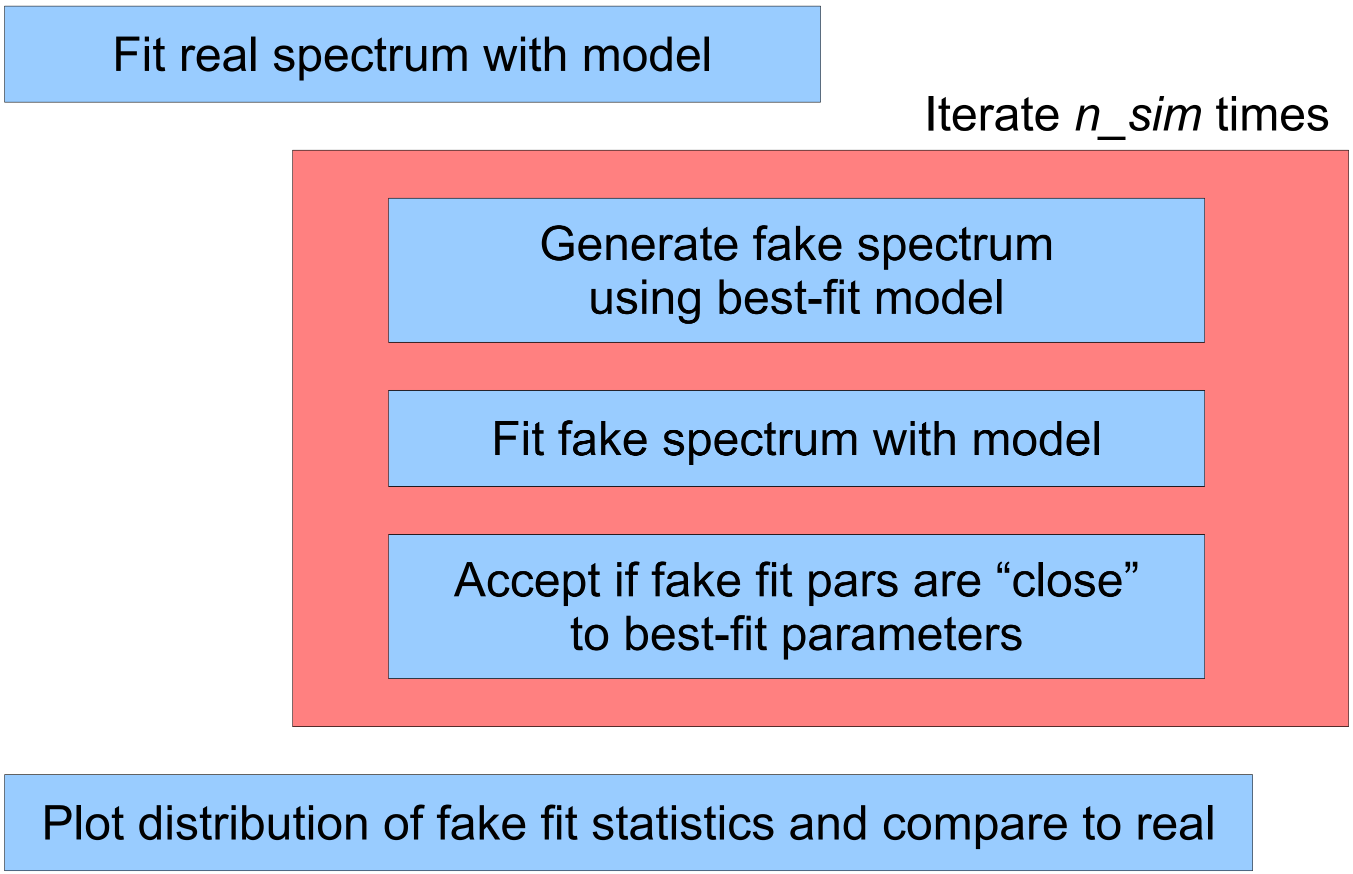
Goodness of fit by simulation

- Measuring the *goodness of fit* is challenging for low-count spectra
  - “Is the X-ray emission thermal or non-thermal (power-law)?”
  - Cannot easily eye-ball the data and model fit
  - No simple analogs to reduced Chi^2 distribution for Cash (likelihood) statistic
  - Challenging, but not impossible, and worth the trouble



