An Introduction to ISIS

Michael A. Nowak (MIT-Kavli Institute)
- with useful advice over the years from-
John Houck, John Davis, Dave Huenemoerder, Jörn Wilms

An Introduction to ISIS

(The Greatest Analysis Software Ever Written)

Michael A. Nowak (MIT-Kavli Institute)
- with useful advice over the years from-
John Houck, John Davis, Dave Huenemoerder, Jörn Wilms

Data (Spectral) Analysis

- Four major choices: XSPEC, SPEX, Sherpa, ISIS
  - XSPEC is the oldest & most established, so many models will be written with XSPEC in mind
  - SPEX is in some ways a “specialty” package for high-resolution X-ray spectroscopy
  - Sherpa is ≈ the youngest & 2nd most programmable (Sherpa & XSPEC both use Python, but Sherpa was built “ground up” in Python; XSPEC added it later)
  - ISIS is what I use for almost all mathematical analysis (spectra, timing, & other things). It uses the XSPEC models, but can do many other things

SCRIPTS

XSELECT

DS9

CIAO/
DMTOOLS

prism

CFITSIO/
FTOOLS

fv

SAS

XMM

Chandra

Suzaku

Swift

NuSTAR

...

IDL

Custom
Analysis

XRONOS

Timing

SPEX

ISIS

Sherpa

XSPEC

Spectra

"High Level"
Data Analysis

Spacecraft
Specific
Software

Data Storage,
Manipulation,
Visualization

SCRIPTS

XSELECT

DS9

CIAO/
DMTOOLS

prism

CFITSIO/
FTOOLS

fv

SAS

XMM

Chandra

Suzaku

Swift

NuSTAR
...

IDL

Custom
Analysis

XRONOS

Timing

SPEX

ISIS

Sherpa

XSPEC

Spectra

Data Storage,
Manipulation,
Visualization

Spacecraft
Specific
Software

"High Level"
Data Analysis

SCRIPTS

XSELECT

DS9

CIAO/
DMTOOLS

prism

CFITSIO/
FTOOLS

fv

SAS

XMM

Chandra

Suzaku

Swift

NuSTAR

...

IDL

Custom
Analysis

XRONOS

Timing

XSPEC

Sherpa

ISIS

SPEX

Spectra

Data Storage,
Manipulation,
Visualization

Spacecraft
Specific
Software

"High Level"
Data Analysis

SCRIPTS

XSELECT

DS9

CIAO/
DMTOOLS

prism

CFITSIO/
FTOOLS

fv

SAS

XMM

Chandra

Suzaku

Swift

NuSTAR

...

IDL

Custom
Analysis

XRONOS

Timing

XSPEC

Sherpa

ISIS

SPEX

Spectra

Data Storage,
Manipulation,
Visualization

Spacecraft
Specific
Software

"High Level"
Data Analysis

Data Storage, Manipulation, Visualization

CFITSIO/ FTOOLS

CIAO/ DMTOOLS

DS9

fv

prism

XSELECT

SCRIPTS

Spacecraft Specific Software

Suzaku

Chandra

SAS

Swift

NuSTAR

...

XMM

"High Level" Data Analysis

XSPEC

Sherpa

ISIS

SPEX

XRONOS

IDL

Spectra

Timing

Custom Analysis

Custom Analysis

Data Storage, Manipulation, Visualization

CFITSIO/ FTOOLS

CIAO/ DMTOOLS

DS9

Python

Python

S-lang

"High Level" Data Analysis

XSPEC

Sherpa

ISIS

Modules

Custom Analysis

ISIS Philosophy

- All Commands are S-lang Functions or Subroutines
- Models and Data are Just Vectors of Numbers
  - When in Doubt, It's Just Math...
  - No additive/multiplicative (but convolution different)!
- You Should Have Access to and Ability to Manipulate these Numbers
  - Vector Math is Intrinsic to the S-lang Language
- You Should Be Able to Customize Whatever You Want
- 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...

```
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 Orange for Functions I Define in My .isisrc)

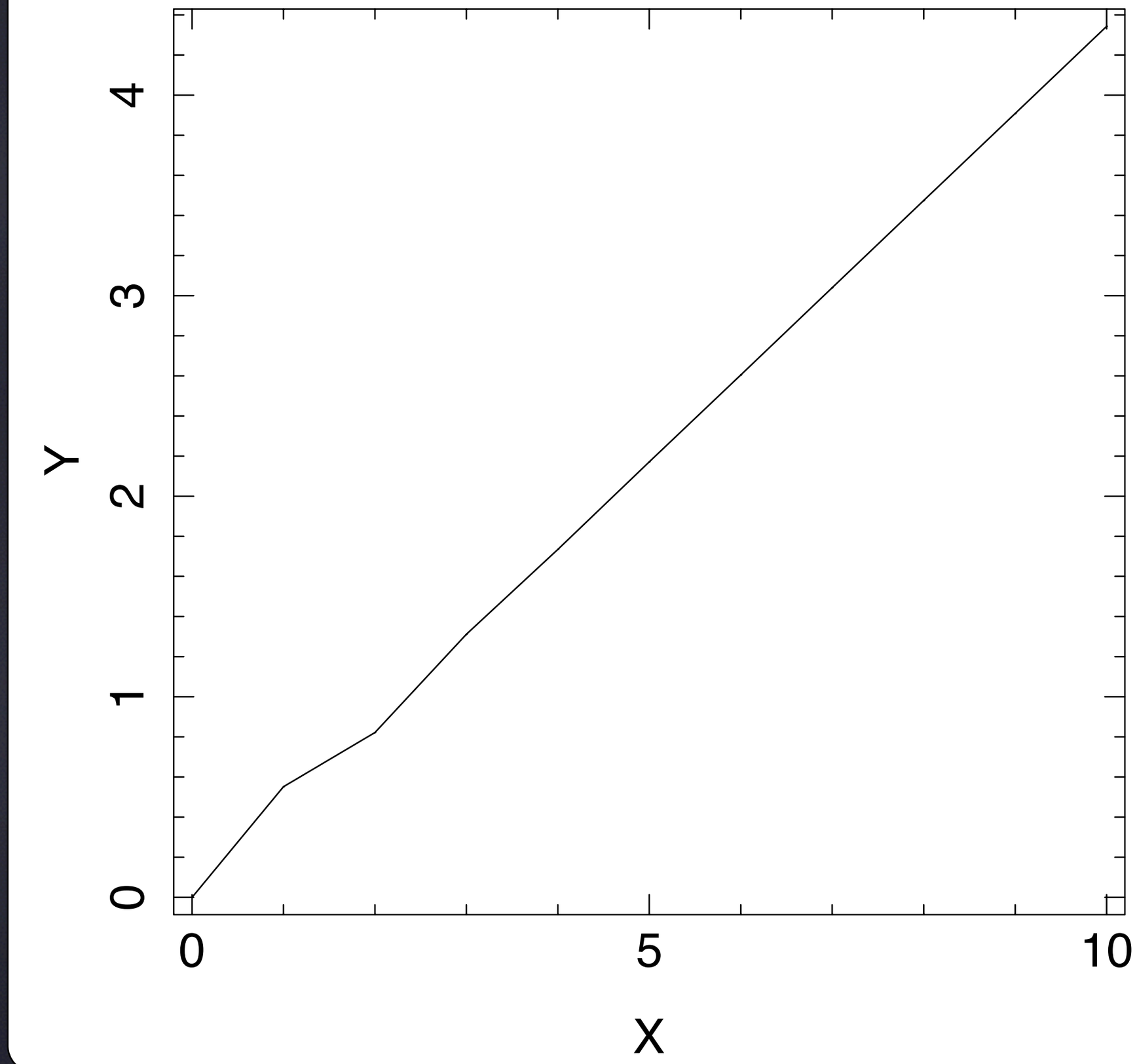
```
isis> print
                (
  "hello world"
                )
.
;
"hello world"
```

