Guide to Using Multiple Order Responses in Analysis Software

Overview

Unlike previous iterations of the Arcus responses, we are now including order-overlap (in CCD energy space) from one order into another. This has the effect of introducing photons from, say, the -6th order which are instead detected as -5th order photons.

This effect is significant for all orders, leading to contributions to the nth order from n-1th, nth and n+1th order needing to be modeled.

There are different ways to change the extraction regions which are more thoroughly explained on Moritz's site, but what is definitely clear is that using the merged response (often used last time round) and/or ignoring this cross talk between orders is going to lead to significant errors in the analysis.

As a model now requires loading up to 12 orders, sometimes with 3 responses each, and 3 cross-arfs for each of these, actually just getting the model into [your analysis tool] can be quite complex. This document will share how to actually do that on a software program by software program basis.

PyXSPEC (Adam Foster)

I have uploaded a detailed example, example_pyxspec_v1.py. This includes loops over the various responses to provide one-click generation of all the faked responses. Below I break down the actual steps required to make a single spectrum with cross ARFs included.

Note that there is no handling of any background in this yet.

Generating Fake Spectra

To generate the fake spectra for each order, you will have to follow this order of operations:

1. Define any model
2. Fake a spectrum using that model for the in-order RMF and ARF you are interested in
3. Load in that fake spectrum (may be done automatically when generated)
4. Add 2 more responses for that spectrum, using the same RMF but the cross ARF.
5. Create sources for each of these orders, which should (presumably) all be identical.
6. Run Fakeit

To go through these in detail:

1. Define any model

Exactly what it says. This can be, but doesn't have to be, the actual model you want to use. It's just to get the responses set up.

```
m1 = xspec.Model("apec")
```

2. Fake a spectrum

For this example, I will be making a spectrum for the n=-6 order. You have to define a FakeitSettings object, then run fakeit. You can do this without writing the file to disk.

```
fs1 = xspec.FakeitSettings(response=rmfname,
                           arf = arfname,
                           exposure=100)
xspec.AllData.fakeit(1,fs1, noWrite=True, applyStats=True)
```

Here rmfname and arfname are the files for in-order response, e.g.

```
rmfname = <respdir>/far_chan_all_-6.rmf
arfname = <respdir>/osip60/far_chan_all_ccdord_-6_true_-6.arf
```

3. Load in the dummy fake spectrum if required

This happens automatically if done by the above. Otherwise you can load a spectrum you generated another time using:

```
xspec.AllData+=<filename>
```

4. Add cross arf responses

For each add in the different RMFs and the relevant cross ARF

```
s1 = xspec.AllData(1) # this will automatically have
                      # the right ARF and RMF
# add other sources with same RMF and cross ARF
s1.multiresponse[1] = <respdir>/far_chan_all_-5.rmf
s1.multiresponse[1].arf =\
         "<respdir>/osip60/far_chan_all_ccdord_-6_true_-5.arf`"
s1.multiresponse[2] = <respdir>/far_chan_all_-7.rmf
s1.multiresponse[2].arf =\
         "<respdir>/osip60/far_chan_all_ccdord_-6_true_-7.arf"
```

5. Create Sources

For each of the 3 responses, create sources
You may want to clear out your models (perhaps):

```
xspec.AllModels.clear()
```

If you need to, recreate the initial model

```
m1 = xspec.Model('apec', 'source1')
 # add the other models
m2=xspec.Model('apec', 'source2', 2)
m3=xspec.Model('apec', 'source3', 3)

# trick to quickly link all the parameters
for iparam in range(m2.nParameters):
          m2(iparam+1).link='source1:%i'%(iparam+1)
          m3(iparam+1).link='source1:%i'%(iparam+1)
```

6. Run Fakeit

```
fs2 = xspec.FakeitSettings(exposure=5e5,\
                           fileName='order_-6.fak')

xspec.AllData.fakeit(settings=fs2, applyStats=True)
```

Loading & Using Faked Spectra

1. Load each faked spectrum
2. Set the correct responses
3. Make models
4. Fit, plot, or whatever you need to do

1. Load faked spectrum

For each order (load them all)

```
s=xspec.AllData+='order_-6.fak'
```

2. Set the responses

By default only the 1st response is set, we need to set the cross arf and rmf still. We don't know for sure how many there are (typically 3, but not always) so we will get the response files used from the spectrum file FKRSPXXX and FKARFXXX keywords

```
for irsp in range(100):
        # yes, 100 is massive overkill.
    try:
        # check both RSP and ARF keywords exist
        rspkwd = 'FKRSP%03i'%(irsp+1)
        arfkwd = 'FKARF%03i'%(irsp+1)
        rsp = s.fileinfo(rspkwd)
        arf = s.fileinfo(arfkwd)

    except:
        # if there are no more FKRSP keyowrds,
        # break out of loop
        break

    # assign these responses
    s.multiresponse[irsp] = rsp
    s.multiresponse[irsp].arf = arf
```

3. Make models

This is done in exactly the same way as (5) above. Again, as the models should all be identical, I suggest linking them.

4. Fitting and Plotting

I leave this part up to you… this is the science bit.

Handling Arcus Response Order Crosstalk in ISIS

(David Huenemoerder)

The following assumes basic knowledge of spectral analysis in ISIS. If you want a general introduction, this is the wrong tutoria! (in that case, for starters, try <https://space.mit.edu/ASC/isis2015/Chandra_HiRes_2015_Intro_to_ISIS.pdf> or other links at <https://space.mit.edu/ASC/isis2015/tutorials.html>).

To support LETG/HRC-S analysis, for which there is no order-sorting capability, ISIS provides assignment of multiple responses to a single PHA counts spectrum, that is, an ARF-RMF pair for each relevant diffraction order.

The Arcus case is similar, except that we are mostly resolved, so instead of having one global sum over orders, we have about 10 extracted counts spectra, each of which needs two or three ARF-RMF pairs, one pair for the primary order, two or one for cross-talk.

First I will give the shortcut version, which utilizes functions defined in **`arcus_utils.sl`**, which will load all the ARFs and RMFs from a set for orders -2 to -11, including cross-talk, assign them to null datasets, ready for defining a model and faking data. Then, for anyone interested in the details of what happens at a lower level (and how much typing we have saved), I'll give an example of the response loading and assignment for one primary order with its cross-talk responses.