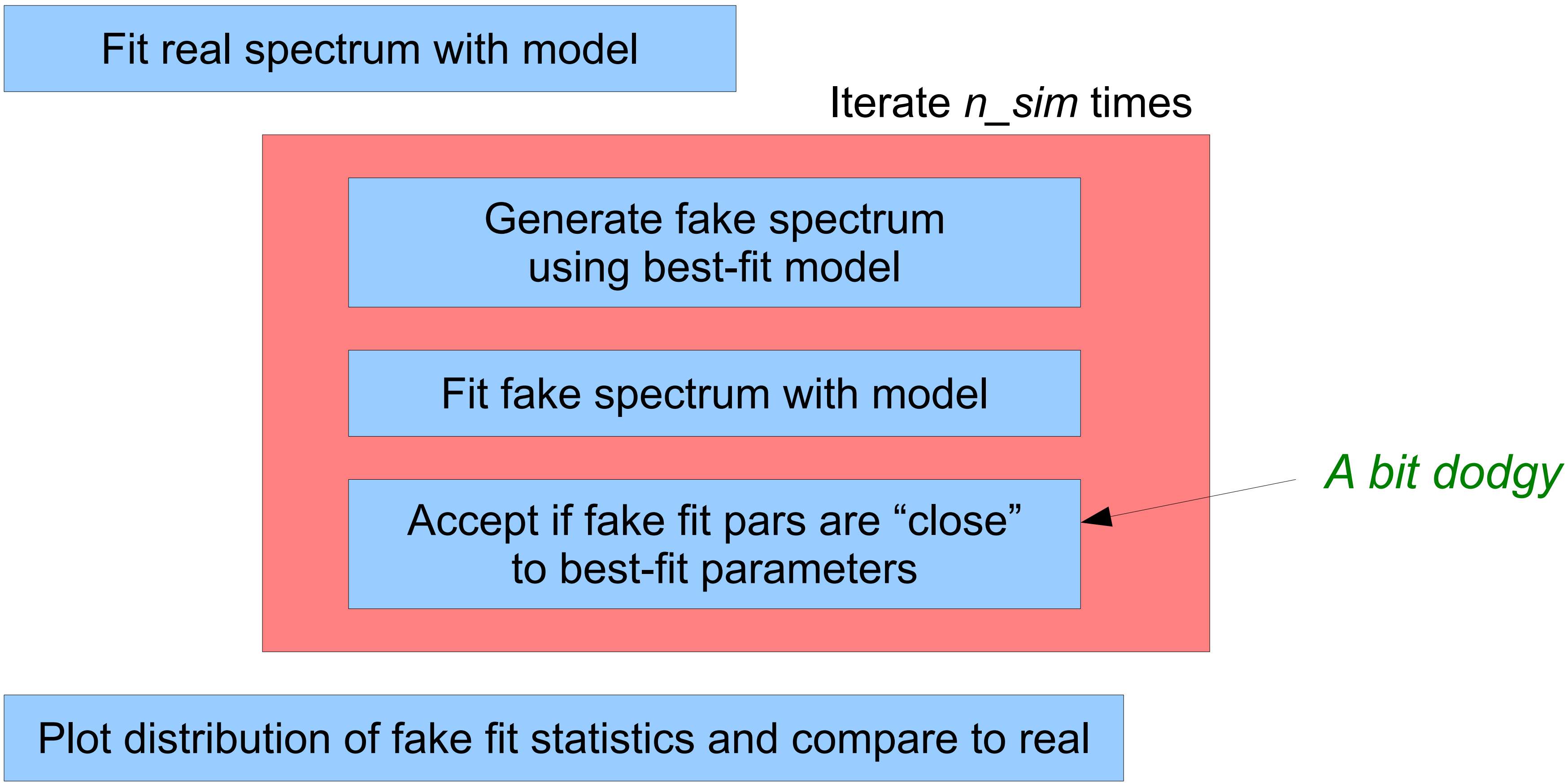
Goodness of fit by simulation

- Simulation provides a way to estimate whether the observed fit statistic would be unusual for an ensemble of data realizations of the source model[1].