```
% isn't read by the program

;       % Commands end with ;


        % Vector math is intrinsic
        % No declaration on command line
(a));   % Common math functions exist


                % Results can be plotted

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

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


Y
0
1
2
3
4
0
5
10
X

```
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
```

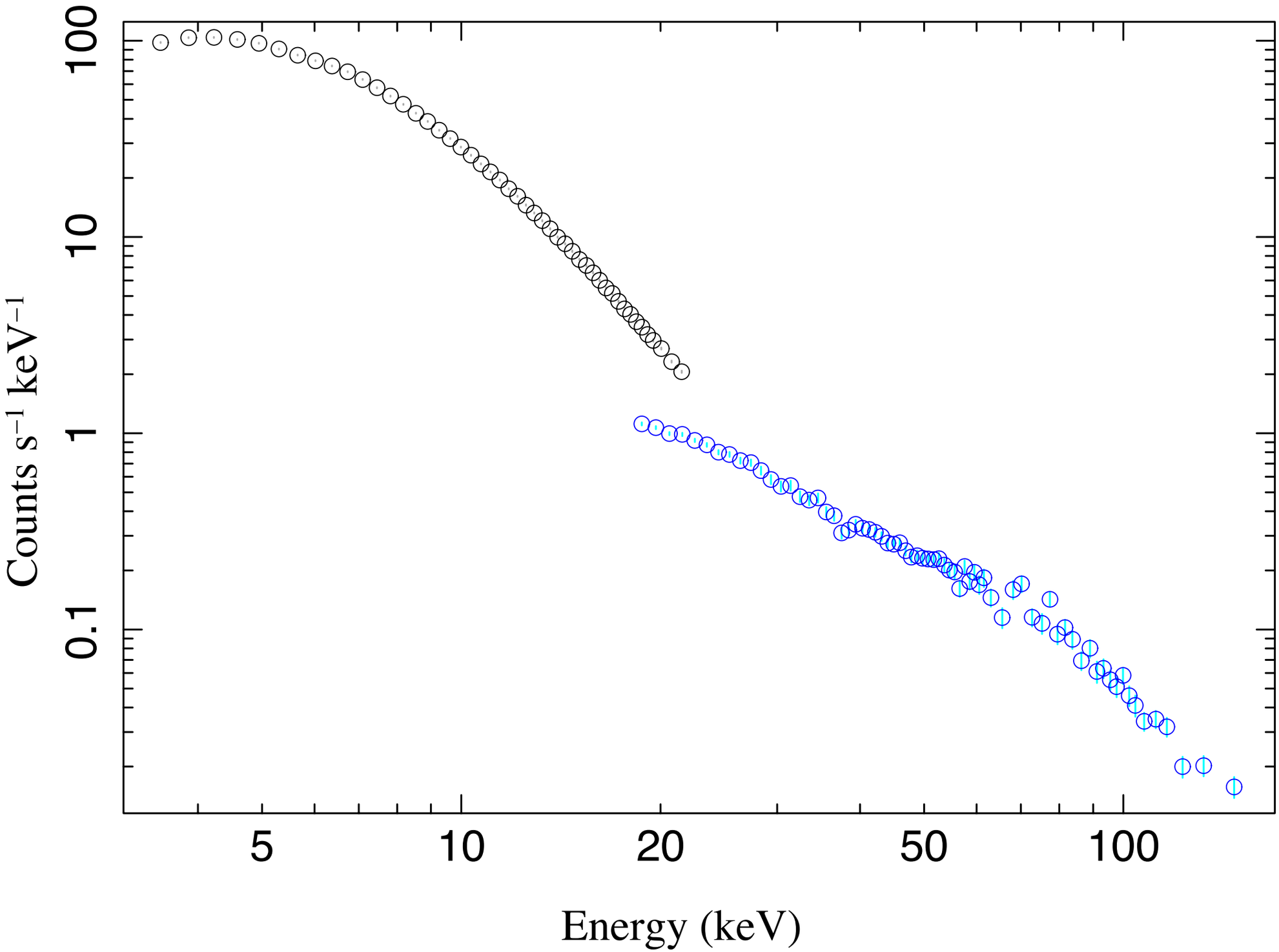
If ARF, RMF, and Background information are in the FITS headers, then these components will also be automatically loaded & associated with spectra.