I. Using helper functions in `arcus_utils.sl`

1. Set two environment variable which point to where you unpacked the response distribution, and which version (directory name) it is:

```
export ARCUS_ROOT="$HOME/h3/Analysis/Arcus/Data/responses"
export ARCUS_RSP_VERSION=20210501ordersorting
```

2. Start isis and do any of your local setup (I have "./scripts" first in my path, and I always have a "./scripts/setup.sl" file in my analysis directories, but this is a matter of taste):

```
isis
.load setup
```

3. Load the helper functions; if the file isn't in your default path, prefix the path:

```
.load arcus_utils
```

4. Set the orders, exposure time, and initialize a fake dataset. The default is for the "osip60" set.

```
mords = -[2:11];
texp  = 10000.0 ;

d60   = n_init_fake_arcus_data( mords, texp );
```

That's it. You now have 10 spectral histograms defined, with responses assigned, and you are ready for simulation. The variable, d60, is an array of structures, each element of which has fields which give the associated spectral index and the assigned response indices. This might be convenient later, but is not necessary. You can see the datasets and responses as usual via:

```
list_data;
list_arf;
list_rmf;
```

5. If you want to use the "osiptouch" case, we can continue. We will print the ARF sub-directory, change it, and create additional datasets:

```
get_arf_dir;
set_arf_subdir( "osiptouch" );
get_arf_dir;

dtch = n_init_fake_arcus_data( mords, texp );
```

You now have another 10 datasets defined.

So far, though, there are no faked counts. You can now define a model, fake data, then continue with your analysis. As a simple example, we'll use a power-law:

6. Define a simple model. We'll use the ISIS "Powerlaw" function (slightly different from Xspec's which is available as small-”p” "powerlaw"). We'll set the exponent to -2 so that it is flat in wavelength space. We'll adjust the norm so that we get 1 photon/cm^2/s/A, which means that the model will approximately be the effective area.
   We will also add three Gaussians in adjacent orders so that the crosstalk will be visible.

```
fit_fun( "Powerlaw(1)+ gauss(1) + gauss(2) + gauss(3)" );

set_par( "Powerlaw(1).norm", _A(1) );
```

```
set_par( "Powerlaw(1).alpha", -2 );
set_par( "gauss(*).area", 1.0 );
set_par( "gauss(*).sigma", 0.01 );
set_par( "gauss(*).center", [23.145, 19.454, 16.818 ] );
```

(you could at this point load any parameter file or scripts which define a model).

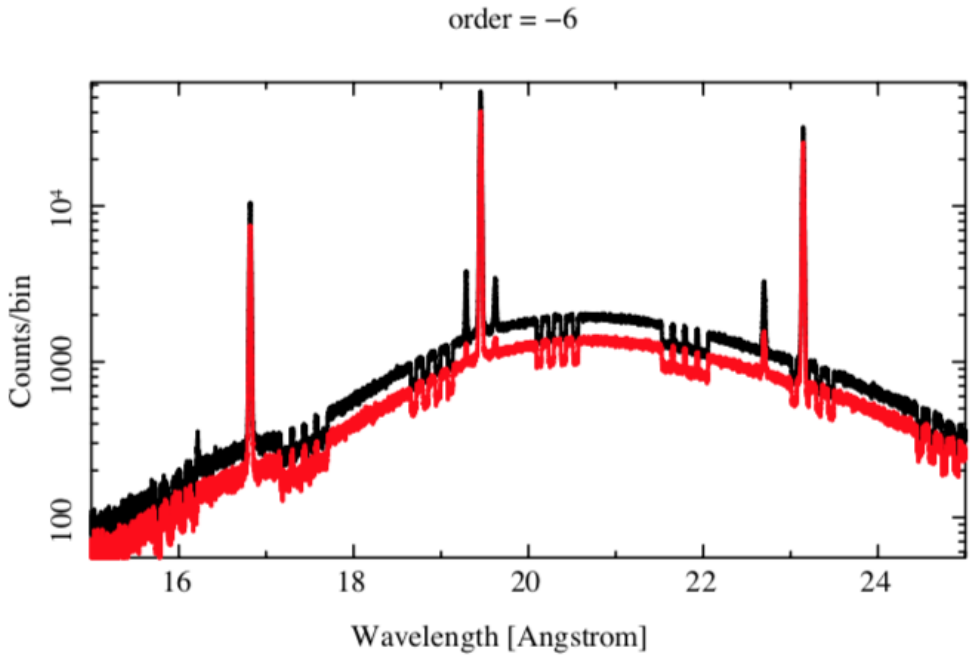
7. Fake the data:

```
fakeit;
```

And you're off. You have 20 faked flat spectra, 10 orders with "osip60" and 10 with "osiptouch", which you can plot, bin, and measure as usual (which is the subject of other tutorials and won't be covered here).

8. Take a look: to compare the "osip60" and "osiptouch" spectra for order -6:

```
title("order = " + string( get_data_info( dtch[4].h).order )
);
xrange( 15, 25 );  ylog;
plot_data_counts( dtch[4].h ); % plot order -7 for “touch”
oplot_data_counts( d60[4].h ); % overplot order -7 for “60”
```



II. Under the Hood

How to assign multiple responses to one count spectrum in ISIS and create fake data. (For simplicity, I'm going to omit paths, and also variable declarations.)

For this example, we will load responses for -6th order and crosstalk, and I will refer to these as "66" for primary order, "65" for primary and crosstalk from -5th, and "67" for primary and crosstalk from -7th.