Goodness of fit by simulation

- Simulation provides a way to estimate whether the observed fit statistic would be unusual for an ensemble of data realizations of the source model[1].



[1] *Warning: this algorithm is not statistician-approved and needs more testing.*

Sherpa

Goodness of fit by simulation

Model selection simulation:

- Generate a spectrum using a Raymond-Smith source model.
- Try to decide if a power-law model is consistent with the spectrum

```
import numpy as np

n_sim = 2000
exposure0 = 200
spectype = 'raymond'

# Generate the simulated "real" data (Raymond-Smith plasma kT=1.5 keV)
set_source(1, "xsraymond.ray1")
ray1.kt = 1.5
ray1.norm = 3e-3
arf = unpack_arf('acis7s.arf')
rmf = unpack_rmf('acis7s.rmf')
fake_pha(1, arf=arf, rmf=rmf, exposure=exposure0)

notice(0.5, 8)
set_method('levmar')
set_stat('cash')
```

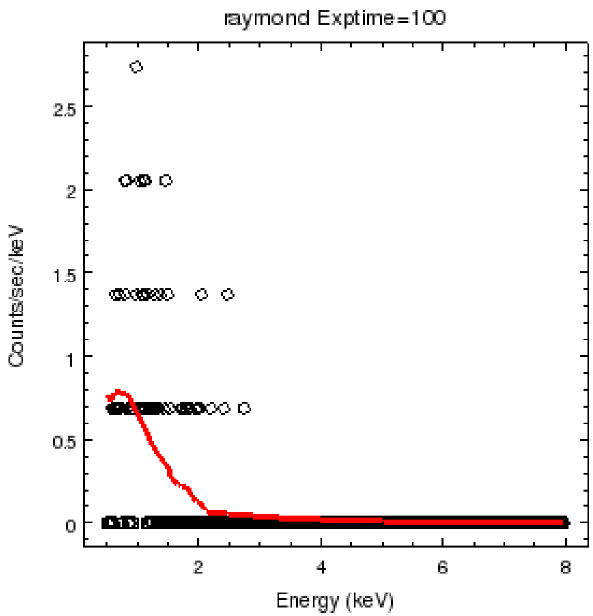
Goodness of fit by simulation

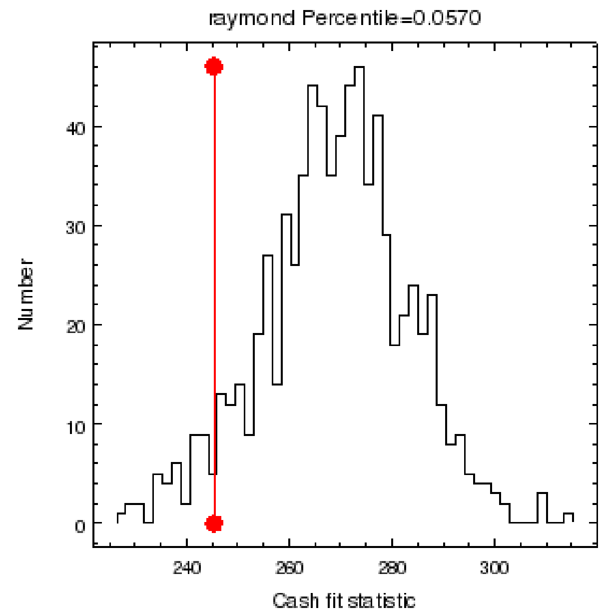
```
# Model this spectrum with an unabsorbed power law
set_source(1, "powlaw1d.pow1")
fit()
gamma0 = pow1.gamma.val
ampl0 = pow1.ampl.val

# Store results from initial fit of “real” data
fit_results = get_fit_results()
stat0 = fit_results.statval
flux0 = calc_energy_flux(lo=0.5, hi=8.0)

stats = np.zeros(n_sim)
gammas = np.zeros(n_sim)

for i in range(n_sim):
    # Reset parameter values to best fit and generate fake spectrum
    pow1.gamma = gamma0
    pow1.ampl = ampl0
    fake_pha(1, arf=arf, rmf=rmf, exposure=exposure0)

    # Fit and record statistics
    fit()
    stats[i] = calc_stat()
    gammas[i] = pow1.gamma.val
    print i
```