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

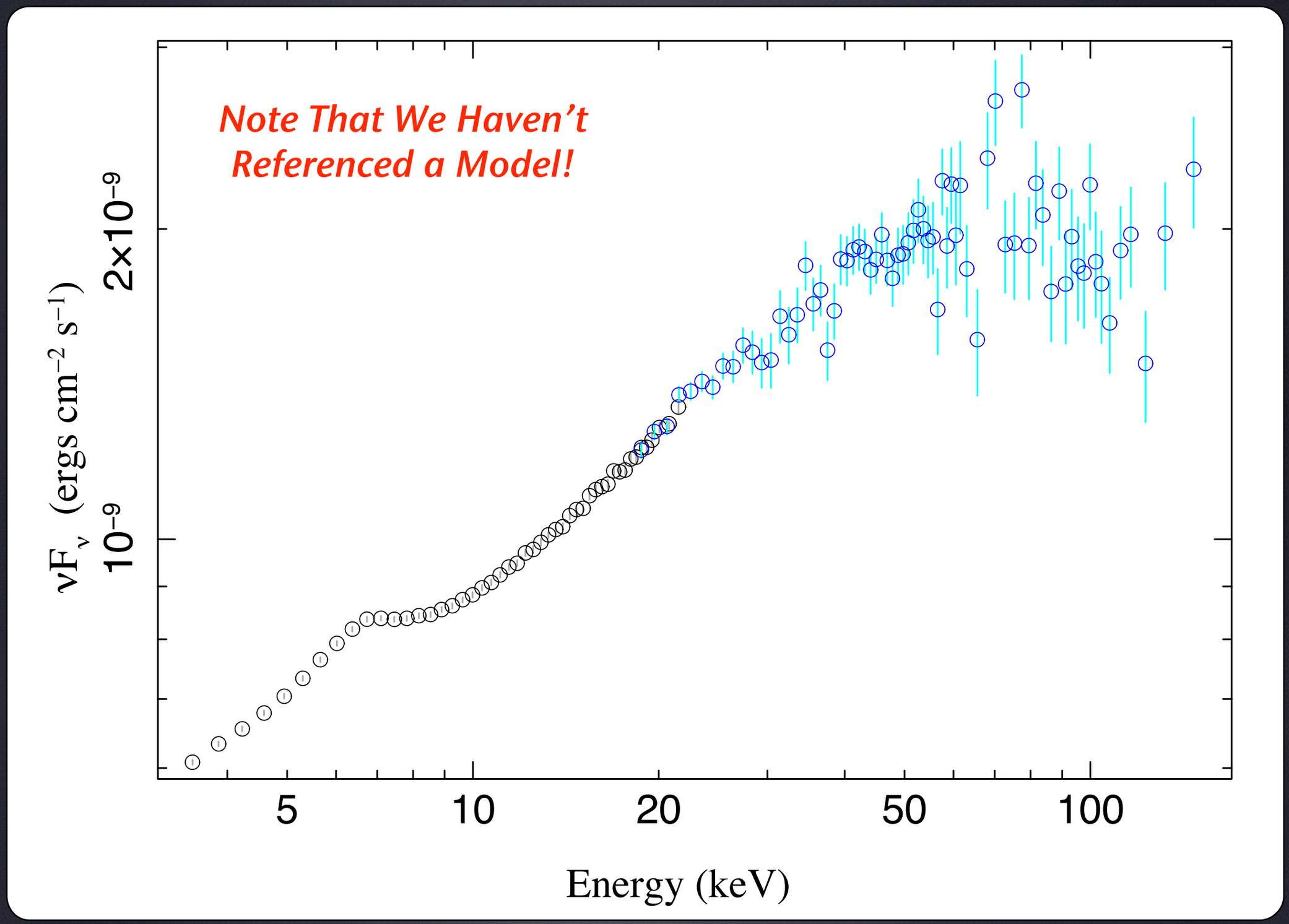
File *grouping will not be automatically applied* unless Isis_Use_PHA_Grouping > 0.

ISIS Let's You Group Data on the Fly!

