X-ray Spectra Part I: Loading, Visualizing, & Grouping Data

Michael Nowak, mnowak@space.mit.edu

July 2, 2010

Starting up ISIS

This exercise presumes that you've downloaded and installed the `.isisrc` files located at:
`http://space.mit.edu/home/mnowak/isis_vs_xspec/download.html`
If these files are placed in your home directory, and the path variable in the main `.isisrc` file is edited to point to your home directory, then these will automatically be loaded when you start ISIS. Further you need to have downloaded the data from the web page:
`http://space.mit.edu/home/mnowak/isis_vs_xspec/data.tar.gz`
These files need to be placed in whatever directory where you will be running ISIS.

1. Start up ISIS, and get used to treating it like a programming environment, and not simply a spectral analysis tool. To begin with, try defining some variables, doing simple functions, and plotting these. For example:

```
variable a = [0:6:0.01];  % 0 -> 6 in 0.01 steps
variable b = a^2;
plot(a,b);
variable c = sin(b+0.5);  % Note that scalars & arrays mix naturally
plot(a,c);
```

There's a few data types that you need to be aware of:

```
variable a = 1;              % Integer
variable b = 1.5;            % Float
variable c = PI;             % There are a few standard numbers
variable d = Double_Type[10];  % Double precision array of zeros
variable e = [0:18:2];       % Integer type array stepping by 2
variable f = d+e;            % f is now a Double_Type stepping by 2
print(f[5]);                 % 10.0
variable g={"pizza",PI,NULL};  % Lists are heterogeneous containers
                             % used a lot in the plotting functions
print( g[1] );               % 3.141592653589793 - arrays index from 0
print( g[2] );               % NULL - used as "default" for plot ranges
```

Note that in the above, the `variable` command is only needed in scripts, but *not* on the command line. From here on out, we drop the `variable`, but in the script version of this exercise, they are included.
All ISIS commands are functions that can be added to scripts. Many functions exist to return information about the data that you've loaded. Furthermore, the data themselves can be loaded to variables.

Loading the Data and Looking at Related Files

2. Load the PCA data, and look at the list of files that just got added. Load the pca.pha file, then use the list functions (`list_data`, `list_arf`, `list_rmf`) to see what you've got. The load data command is:

```
pca = load_data("pca.pha");
print(pca);        % Note that this agrees with the # seen in list data
```

3. Now take a look at the data "by hand" by loading the data in a "structure variable" using the `get_data_counts` command. Print out the pieces of this, and compare this to the returns of the `list_*` functions.

```
variable cts = get_data_counts(pca);  % This is a structure variable

print( cts );                         % A list of its contents
print( sum(cts.value) );              % The total # of counts
print( min(cts.bin_lo) );             % Minimum wavelength in the data
print( max(cts.bin_hi) );             % Maximum wavelength in the data
print( min( _A(cts.bin_hi) ) );       % Minimum energy in the data
print( max( _A(cts.bin_lo) ) );       % Maximum energy in the data
```

4. Take a look at the response that got loaded by defining a delta function model and evaluating it. We'll talk more about fit functions in the next set of lectures. Fit functions are loaded via `fit_fun`, and their parameters can be edited via `set_par` or `edit_par`. Try both. Evaluate the model at 6 keV, and plot it with the `plot_model` command.

```
fit_fun("delta(1)"); % The (#) allows multiple instances to be used
set_par(1,1.);
set_par(2,_A(6));    % delta takes Angstroms: convert from keV with _A()
() = eval_counts;    % returns a success/fail flag caught by the () =
plot_model(pca);
```

Repeat the above for several different energies: 10, 15, 20, 60 keV. Trying changing parameters with `edit_par`.