Goodness of fit by simulation

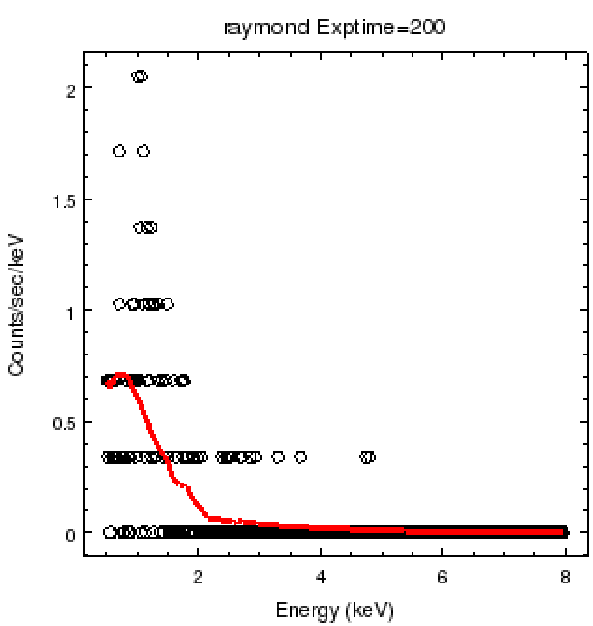
```
# Select the simulations that were "close" to the best-fit gamma
ok = (gammas > gamma0 - 0.1) & (gammas < gamma0 + 0.1)
stats = np.sort(stats[ok])

# Sort the statistics in order and rank the fit stat from the "real" data
n_stats = len(stats)
rank = np.searchsorted(stats, stat0)
percentile = float(rank) / n_stats
print 'fit stats (rank, n_stats, percentile) : {0} {1} {2:.4f}'.format(
    rank, n_stats, percentile)

# Make some plots
bin_vals, bin_edges = np.histogram(stats, bins=50)
bin_lefts = bin_edges[:-1]
bin_rights = bin_edges[1:]
add_window()
add_histogram(bin_lefts, bin_rights, bin_vals)

add_curve([stat0, stat0], [0, bin_vals.max()])
set_curve(['symbol.color', 'red',
           'line.color', 'red',
           'symbol.style', 'circle'])
set_plot_title('{0} Percentile={2:.4f}'.format(spectype, exposure0,
percentile))
set_plot_xlabel('Cash fit statistic')
set_plot_ylabel('Number')
print_window('{0}_hist_{1}.png'.format(spectype, exposure0))
```