1. Load the ARFs, and keep the returned index in a variable:

```
a66 = load_arf( "far_chan_all_ccdord_-6_true_-6.arf" );
a67 = load_arf( "far_chan_all_ccdord_-6_true_-7.arf" );
a65 = load_arf( "far_chan_all_ccdord_-6_true_-5.arf" );
```

2. Likewise, load the associated RMFs:

```
r6 = load_rmf( "far_chan_all_-6.rmf" );
r7 = load_rmf( "far_chan_all_-7.rmf" );
r5 = load_rmf( "far_chan_all_-5.rmf" );
```

3. Define an index for the counts spectrum. This should be an unused index (that is, not an existing index in the "list_data" output).

```
h = 1;   % for example, if you have no datasets defined.
```

   Or, to determine it from our current state:

```
h =  max( all_data ) + 1 ; % if you have datasets defined
```

4. Assign the responses. Assigning them to a non-existent dataset will create an empty dataset with a fake-data flag set to 1 ("h" is the index, there is as of yet no counts histogram):

```
assign_rsp( [a66, a67, a65], [r6, r7, r5], h );
```

   You can now continue with the definition of a model and faking data. Model evaluation will fold the model through each assigned response and sum the results into the counts and model histograms associated with index "`h`".

   What the `arcus_utils.sl` functions do is construct all the response name multiplets, load them, and assign them, saving you much typing.

III. Bonus Material!

There are various other utility functions in arcus_utils.sl, but none very interesting unless you need to customize or debug. Ask for help (which is easier than documenting things right now).

However, there is one interesting and useful function: defining a common grid. While it will waste some storage by creating many bins with zero counts, it does allow you to plot combined spectra for a global view (just don't do it before detailed analysis, because it will also combine data in inappropriate ways for that).

1. Get a merged grid for the 10 faked orders: This function will determine where orders overlap and take the smaller of the bin sizes in that range.

   First we need an array of the spectral indices. We can get that from "list_data", or from our info structure made when we initialized the fake dataset:

   ```
   h60 = array_struct_field( d60, "h" );
   ```

   This is nominally the same as saying: `h60 = [1:10] ;`
   But if you had other data defined or loaded first, indices 1--10 might not be correct.

   Form the common grid:

   ```
   g = common_grid( h60 );
   ```

2. Use the grid to evaluate your model in model space:

   Spectral grids in ISIS generally require both low and high bounds, so first make the high-bounds grid:

   ```
   ghi = make_hi_grid( g );
   ```

   Use the grid to evaluate the currently defined model:

   ```
   f = eval_fun( g, ghi )/(ghi-g);% [photon/cm^2/s/A];
   ```

   Take a look:

   ```
   limits; xlog; ylog;
   hplot( g, ghi, f );
   ```

(If you want some other model, see the help for eval_fun2().)

3. Regrid all your faked data:

This will determine the common grid and match all data and responses to this grid:

```
regrid_orders( h60 );
```

Since we didn't set the fake flag to 0 on the data sets, we have to redo this:

```
fakeit;
```

(see the help for set_fake(); if the flags are unset first, this `fakeit` probably isn't necessary).

4. Use Mike Nowak's fancy-plotting functions:

```
require("fancy_plots" ); %or get from Remeis'isisscripts.sl
```

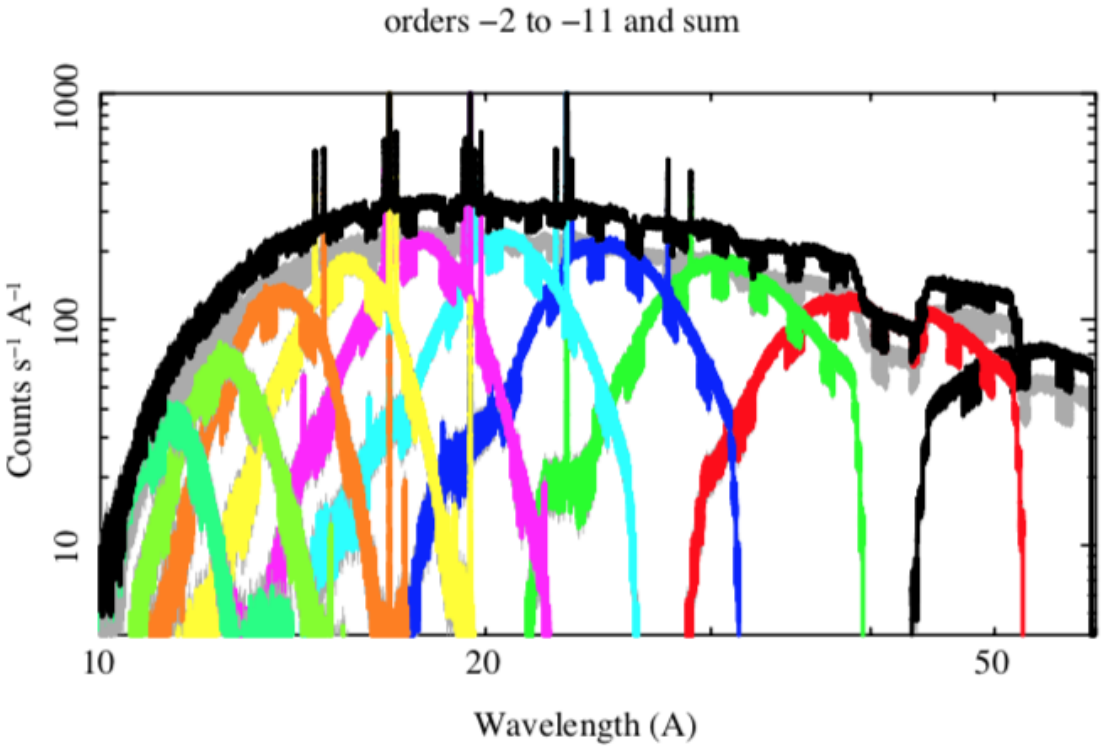
Customize some parameters:

```
set_plot_widths( ; d_width=5, m_width = 3, r_width=5);

popt.res   =  0 ;   popt.dsym  =  0 ;   popt.dcol  =  1 ;
popt.mcol  =  2 ;   popt.rcol =   4 ;   popt.decol = 15 ;
popt.recol =  0 ;   popt.rsym  =  0 ;   popt.con_mod = 1 ;
popt.xrange = NULL; popt.yrange = NULL;


  %%% next line is CRUCIAL for combined orders in single
"observation":
    popt.sum_exp = 0 ;
```

And plot the summed orders:

```
xrange( 10, 60); xlog;
title("orders -2 to -11 and sum");
plot_data( {htch},popt; yrange=[4,1000] );
plot_data( {h60},popt; oplt=1, dcol=15 );
```

```
_for i (0, 9) plot_data({htch[i]},popt; oplt=1, dcol=i+1);
```



Note: *addition happens dynamically*; there are no files or summed data or new response files required.

