

An Introduction to ISIS

(A Scriptable, Extensible Fitting Package)

Michael A. Nowak (MIT-Kavli Institute)

- with useful advice over the years from-

John Houck, John Davis, Dave Huenemoerder, Jörn Wilms

“High Level”
Data Analysis

Spectra

XSPEC

Sherpa

ISIS

SPEX

XRONOS

IDL

AstroPy

Timing

Custom
Analysis

Spacecraft
Specific
Software

Suzaku

Swift

NuSTAR

...

Chandra

XMM

SAS



Data Storage,
Manipulation,
Visualization

fv

CFITSIO/
FTOOLS

prism

CIAO/
DMTOOLS

DS9

XSELECT

“High Level”
Data Analysis

Spectra

XSPEC

Sherpa

ISIS

SPEX

XRONOS

IDL

AstroPy

Timing

Custom
Analysis

Spacecraft
Specific
Software

Suzaku

Swift

NuSTAR

...

Chandra

XMM

SAS

Data Storage,
Manipulation,
Visualization

fv

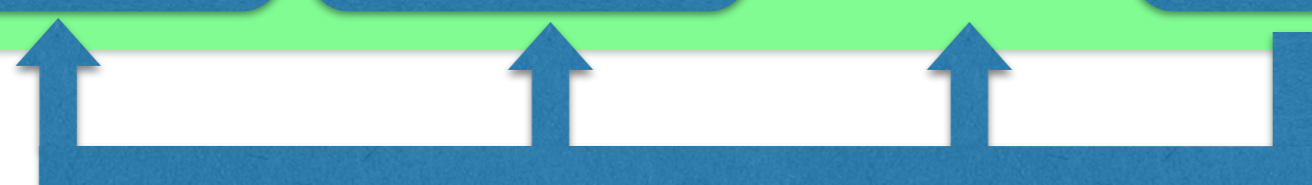
prism

CFITSIO/
FTOOLS

CIAO/
DMTOOLS

DS9

XSELECT



“High Level”
Data Analysis

Spectra

XSPEC

Sherpa

ISIS

SPEX

XRONOS

IDL

AstroPy

Timing

Custom
Analysis

Spacecraft
Specific
Software

Suzaku

Swift

NuSTAR

...

