# Excitations Data Analysis Training Course – Solutions 5

## Horace Introduction

Horace is a program to view and analyse single crystal inelastic neutron scattering data from time-of-flight (ToF) spectrometers. These machines have a large area detector and measure the neutron’s ToF in order to determine their velocity (energy). Combined with a way of defining either the neutron’s energy before or after it scatters from a sample the ToF information allow the neutron’s energy transfer to or from the sample to be determined. The location of the detected neutron on the 2D area detector then allows two of the three components of the momentum transfer vector to be determined.

Horace is based on the observation that depending on the orientation of a single crystal sample with respects to the incident beam, the projection of the 2D detector surface on the 3D momentum transfer space changes. Thus a 4D *S*(***Q***,ω) dataset can be built up from multiple measurements of a single crystal sample at different orientations. Horace is designed to perform such a recombination and then to quickly access and rebin the data in 3D, 2D, or 1D for further analysis. The data structures which permits this quick access is describe in more detail in the Horace paper [[1]](#_References).

## Fitting a single 1d cut with some peaks

This session continues from yesterday’s session. To start you off, you should have made some cuts as follows:

% Create cuts and slices for use later

sqw\_file = '/mnt/ceph/auxiliary/excitations/edatc/iron.sqw';

proj = projaxes([1,1,0], [-1,1,0], 'tpye', 'rrr');

% Make our usual 2d slice

my\_slice = cut\_sqw(sqw\_file, proj, ...

[-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], [0,4,280]);

% Make the array of 1d cuts

energy\_range = [80:20:160];

for i = 1:numel(energy\_range)

my\_cuts(i) = cut\_sqw(sqw\_file, proj, ...

[-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], ...

[-10 10]+energy\_range(i));

end

We will now try to fit the mgauss function to the first 1d cut you made above: my\_cuts(1). An introduction of how to do this is documented here: <https://pace-neutrons.github.io/horace-docs/3.5.0/Multifit.html>. The fitting syntax is very rich and flexible, and is common across fitting of functions, S(Q,w) models and resolution convolution. It may take a bit of time, but is worth the effort of becoming familiar with the fitting syntax.

1. Subtract a scalar from your 1d cut, so that the background level is approximately zero (we will deal with accounting for background later).
2. Use the parameters you used for your simulation as your initial guess, and do not use any of the fitting options yet – this will mean that all of the fit parameters are free to vary. Once the fit has run, overplot the result on the data as a line, like you did before for the simulation.
3. The results of the fit, i.e. chi-squared, fit parameters and errors, covariance matrix, etc, are held in a structure array. Inspect the structure array to ensure that all of the parameter values and errors are sensible.
4. Now we will explore keeping some parameters free and some fixed. You specify this by adding an extra argument that is a vector of zeros and ones the same length as that which specifies the input parameters. If the nth element of this array is 1, then the nth input parameter is allowed to vary, and if it is zero then the parameter is held fixed. Set all of the peak widths to be constant and run the fit again, again checking the results are sensible.
5. Now we will use parameter binding (use doc sqw/multifit\_func and follow the links to get documentation for the syntax). You do this by creating a cell array of cell arrays, each of which states which parameter is bound to which, and in what ratio. So to bind the 3rd and 4th parameters together in a ratio of 0.2, and the 5th and 6th together in a ratio of 1, you would have {{3,4,0.2}, {5,6,1}}. In our example of multiple Gaussians, set bindings that mean the peak positions are symmetric about x=0.
6. Re-run the above, but now using some of the further options. Set things up so that a fully verbose output is given in the Matlab command window during the fit, and select a restricted Q-range (of your choice) over which to perform the fit.
7. Now let us deal with the background. While exploring the documentation pages for multifit you probably spotted that you can set a background function as well. Use the built-in function linear\_bg, which takes two parameters [constant, gradient]. Set the initial value of the constant to the value you subtracted at the beginning of this fitting exercise, and fit.
8. You can plot the background function and foreground functions separately if you like. Have a look at the help for the fit method: if you use the keyword ‘components’ then the returned fit is a structure with foreground, background and sum as three separate fields.

Solution script:

%% ========================================================================

% Fit a single cut with a peak function

% =========================================================================

% Allow all parameters to be free

pars\_in = [0.4,-0.7,0.1, 0.5,-0.2,0.1, 0.5,0.2,0.1, 0.4,0.6,0.1, 0.4,1.3,0.1];

kk = multifit\_func (my\_cuts(1)-0.3);

kk = kk.set\_fun (@mgauss, pars\_in);

[wfit, fitdata] = kk.fit;

acolor black

plot(my\_cuts(1)-0.3);

acolor red

pl(wfit);

keep\_figure;

% That was not very good. Let's keep the widths of all the peaks fixed, but

% allow heights and centres to vary;

pars\_in = [0.4,-0.7,0.1, 0.5,-0.2,0.1, 0.5,0.2,0.1, 0.4,0.6,0.1, 0.4,1.3,0.1];

pars\_free = [1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0];

kk = multifit\_func (my\_cuts(1)-0.35);

kk = kk.set\_fun (@mgauss, pars\_in);

kk = kk.set\_free (pars\_free);