(If you are energy-centric, your plot units could be changed to keV.)

Example ISIS commands are also in arcus_utils.sl, following the "`#stop`" line (which stops interpretation when you load the file).

XSPEC (Eric Miller)

Introduction

These Xspec commands provide an example of simulating an Arcus observation including the effects of cross-talk from neighboring orders. We first simulate a spectrum extracted from order 6, and so include cross-talk from orders 5 and 7. In order to analyze the order 6 spectrum, we need to simultaneously fit spectra extracted from orders 5 and 7, both of which also include cross-talk from order 6, so we simulate those spectra as well. The model used here for demonstration purposes is a constant continuum plus 7 narrow Gaussian emission lines. The lines are spaced 0.7 keV apart from each other to ensure the cross-talk components fall at different energies in each order (i.e. m*lambda is not divisible by 0.7 keV). The simulated observation is 100 ksec.

The remainder of this section, including comments, can be copied and pasted directly into XSPEC or saved as a .xcm file and edited. Actual XSPEC commands are highlighted in blue. This assumes the working directory contains the RMF files, with the cross-talk ARF files contained in the subdirectory 'osiptouch/':

```
/path/to/my/dir > find .
far_chan_all_-10.rmf
far_chan_all_-11.rmf
far_chan_all_-2.rmf
far_chan_all_-3.rmf
far_chan_all_-4.rmf
far_chan_all_-5.rmf
far_chan_all_-6.rmf
far_chan_all_-7.rmf
far_chan_all_-8.rmf
far_chan_all_-9.rmf
near_chan_all_+1.rmf
near_chan_all_+2.rmf
near_chan_all_-1.rmf
near_chan_all_-2.rmf
osiptouch/far_OSIP_regions.pdf
osiptouch/far_README.md
osiptouch/far_chan_all_ccdord_-10_true_-10.arf
osiptouch/far_chan_all_ccdord_-10_true_-11.arf
osiptouch/far_chan_all_ccdord_-10_true_-9.arf
osiptouch/far_chan_all_ccdord_-11_true_-10.arf
osiptouch/far_chan_all_ccdord_-11_true_-11.arf
osiptouch/far_chan_all_ccdord_-2_true_-2.arf
osiptouch/far_chan_all_ccdord_-2_true_-3.arf
osiptouch/far_chan_all_ccdord_-3_true_-2.arf
osiptouch/far_chan_all_ccdord_-3_true_-3.arf
osiptouch/far_chan_all_ccdord_-3_true_-4.arf
osiptouch/far_chan_all_ccdord_-4_true_-3.arf
osiptouch/far_chan_all_ccdord_-4_true_-4.arf
osiptouch/far_chan_all_ccdord_-4_true_-5.arf
osiptouch/far_chan_all_ccdord_-5_true_-4.arf
osiptouch/far_chan_all_ccdord_-5_true_-5.arf
osiptouch/far_chan_all_ccdord_-5_true_-6.arf
osiptouch/far_chan_all_ccdord_-6_true_-5.arf
osiptouch/far_chan_all_ccdord_-6_true_-6.arf
osiptouch/far_chan_all_ccdord_-6_true_-7.arf
osiptouch/far_chan_all_ccdord_-7_true_-6.arf
osiptouch/far_chan_all_ccdord_-7_true_-7.arf
osiptouch/far_chan_all_ccdord_-7_true_-8.arf
osiptouch/far_chan_all_ccdord_-8_true_-7.arf
osiptouch/far_chan_all_ccdord_-8_true_-8.arf
osiptouch/far_chan_all_ccdord_-8_true_-9.arf
osiptouch/far_chan_all_ccdord_-9_true_-10.arf
osiptouch/far_chan_all_ccdord_-9_true_-8.arf
osiptouch/far_chan_all_ccdord_-9_true_-9.arf
```

Starting XSPEC

```
#################################################################
# Start XSPEC commands.
#################################################################
# Xspec confiuration preamble.
method leven 10 0.01
abund wilm
xsect vern
```

```
cosmo 70 0 0.73
xset delta 0.01
setplot energy
setplot xlog off

######################################################################
# Fake the three spectra that contribute to our extracted order 6 spectrum.
######################################################################

# First we have to fake a spectrum just so we have some data read in to define 'sources',
# because XSPEC does this based on responses, not based on models, and one must have a
# spectrum loaded to use the ‘response’ command. (A chicken and egg problem.) If we
# already have a spectrum with the proper Arcus PHA channels we can just read that in and
# skip this initial 'fakeit' step.  First define an arbitrary model.
model 1:order6 powerlaw
0
1

# Run fakeit to get a temporary (arbitrary) spectrum. Note the blank line after ‘y’.
fakeit none
far_chan_all_-6.rmf
osiptouch/far_chan_all_ccdord_-6_true_-6.arf
y

tmp.pha
100
```

Defining model sources

```
# Define the model sources, one for each order that contributes to the spectrum; have
# to do this with the command
#    response [source]:[spectrum] <rmffile>
# which is why we need a spectrum already loaded with the proper channels defined.
# In fact the model is the same for all three sources, we're just sampling it
# differently (via cross-talk from the neighboring orders).
resp 1:1 far_chan_all_-6.rmf
arf  1:1 osiptouch/far_chan_all_ccdord_-6_true_-6.arf
resp 2:1 far_chan_all_-5.rmf
arf  2:1 osiptouch/far_chan_all_ccdord_-6_true_-5.arf
resp 3:1 far_chan_all_-7.rmf
arf  3:1 osiptouch/far_chan_all_ccdord_-6_true_-7.arf

# Now define the *real* model for all three sources. They should probably all be the
# same, which means you need to ensure that you have some reliable broad-band model
```