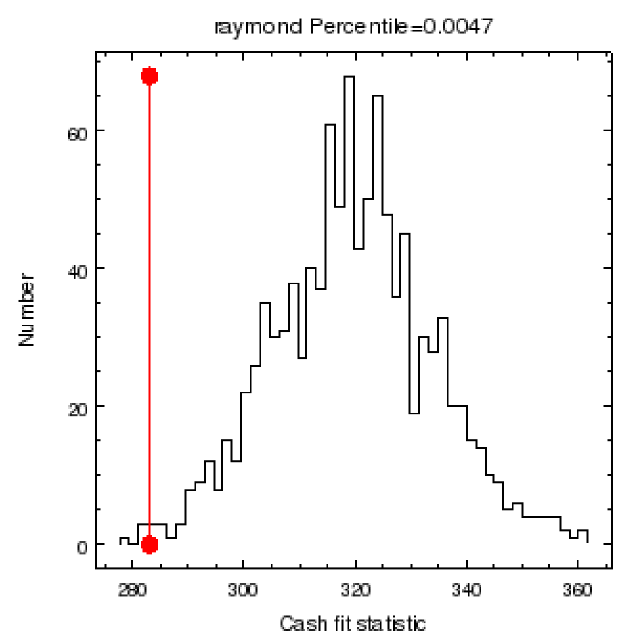
Goodness of fit by simulation





70 counts
Confidence ~0.94



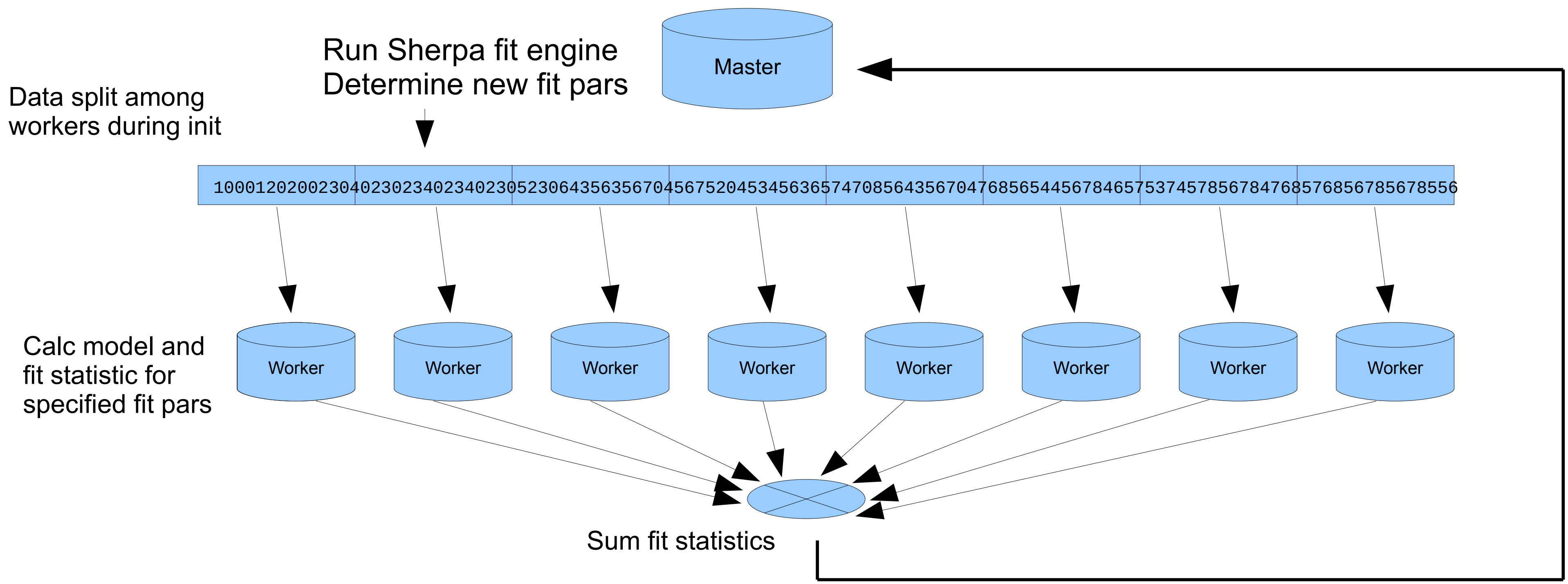


140 counts
Confidence ~0.995

“Power-law model is not a likely source for the spectrum”
(truth =Raymond-Smith kT=1.5)

Parallelization

- For some problems with large datasets or computationally intensive models it may be possible to improve fit performance by using multiple processors.
- Processors can be on the same machine or in a networked cluster.
- Sherpa already takes advantage of multiple cores in projection and conf.
- Improving fit performance is tricky for convolved models but easy for models that can be split in data space[1].