[wfit, fitdata] = kk.fit;

acolor black

plot(my\_cuts(1)-0.35);

acolor red

pl(wfit);

keep\_figure;

% That still wasn't too good. Now start with more realistic values for the

% widths, and bind some of the peak positions to follow symmetry, i.e. the

% position of peaks for Q<0 are reflection of those at Q>0

pars\_in = [0.4,-0.8,0.07, 0.5,-0.22,0.07, 0.5,0.22,0.07, 0.4,0.8,0.07, 0.4,1.2,0.07];

pars\_free = [1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0];

pars\_bind = {{2,11,-1}, {5,8,-1}}; % ensures symmetry about x=0

% The syntax above is that each binding is a cell array, which has the form:

% {ipar\_bound, ipar\_free, ratio}

% More generally, if there are several functions (see later) then we would

% use the form:

% {[par\_bound,ifun\_bound], [ipar\_free,ifun\_free], ratio}

% However, in this case, there is only one function and it is assumed.

kk = multifit\_func (my\_cuts(1)-0.35);

kk = kk.set\_fun (@mgauss, pars\_in);

kk = kk.set\_free (pars\_free);

kk = kk.set\_bind (pars\_bind);

[wfit, fitdata] = kk.fit;

acolor black

plot(my\_cuts(1)-0.35);

acolor red

pl(wfit);

keep\_figure;

% Repeat the above, but using some of the options to restrict the range of fitting

% and produce verbose output

kk = multifit\_func (my\_cuts(1)-0.35);

kk = kk.set\_fun (@mgauss, pars\_in);

kk = kk.set\_free (pars\_free);

kk = kk.set\_bind (pars\_bind);

kk = kk.set\_mask ('keep',[-1,1.5]);

kk = kk.set\_options ('list',2);

[wfit, fitdata] = kk.fit;

acolor black

plot(my\_cuts(1)-0.35);

acolor red

pl(wfit);

keep\_figure;

% Lastly, set a background function as well. In full:

pars\_in = [0.4,-0.8,0.07, 0.5,-0.22,0.07, 0.5,0.22,0.07, 0.4,0.8,0.07, 0.4,1.2,0.07];

pars\_free = [1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0];

pars\_bind = {{2,11,-1}, {5,8,-1}}; % ensures symmetry about x=0

kk = multifit\_func (my\_cuts(1));

kk = kk.set\_fun (@mgauss, pars\_in);

kk = kk.set\_free (pars\_free);

kk = kk.set\_bind (pars\_bind);

kk = kk.set\_bfun (@linear\_bg, [0.35,0]);

kk = kk.set\_bfree ([1,0]);

[wfit, fitdata] = kk.fit;

acolor black

plot(my\_cuts(1));

acolor red

pl(wfit);

keep\_figure;

% Plot the background and sum on one plot

[wfit, fitdata] = kk.fit('components');

acolor black

plot(my\_cuts(1));

acolor red

pl(wfit.sum);

pl(wfit.back);

keep\_figure;

## Fitting a single 1d cut with an S(Q,w) model (including coding your own model)

We will now fit part of the same 1d cut we used before with multifit\_func, but with multifit\_sqw, which is for the case where you have a model for S(Q,w). The function syntax (parameters, free/fixed, binding) is exactly the same. Make a new 1d cut that is the same as the previous one, with energy integration range 100 to 120 meV, but with a Q range of 0.5 to 1.5 in steps of 0.05.

1. Try fitting your new cut on a linear background using multifit\_sqw to the model for S(Q,w) in sr122\_xsec. Good starting parameters are the ones we used at the beginning of this session when simulating the Q-E slice, namely [1, 0, 0, 35, -5, 15, 10, 0.1]. Remember that parameters 2 and 3 are redundant, so fix these in the fit. Use a linear background model linear\_bg (the same one you were using before) with appropriate background constant and gradient as starting parameters.
2. You will remember from when we simulated the Q-E slice that the model we have been using so far will not be all that good for globally fitting our data! Using the file sr122\_xsec.m as a starting template, write your own Matlab file that codes the following model:

https://latex.codecogs.com/png.latex?%5Clarge%20E_0%20%3D%20%5CDelta%20&plus;%208J%281%20-%20%5Ccos%28%5Cpi%20h%29%5Ccos%28%5Cpi%20k%29%5Ccos%28%5Cpi%20l%29%29

https://latex.codecogs.com/png.latex?%5Clarge%20B%20%3D%20%5Cfrac%7BE%7D%7B1%20-%20%5Cexp%28-%5Cfrac%7B1.602%20E%7D%7BT%7D%29%7D

https://latex.codecogs.com/png.latex?%5Clarge%20S%28%5Cmathbf%7BQ%7D%2CE%29%20%3D%20AB%20%5Cfrac%7B4%5CGamma%20E_0/%5Cpi%7D%7B%28E%5E2%20-%20E_0%5E2%29%5E2%20&plus;%204%28%5CGamma%20E%29%5E2%7D

1. Where *A* is an amplitude scale factor, *T* is the temperature in Kelvin, *J* is a ferromagnetic exchange constant in meV, Γ is an energy broadening term in meV, and Δ is an energy gap in meV .
2. Run a simulation of our favourite Q-E slice using this new function, and play around with the input parameters to get something that looks similar to the data. Note that the temperature should be fixed at 10 K.
3. Now try fitting the 1d cut, allowing for a linear background model as well. Play with fixing different parameters in the fit. What sensitivity does the data have to the different parameters, and how might pairs of parameters be coupled in a fit?