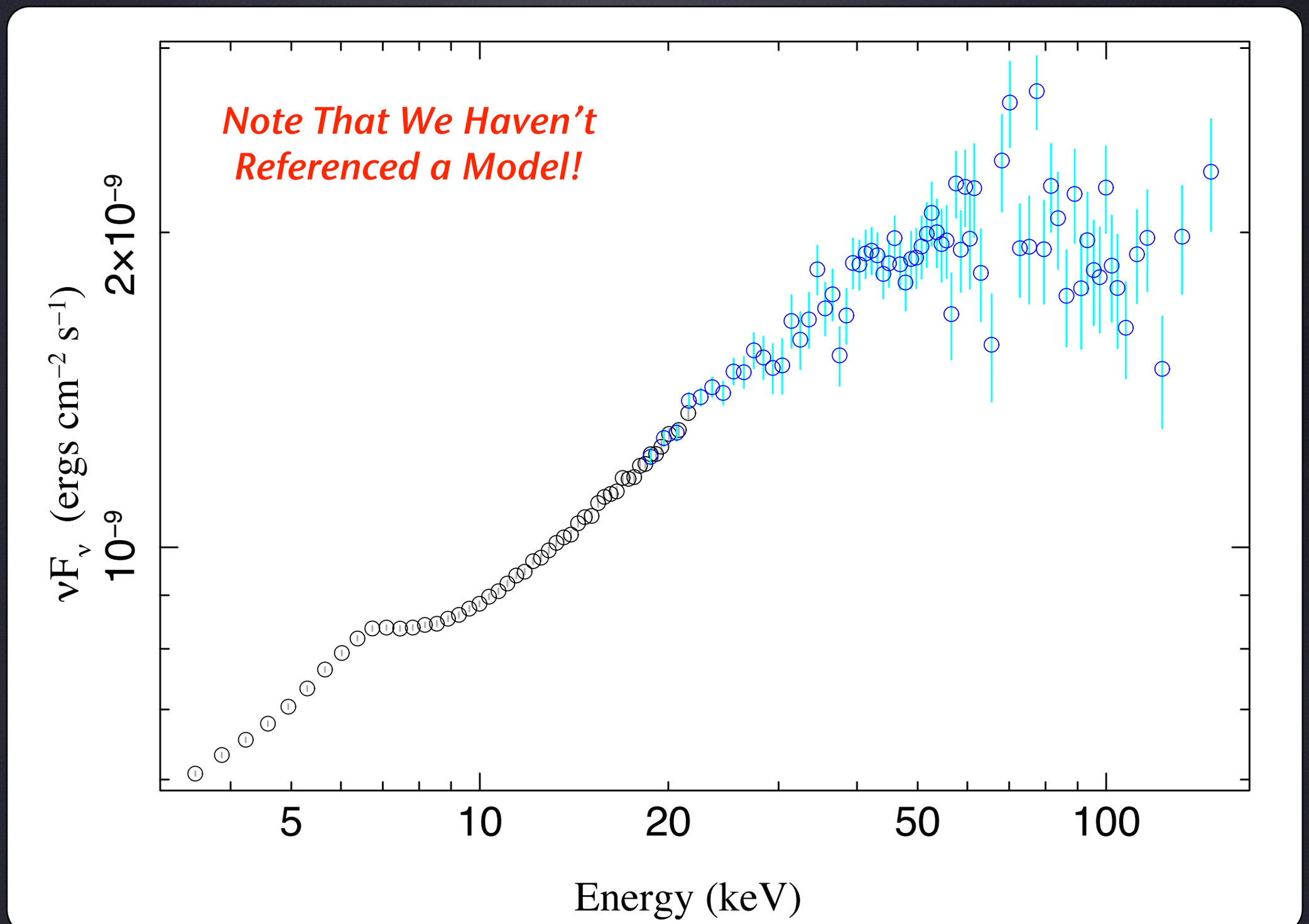
```
isis> group(pca_id; min_sn=4.5, bounds=3, unit="kev");
isis> notice_values(pca_id,3,22; unit="kev");


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});
```


100
10
1
0.1
Counts s^{-1} keV^{-1}
5
10
20
50
100
Energy (keV)

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

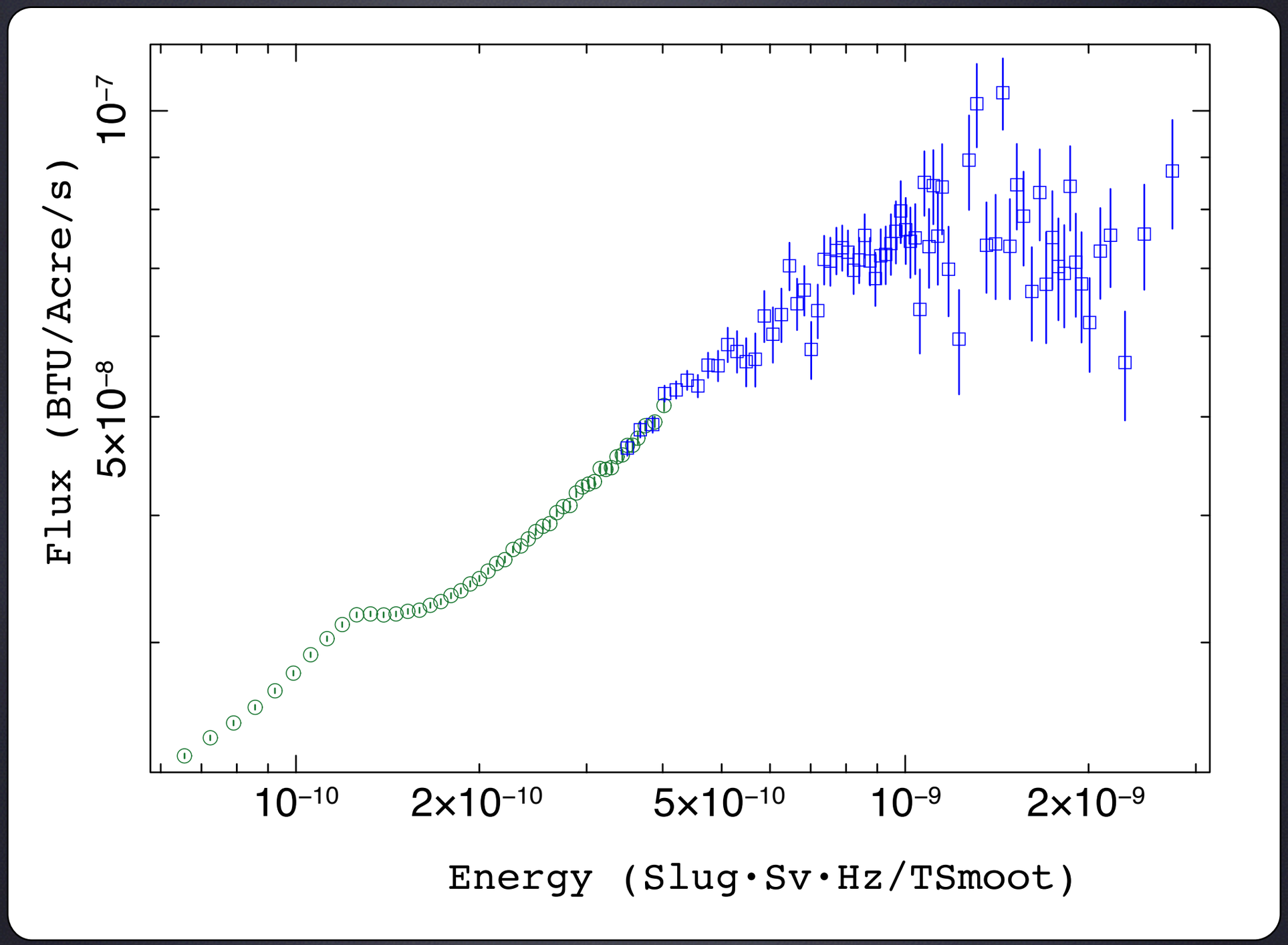


```
isis> plot_unfold({1,2};dcol={1,4},decol={15,5},scale={1,1.07});
```