for this object, because this scatters other energy bands of the spectrum into your
spectrum. The model used here for demonstration purposes is a constant continuum
plus 7 narrow Gaussian emission lines. The lines are spaced 0.7 keV apart from
each other to ensure the cross-talk components fall at different energies in each
order (i.e. m*lambda is not divisible by 0.7 keV). There are three model 'sources'
that we initialized above using the 'resp' commands, and which we define here:
Source 1 = flux produced in order 6 energy range ('true' = 6)
Source 2 = flux produced in order 5 energy range ('true' = 5)
Source 3 = flux produced in order 7 energy range ('true' = 7)
```
model 1:order6 powerlaw + gaussian + gaussian + gaussian + gaussian +
gaussian + gaussian + gaussian
0
1
.450
.0001
1
.520
.0001
1
.590
.0001
1
.660
.0001
1
.730
.0001
1
.800
.0001
1
.870
.0001
1
model 2:order5 powerlaw + gaussian + gaussian + gaussian + gaussian +
gaussian + gaussian + gaussian
=order6:1
=order6:2
=order6:3
=order6:4
=order6:5
=order6:6
=order6:7
=order6:8
=order6:9
```

```
=order6:10
=order6:11
=order6:12
=order6:13
=order6:14
=order6:15
=order6:16
=order6:17
=order6:18
=order6:19
=order6:20
=order6:21
=order6:22
=order6:23
model 3:order7 powerlaw + gaussian + gaussian + gaussian + gaussian +
gaussian + gaussian + gaussian
=order6:1
=order6:2
=order6:3
=order6:4
=order6:5
=order6:6
=order6:7
=order6:8
=order6:9
=order6:10
=order6:11
=order6:12
=order6:13
=order6:14
=order6:15
=order6:16
=order6:17
=order6:18
=order6:19
=order6:20
=order6:21
=order6:22
=order6:23

# Save the model so we can reload it later if it gets mangled.
save model arcus_fakeit_example_model.xcm
```

Simulating the contributing spectra

```
# Now fake the central order spectrum for realz; don't use 'fakeit none' because if
# you do it will just use a single RMF/ARF, rather than the full complement of
# source responses. Note the blank line after ‘y’ needs to stay.
fakeit
y

order6_faked.pha
100000.

# Now you should have a single spectrum which has summed components from the three
# contributing orders.
sho dat

# Now we have to fake the lower order spectrum, which has contributions from itself
# and the central order. Note we are ignoring cross-talk from orders beyond the three
# we are considering, which are a second-order (so to speak) effect.  First define
# the source responses.
resp 1:1 far_chan_all_-5.rmf
arf  1:1 osiptouch/far_chan_all_ccdord_-5_true_-5.arf
resp 2:1 far_chan_all_-5.rmf
arf  2:1 osiptouch/far_chan_all_ccdord_-5_true_-6.arf
resp 3:1 none

# We don't have to re-define the model, although technically the labels are incorrect
# now. But it's all the same model, so just run fakeit. Again note the blank line.
fakeit
y

order5_faked.pha
100000.

# Now we fake the upper order spectrum, which has contributions from itself and the
# central order. First define the source responses.
resp 1:1 far_chan_all_-7.rmf
arf  1:1 osiptouch/far_chan_all_ccdord_-7_true_-7.arf
resp 2:1 far_chan_all_-7.rmf
arf  2:1 osiptouch/far_chan_all_ccdord_-7_true_-6.arf
resp 3:1 none

# Run fakeit. Again note the blank line.
fakeit
y
```

```
order7_faked.pha
100000.
```

Loading and analyzing the simulated spectra

```
#####################################################################
# Analyze the three spectra that contribute to our extracted order 6 spectrum.
#####################################################################
```

First reset data and model just to be safe.

```
model clear
data none
```

Load the central order spectrum and assign responses to model sources.

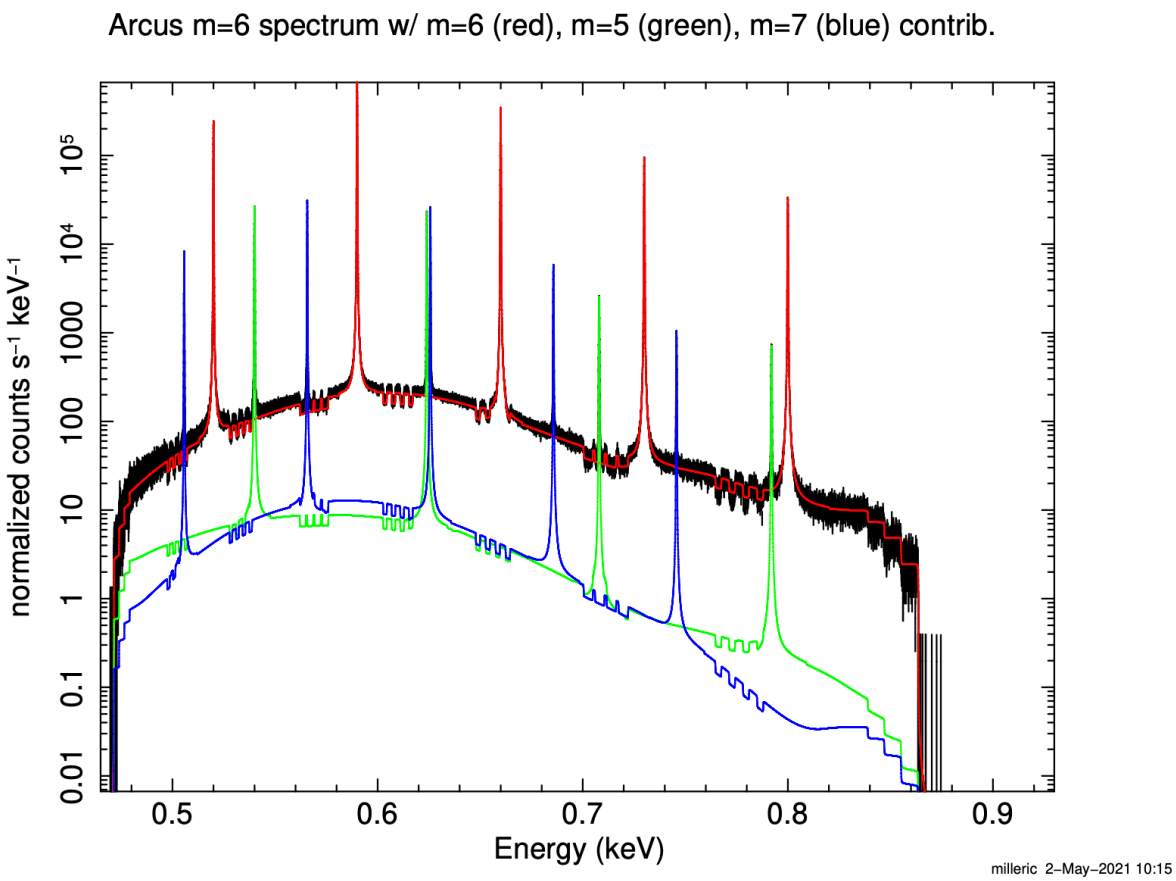
```
data 1:1 order6_faked.pha
resp 1:1 far_chan_all_-6.rmf
arf  1:1 osiptouch/far_chan_all_ccdord_-6_true_-6.arf
resp 2:1 far_chan_all_-5.rmf
arf  2:1 osiptouch/far_chan_all_ccdord_-6_true_-5.arf
resp 3:1 far_chan_all_-7.rmf
arf  3:1 osiptouch/far_chan_all_ccdord_-6_true_-7.arf
```