Solutions script:

%% ========================================================================

% Fit a single cut with an S(Q,w) model

% =========================================================================

% Use the sr122 cross-section here, and fit only a small section, because

% calculating the cross-section is a lot slower than the dispersion or

% just fitting peaks.

my\_new\_cut = cut\_sqw(sqw\_file, proj, [0.5,0.05,1.5], [-1.1,-0.9], [-0.1,0.1], [100,120]);

pars = [1,0,0,35,-5,15,10,0.1];

pfree = [1,0,0,1,1,1,1,1];

kk = multifit\_sqw (my\_new\_cut);

kk = kk.set\_fun (@sr122\_xsec, pars);

kk = kk.set\_free (pfree);

kk = kk.set\_bfun (@linear\_bg, [0.35,0]);

kk = kk.set\_options ('list',2);

[wfit, fitdata] = kk.fit('components');

acolor black

plot(my\_new\_cut);

acolor red

pl(wfit.sum);

pl(wfit.back);

keep\_figure;

% Correct ferromagnetic spin-waves function

% -----------------------------------------

% So far we have used a library function (for SrFe2As2) which

% is not suitable for the dataset we actually have (bcc-Fe). The worksheet

% gave an analytical expression for the cross-section for the expected

% ferromagnetic spin waves and instructed you to modify the library model

% to use the new function which the commands below assumes will be in the

% file Fe\_FM\_spinwaves.m

%

% In addition, we have also included a "model answer" version which also

% includes the Fe form factor in the file Fe\_FM\_spinwaves\_FF.m

% Simulate our favourite Q-E slice with your cross-section model

my\_slice = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], [0,4,280]);

test\_slice = sqw\_eval(my\_slice, @Fe\_FM\_spinwaves\_FF, [35 0 30 10 1000]);

plot(test\_slice);

% Fit our newly created short cut. There is no separate sensitivity to the

% gap and the exchange constant; oneof them has to be fixed. The temeprature

% also has to be fixed. Also, the background gradient is not well-defined,

% so we fix that too.

pars = [35, 0, 30, 10, 1000];

pfree = [1, 0, 1, 0, 1];

kk = multifit\_sqw (my\_new\_cut);

kk = kk.set\_fun (@Fe\_FM\_spinwaves\_FF, pars);

kk = kk.set\_free (pfree);

kk = kk.set\_bfun (@linear\_bg, [0.15,0]);

kk = kk.set\_bfree ([1,0]);

kk = kk.set\_options ('list',2);

[wfit, fitdata] = kk.fit('components');

acolor black

plot(my\_new\_cut);

acolor red

pl(wfit.sum);

pl(wfit.back);

keep\_figure;

The Fe\_FM\_spinwaves.m file:

function y=Fe\_FM\_spinwaves(qh,qk,ql,en,pars)

% Calculate the spectral weight from a Heisenberg ferromagnet for a Q-E slice

% with nearest-neighbour interactions only.

%

% >> wout= sqw\_eval(win,@Fe\_FM\_spinwaves,[pars],options)

%

% Input for function:

% ------

% qh,qk,ql,en - arrays that are all the same size (or are scalars)

% that specify the co-ordinates for point in (Q,E)-space

% where you wish to calculate the intensity

%

% pars [js, delta, gam, temp, amp]

% js nearest neighbour exchange constant (meV)

% delta energy gap at the zone centre (meV)

% gam inverse lifetime (meV)

% temp temperature (K)

% amp intensity scale factor

%

% Output:

% -------

% y array the same size as the input arrays qh, qk, ql and en.

% A trick to avoid divide by zero warning

warning\_status = warning('query');

warning off

%=============================================

js=pars(1); delta=pars(2);

gam=pars(3); temp=pars(4); amp=pars(5);

omega0 = delta + (8\*js)\*(1-cos(pi\*qh).\*cos(pi\*qk).\*cos(pi\*ql));

Bose= en./ (1-exp(-11.602.\*en./temp));%Bose factor from Tobyfit.