Plotting the Data

5. This is where we'll start to heavily use the functions from the `.isisrc` files. Choose logarithmic axes, and then plot the counts, with & without the background, and then plot both on the same plot:

```
xlog;
ylog;
plot_counts(pca;mcol=0);          % Don't plot the model defined above!
plot_counts(pca;bkg=1,mcol=0);
plot_counts({pca,pca}; mcol={0,0}, bkg={1,0});
```

There are several ways of setting plot options. The above used the `S-lang` feature of "qualifiers". These are named variables passed after a semi-colon (;). If the qualifier is defined in the function, it will affect the function behavior. Often a qualifier will have a defined default value. (If the name you pass is not defined, it is simply ignored. The order of the passed qualifiers does not matter.) See the options you have by typing `plot_counts` without any arguments, then play around with it.

```
plot_counts({pca,pca}; mcol={0,0},dcol={4,8},decol={5,7},bkg={1,0});
```

To learn what colors and symbol values you have, type the `pg_info` function, which is also defined in the `.isisrc` startup files. (I.e., that's not an intrinsic function in ISIS.)

Note that for the purposes of these plotting functions, most arguments are passed as lists. using the curly brackets, {}, to denote a list. This allow us to mix data types, e.g., `NULL` with integers. `NULL` is used for "auto-scale" the plot axis:

```
plot_counts({pca,pca}; mcol={0,0},dcol={4,8},decol={5,7},bkg={1,0},
                       xrange={NULL,10},yrange={3,NULL});
xrange(NULL,NULL); % When called as a *function*, use () *not* {}
yrange(1,300);     % xrange() choices are respected by .isisrc-defined
                   % plot functions, yrange() only works for ISIS
                   % intrinsic plots: plot, hplot, plot_model, ...
```

Instead of qualifiers, we can set plot options via a structure variable. A structure for doing this, `popt` has already been globally defined by the script. *Any* structure variable with the proper fields can be used, which allows you to define multiple ones for different sets of plots.

```
popt.dcol={4,8};
popt.decol={5,7};
popt.bkg={1,0};
popt.mcol={0,0};
plot_counts({pca,pca},popt);
```

You can also mix the structure variable with qualifiers. Qualifiers will override the values in the structure. (Structures are more convenient when setting lots of options, but sometimes you want to change just a couple of those.)

```
plot_counts({pca,pca},popt;dcol={5,7});
```

Now try the above with the `plot_data` function instead, and note the differences.

```
plot_data({pca,pca},popt);
```

Loading a Second Data Set

Let's load a second data set, which in this case will be the HEXTE data that was taken simultaneously with the PCA data already loaded.

```
hxt = load_data("hxt.pha");
```

6. Repeat the same exercises as above, specifically use the `list_data`, `list_rmf`, and `list_arf` functions to see what you've loaded. How do the numbers compare to the PCA data? Note that the HEXTE data *have a separate arf and rmf file, unlike the PCA data which only has a combined arf/rmf.* Use the `get_data_counts` function to look at the total counts, the minimum and maximum energy bins in the data, etc.

7. The delta function fit function still remains defined from before. Now choose an energy value appropriate to the HEXTE data:

```
set_par(2,_A(30));  % 30 keV
() = eval_counts;
plot_model(2);      % This almost all background, and not the response
```

Since the plot above is mostly comprised of the background data, use the `.isisrc` defined function `plot_fit_model` instead. This latter plot function subtracts the background from the fit model.

```
plot_fit_model(2);
```

8. Since the HEXTE has a separate arf file, let's take a look at that by itself:

```
variable arf = get_arf(1);  % The HEXTE arf is #1, since PCA has no arf
hplot(_A(arf.bin_hi),_A(arf.bin_lo),reverse(arf.value));
```

9. As for the PCA data, plot the counts, with & without the background, and then plot both together on the same plot. Do this for both the counts per bin (`plot_counts`) and the counts per unit energy per second (`plot_data`).

```
plot_counts({hxt,hxt},popt);
plot_data({hxt,hxt},popt);
```

10. Let's put the PCA and HEXTE data together on the same plot. If you are using the `popt` structure variable, remember that although we have it set up for two data sets, one is set to have no background subtraction.

```
popt.bkg={0,0};
plot_data({pca,hxt},popt);
```

'Flux Corrected' Data, Grouping, and Noticing

In the next set of lectures we will discuss the extent to which we can trust our ability to "undo" the effects of the response and plot the data in terms of physical units such as flux. The short answer is: *this can be very misleading, so is dangerous to do.* In XSPEC, this can only be done in reference to a fitted model, which only makes it *more misleading, not less*. In ISIS, we only do it in reference to the detector responses. (This will be explained more in the next lecture.) Still, this can be a helpful thing to do, if approached with caution, so we will slowly move forward and try it.