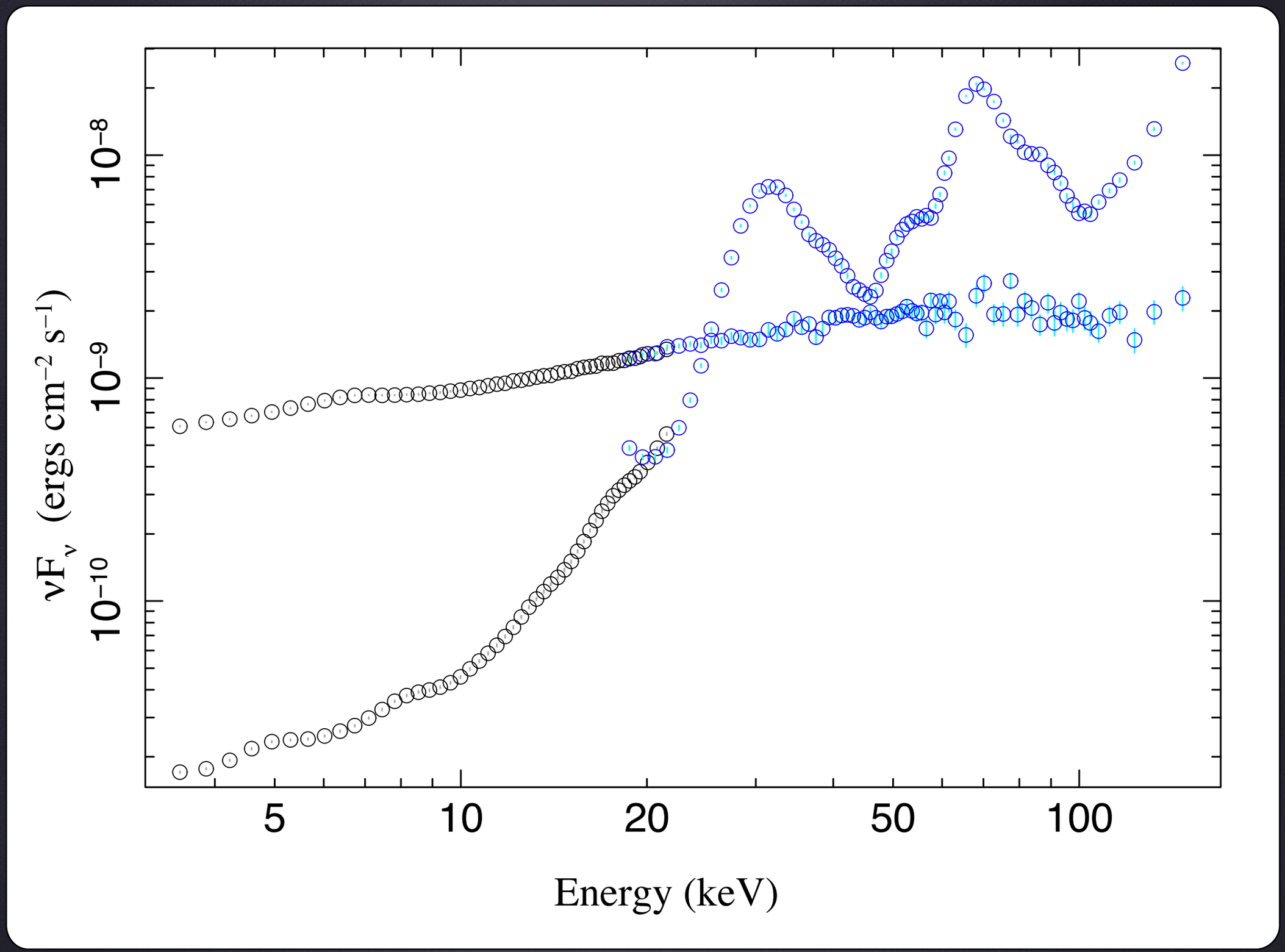


Use with Caution! And Never, Ever, Under Any Circumstances Use the XSPEC: plot eeufs!

```
isis> add_plot_unit("new_energy","new_flux";is_energy=1,
                                    xscale=5.353038e10,yscale=3274.3436);
isis> fancy_plot_unit("new_energy","new_flux");
isis> plot_unfold({1,2};power=3,dsym={4,6},dcol={17,4},
                        xlabel="\\fs Energy (Slug\\.Sv\\.Hz/TSmoot)",
                        ylabel="\\fs Flux (BTU/Acre/s)",scale={1,1.07});
```



```
isis> fancy_plot_unit("kev","ergs");
isis> plot_unfold({1,1,2,2};dcol={1,1,4,4},decol={15,15,5,5},
                            scale={1,1,1.07,1.07},bkg={0,-1,0,-1});
```