Chandra

XMM

SAS

Data Storage,
Manipulation,
Visualization

fv

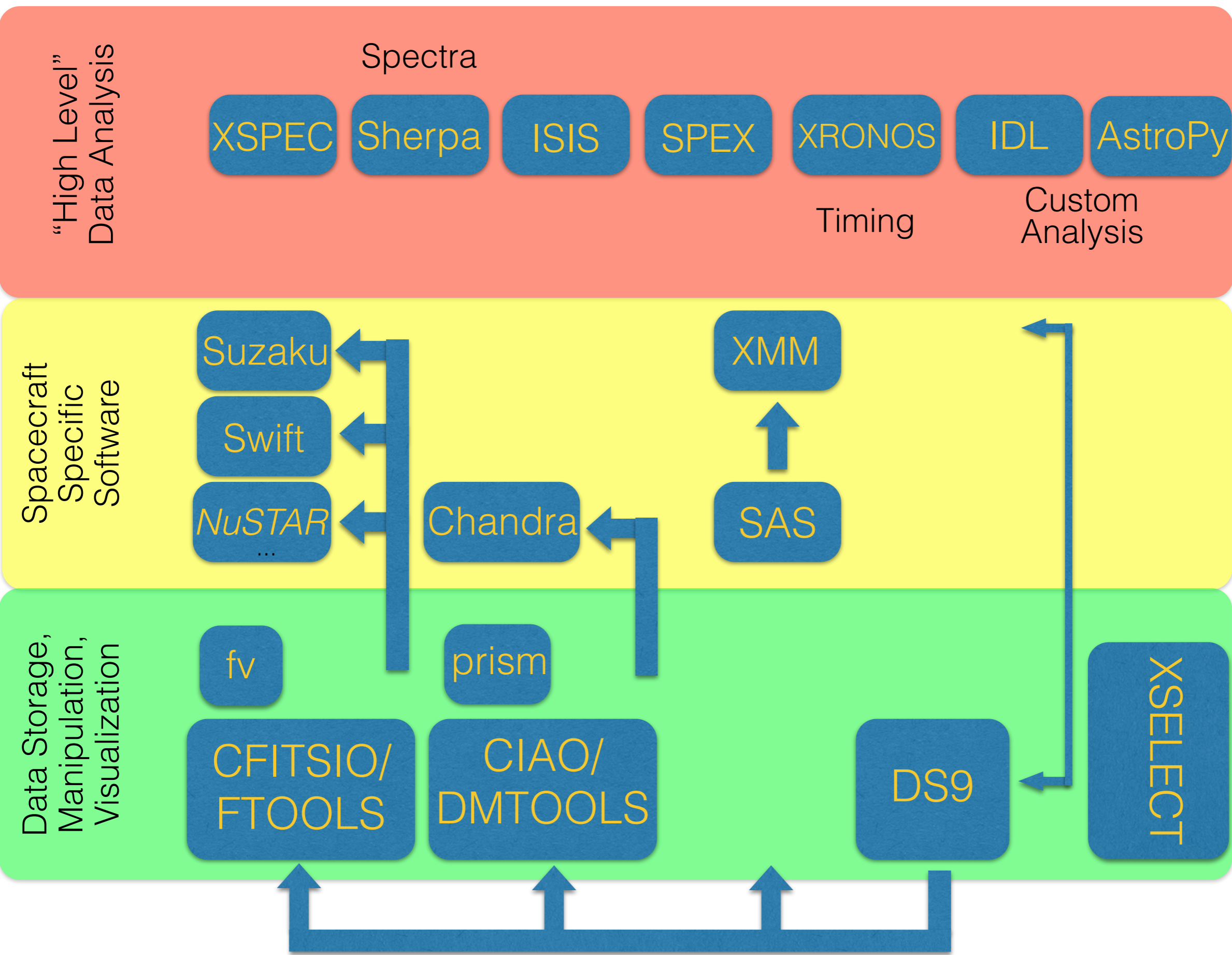
prism

CFITSIO/
FTOOLS

CIAO/
DMTOOLS

DS9

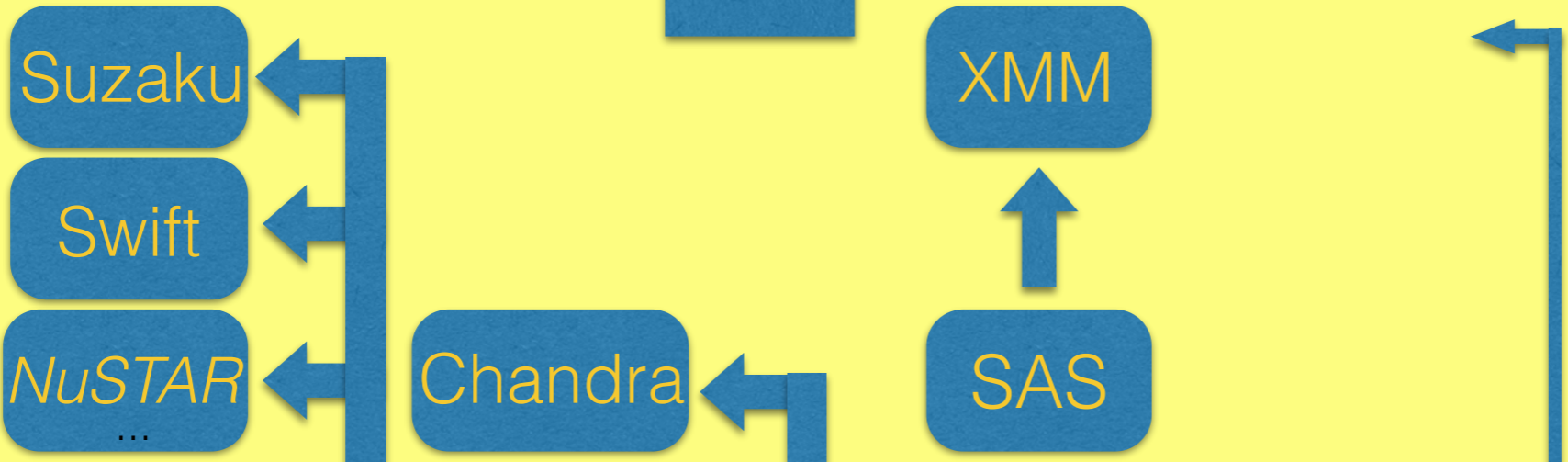
XSELECT



“High Level”
Data Analysis

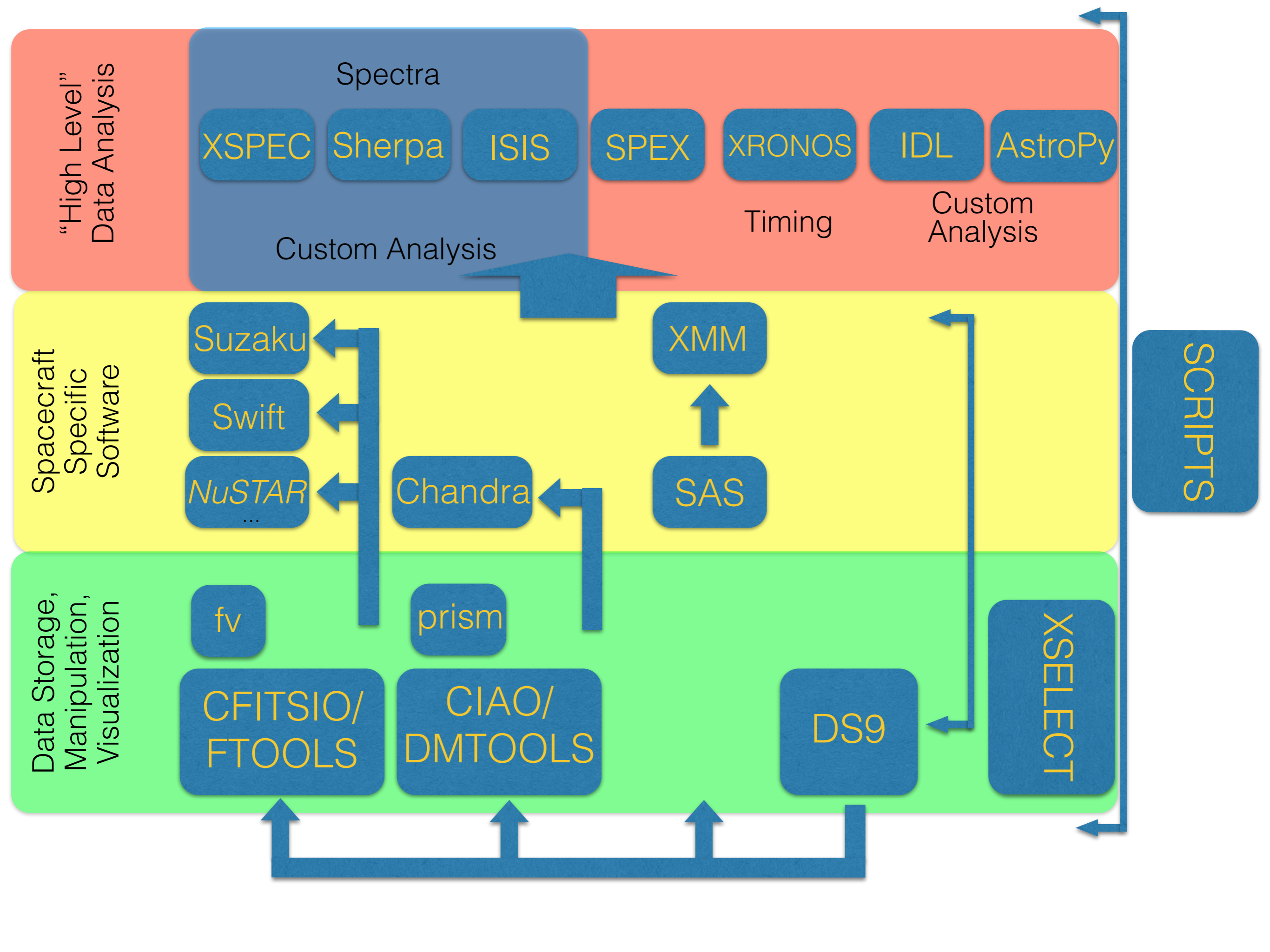


Spacecraft
Specific
Software



Data Storage,
Manipulation,
Visualization





“High Level”
Data Analysis

Custom Analysis

XSPEC

Sherpa

ISIS

Modules

Python

Python

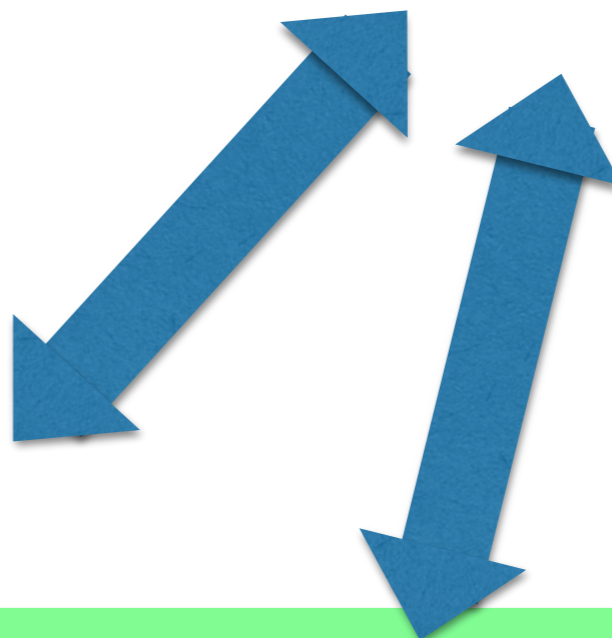
S-lang

Data Storage,
Manipulation,
Visualization

CFITSIO/
FTOOLS

CIAO/
DMTOOLS

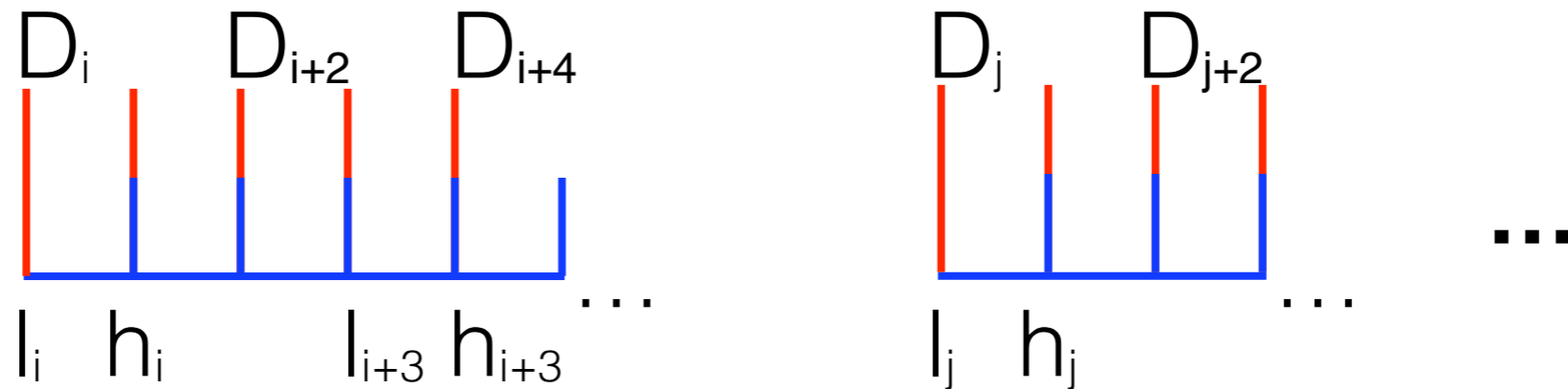
DS9



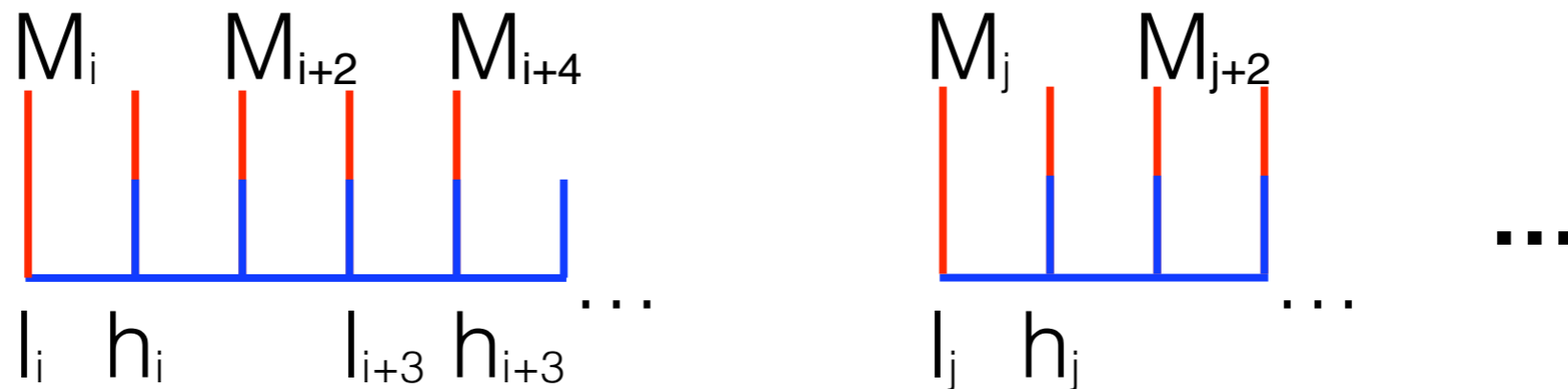
Data (Spectral) Analysis

- Four major choices: XSPEC, SPEX, Sherpa, ISIS
 - XSPEC – oldest & most established; many models written for it
 - SPEX – “specialty” package for high-resolution X-ray spectroscopy
 - Sherpa – youngest & 2nd most programmable (Sherpa & XSPEC use Python; Sherpa was built “ground up” in Python)
 - ISIS – I use for almost all mathematical analysis (spectra, timing, etc.). ISIS uses XSPEC models, and does many other things