% Use DSHO model to give intensity:

y = amp .\* (Bose .\* (4.\*gam.\*omega0)./(pi.\*((en.^2-omega0.^2).^2 + 4.\*(gam.\*en).^2)));

%=============================================

% Return to original warning status

warning(warning\_status);

The Fe\_FM\_spinwaves\_FF.m file (with Fe2+ magnetic form factor):

function y=Fe\_FM\_spinwaves\_FF(qh,qk,ql,en,pars)

% Calculate the spectral weight from a Heisenberg ferromagnet for a Q-E slice

% with nearest-neighbour interactions only, including magnetic form factor

%

% >> wout= sqw\_eval(win,@Fe\_FM\_spinwaves,[pars],options)

%

% Input for function:

% ------

% qh,qk,ql,en - arrays that are all the same size (or are scalars)

% that specify the co-ordinates for point in (Q,E)-space

% where you wish to calculate the intensity

%

% pars [js, delta, gam, temp, amp]

% js nearest neighbour exchange constant (meV)

% delta energy gap at the zone centre (meV)

% gam inverse lifetime (meV)

% temp temperature (K)

% amp intensity scale factor

%

% Output:

% -------

% y array the same size as the input arrays qh, qk, ql and en.

% Trick to avoid divide by zero warning

warning\_status = warning('query');

warning off

%=============================================

js=pars(1); delta=pars(2);

gam=pars(3); temp=pars(4); amp=pars(5);

omega0 = delta + (8\*js)\*(1-cos(pi\*qh).\*cos(pi\*qk).\*cos(pi\*ql));

Bose= en./ (1-exp(-11.602.\*en./temp));%Bose factor from Tobyfit.

% Use DSHO model to give intensity:

y = amp .\* (Bose .\* (4.\*gam.\*omega0)./(pi.\*((en.^2-omega0.^2).^2 + 4.\*(gam.\*en).^2)));

%==============

% Now multiply by the magnetic form factor for elemental iron

A=0.0706; a=35.008; B=0.3589; b=15.358; C=0.5819; c=5.561; D=-0.0114;

modQ=(2\*pi/2.87)\*sqrt(qh.^2+qk.^2+ql.^2);

ssqr=(modQ./(4\*pi)).^2;

j0=A\*exp(-a\*ssqr) + B\*exp(-b\*ssqr) + C\*exp(-c\*ssqr) + D;

FF=j0;

%==============

y=y.\*(FF.^2);

%=============================================

% Return to original warning status

warning(warning\_status);

## Fitting multiple cuts simultaneously with a single S(Q,w) model

One of the key features of Horace is the ability to fit many objects to a single model and single set of parameters, but with independent backgrounds. This can be done to an arbitrary collection of 1d, 2d, 3d and 4d objects, not just 1d cuts, although that is all we will do here.

1. Use the array of 1d cuts created at the start of this exercise that all cover the same Q-range for a series of energies.
2. Run multifit\_sqw but this time initialise with the array of cuts. Make a series of plots of the results (i.e. plot the cut and add a line for its corresponding fit, and repeat for every cut). Also look at the structure array that contains the fit data, to understand how the information is now stored there.
3. For our final fitting exercise we are going to use different background functions for different cuts. To do this you need to create a cell array whose number of elements is equal to the number of cuts. Each element should be the handle to the background function you are going to use – use the linear\_bg function (Herbert built-in) for the first three cuts, and quad\_bg for the last two. The input parameters for the background functions similarly have to be contained in a cell array, each element of which is a vector of input parameters; likewise the free parameter list.
4. As an exercise in using bindings with multiple datasets, set the bindings so that the gradients are all the same for the linear backgrounds, and the x2 terms are all the same for the quadratic backgrounds.
5. Once the fit above has run and converged, plot the results of the cuts and fits. Also simulate the Q-E slice with the fit parameters you found, as a sanity check.

Solutions script:

%% ========================================================================

% Fit multiple cuts simultaneously with a single S(Q,w) model

% =========================================================================

% Make the array of 1d cuts previous made in the advance plotting session

energy\_range = [80:20:160];

for i = 1:numel(energy\_range)

my\_cuts(i) = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], ...

[-10 10]+energy\_range(i));

end

% ------------------------------------------------

% To begin just use the same input as above, i.e. single parameter set and a

% single set of parameters for the background functions

kk = multifit\_sqw (my\_cuts);

kk = kk.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

kk = kk.set\_free ([1, 0, 1, 0, 1]);

kk = kk.set\_bfun (@linear\_bg, [0.1,0]);

kk = kk.set\_bfree ([1,0]);

kk = kk.set\_options ('list',2);

[wfit, fitdata] = kk.fit('comp');

for i=1:numel(my\_cuts)

acolor black

plot(my\_cuts(i));

acolor red

pl(wfit.sum(i));

pl(wfit.back(i));

keep\_figure;

end

% ------------------------------------------------

% These fits are pretty good, but let's do something a bit more

% sophisticated with the backgrounds: set different background functions

% for different cuts

% Initialise the fitting object

kk = multifit\_sqw (my\_cuts);

% Our usual starting parameters and bindings for the cross-section:

kk = kk.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

kk = kk.set\_free ([1, 0, 1, 0, 1]);