Load the model.

```
@arcus_fakeit_example_model.xcm
```

Plot it.

```
pl ldat
```



The plot shows the simulated order 6 spectrum, with the in-order model counts shown in red.
The off-order or cross-talk model counts are shown in green from order 5 and blue from order 7.

Load the lower order spectrum and assign responses to model sources.
Note we can load the spectrum into the same data group as the first spectrum, since
they all use the same model parameters.
data <datagroup>:<spectrum number>
```
data 1:2 order5_faked.pha
```
And we'll be more careful with assigning source numbers correctly:
Source 1 = flux produced in order 6 energy range ('true' = 6)
Source 2 = flux produced in order 5 energy range ('true' = 5)
Source 3 = flux produced in order 7 energy range ('true' = 7)
although here it's a detail because, again, the parameters are all tied between
source models. But best to get the bookkeeping correct in case you want to vary
some cross-talk components to see what happens.
Response and ARF loading syntax is:
resp <source number>:<spectrum number>
```
resp 1:2 far_chan_all_-5.rmf
arf  1:2 osiptouch/far_chan_all_ccdord_-5_true_-5.arf
```

```
resp 2:2 far_chan_all_-5.rmf
arf  2:2 osiptouch/far_chan_all_ccdord_-5_true_-6.arf
resp 3:2 none
```

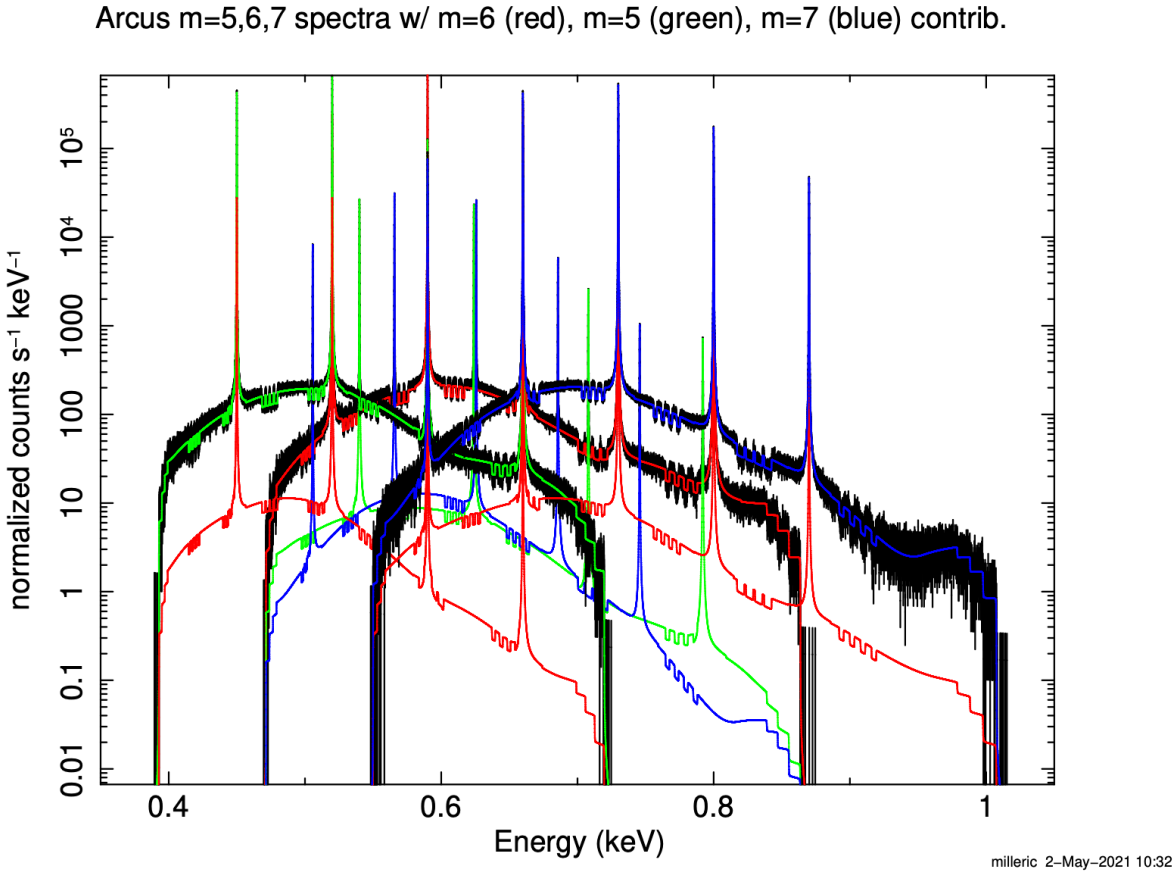
Load the upper order spectrum and assign responses to model sources.

```
data 3:3 order7_faked.pha
resp 1:3 none
resp 2:3 far_chan_all_-7.rmf
arf  2:3 osiptouch/far_chan_all_ccdord_-7_true_-6.arf
resp 3:3 far_chan_all_-7.rmf
arf  3:3 osiptouch/far_chan_all_ccdord_-7_true_-7.arf

sho dat

# 3 files 3 spectra
# Spectrum 1  Spectral Data File: order6_faked.pha
# Net count rate (cts/s) for Spectrum:1  5.099e+02 +/- 7.141e-02
#  Assigned to Data Group 1 and Plot Group 1
#   Noticed Channels:  1-15999
#   Telescope: ARCUS Instrument: far  Channel Type: PHA
#   Exposure Time: 1e+05 sec
#  Using fit statistic: cstat
#  Using Response (RMF) File            far_chan_all_-6.rmf for Source 1
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
6_true_-6.arf
#  Using Response (RMF) File            far_chan_all_-5.rmf for Source 2
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
6_true_-5.arf
#  Using Response (RMF) File            far_chan_all_-7.rmf for Source 3
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
6_true_-7.arf
#
# Spectrum 2  Spectral Data File: order5_faked.pha
# Net count rate (cts/s) for Spectrum:2  4.214e+02 +/- 6.492e-02
#  Assigned to Data Group 1 and Plot Group 2
#   Noticed Channels:  1-15999
#   Telescope: ARCUS Instrument: far  Channel Type: PHA
#   Exposure Time: 1e+05 sec
#  Using fit statistic: cstat
#  Using Response (RMF) File            far_chan_all_-5.rmf for Source 1
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
5_true_-5.arf
#  Using Response (RMF) File            far_chan_all_-5.rmf for Source 2
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
5_true_-6.arf
#
# Spectrum 3  Spectral Data File: order7_faked.pha
# Net count rate (cts/s) for Spectrum:3  4.659e+02 +/- 6.825e-02
```

```
#  Assigned to Data Group 2 and Plot Group 3
#   Noticed Channels:  1-15999
#   Telescope: ARCUS Instrument: far  Channel Type: PHA
#   Exposure Time: 1e+05 sec
#  Using fit statistic: cstat
#  Using Response (RMF) File            far_chan_all_-7.rmf for Source 2
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
7_true_-6.arf
#  Using Response (RMF) File            far_chan_all_-7.rmf for Source 3
#  Using Auxiliary Response (ARF) File  osiptouch/far_chan_all_ccdord_-
7_true_-7.arf
```