ISIS Mechanics



3 Vectors: Lower Bin, Higher Bin, (Data) Value of Bin



3 Vectors: Lower Bin, Higher Bin, (Model) Value of Bin

`convol_fun(1, exp(-fun_A)*fun_B*(func+...)) + 0.*fun_D...`

ISIS Models

phabs*(diskbb+powerlaw+gaussian)

phabs*(diskbb+powerlaw+gaussian(1)+gaussian(2))

phabs***exp(lines()/bin_width)***(diskbb+powerlaw)

ISIS Models

`phabs*(diskbb+powerlaw+gaussian)`

`phabs*(diskbb+powerlaw+gaussian(1)+gaussian(2))`

`phabs*exp(lines()/bin_width)*(diskbb+powerlaw)`



Custom function,
user redefined during fit

ISIS Philosophy

- Models and Data are Vectors, Parameters are Numbers
 - When in doubt, it's just Math.
 - No additive/multiplicative (but convolution different)!
- All Commands are S-lang Functions or Subroutines
 - Vector Math is intrinsic to S-lang
- Access/Ability to Manipulate these Numbers
- You Can Customize Many, Many Things
- With Great Power Comes Great Responsibility
 - ISIS will let you do what you want, but is less likely than XSPEC to try and guess what you want...

Default ISIS Fitting

The diagram illustrates the ISIS fitting equation with the following components and flow:

- Input or Diagonal**: Points down to the $R_i(h, E)$ term.
- Input or 1**: Points down to the $A_i(E)$ term.
- Input or 1**: Points down to the $S_i(E)$ term.
- Detected Counts**: Points up to the $C(h)$ term.
- Response Matrix**: Points up to the $R_i(h, E)$ term.
- Effective Area**: Points up to the $A_i(E)$ term.
- Spectral Model**: Points up to the $S_i(E)$ term.
- Background Counts**: Points up to the $B(h)$ term.

$$C(h) = \int_0^\infty \sum_i R_i(h, E) A_i(E) S_i(E) dE dT + B(h)$$

- Poisson statistics, but can be changed
- Regrouping (normally not done in XSPEC) sums channels
- Error propagation is Poisson, but can be changed
- Customization via S-lang scripting


```
mnowak%> isis -g
```

```
isis> % What follows after % isn't read by the program
```

```
isis> print("hello world");           % Commands end with ;  
"hello world"
```

```
isis> variable a = [0:10];           % Vector math is intrinsic  
isis> b = a^2;                       % No declaration on command line  
isis> c = log10(sin(b)+exp(a));      % Common math functions exist
```

```
isis> plot(a,c);                     % Results can be plotted
```

```
isis> id = open_plot("my_first_plot.ps/vcps");  
isis>     apj_size; nice_width;  
isis>     xlabel("X"); ylabel("Y");  
isis>     plot(a,c);  
isis> close_plot(id);
```

(I Will Use Purple for Functions I Define in My .isisrc)

```
isis> print
```

```
(
```

```
"hello world"
```

```
));
```

```
)
```

```
;
```

```
"hello world"
```

```
isis> close_plot(id);
```

isn't read by the program

```
% Commands end with ;
```

```
% Vector math is intrinsic
```

```
% No declaration on command line
```

```
)); % Common math functions exist
```

```
% Results can be plotted
```

```
st_plot.ps/vcps");
```

```
;
```

```
"Y");
```

*(I Will Use **Purple** for Functions I Define in My **.isisrc**)*

```
mnowak%> isis -g
```

```
isis> % What follows after % isn't read by the program
```

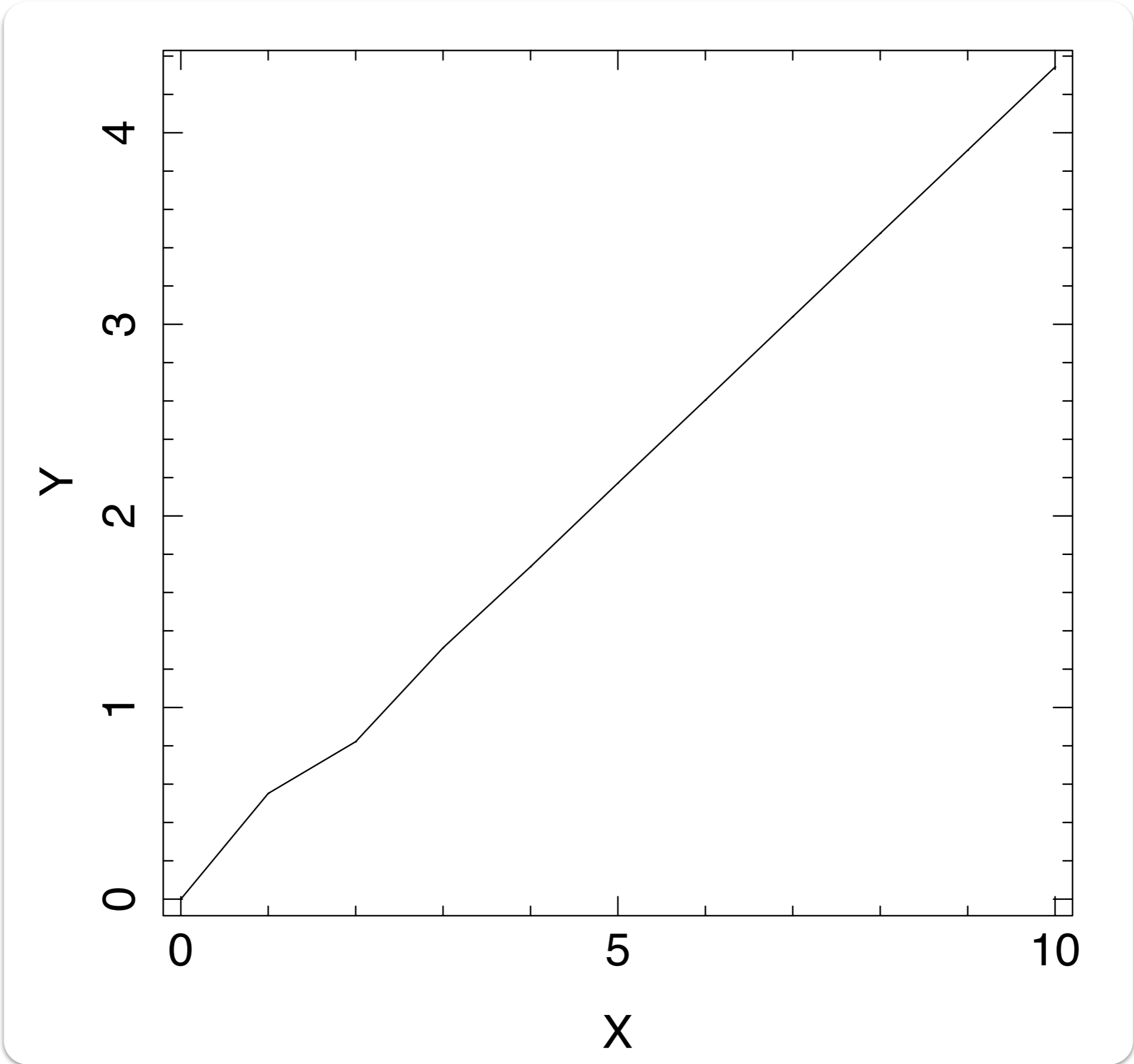
```
isis> print("hello world");           % Commands end with ;  
"hello world"
```

```
isis> variable a = [0:10];           % Vector math is intrinsic  
isis> b = a^2;                       % No declaration on command line  
isis> c = log10(sin(b)+exp(a));      % Common math functions exist
```

```
isis> plot(a,c);                     % Results can be plotted
```

```
isis> id = open_plot("my_first_plot.ps/vcps");  
isis>     apj_size; nice_width;  
isis>     xlabel("X"); ylabel("Y");  
isis>     plot(a,c);  
isis> close_plot(id);
```

(I Will Use Purple for Functions I Define in My .isisrc)



```
isis> () = evalfile("/path/my_script.sl");
isis> () = evalfile("/path/.isisrc");

isis> public define returns_abc()
    {
        variable a="a";
        variable b="b";
        variable c="c";
        return a,b,c;
    }
isis>

isis> returns_abc;                % Dump to screen
c
b
a
isis> (a,b,c) = returns_abc();    % Capture output
isis> (,,) = returns_abc();       % Discard output
```



```
isis> () = evalfile("/path/my_script.sl").
```

File: unix%> ~myhome/.isisrc

Many programs look for an "rc" file in your home directory.