[1]Splitting in data space is just one of many possible strategies

Parallelization with MPI

- Can do parallel processing using C and Python implementations of the widely used Message Passing Interface standard.

```
class CalcModel(object):
    def __init__(self, x, y):
        msg = {'cmd': 'init', 'x': x, 'y': y}
        comm.bcast(msg, root=MPI.ROOT)

    def __call__(self, pars, x):
        comm.bcast(msg={'cmd': 'calc_model', 'par': par}, root=MPI.ROOT)
        return numpy.ones_like(x)    # Dummy value of correct length

def calc_staterror(data):
    return numpy.ones_like(data)

class CalcStat(object):
    def __call__(self, data, model, staterror=None, syserror=None, weight=None):
        msg = {'cmd': 'calc_statistic'}
        comm.bcast(msg, root=MPI.ROOT)
        fit_stat = numpy.array(0.0, 'd')
        comm.Reduce(None, [fit_stat, MPI.DOUBLE], op=MPI.SUM, root=MPI.ROOT)
        return fit_stat.tolist(), numpy.ones_like(data)

comm = MPI.COMM_SELF.Spawn(sys.executable, args=['fit_worker.py'], maxprocs=nproc)
load_arrays(1, x, y)
load_user_model(CalcModel(x, y), 'mpimod')
add_user_pars('mpimod', parnames)
set_model(1, mpimod)
load_user_stat('mpistat', CalcStat(), calc_staterror)
set_stat(mpistat)
fit(1)
```

Parallelization with MPI

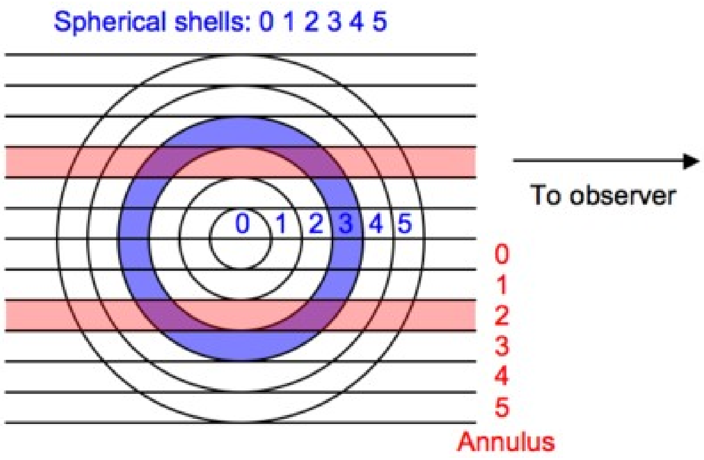
The fit_worker code just waits around to get instructions.

```
def calc_model(pars, x):
    # calculate the model values
    return model

comm = MPI.Comm.Get_parent()
size = comm.Get_size()
rank = comm.Get_rank()

while True:
    msg = comm.bcast(None, root=0)

    if msg['cmd'] == 'stop':
        break

    elif msg['cmd'] == 'init':
        i = numpy.int32(numpy.linspace(0.0, len(msg['x'], size+1))
        i0 = i[rank]
        i1 = i[rank+1]
        data_x = msg['x'][i0:i1]
        data_y = msg['y'][i0:i1]

    elif msg['cmd'] == 'calc_model':
        model = calc_model(msg['pars'], data_x)

    elif msg['cmd'] == 'calc_statistic':
        fit_stat = numpy.sum((data_y - model)**2)
        comm.Reduce([fit_stat, MPI.DOUBLE], None, op=MPI.SUM, root=0)

comm.Disconnect()
```

Deproject: a Sherpa extension module

Deproject is a CIAO Sherpa extension package to facilitate deprojection of two-dimensional annular X-ray spectra to recover the three-dimensional source properties.

- The deproject module creates a framework for manipulation of a stack of related input datasets and their models.
- Most of the functions resemble ordinary Sherpa commands (e.g. set_par, set_source, ignore) but operate on a stack of spectra.