% Set background functions, one per dataset

bgfunc = {@linear\_bg, @linear\_bg, @linear\_bg, @quad\_bg, @quad\_bg};

bgpars = {[0.37,0], [0.2,0], [0.14,0], [0.08,0,0], [0.03,0,0]};

bgfree = {[1,1], [1,1], [1,1], [1,1,1], [1,1,1]};

kk = kk.set\_bfun (bgfunc, bgpars);

kk = kk.set\_bfree (bgfree);

% Bind the linear background gradients together. Note how you can accumulate

% bindings a bit at a time

kk = kk.set\_bbind({[2,2],[2,1],1},{[2,3],[2,1],1}); % bind gradients for linear bg

kk = kk.add\_bbind({[3,5],[3,4],1});

% Ask for copious output

kk = kk.set\_options ('list',2);

% Now perform the fit and plot the results

[wfit, fitdata] = kk.fit('comp');

for i=1:numel(my\_cuts)

acolor black

plot(my\_cuts(i));

acolor red

pl(wfit.sum(i));

pl(wfit.back(i));

keep\_figure;

end

% ------------------------------------------------

% Look at a slice with our fit parameters:

test\_slice = sqw\_eval(my\_slice, @Fe\_FM\_spinwaves\_FF, fitdata.p);

plot(my\_slice); keep\_figure;

plot(test\_slice); keep\_figure;

## Fitting with Resolution Convolution

Often the resolution function of the instrument can make a significant quantitative difference to the result of a fit. In this session we are going to repeat the simultaneous fit to multiple cuts but this time include the effects of resolution. The program which does this is the eponymous tobyfit, which has a very similar syntax to multifit and its variants. Before yu can use it, additional information needs to be added to the cuts that describes the configuration of the instrument – moderator and chopper pulse shapes, the size of the sample, information about the beam divergence and the size of the detectors. Use the on-line documentation for Tobyfit as a guide in what follows.

1. First create a sample description. Create an object of type IX\_sample. The sample can be approximated as a cuboid 3cm x 3cm x 4cm high, with the sides parallel to a\*, b\* and c\* respectively.
2. Create an instrument object using the function maps\_instrument. This function contains much of the fixed instrument description, and just needs a few parameters specific to the particular experiment. For reference, there are equivalent instrument generation functions for maps, mari and let. You already know that the incident energy was 401 meV. The monochromating chopper was the ‘Sloppy’ chopper running at 600Hz. Have a browse through the output of this function.
3. Retaining your original cuts, set the sample and instrument descriptions using set\_sample and set\_instrument. Have a look at the header section of one of the cuts. Can you see where the information is stored?
4. Now create a fitting object using Tobyfit. This works just like multifit – type doc sqw/tobyfit to open a help window. If you look closely at the list of methods you’ll see that there are a few extra ones that are specific to Tobyfit. We’ll one or two of these later on. Set the cross-section model to the one you wrote for use with multifit\_sqw, and set the starting parameter values (and which parameters are free) to the same ones you used in the previous exercise. To start with, just initialise Tobyfit with one of the cuts in your array, and set a linear background. Fit the parameters and compare the results with those you get using multifit\_sqw.
5. Do the same, but now for all five datasets simultaneously. Follow a similar procedure as you did when you fitted all five cuts simultaneously with multifit\_sqw earlier on, namely a single function for the S(Q,w) model but independent linear backgrounds for each cut.
6. One of the control parameters you have in Tobyfit is the number of Monte Carlo points per detector-energy pixel. Use the documentation doc sqw/tobyfit and navigate to the list of methods to work out how to set the number of Monte Carlo points. The default is 10. Why does the resolution convolution algorithm give useful results with such a small number?
7. There are various contributions to the resolution function of the instrument, and you can disable one or more of these contributions. Experiment with the effect of turning off the moderator contribution, the chopper contribution, and others too. Just to keep the fitting time reasonable, return to fitting just the second cut of the array. What are the major contributors to the resolution of the instrument?
8. As a final exercise, set the foreground function to be local and fit the data with the exchange constant to be constrained to be the same for all cuts, but the intensity and lifetimes to vary. Why would you want to do this? At this point you will have become an expert!

Solutions script:

%% ========================================================================

% Resolution Convolution

% =========================================================================

% NOTE - For help about the syntax of any command, type in Matlab:

% >> help routine\_name

% or

% >> doc routine\_name

%

% EXAMPLES

% To prints in the Matlab command window the help for the gen\_sqw routine

% >> help gen\_sqw

%

% To displays the help for gen\_sqw in the Matlab documentation window

% >> doc gen\_sqw

clear variables

%% ========================================================================

% Fitting with resolution convolution

% =========================================================================

% We will use the same array of 1d cuts but now account for resolution

%-----------------------------------

% Create cuts and slices for use later

sqw\_file = '/mnt/ceph/auxiliary/excitations/edatc/iron.sqw';

proj.u = [1,1,0]; proj.v = [-1,1,0]; proj.uoffset = [0,0,0,0]; proj.type = 'rrr';

% Make our usual 2d slice

my\_slice = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], [0,4,280]);

% Make the array of 1d cuts previous made in the advance plotting session

energy\_range = [80:20:160];

for i = 1:numel(energy\_range)

my\_cuts(i) = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], ...