.xspecrc, .sherparc, .ciaorc, .xinitrc

*These are places where you can put **your customizations**.*

*.isisrc is a good place to put customizations and programs you will use over and over again. **No recompiling of ISIS is required.***

Environment variable ISIS_HISTORY_FILE keeps a record of ISIS commands, which one can scroll backwards through (GNU readline installed).

```
ISIS> (,,) = returns_abc();          % Discard output
```

```
isis> () = evalfile("/path/my_script.sl");
isis> () = evalfile("/path/.isisrc");

isis> public define returns_abc()
    {
        variable a="a";
        variable b="b";
        variable c="c";
        return a,b,c;
    }
isis>

isis> returns_abc;                % Dump to screen
c
b
a
isis> (a,b,c) = returns_abc();    % Capture output
isis> (,,) = returns_abc();       % Discard output
```

```
isis> a = [0:10]; b = 3.5; c = "pizza";
```

```
isis> who;
```

```
a: Integer_Type[11]
```

```
b: 3.5
```

```
c: pizza
```

```
isis> .apropos arf
```

```
Found 15 function matches in namespace Global:
```

_nonstandard_arf_hdu_names	all_arfs	assign_arf
define_arf	delete_arf	get_arf
get_arf_exposure	get_arf_info	get_rmf_arf_grid
list_arf	load_arf	put_arf
set_arf_exposure	set_arf_info	unassign_arf

```
Found 6 variable matches in namespace Global:
```

Allow_Multiple_Arf_Factors	Assigned_ARFRMF	Assigned_RMFARF
Ideal_ARF	Ideal_ARFRMF	Ideal_RMFARF

```
isis> .help load_arf
load_arf
```

SYNOPSIS

Load an effective area (ARF) file

USAGE

```
status = load_arf ("filename")
```

DESCRIPTION

This function loads either a FITS Type I or Type II ARF file; the updated list of currently loaded ARFs is automatically displayed. On return, status is equal to the integer index of the ARF just loaded (status > 0); a return value of status = -1 is used to indicate failure. (For Type II ARF input, a return value of zero indicates success).

SEE ALSO

load_dataset, list_arf, delete_arf, assign_arf, unassign_arf

```
isis> alias("load_arf", "arf");
isis> alias("fit_fun", "model");
```

```
isis> pca_id = load_data("pca.pha");      % Read pca data
isis> hexte_id = load_data("hxt.pha");    % Read hexte data

isis> % pca_id = 1, hexte_id =2
```

If ARF, RMF, and Background information are in the FITS headers, then these components will also be automatically loaded & associated with spectra.

Data errors are taken from file (STAT_ERR column, combined with SYS_ERR column or keyword), ***unless grouping is applied to the data.***

File ***grouping will not be automatically applied*** unless
Isis_Use_PHA_Grouping > 0.

ISIS Let's You Group Data on the Fly!

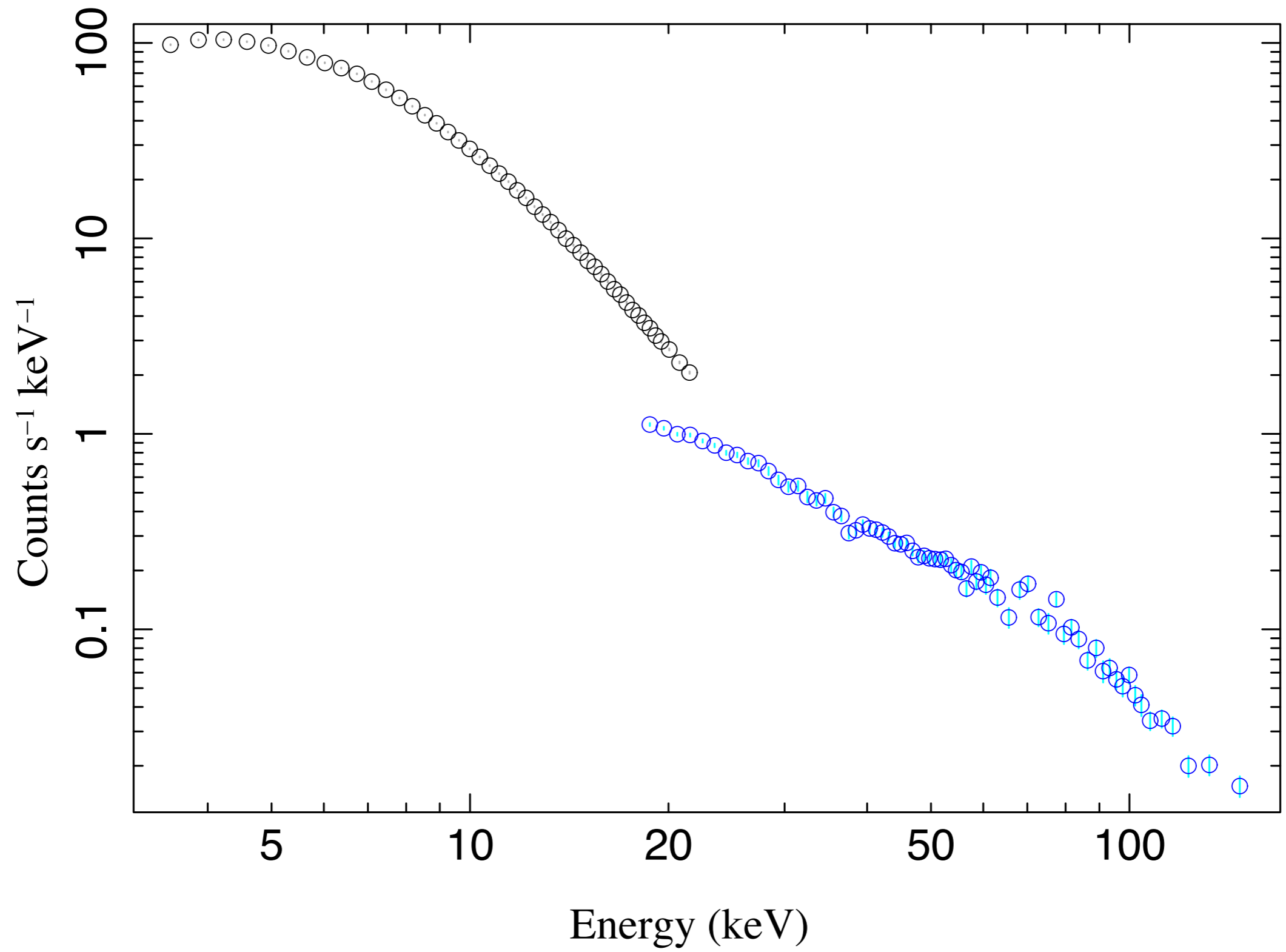
```
isis> group(pca_id; min_sn=4.5, bounds=3, unit="kev");  
isis> notice_values(pca_id,3,22; unit="kev");
```

```
isis> group(hexte_id; min_sn=8, bounds=18, unit="kev");  
isis> notice_values(hexte_id,18,200; unit="kev");
```