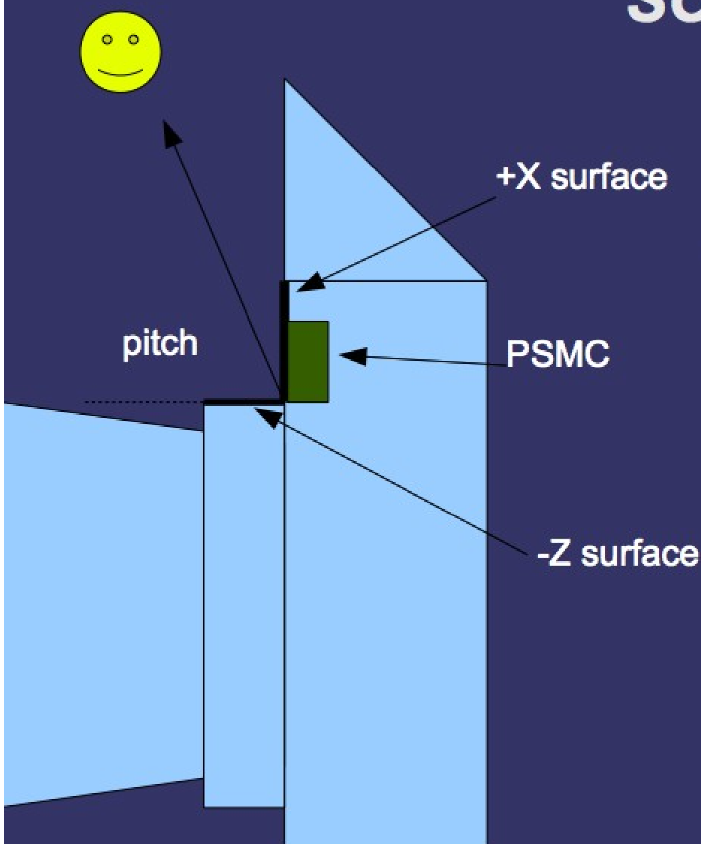


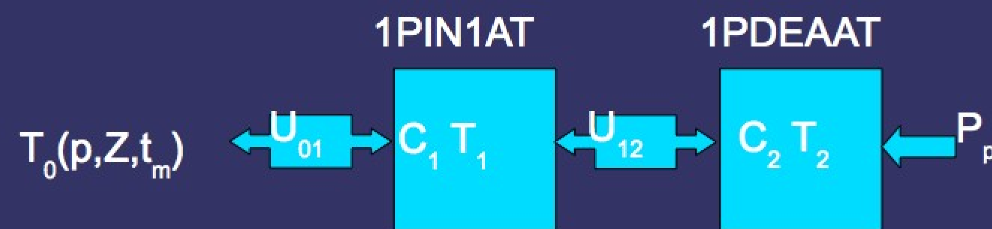
Keeping Chandra cool: a Sherpa success story