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

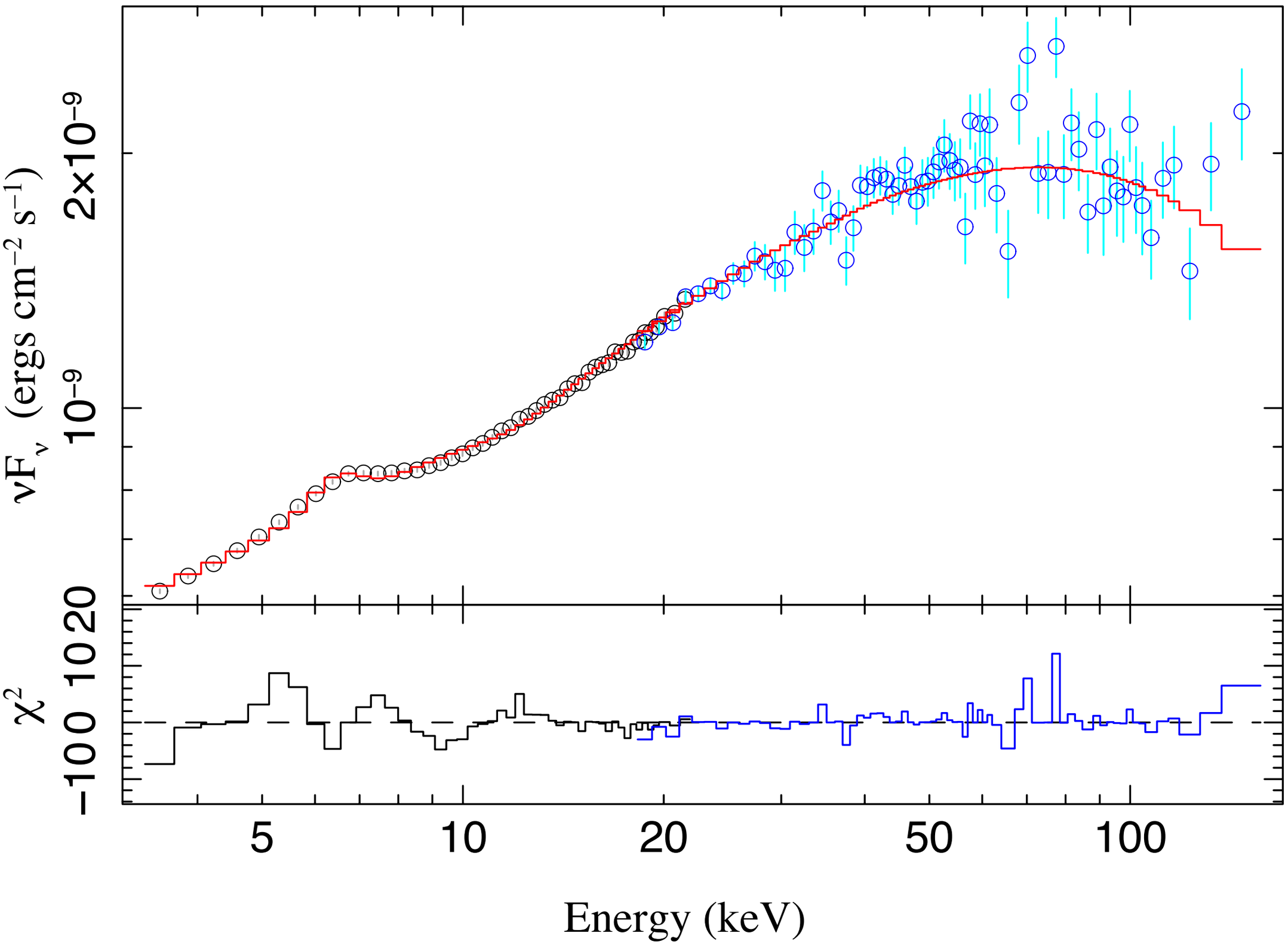
Model is: A Power Law with a "Break" Plus a Line,
that "exponentially rolls over" at high energies,
that is absorbed at low energies,
and
has a different normalization for PCA and HEXTE

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*highecut*(bknpower+gaussian)
 idx  param              tie-to  freeze          value           min           max
  2  phabs(1).nH             0       0            0.5             0             5  10^22
  3  highecut(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 Index
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);
```


νF_ν (ergs cm^{-2} s^{-1})
10^{-9}
2×10^{-9}
χ^2
−10 0 10 20
5
10
20
50
100
Energy (keV)

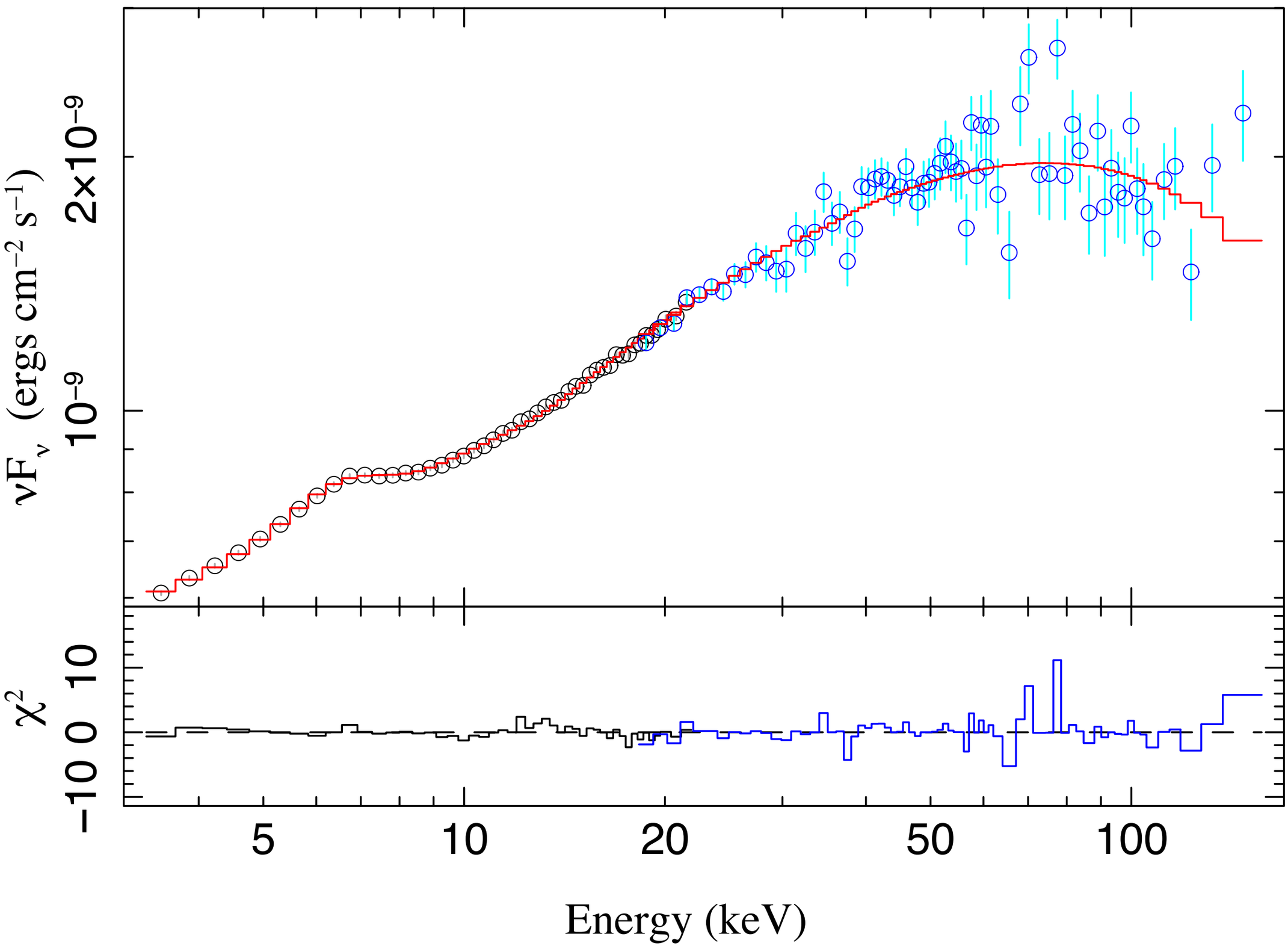
Confidence Contours are Parallelized!