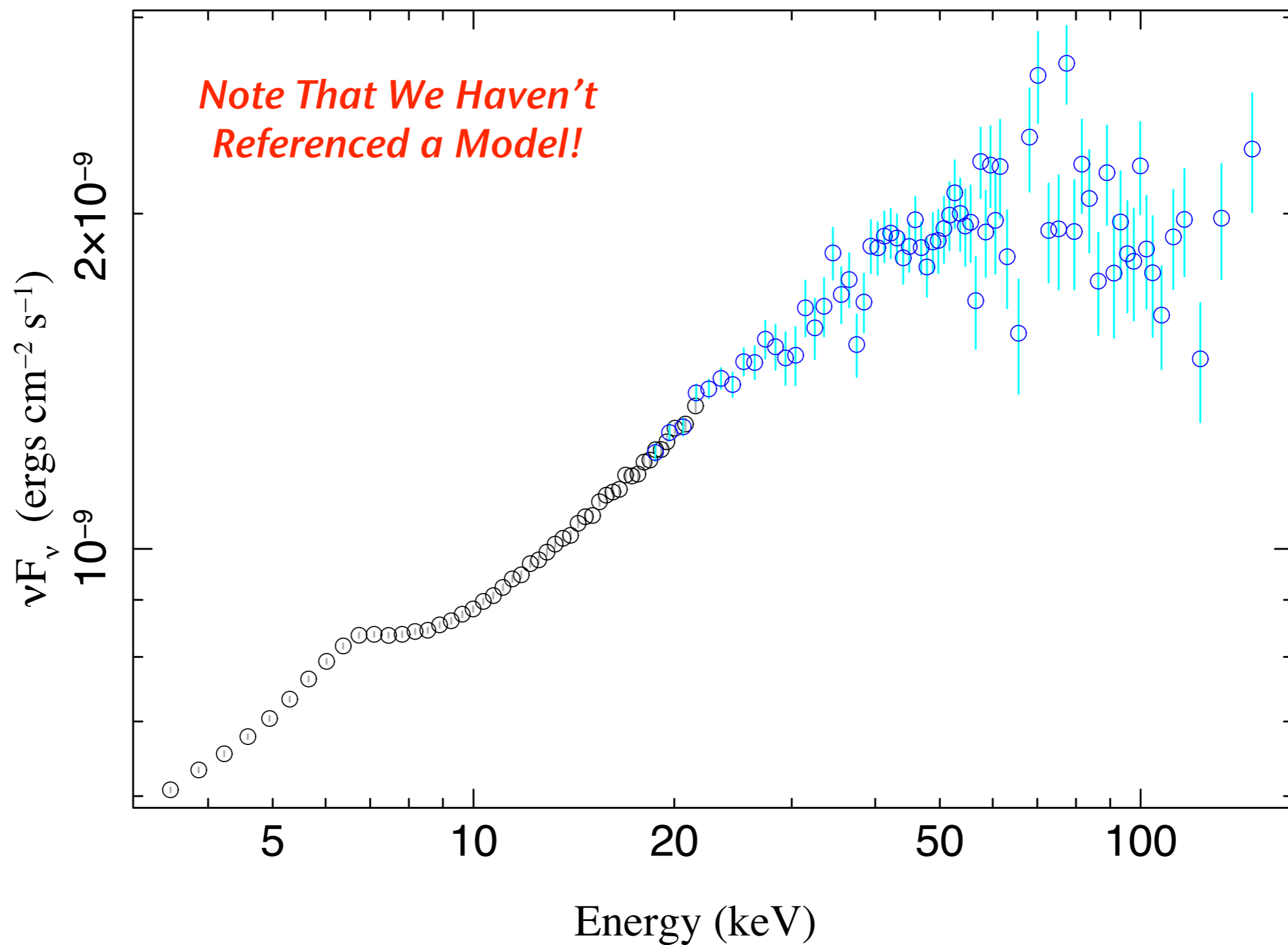
```
isis> xlog; ylog;  
isis> fancy_plot_unit("kev","ergs");
```

Also setting ISIS intrinsic plot units to: kev

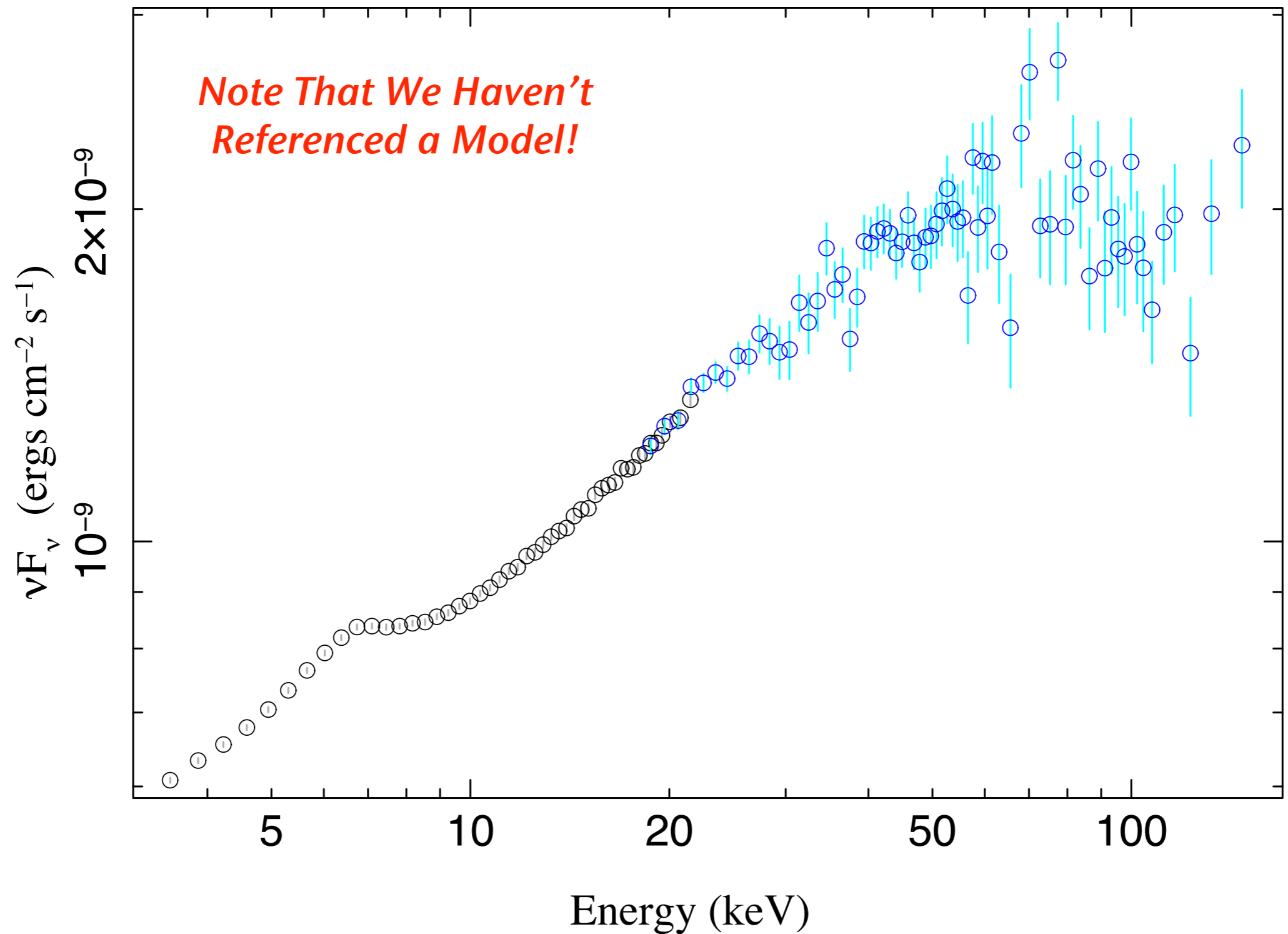
```
isis> plot_data({pca_id,hexte_id};dcol={1,4},decol={15,5});  
isis> plot_data({1,2};dcol={1,4},decol={15,5});
```



```
plot_unfold({1,2};dcol={1,4},decol={15,5},scale={1,1.07});
```



```
plot_unfold({1,2};dcol={1,4},decol={15,5},scale={1,1.07});
```



Use with Caution! And Never, Ever, Under Any Circumstances Use the XSPEC: plot eeufs!

```
isis> model("constant(Isis_Active_Dataset)*phabs*highcut*(bknpower+gaussian)");
```

*Model is: A Power Law with a “Break” Plus a Line,
that “exponentially rolls over” at high energies,
that is absorbed at low energies,
and
has a different normalization for PCA and HEXTE*

*Isis_Active_Dataset is a powerful concept that let's you
alter how the model is applied to different data sets, and
even allows you to alter how models behave!*

```
isis> list_free;
```

```
constant(Isis_Active_Dataset)*phabs*highcut*(bknpower+gaussian)
```

idx	param	tie-to	freeze	value	min	max	
2	phabs(1).nH	0	0	0.5	0	5	10 ²²
3	highcut(1).cutoffE	0	0	10	8	300	keV
5	bknpower(1).norm	0	0	1	0	1e+10	
6	bknpower(1).PhoIndx1	0	0	1.8	1	3	
7	bknpower(1).BreakE	0	0	10	8	15	keV
8	bknpower(1).PhoIndx2	0	0	1.6	1	3	
9	gaussian(1).norm	0	0	0.1	0	10	
10	gaussian(1).LineE	0	0	6.4	6	7	keV
11	gaussian(1).Sigma	0	0	0.3	0	1	keV
12	constant(2).factor	0	0	0.9345794	0.75	1.25	

```
isis> edit_par;          % Starts an editor, or use the commands below ...

isis> set_par("constant(1).factor",1,1);          % Second 1 means => frozen
isis> set_par("phabs(1).nH",0.5,0,0,5);          % Limited range in search
isis> set_par("high*cutoffE",10,0,8,300);        % Wild cards OK
isis> set_par(4,100,10,0.01,1000);              % Or use parameter numbers
isis> set_par(6,1.8,0,1,3);                      % Photon Index 1
isis> set_par(7,10,0,8,15);                      % Break Energy
isis> set_par(8,1.6,0,1,3);                      % Photon Index 2
isis> set_par(9,0.1,0,0,10);                    % Gaussian Norm
isis> set_par("*LineE",6.4,0,6,7);              % Line Energy
isis> set_par("*Sigma",0.3,0,0,1);              % Line Width
isis> set_par("constant(2).factor",1/1.07,0,0.75,1.25);
```

```
isis> () = fit_counts;
Parameters[Variable] = 12[10]
      Data bins = 118
      Chi-square = 171.7534
      Reduced chi-square = 1.590309
```