Keeping Chandra cool: a Sherpa success story

Keeping Chandra cool: a Sherpa success story

SOT PSMC model



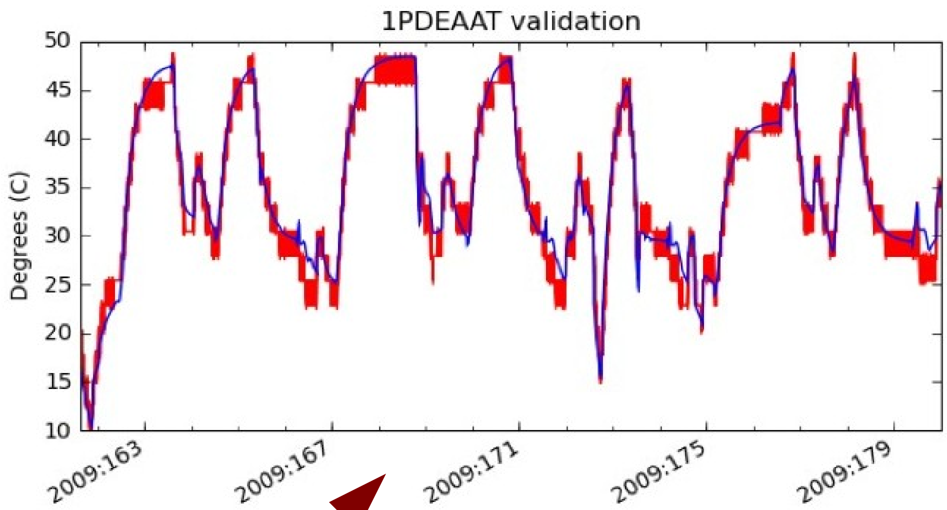


$$C_1 \frac{dT_1}{dt} = U_{01}(T_0 - T_1) + U_{12}(T_2 - T_1)$$

$$C_2 \frac{dT_2}{dt} = P_p + U_{12}(T_1 - T_2)$$

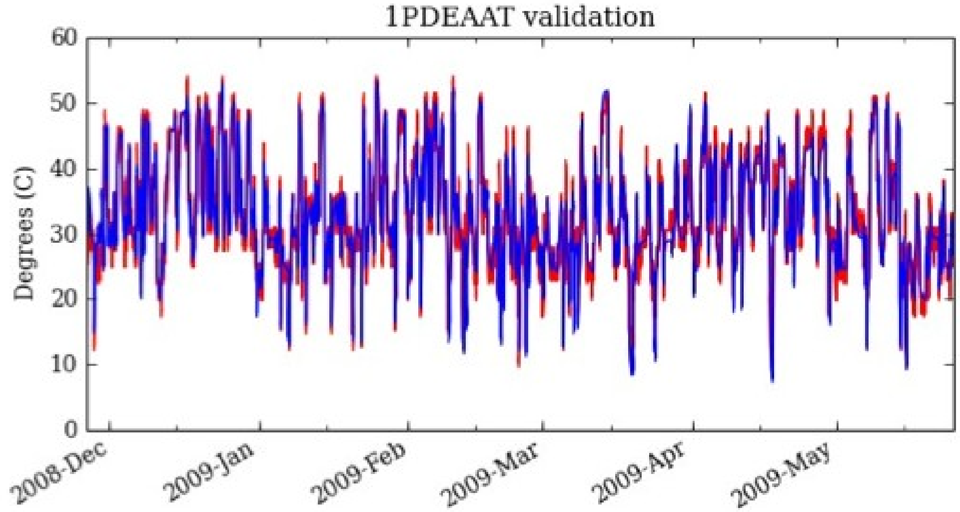
- Key inputs to model are pitch angle, SIM-Z position and ACIS power.
- Total of 13 model coefficients.

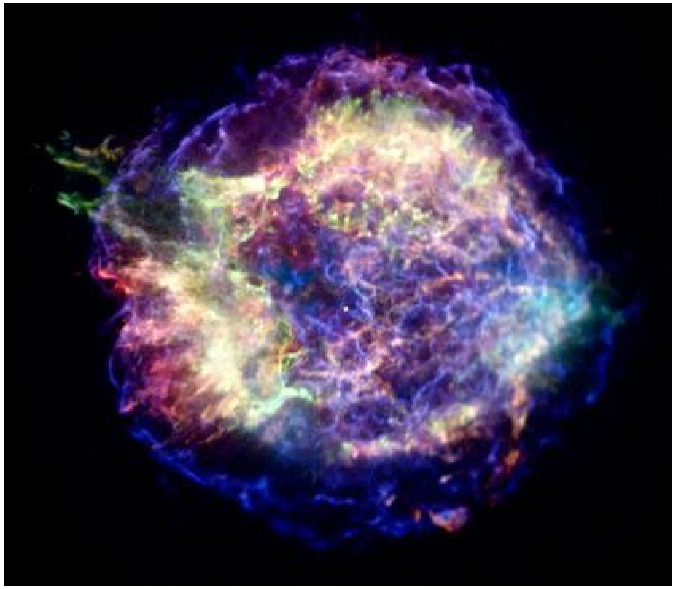
Keeping Chandra cool: a Sherpa success story



Predictions for mission planning

Calibration: Sherpa





Conclusions

- Sherpa provides next-generation capability through Python scripting
- Your time investment to learn Sherpa will pay off
- Learn Python!
  - http://python4astronomers.github.com
  - Python + analytic skills = Job security