Plot it.

```
pl ldat
```



The plot shows the simulated order 5, 6, and 7 spectra, with the order-specific model counts shown in red for order 6, green for 5, and blue for 7.

Now one can fit and get errors.

```
sho free
fit
```

```
# Free parameters defined:
# ========================================================================
# Model order5:powerlaw<1> + gaussian<2> + gaussian<3> + gaussian<4> +
gaussian<5> + gaussian<6> + gaussian<7> + gaussian<8> Source No.: 2
Active/On
# Model Model Component  Parameter  Unit     Value
#  par  comp
#                            Data group: 1
#                            Data group: 2
# ____________________________________________________________________
#
# ========================================================================
# Model order6:powerlaw<1> + gaussian<2> + gaussian<3> + gaussian<4> +
gaussian<5> + gaussian<6> + gaussian<7> + gaussian<8> Source No.: 1
Active/On
# Model Model Component  Parameter  Unit     Value
#  par  comp
#                            Data group: 1
#    1    1   powerlaw   PhoIndex            0.0          +/-  0.0
#    2    1   powerlaw   norm                1.00000      +/-  0.0
#    3    2   gaussian   LineE      keV      0.450000     +/-  0.0
#    4    2   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#    5    2   gaussian   norm                1.00000      +/-  0.0
#    6    3   gaussian   LineE      keV      0.520000     +/-  0.0
#    7    3   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#    8    3   gaussian   norm                1.00000      +/-  0.0
#    9    4   gaussian   LineE      keV      0.590000     +/-  0.0
#   10    4   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#   11    4   gaussian   norm                1.00000      +/-  0.0
#   12    5   gaussian   LineE      keV      0.660000     +/-  0.0
#   13    5   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#   14    5   gaussian   norm                1.00000      +/-  0.0
#   15    6   gaussian   LineE      keV      0.730000     +/-  0.0
#   16    6   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#   17    6   gaussian   norm                1.00000      +/-  0.0
#   18    7   gaussian   LineE      keV      0.800000     +/-  0.0
#   19    7   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#   20    7   gaussian   norm                1.00000      +/-  0.0
#   21    8   gaussian   LineE      keV      0.870000     +/-  0.0
#   22    8   gaussian   Sigma      keV      1.00000E-04  +/-  0.0
#   23    8   gaussian   norm                1.00000      +/-  0.0
# ____________________________________________________________________
#
# ========================================================================
# Model order7:powerlaw<1> + gaussian<2> + gaussian<3> + gaussian<4> +
gaussian<5> + gaussian<6> + gaussian<7> + gaussian<8> Source No.: 3
Active/On
# Model Model Component  Parameter  Unit     Value
```

```
#  par  comp
#                            Data group: 1
#                            Data group: 2
# ------------------------------------------------------------------------------
# ...
# ...
# ...
# ==============================================================================
# Model order6:powerlaw<1> + gaussian<2> + gaussian<3> + gaussian<4> +
gaussian<5> + gaussian<6> + gaussian<7> + gaussian<8> Source No.: 1
Active/On
# Model Model Component  Parameter  Unit     Value
#  par  comp
#                            Data group: 1
#    1    1   powerlaw   PhoIndex            8.66999E-04  +/-  2.05693E-03
#    2    1   powerlaw   norm                0.999493     +/-  1.08088E-03
#    3    2   gaussian   LineE      keV      0.450000     +/-  4.14788E-08
#    4    2   gaussian   Sigma      keV      1.00037E-04  +/-  2.66731E-08
#    5    2   gaussian   norm                0.999748     +/-  2.69443E-04
#    6    3   gaussian   LineE      keV      0.520000     +/-  3.16520E-08
#    7    3   gaussian   Sigma      keV      1.00055E-04  +/-  2.06707E-08
#    8    3   gaussian   norm                1.00030      +/-  1.89658E-04
#    9    4   gaussian   LineE      keV      0.590000     +/-  3.70940E-08
#   10    4   gaussian   Sigma      keV      1.00008E-04  +/-  2.22779E-08
#   11    4   gaussian   norm                0.999856     +/-  1.90297E-04
#   12    5   gaussian   LineE      keV      0.660000     +/-  4.18888E-08
#   13    5   gaussian   Sigma      keV      9.99867E-05  +/-  2.37125E-08
#   14    5   gaussian   norm                0.999780     +/-  1.86646E-04
#   15    6   gaussian   LineE      keV      0.730000     +/-  5.18564E-08
#   16    6   gaussian   Sigma      keV      1.00038E-04  +/-  3.00800E-08
#   17    6   gaussian   norm                0.999771     +/-  2.12163E-04
#   18    7   gaussian   LineE      keV      0.800000     +/-  1.03580E-07
#   19    7   gaussian   Sigma      keV      1.00053E-04  +/-  5.95374E-08
#   20    7   gaussian   norm                0.999469     +/-  3.56751E-04
#   21    8   gaussian   LineE      keV      0.870000     +/-  2.17165E-07
#   22    8   gaussian   Sigma      keV      9.98345E-05  +/-  1.38329E-07
#   23    8   gaussian   norm                0.999321     +/-  7.29960E-04
# ------------------------------------------------------------------------------
# Fit statistic  : C-Statistic                13860.60     using 15999 bins.
#                  C-Statistic                14441.41     using 15999 bins.
#                  C-Statistic                14586.95     using 15467 bins.
# Total fit statistic                         42888.96     with 47442 d.o.f.
#
# Test statistic : Pearson Chi-Squared        46295.98     using 47465 bins.
#
# ***Warning: Pearson Chi-square may not be valid due to bins with zero model
value
#             in spectrum number(s): 1 2 3
#
```

```
#  Null hypothesis probability of 1.00e+00 with 47442 degrees of freedom
```