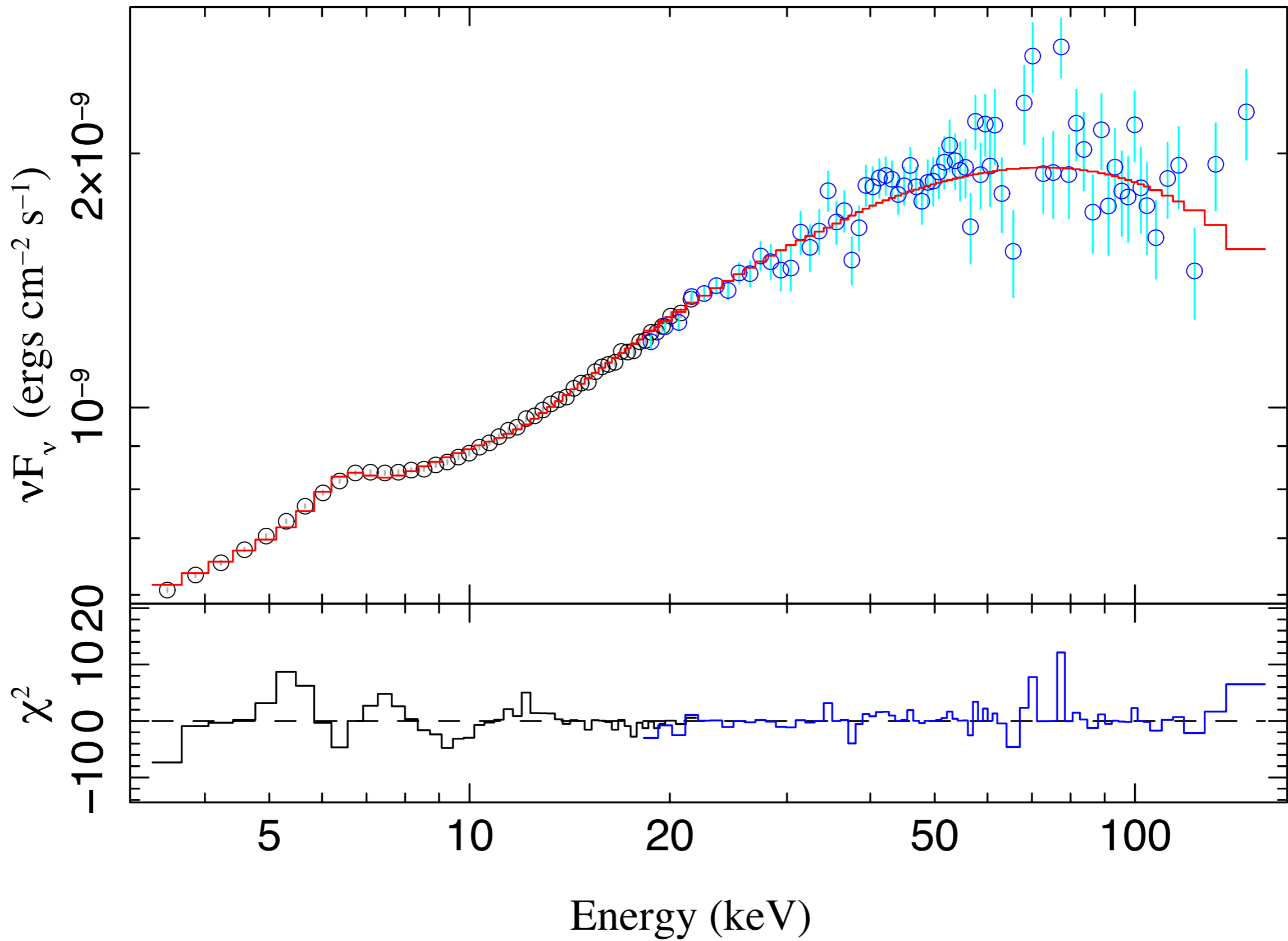
```
isis> plot_unfold({1,2};dcol={1,4},decol={15,5},scale={1,1/get_par(12)},res=2);
```

```
isis> edit_par;           % Starts an editor, or use the commands below ...

isis> set_par("constant(1).factor",1,1);           % Second 1 means => frozen
isis> set_par("phabs(1).nH",0.5,0,0,5);           % Limited range in search
isis> set_par("high*cutoffE",10,0,8,300);         % Wild cards OK
isis> set_par(4,100,10,0.01,1000);               % Or use parameter numbers
isis> set_par(6,1.8,0,1,3);                       % Photon Index 1
isis> set_par(7,10,0,8,15);                       % Break Energy
isis> set_par(8,1.6,0,1,3);                       % Photon Index 2
isis> set_par(9,0.1,0,0,10);                     % Gaussian Norm
isis> set_par("*LineE",6.4,0,6,7);               % Line Energy
isis> set_par("*Sigma",0.3,0,0,1);              % Line Width
isis> set_par("constant(2).factor",1/1.07,0,0.75,1.25);
```

```
isis> () = fit_counts;
Parameters[Variable] = 12[10]
      Data bins = 118
      Chi-square = 171.7534
      Reduced chi-square = 1.590309
```

```
isis> plot_unfold({1,2};dcol={1,4},decol={15,5},scale={1,1/get_par(12)},res=2);
```

Confidence Ranges are Parallelized!

```
isis> (,) = conf_loop(,1,0.1;save,prefix="best_fit.",num_slaves=2,nice=19);
```

```
...
```

```
isis> save_par("best_fit.par");
```

```
isis> load_par("best_fit.save");
```

```
isis> () = eval_counts;
```

```
Parameters[Variable] = 12[10]
```

```
    Data bins = 118
```

```
    Chi-square = 111.0569
```

```
    Reduced chi-square = 1.028305
```

```
isis> list_free;
```

```
constant(isis_Active_Dataset)*phabs*highcut*(bknpower+gaussian)
```

idx	param	tie-to	freeze	value	min	max	
2	phabs(1).nH	0	0	0	0	0.4477608	10 ²²
3	highcut(1).cutoffE	0	0	12.86972	11.55878	13.99137	keV
5	bknpower(1).norm	0	0	0.2360461	0.2328648	0.240283	
6	bknpower(1).PhoIndx1	0	0	1.628581	1.622399	1.637057	
7	bknpower(1).BreakE	0	0	12.81679	12.11459	13.35711	keV
8	bknpower(1).PhoIndx2	0	0	1.249342	1.231995	1.269046	
9	gaussian(1).norm	0	0	0.002511571	0.002188707	0.002860817	
10	gaussian(1).LineE	0	0	6.19454	6.083825	6.302416	keV
11	gaussian(1).Sigma	0	0	0.8079406	0.6850875	0.9338861	keV
12	constant(2).factor	0	0	0.9480235	0.9355516	0.9613026	

Confidence Ranges are Parallelized!

```
isis> (,) = conf_loop(,1,0.1;save,prefix="best_fit.",num_slaves=2,nice=19);
```

```
...
```

```
isis> save_par("best_fit.par");  
isis> load_par("best_fit.save");
```

```
isis> () = eval_counts;  
Parameters[Variable] = 12[10]  
Data bins = 118  
Chi-square = 111.0569  
Reduced chi-square = 1.028305
```

```
isis> list_free;
```