```
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*highecut*(bknpower+gaussian)
 idx  param                tie-to  freeze          value          min          max
  2  phabs(1).nH               0       0              0            0    0.4477608  10^22
  3  highecut(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

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

Confidence Contours 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*highecut*(bknpower+gaussian)
 idx  param                tie-to  freeze          value          min          max
  2   phabs(1).nH               0       0              0            0    0.4477608  10^22
  3   highecut(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

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

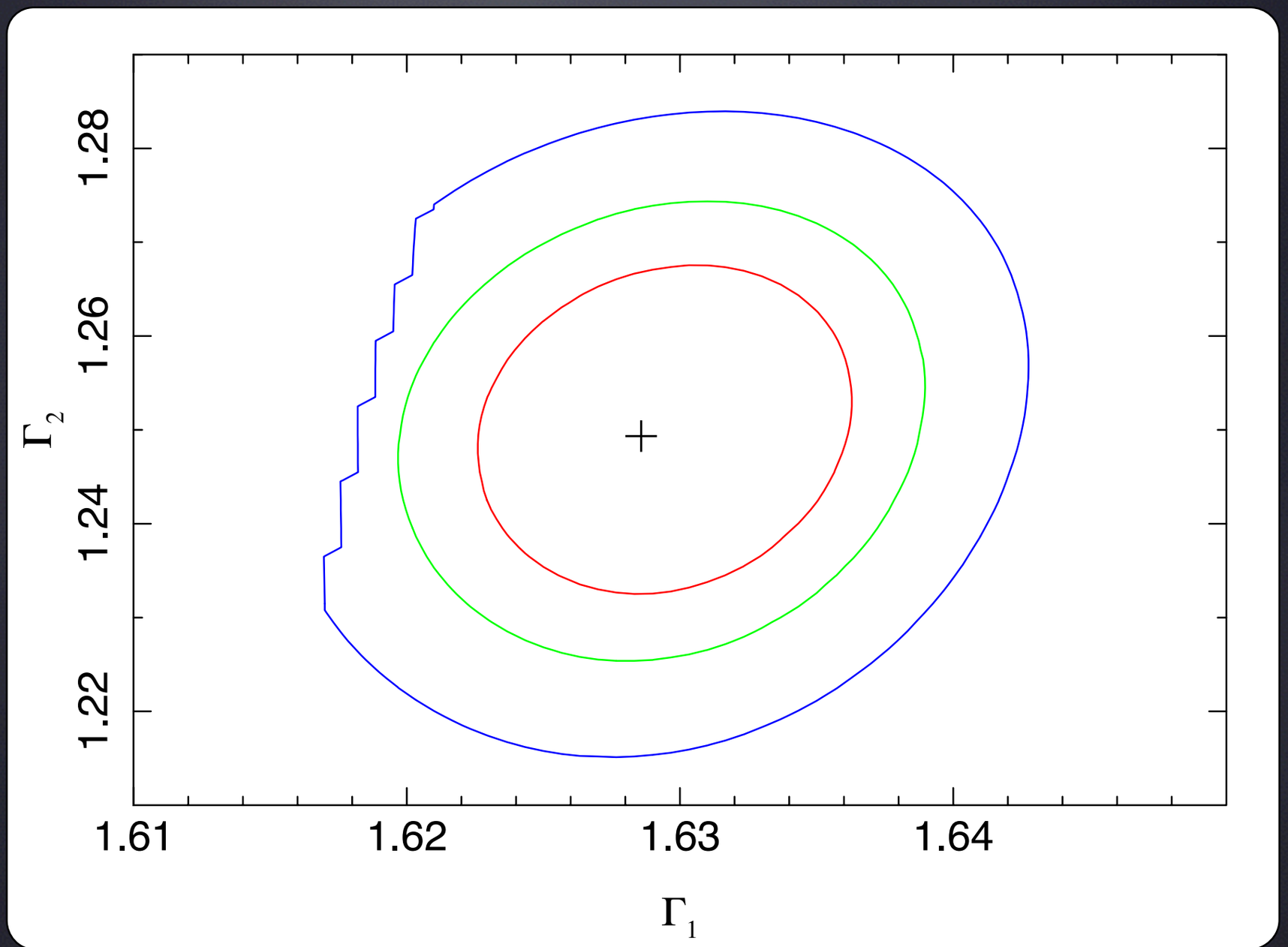
90% Error Bars!