[-10 10]+energy\_range(i));

end

%-----------------------------------

% The first thing we need to do is attach instrument and sample size

% information to the cuts. We can do this to the sqw file itself, and the

% information will be propagated to any cut that outputs an sqw object.

% We can attach it to the cuts themselves, and that is what we will do here.

% Create a sample - something not totally dissimilar to the Fe crystal shape

sample = IX\_sample(true,[1,0,0],[0,1,0],'cuboid',[0.03,0.03,0.04]);

% Create an instrument description

ei = 401;

freq = 600;

chopper = 'S';

instru = maps\_instrument(401, freq, chopper);

% We can now set the sample and instrument

%my\_cuts\_tf = set\_sample\_horace ('my\_file.sqw', sample);

my\_cuts\_tf = set\_sample (my\_cuts, sample);

my\_cuts\_tf = set\_instrument (my\_cuts\_tf, instru);

%-----------------------------------

% Now fit one of the cuts

% ------------------------

% Initialise the fitting object

tf = tobyfit (my\_cuts\_tf(1));

tf = tf.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

tf = tf.set\_free ([1, 0, 1, 0, 1]);

tf = tf.set\_bfun (@linear\_bg, [0.37, 0]);

tf = tf.set\_bfree ([1, 0]);

tf = tf.set\_options ('list',2);

% Now perform the fit and plot the results

[wfit\_tf, fitdata\_tf] = tf.fit('comp');

acolor black

plot(my\_cuts\_tf(1));

acolor red

pl(wfit\_tf.sum);

pl(wfit\_tf.back);

keep\_figure;

% Compare with no including resolution convolution

kk = multifit\_sqw (my\_cuts\_tf(1));

kk = kk.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

kk = kk.set\_free ([1, 0, 1, 0, 1]);

kk = kk.set\_bfun (@linear\_bg, [0.37,0]);

kk = kk.set\_bfree ([1,0]);

kk = kk.set\_options ('list',2);

% Now perform the fit and plot the results

[wfit, fitdata] = kk.fit('comp');

acolor black

plot(my\_cuts\_tf(1));

acolor red

pl(wfit.sum);

pl(wfit.back);

keep\_figure;

%-----------------------------------

% Now fit all five cuts simultaneously

% ------------------------------------

% Initialise the fitting object

tf = tobyfit (my\_cuts\_tf);

tf = tf.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

tf = tf.set\_free ([1, 0, 1, 0, 1]);

tf = tf.set\_bfun (@linear\_bg, [0.37,0]);

tf = tf.set\_bfree ([1,0]);

tf = tf.set\_options ('list',2);

% Now perform the fit and plot the results

[wfit\_tf, fitdata\_tf] = tf.fit('comp');

for i=1:numel(my\_cuts\_tf)

acolor black

plot(my\_cuts\_tf(i));

acolor red

pl(wfit\_tf.sum(i));

pl(wfit\_tf.back(i));

keep\_figure;

end

% Compare with no including resolution convolution

kk = multifit\_sqw (my\_cuts\_tf);

kk = kk.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

kk = kk.set\_free ([1, 0, 1, 0, 1]);

kk = kk.set\_bfun (@linear\_bg, [0.37,0]);

kk = kk.set\_bfree ([1,0]);

kk = kk.set\_options ('list',2);

% Now perform the fit and plot the results

[wfit, fitdata] = kk.fit('comp');

for i=1:numel(my\_cuts\_tf)

acolor black

plot(my\_cuts\_tf(i));

acolor red

pl(wfit.sum(i));

pl(wfit.back(i));

keep\_figure;

end

%-----------------------------------

% Now fit one of the cuts, change the contributions

% ------------------------

% Now we will change the number of Monte Carlo points per and fit turning

% off the moderator pulse width

% Initialise the fitting object

tf = tobyfit (my\_cuts\_tf(2));

tf = tf.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

tf = tf.set\_free ([1, 0, 1, 0, 1]);

tf = tf.set\_bfun (@linear\_bg, [0.37,0]);

tf = tf.set\_bfree ([1,0]);

tf = tf.set\_options ('list',2);

% Now change number of Monte Carlo points to 15

tf = tf.set\_mc\_points(15);

% Fit

[wfit\_ref, fitdata\_ref] = tf.fit('comp');

% Fit with no chopper contribution

tf = tf.set\_mc\_contributions('nochop');

[wfit\_nochop, fitdata\_nochop] = tf.fit('comp');

%-----------------------------------

% Semi-global foreground

% Now fit all five cuts simultaneously

% ------------------------------------

% We will allow the intensities and lifetime to vary for each of the cuts,

% but constrain the exchnage constant to be the same for all cuts

% Initialise the fitting object

tf = tobyfit (my\_cuts\_tf);

tf = tf.set\_local\_foreground;

tf = tf.set\_fun (@Fe\_FM\_spinwaves\_FF, [35, 0, 30, 10, 300]);

tf = tf.set\_free ([1, 0, 1, 0, 1]);

tf = tf.set\_bind ({1,[1,1],1}); % check you understand this syntax

tf = tf.set\_bfun (@linear\_bg, [0.37,0]);

tf = tf.set\_bfree ([1,0]);

tf = tf.set\_options ('list',2);