Sherpa (Moritz Günther)

Instructions for Sherpa are written up as an IPython notebook. You can download the notebook like this:
(unix%) wget -L https://raw.githubusercontent.com/hamogu/arcus-analysis/master/notebooks/SimMultiOrder.ipynb
I recommend to run this notebook inside your CIAO environment. To do so, activate your CIAO environment as normal and then type: “jupyter notebook”. A new tab in your webbrower will open. Navigate to where you save your notebook and open it. Simply edit the cells where I specified the spectral model and put in your model.
Alternatively, you can see a rendered version of the notebook here:
https://space.mit.edu/home/guenther/ARCUS/SimMultiOrder.html
Click on the blue button at the top of the page to toggle on the code view and copy and paste the lines that you need into your own Sherpa session.

SPEX (Lynne Valencic)

1) To make the usual "spex-ified" response files, you will need to use the 'simres' task in PySpextools (‘trafo’ has issues). The PySpextools code and instructions on how to install it are here:
https://github.com/spex-xray/pyspextools

2) When it is installed, activate it and run simres on the arf and rmf files (this can take a minute or so. It will give a warning about a shift in the response array that you can safely ignore.) For this example, let's focus on order -7. Due to crosstalk, you need to simulate the spectra in not only the order -7, but also the orders to either side of it (-6 and -8):

```
conda activate spex

simres --rmffile far_chan_all_-6.rmf --arffile far_chan_all_ccdord_-7_true_-6.arf --spofile far_chan_all_-7_-6.spo --resfile far_chan_all_-7_-6.res

simres --rmffile far_chan_all_-7.rmf --arffile far_chan_all_ccdord_-7_true_-7.arf --spofile far_chan_all_-7_-7.spo --resfile far_chan_all_-7_-7.res

simres --rmffile far_chan_all_-8.rmf --arffile far_chan_all_ccdord_-7_true_-8.arf --spofile far_chan_all_-7_-8.spo --resfile far_chan_all_-7_-8.res
```

3) Now that we have proper res and spo files, we need to arrange them so that SPEX will know how to deal with them. We will have one source model and three res-spo file pairs which need to

be fitted simultaneously. This means using “regions”, which must be specified with pyspextools. For our example, you would call python in the window where SPEX is active, then specify the regions:

```
import pyspextools.io
data = pyspextools.io.Dataset()

data.read_all_regions("far_chan_all_-7_-6.spo","far_chan_all_-7_-6.res")
data.read_all_regions("far_chan_all_-7_-7.spo","far_chan_all_-7_-7.res")
data.read_all_regions("far_chan_all_-7_-8.spo","far_chan_all_-7_-8.res")
```

If you want to verify that the files were read in correctly and assigned their own region:

```
data.show()
```

Now merge the three pairs into one res-spo file pair:

```
data.write_all_regions("far_chan_all_-7_-6to-8.spo","far_chan_all_-7_-6to-
8.res")
```

4) The merged res-spo files can now be used to simulate data. Call SPEX and read in the merged files:

```
spex
data far_chan_all_-7_-6to-8 far_chan_all_-7_-6to-8
```

5) Set the model and make a spectrum. To demonstrate the cross talk, I will blatantly copy what other folks have done and use a power law and gaussians.

```
com po
com ga
com ga
com ga
com ga
com ga
com ga

par 1 1 gamm v 2
par 1 1 norm v 1e5

par 1 2 e v 0.6
par 1 2 norm v 1e6
par 1 2 fwhm v 1e-3

par 1 3 e v 0.7
par 1 3 norm v 1e6
par 1 3 fwhm v 1e-3
```

```
par 1 4 e v 0.8
par 1 4 norm v 1e6
par 1 4 fwhm v 1e-3

par 1 5 e v 0.9
par 1 5 norm v 1e6
par 1 5 fwhm v 1e-3

cal
par sh fr
sim region 1:3
sim 1e6
```

Fitting data with regions is the same as without; SPEX automatically applies the model you set for fitting to all the regions. *Note that for simulating with regions, SPEX does **not** automatically apply the model to all regions!* The default applies it only to region 1. For all regions, you must use the "`sim region minregion:maxregion`" command, as above. If you are simulating over a bunch of regions, you can just set `maxregion` to a really big number, and SPEX will skip over the missing regions.