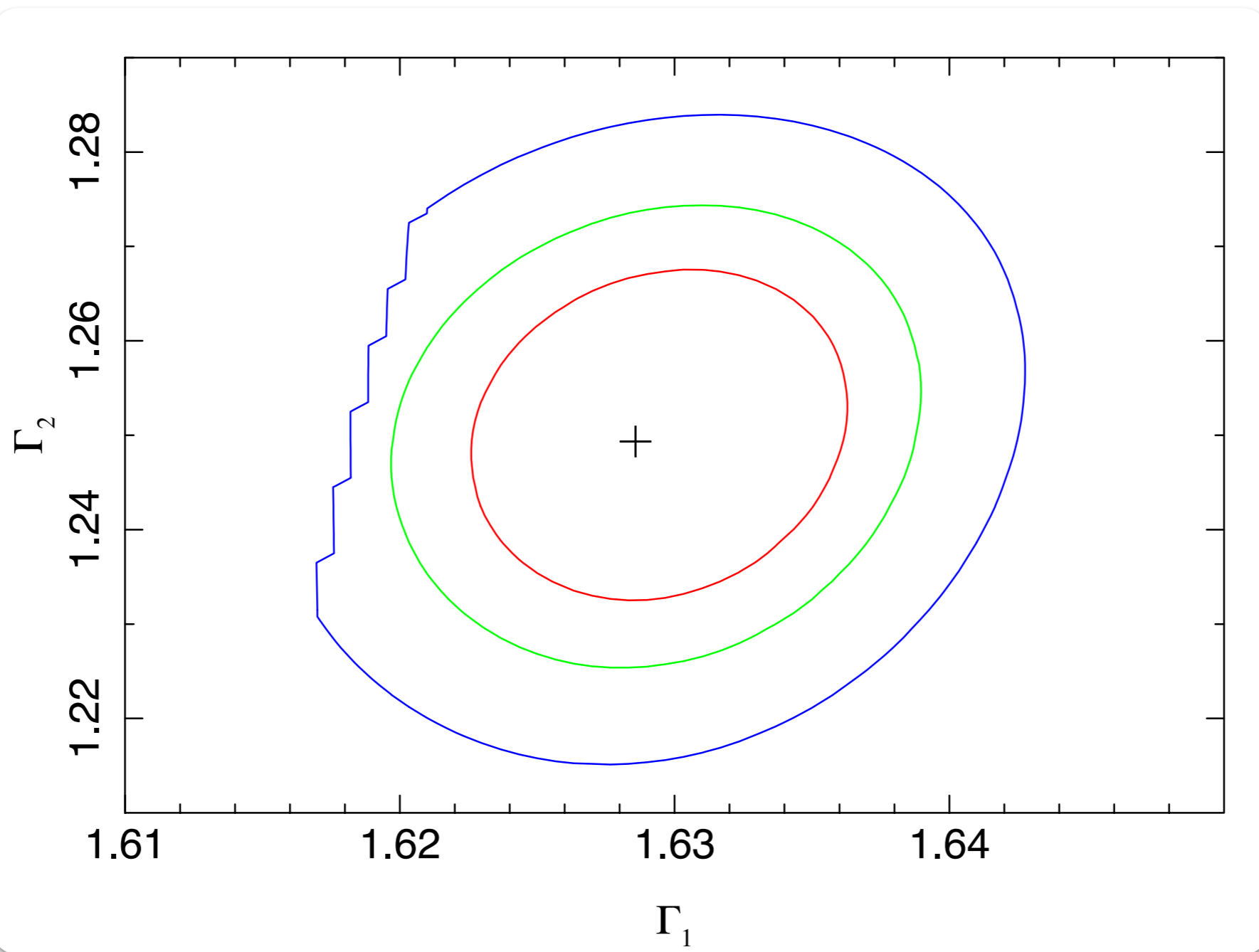
```
constant(isis_Active_Dataset)*phabs*highcut*(bknpower+gaussian)
```

90% Error Bars!

idx	param	tie-to	freeze	value	min	max	
2	phabs(1).nH	0	0	0	0	0.4477608	10 ²²
3	highcut(1).cutoffE	0	0	12.86972	11.55878	13.99137	keV
5	bknpower(1).norm	0	0	0.2360461	0.2328648	0.240283	
6	bknpower(1).PhoIndx1	0	0	1.628581	1.622399	1.637057	
7	bknpower(1).BreakE	0	0	12.81679	12.11459	13.35711	keV
8	bknpower(1).PhoIndx2	0	0	1.249342	1.231995	1.269046	
9	gaussian(1).norm	0	0	0.002511571	0.002188707	0.002860817	
10	gaussian(1).LineE	0	0	6.19454	6.083825	6.302416	keV
11	gaussian(1).Sigma	0	0	0.8079406	0.6850875	0.9338861	keV
12	constant(2).factor	0	0	0.9480235	0.9355516	0.9613026	

Contours are Parallelized

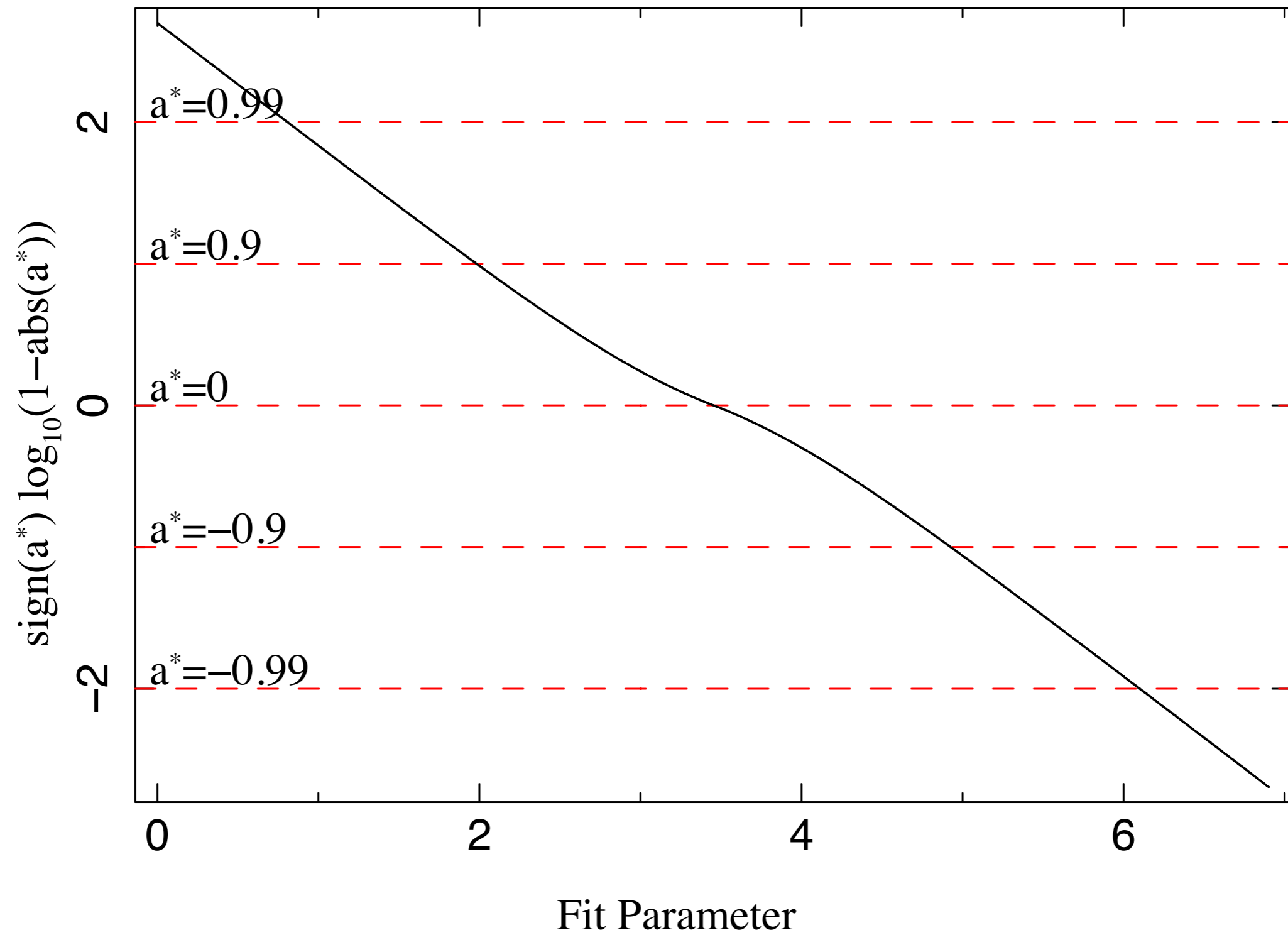
```
isis> variable px = conf_grid(6,1.61,1.65,61);  
isis> variable py = conf_grid(8,1.21,1.29,81);  
isis> variable cntr = conf_map(px,py;flood,num_slaves=2);  
isis> xlabel("\fr\gG\d1"); ylabel("\fr\gG\d2");  
isis> plot_conf(cntr);
```



- It's just math! If it makes math sense, it “works”
 - $\text{constant} * (\text{stuff}) - \text{or} - (\text{stuff} + \text{constant}) - \text{or} - \text{stuff} + 0. * \text{constant}$
 - ```
isis> model(“(constant-edge)*powerlaw”);
xspec> model (constant-edge)*powerlaw
```
  - $\text{stuff} + 0. * (\text{constant}(1) + \text{constant}(2) + \dots)$   
the constants become “dummy parameters”
  - “Works like math” carries over to parameters using `set_par_fun`

- It's just math! If it makes math sense, it “works”
  - $\text{constant} * (\text{stuff})$  – or –  $(\text{stuff} + \text{constant})$  – or –  $\text{stuff} + 0.*\text{constant}$
  - `isis> model(“(constant-edge)*powerlaw”);`  
~~`xspec> model (constant edge)*powerlaw`~~
  - $\text{stuff} + 0.*(\text{constant}(1)+\text{constant}(2)+\dots)$   
the constants become “dummy parameters”
  - “Works like math” carries over to parameters using `set_par_fun`

```
set_par_fun("relconv(1).a" , "tanh(constant(1).factor - atanh(0.998))")
```