% Now perform the fit and plot the results

[wfit\_tf, fitdata\_tf] = tf.fit('comp');

for i=1:numel(my\_cuts\_tf)

acolor black

plot(my\_cuts\_tf(i));

acolor red

pl(wfit\_tf.sum(i));

pl(wfit\_tf.back(i));

keep\_figure;

end

## A worked example using CuGeO3 (Optional)

We’ve now gone through a full example using the bcc-iron dataset from generating the 4D S(***Q***,ω) file, making cuts and plotting them to simulating a model and including resolution convolution. As an optional exercise, to confirm what you’ve learnt, you also apply this analysis to CuGeO3, a 1D spin-1/2 chain material with spinon excitations. The theory is described in an early work by Nagler et al. [2] and the first experimental inelastic neutron work is by Arai et al. [3]. The theoretical S(q, ω) model (the Muller Ansatz) is coded in a file /home/dl11170/edatc/scripts/Bethe\_Ansatz\_CuGeO3.m but you could equally write it from eqs (3), (4) and (7) of ref [2].

Because the system is 1D, the (magnetic) dispersion is only along a single Q direction (00L) so as long as this direction is perpendicular to the incident beam, it will be imaged in the detectors. As such the data was taken at a single rotation angle. The data file is: /home/dl11170/edatc/data/MER18519\_Ei50.00meV\_One2One.nxspe

It was measured with the *b* axis parallel to the incident beam (ki || [010]) and with the *c* axis [001] horizontal and perpendicular to the incident beam. The lattice parameters are *a*=4.81Å, *b*=8.47 Å, *c*=2.94Å, and α=β=γ=90°. You can assume all the goniometer angles are zero.

The measurement (from an earlier in-person neutron training course) used the MERLIN spectrometer with the gadolinium (‘G’) chopper running at 250 Hz with Ei=50 meV.

1. Create an sqw file from the nxspe data file using the information given above.
2. Make a Q-E cut along the dispersion direction [00L] similar to that shown in Fig. 1 of [3].
3. Evaluate the theoretical (Muller Ansatz / Bethe\_Ansatz\_CuGeO3.m) model on your cut. The theoretical model function takes two parameters: [scale\_factor, JS]. Note that the exchange parameter used in the model function is J\*S which is actually half of the J value quoted in ref [3] because S=1/2.
4. Make some 1D cuts to compare the data and the model calculations – which part of the spectrum should you use to estimate the background?
5. Subtract the background and fit the data to the model. Plot the results – what do you notice about the widths of the data and model?
6. Simulate some spectra including resolution effect and compare that to the data.
7. (Optional) Try to fit the data to the model including instrument resolution effects

Solution script:

%% ========================================================================

% CuGeO3 spinon full example

% =========================================================================

clear variables

addpath('/mnt/ceph/auxiliary/excitations/edatc/scripts');

% Generate the sqw file

data\_path = '/mnt/ceph/auxiliary/excitations/edatc/data/';

par\_file = '';

sqw\_file = './cugeo3.sqw';

efix = 50; emode = 1;

spefile = fullfile(data\_path, 'MER18519\_Ei50.00meV\_One2One.nxspe');

alatt = [4.81, 8.47, 2.94]; angdeg = [90, 90, 90];

u = [0,1,0]; v = [0,0,1];

psi = 0; omega=0; dpsi=0; gl=0; gs=0;

gen\_sqw (spefile, par\_file, sqw\_file, efix, emode, alatt, angdeg,...

u, v, psi, omega, dpsi, gl, gs);

%% Make a Q-E cut showing the spinon excitations and plot it

proj = projaxes([0,0,1],[1,0,0],'type','rrr');

cgo\_cut = cut\_sqw('cugeo3.sqw', proj, [-1,0.05,1], [-inf,inf], [-inf,inf], [0.3]);

plot(cgo\_cut); lz(0, 200);

keep\_figure;

%% Evaluate the theoretical S(q,w) model

scale\_fac = 1;

JS = 9.8 / 2; % Value from PRL 77 3649

cgo\_theory = sqw\_eval(cgo\_cut, @Bethe\_Ansatz\_CuGeO3, [scale\_fac, JS]);

plot(cgo\_theory);

keep\_figure;

%% Plot some 1D cuts

ql = [-0.5 -0.25 0]; % Locations to cut at

colors = 'rgb';

for ii = 1:numel(ql)

cgo\_1d(ii) = cut(cgo\_cut, [-0.05 0.05]+ql(ii), []);

acolor(colors(ii)); pm(cgo\_1d(ii));

end

ly(0,500); keep\_figure;

% Also plot the equivalent cuts from the model calculation

for ii = 1:numel(ql)

cgo\_1d\_calc(ii) = cut(cgo\_theory, [-0.05 0.05]+ql(ii), []);

acolor(colors(ii)); pl(cgo\_1d\_calc(ii));

end

keep\_figure;