νF_ν (ergs cm^−2 s^−1)
10^−9
2×10^−9
χ^2
−10 0 10
5
10
20
50
100
Energy (keV)

```
isis> variable px = conf_grid(6,1.61,1.65,61);
isis> variable py = conf_grid(8,1.21,1.29,81);
```

Contours are Parallelized

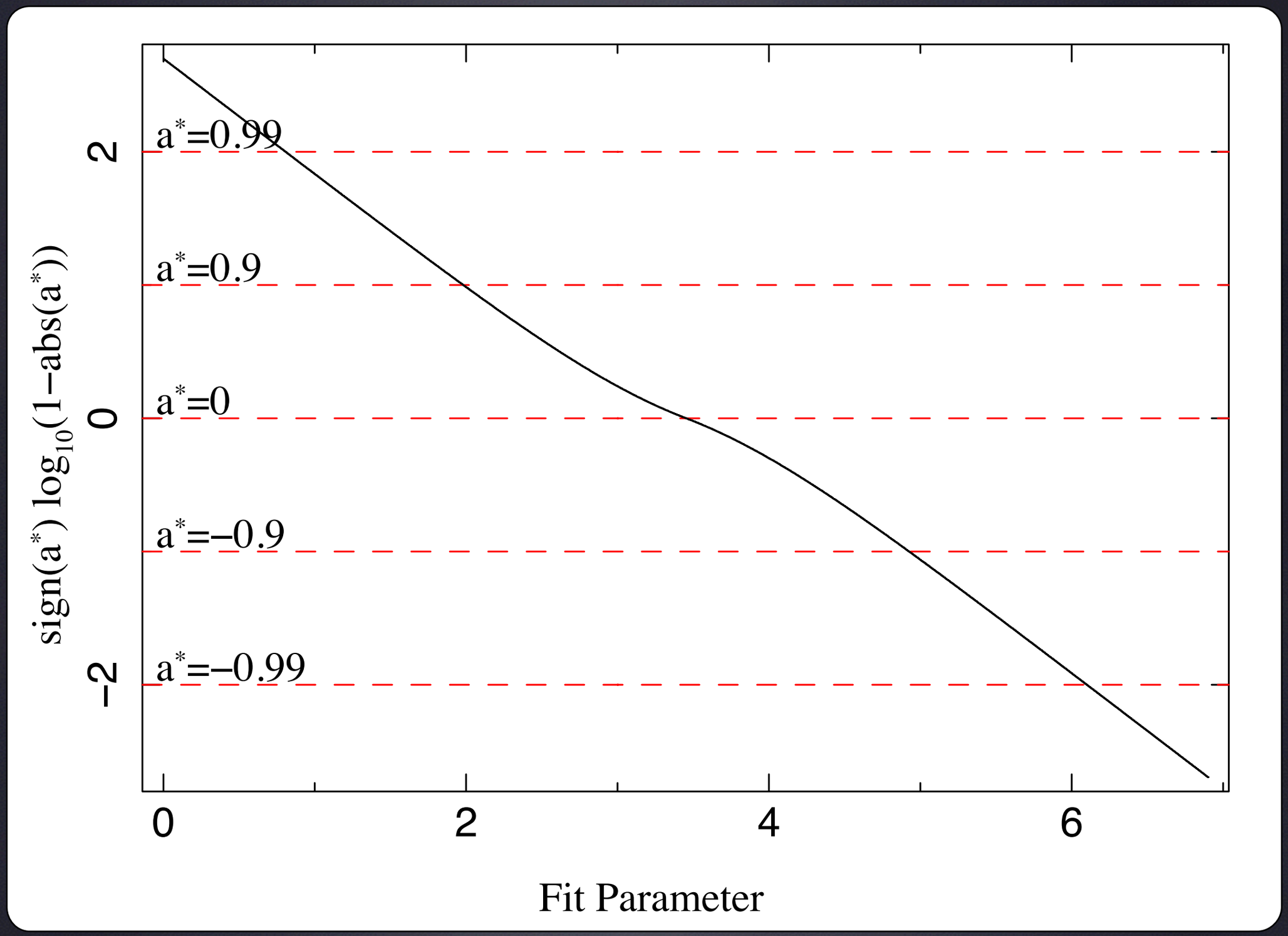
```
isis> variable cntr = conf_map(px,py;flood,num_slaves=2);
isis> xlabel("\\fr\\gG\\d1"); ylabel("\\fr\\gG\\d2");
isis> plot_conf(cntr);
```



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

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

set_par_fun(“relconv(1).a” , ”tanh(constant(1).factor - atanh(0.998))”)



Things You Can Control

- Fit methods: “Levenberg-Marquadt” (mpfit) is default, subplex (slow but robust), diffevol (very slow!)
- Fit statistics: χ^2 with Data or 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
- Put values of data & models into vectors; overwrite values of data and data errors
- Get values of responses, effective areas, overwrite

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 χ^2 data, but not for χ^2 model
  - back_fun: don't increase the error
  - But both these rules sometimes *should* be broken

ISIS Let's You Alter Backgrounds

- A complicated background function I wrote:
  - corfile(indx1, indx2, scale);
  - $Source_1 = Total_1 - Back_1 - scale*(Total_2 - Back_2)$ where errors are a *user-defined* sum in quadrature of *Poisson & Fractional (systematic)* pieces
    - Something I had to write after talking to Phil Uttley, who does RXTE-PCA spectra & timing of AGN
    - Faint Binary + Galactic Ridge + Background & Second Observation Galactic Ridge + Background
- Complicated example of a user-defined function, but ...
- User-defined functions can also be simple

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

$$\mathrm{\_A}(\lambda_{lo},\lambda_{hi}) \Rightarrow \mathrm{keV}_{lo},\mathrm{keV}_{hi}\ ,\ \ \mathrm{\_A}(\mathrm{keV}_{lo},\mathrm{keV}_{hi}) \Rightarrow \lambda_{lo},\lambda_{hi}$$

$$\mathrm{\_A}(x) \ = \ \mathrm{\_A}(1)/\mathrm{reverse}(x)$$

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

Going Nuts with User Models

- Rewrite line models: find_line, add_line, delete_line
- model(“powerlaw+lines()”);
  I defined lines()/add_line/delete_line such that
  add_line(“Ne9r”) will:
  - add a guassian 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/
- 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