# Hi-Res Spectroscopy Example

```
isis> alias_fun("gaussian", "ne9r");
isis> alias_fun("gaussian", "ne9i");
isis> alias_fun("gaussian", "ne9f");
isis> alias_fun("constant", "R");
isis> alias_fun("constant", "G");
```

```
isis> fit_fun("ne9r + ne9i + ne9f + 0.*(R+G)");
```

```
isis> set_par_fun("ne9i(1).norm",
 "G(1).factor*ne9r(1).norm/(1+R(1).factor)");
```

```
isis> set_par_fun("ne9f(1).norm",
 "G(1).factor*R(1).factor*ne9r(1).norm/(1+R(1).factor)");
```

```
isis> list_free;
```

```
ne9r + ne9i + ne9f + 0.*(R+G)
```

| idx | param         | tie-to | freeze | value | min | max      |     |
|-----|---------------|--------|--------|-------|-----|----------|-----|
| 1   | ne9r(1).norm  | 0      | 0      | 1     | 0   | 1e+10    |     |
| 2   | ne9r(1).LineE | 0      | 0      | 6.5   | 0   | 10000000 | keV |
| 3   | ne9r(1).Sigma | 0      | 0      | 0.1   | 0   | 10       | keV |
| 5   | ne9i(1).LineE | 0      | 0      | 6.5   | 0   | 10000000 | keV |
| 6   | ne9i(1).Sigma | 0      | 0      | 0.1   | 0   | 10       | keV |
| 8   | ne9f(1).LineE | 0      | 0      | 6.5   | 0   | 10000000 | keV |
| 9   | ne9f(1).Sigma | 0      | 0      | 0.1   | 0   | 10       | keV |
| 10  | R(1).factor   | 0      | 0      | 1     | 0   | 1e+10    |     |
| 11  | G(1).factor   | 0      | 0      | 1     | 0   | 1e+10    |     |



# (Partial List of) Things You Can Control

- Fit methods: “Levenberg-Marquadt” (mpfit) is default, subplex (slow but robust), diffevol (very slow!)
- Fit statistics: with  $\chi^2$  Data or  $\chi^2$  Model Variance, Cash statistics, W statistics (script), or **define your own**
  - Cash/W statistics must use certain fit methods (not Levenberg-Marquadt! Use subplex!)
- Custom define propagation of errors
- Group data, combine data, exclude data on the fly
- Put values of data & models into vectors; overwrite values of data and data errors
- Get values of responses, effective areas, overwrite effective area, create “software responses”

# A Note on Backgrounds

- ISIS doesn't "subtract" backgrounds –
  - Total Counts = (Model Prediction) + Background
- We often don't plot total counts, and background can be measured (file) or modeled (back\_fun)
- Data statistics are based on the background – *this is a tricky topic*
  - file: increase error for  $\chi^2$  data, but not for  $\chi^2$  model
  - back\_fun: don't increase the error
  - These rules can be over-ridden

# User Defined Functions

```
define zfc_fit(lo,hi,par){
 variable a,f,al,ah,l;
 al = _A(lo);
 ah = _A(hi);
 f = par[1];
 a = par[0]^2 * PI/2./f;
 l = a*f/(al-ah) * (atan(al/f) - atan(ah/f));
 l = reverse(l);
 return l; }
```

*ISIS Fit Functions Take/Return Wavelength Order,  
and like XSPEC are Integrated Counts/Bin*

$$\_A(\lambda_{lo}, \lambda_{hi}) \Rightarrow keV_{lo}, keV_{hi} , \_A(keV_{lo}, keV_{hi}) \Rightarrow \lambda_{lo}, \lambda_{hi}$$

$$\_A(x) = \_A(1)/reverse(x)$$

$$\_A(x,y) = \_A(1)/reverse(y) , \_A(1)/reverse(x)$$

$$\_A(x) = \_A(1)/reverse(x)$$

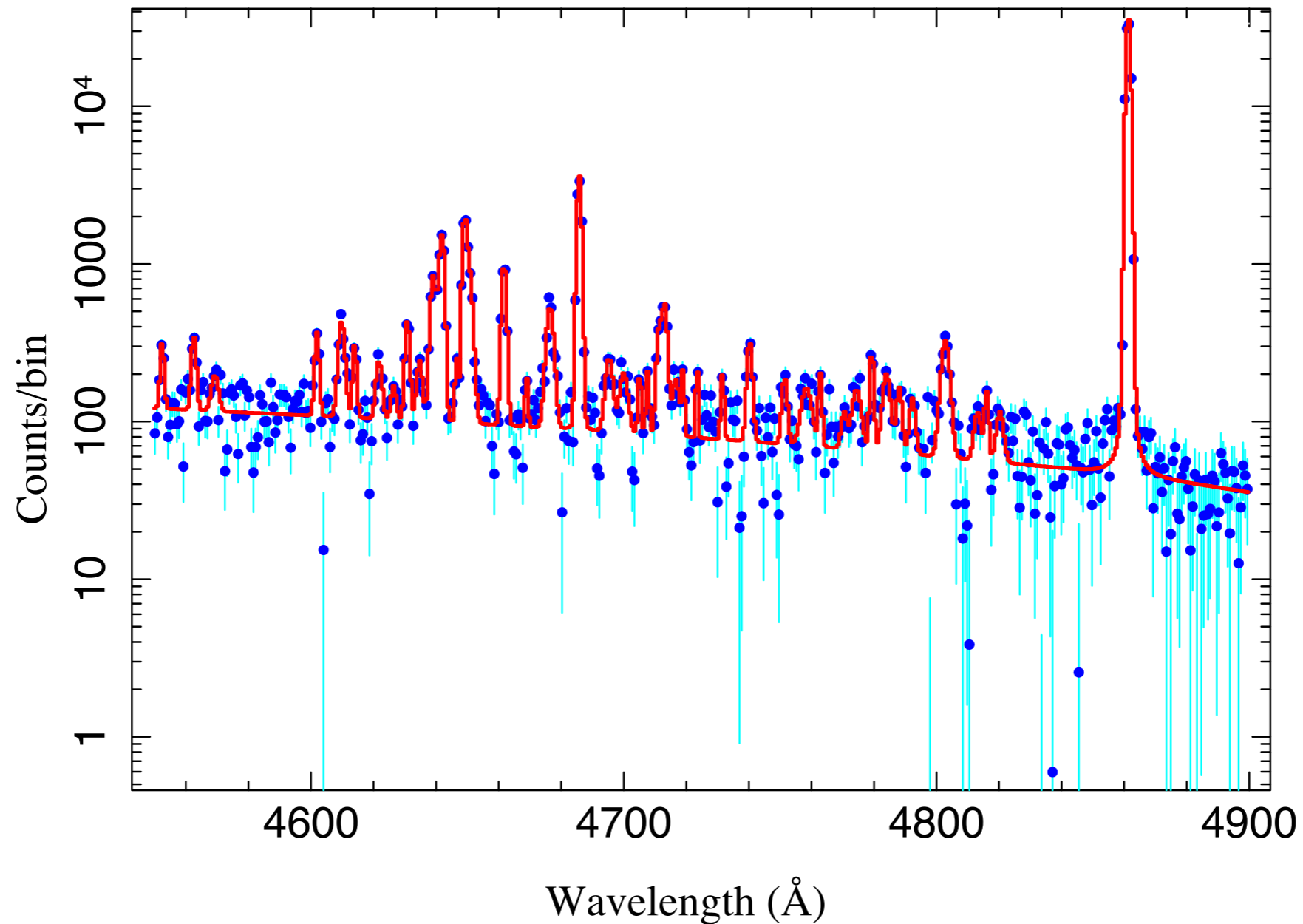
# Going Crazy with User Models

- Rewrite line models: `find_line`, `add_line`, `delete_line`
- `model("powerlaw+lines()");`
  - `lines()/add_line/delete_line` such that `add_line("Ne9r")` will:
    - add a gaussian line component @13.447 A
    - rename it to "ne9r"
    - place it in existing line list in wavelength order
    - `delete_line` will remove the component

# Resources

- S-lang Scripting Language – <http://www.jedsoft.org/slang/>
- ISIS Home Page – <http://space.mit.edu/CXC/ISIS>
- Worked X-ray Fitting Example (& my [.isisrc](#)) – [http://space.mit.edu/home/mnowak/isis\\_vs\\_xspec/](http://space.mit.edu/home/mnowak/isis_vs_xspec/)
- S-lang Timing Analysis Routines – <http://space.mit.edu/cxc/analysis/SITAR/>
- Remeis ISIS Scripts – <http://www.sternwarte.uni-erlangen.de/isis/>
- Mail group for asking questions – [isis-users@space.mit.edu](mailto:isis-users@space.mit.edu)

# Example: ISIS Fit of Optical Data



- Used diagonal RMF/Uniform ARF, bracketed center with lo/hi
- “Reverse engineered” Poisson statistics & background
- Fit 45 Voigt profiles interactively, redefining “lines( )” each time