% Shows that the signal expected at ql=0 is minimal -> could be used as the background

%% Let's take this cut and replicate it and subtract it,

% then fit the result to the model and plot

% (Remember we have to convert the sqw file to d2d)

cgo\_bkg = replicate(cgo\_1d(3), d2d(cgo\_cut));

cgo\_sub = d2d(cgo\_cut) - cgo\_bkg;

% Fit the model:

[wfit, fitdat] = fit\_sqw(cgo\_sub, @Bethe\_Ansatz\_CuGeO3, [scale\_fac, JS], [1,1]);

plot(wfit); keep\_figure;

% Plot 1d cuts

for ii = 1:numel(ql)

cgo\_1d\_sub(ii) = cut(cgo\_sub, [-0.05 0.05]+ql(ii), []);

cgo\_1d\_fit(ii) = cut(wfit, [-0.05 0.05]+ql(ii), []);

acolor(colors(ii)); pm(cgo\_1d\_sub(ii)); pl(cgo\_1d\_fit(ii));

end

ly(0,500); keep\_figure;

%% We see the simulation is ok, but strongly underestimates the widths of the peaks

% This is due to the instrument resolution. Now we include it and simulate the data

sample = IX\_sample(true,[1,1,0],[0,0,1],'cuboid',[0.01,0.03,0.01]);

ei = 50;

freq = 250;

chopper = 'G';

instru = merlin\_instrument(ei, freq, chopper);

% Set up the fit object - note we have to use a sqw object(s) not dnd

cuts\_for\_fit = set\_sample(cgo\_1d, sample);

cuts\_for\_fit = set\_instrument(cuts\_for\_fit, instru);

tf = tobyfit(cuts\_for\_fit);

tf = tf.set\_fun(@Bethe\_Ansatz\_CuGeO3, fitdat.p); % Use the fitted parameters from above

cgo\_sim = tf.simulate();

% Plot it with the data

for ii = 1:numel(ql)

acolor(colors(ii)); pm(cgo\_1d\_sub(ii)); pl(cgo\_sim(ii));

end

ly(0,500); keep\_figure;

%% We can see that a lot of the width is accounted for by the resolution

% In order to do a proper fitting we have to use a background function

% Because we cannot use the replicated-subtracted d1d objects in the fit

% (the resolution calculation needs the pixel data).

% This is a bit complicated because the background is not so well defined.

% For a start we can just truncate the data to remove the elastic line,

% and then use a linear background:

for ii = 1:numel(ql)

cgo\_1d\_truncated(ii) = cut(cgo\_cut, [-0.05 0.05]+ql(ii), [6,0.3,40]);

end

cuts\_for\_fit = set\_sample(cgo\_1d\_truncated, sample);

cuts\_for\_fit = set\_instrument(cuts\_for\_fit, instru);

tf = tobyfit (cuts\_for\_fit);

tf = tf.set\_fun (@Bethe\_Ansatz\_CuGeO3, fitdat.p);

tf = tf.set\_free ([1,1]);

tf = tf.set\_bfun (@linear\_bg, [50 0]);

tf = tf.set\_bfree ([1,1]);

tf = tf.set\_options ('list',2);

% Run the fit and plot it.

% Note this fit can be unstable and crash...

[cgo\_resfit, cgo\_resfit\_dat] = tf.fit('comp')

for ii = 1:numel(ql)

acolor(colors(ii)); pm(cgo\_1d\_truncated(ii)); pl(cgo\_resfit.sum(ii));

end

ly(0,500); keep\_figure;

% This is a bit better but the fit is not varying the J parameter

% which we're interested in much - as there are too many background parameters

% The rest is left as an exercise for the reader...

% You can:

% \* Restrict the fit to a single 1D cut

% \* Use a different / more sophisticated background function

% (use "dir(fileparts(which('mgauss')))" to see what is available or write your own)

% \* Fix the background parameters

## References

[1] R.A. Ewings, A. Buts, M.D. Le, J. van Duijn, I. Bustinduy, T.G. Perring, *Horace: Software for the analysis of data from single crystal spectroscopy experiments at time-of-flight neutron instruments*, [Nucl. Instr. Methods Phys. Res. A, **834** (2016) 132](https://doi.org/10.1016/j.nima.2016.07.036).

[2] S. E. Nagler, D. A. Tennant, R. A. Cowley, T. G. Perring, and S. K. Satija, *Spin dynamics in the quantum antiferromagnetic chain compound KCuF3*, [Phys. Rev. B **44**, 12361 (1991)](https://doi.org/10.1103/PhysRevB.44.12361).

[3] M. Arai, M. Fujita, M. Motokawa, J. Akimitsu, and S. M. Bennington, *Quantum Spin Excitations in the Spin-Peierls System CuGeO3*, [Phys. Rev. Lett. **77** 3649 (1996)](https://doi.org/10.1103/PhysRevLett.77.3649).