11. Plot the data in a "flux corrected" manner using the `plot_unfold` function. (Yes, there is some inconsistency between the procedure and the name of the function!)

```
plot_unfold({pca,hxt},popt);
```

To what extent do the data agree or disagree? Where are the two most different from each other? Which do you trust more? Why? Here is where some knowledge of how the detectors work can come in handy.

You can try playing around with the data plots a little more by using various different combinations of units. The `fancy_plot_unit` function allows you to choose a variety of X- and Y-units, e.g.,

```
fancy_plot_unit("hz","ergs");  % Case insensitive
```

Also, the `plot_unfold` function will make use of the `power` qualifier or structure field. Choosing `power=1` means being proportional to photons/area/sec/(unit x), and then power=0 means dividing by another power of x, power=2–4 means multiplying by additional powers of x. This allows for plotting things like F_λ, λF_λ, νF_ν, etc. Play around with that a little bit, and notice the extent to which you see deviations between PCA and HEXTE become more or less prominent.

12. One of the useful features of the 'flux corrected' data is that at least this gives us some more idea where we can trust the spectra. (Although this is not a substitute for reading the observatory guides for the detectors!) Here clearly there are issues at both the low and high energy ends of both detectors. Let's try cleaning this up a bit by restricting the noticed energy ranges. Before we do that, however, we will first group the data to improve the signal-to-noise in each channel. We accomplish these tasks with the `group` and `notice_values` functions. *Note that these functions need to be told what units you are referring to when making selections.*

```
group(pca;min_sn=5,bounds=3.,unit="kev");
group(hxt;min_sn=5,bounds=18.,unit="kev");

notice_values(pca,3,22;unit="kev");
notice_values(hxt,18,200;unit="kev");
```

Now try replotting the data. Play around with different choices on the signal-to-noise criteria, and see how that affects the plots. Also notice that `notice_values` allows you to slice out specific regions. For example:

```
notice_values(pca,3,5,8,22;unit="kev");
```

will ignore the broad iron line region between 5–8 keV in the PCA data.

One final note: Any time you redo the `group` function, you also have to redo the `notice_values` function. The notice range is *not* preserved after the data bins have been redefined.

Data Fluxes - Part I

In X-ray spectroscopy, fluxes are most reliably defined *in reference to a specific model*, and then only over energy ranges actually detected by the satellite in question. This might seem like an obvious statement, but it's not. In the literature, you will often find fluxes quoted well outside of the bandpass of the detector. Furthermore, you will often find fluxes *corrected for the effects of interstellar absorption*. Given that absorption is an *exponential process*, that depends upon the chosen extinction model and atomic cross sections and elemental abundances assumed, this can be a very, very model-dependent quantity. Again, if you are going to attempt such a thing, it has to be in reference to a specific model.

13. Since we have yet to define and fit a model, we can make a rough calculation of the absorbed flux essentially using the same procedures applied internally in the `plot_unfold` function. Another function

defined by the `.isisrc` files is `data_flux`. This can return values to variables, or print them to a screen. Try it both ways:

```
(p,pe,e,ee) = data_flux(pca,3,8;unit="kev");  % PCA Flux
data_flux(hxt,20,100;unit="kev",print);        % HEXTE Flux
```

Also try it over the same energy range for PCA and HEXTE, and notice the differences:

```
data_flux(pca,20,30;unit="kev",print);   % PCA Flux
data_flux(hxt,20,30;unit="kev",print);   % HEXTE Flux
```

Which of these to trust, if either, will be discussed more in the next lecture. We will then also discuss functions that obtain their estimates of the flux from a fitted model.