# Excitations Data Analysis Training Course – Solutions 4

## 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).

## Background subtraction.

In the GUI practical on Tuesday, we went through a process to take a high |Q| cut, tile it over a 2D slice and use it to subtract off a non-magnetic background. This process could be done with the following script:

% Recreate the Q-E slice from earlier

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

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

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

[-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], [0,4,360]);

plot(my\_slice)

keep\_figure;

lz(0,2)

% Make a 1D cut from the slice at high Q

my\_bg = cut(my\_slice, [1.9,2.1], []);

plot(my\_bg);

% Now tile it (note the conversion to dnd)

my\_bg\_rep = replicate(d1d(my\_bg), d2d(my\_slice));

plot(my\_bg\_rep)

lz 0 2

my\_slice\_subtracted = d2d(my\_slice) - my\_bg\_rep;

plot(my\_slice\_subtracted);

lz 0 2

1. Try repeating the script above with different Q positions for the cut (instead of [1.9,2.1]). In particular, what happens if the cut does not cover the full energy range of the slice?
2. Given this information, why do you think it was chosen to disallow the replicate function to operate on sqw objects (objects with pixel information)? (E.g. if you were to write a replicate function for sqw objects, how would *you* handle the pixels?)

Answer: Because the 1D cut was taken from a different ***Q***,*E* region, replicating it for pixel sqw data we cannot simply replicate the pixels, since the positions would be incorrect. Adding new pixels which would correspond to the replicated bins would essentially be making up data.

## Symmetrisation

We will focus on symmetrisation of sqw objects here. There are symmetrisation methods for d1d and d2d objects, but these are rather slow and their use should be avoided unless strictly necessary. In an extension exercise we will explore the possibility of symmetrising the entire dataset.

In general it is better to use sqw data for symmetrisation, though one should be cautious if the data you wish to symmetrise takes up a lot of memory.

1. Make a constant energy slice with Q axes from -3 to 3 in steps of 0.05 along the (1,1,0) and (-1,1,0) axes, and integration -0.1 to 0.1 along L, and 100 to 120 meV in energy. Plot it to confirm that you see some rings of magnetic scattering. Make sure that the object you created was of sqw type.
2. We will now use symmetrise\_sqw to apply a symmetry operation to this dataset. We want to fold along a vertical line centred at x=0. The way we do this with this command is to figure out what is the reflection plane to apply to the data. We specify two vectors that lie in this plane, and then a third vector that specifies the plane’s offset from the origin. In our case what are these three vectors?
3. Once you have worked out what the vector are, run the symmetrise\_sqw command, and plot the result to check it did what you wanted.
4. Now work out how to specify two planes that run along the diagonals of the plot axes. Apply these two symmetrisation operations to the data (i.e. fold along one diagonal, then the other). You should have a wedge of data. If you find, after plotting, that the wedge is on the negative Q side then try altering the order of the two vectors that define the reflection plane [the reason for this is that the reflection plane is dealt with in the internals of the Horace code by its surface normal, which is determined from the cross product of the two vectors in the plane, and (**a** ⨯ **b**) = -(**b** ⨯ **a**).
5. [Optional extension] Let’s try to do some much more complicated origami on the data! Make a volume cut, using the same Q axes and energy integration as before, but make a plot axis going from -2 to 2 in steps of 0.05. Plot this to see where all the rings of magnetic scattering are. See if you can figure out a way of doing multiple reflections that get all of the rings to overlap onto one.

Solution script:

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

% Symmetrisation

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

my\_slice2 = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-3,0.05,3], [-0.1,0.1], [100,120]);

plot(my\_slice2);

% Fold along vertical:

my\_sym = symmetrise\_sqw(my\_slice2, [-1,1,0], [0,0,1], [0,0,0]);

plot(my\_sym);

% Two folds along diagonals

my\_sym2 = symmetrise\_sqw(my\_slice2, [1,0,0], [0,0,1], [0,0,0]);

my\_sym2 = symmetrise\_sqw(my\_sym2, [0,1,0], [0,0,1], [0,0,0]);

plot(my\_sym2);

% Some origami!

my\_slice3 = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-3,0.05,3], [-2,0.05,2], [100,120]);

plot(my\_slice3)

sym1 = symmetrise\_sqw(my\_slice3, [0,1,0], [1,0,0], [0,0,0]);

plot(sym1);

sym2 = symmetrise\_sqw(sym1, [1,0,0], [0,0,1], [0,0,0]);

sym2 = symmetrise\_sqw(sym2, [0,1,0], [0,0,1], [0,0,0]);

plot(sym2)

% Squeeze out all the dead volume

plot(compact(sym2))

### Whole dataset symmetrisation (optional extra)

The above procedure using symmetrise\_sqw only applies to sqw objects in memory. Horace currently cannot apply this to a sqw file – although such file-backed operations are currently being worked on and will be part of Horace 4.0. At the moment, to symmetrise a whole dataset, you have to do this when the sqw file is generated using gen\_sqw. The gen\_sqw function takes an option called transform\_sqw which is a user defined function which maps an input sqw to an output. We can use symmetrise\_sqw (or a series of calls to symmetrise\_sqw) as this function. For example define a function in a file called my\_sym.m:

function wout = my\_sym(win)

% Fold above the line [1,0,0] in the H-K plane

wout = symmetrise\_sqw(win, [1,0,0], [0,1,0], [0,0,0]);

end

Then you can call gen\_sqw with:

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

u, v, psi, omega, dpsi, gl, gs,'transform\_sqw', @my\_sym)

Use this method to create a symmetrised iron.sqw file with the symmetry operations you worked out in steps 2-4 above.

Make a cut from this symmetrised sqw file and check it looks like you expect (it looks like the symmetrised cuts from memory done above).

## Data Diagnostics (Spurions)

If you were unlucky there may be a problem with one or more of the runs that were combined to make your sqw file. For example, the sample might not have moved to the correct psi, or the detectors failed, etc. Usually the signature of something like this is a plot that looks strange. To diagnose what is wrong you can use the tool run\_inspector. This allows you to look at 1d or 2d cuts (sqw objects only) decomposed into the data from individual runs. You can then determine if the signal from just one run is anomalous far more easily.

The bcc-iron dataset doesn’t have a clear spurion so we will look at a couple of different small dataset instead: /home/dl11170/edatc/data/spurious1.sqw and /home/dl11170/edatc/data/spurious2.sqw. The first file is a 3D sqw object cut from a larger dataset with two ***Q***-axes: **u**=[100], **v**=[010] and the energy transfer axis (it was generated by integrating over **w**=[001]) and saved to a file. The second file is a 2D sqw object cut from a large dataset. You can load the files with:

w\_sp1 = sqw('/mnt/ceph/auxiliary/excitations/edatc/data/spurious1.sqw')

w\_sp2 = sqw('/mnt/ceph/auxiliary/excitations/edatc/data/spurious2.sqw')

Plot w\_sp1 and w\_sp2 and see if you can spot the spurious scattering.

Hint: w\_sp1 has some is anomalously intense scattering at low energies. For w\_sp2, think about the symmetry (you’re looking at the (111) plane of a cubic crystal).

Make a 2D slice (keeping pixels) of w\_sp1 which contains the spurious scattering, and use run\_inspector on it (try a constant energy slice, and a QE slice, remembering that they must be sqw objects) to examine the signal from the contributing runs.

Where do you think the spurious scattering comes from?

Answer: w\_sp1 contains detector “bleed” where very intense scattering from multiple Bragg reflections on a single tube causes the electronics to misrecord the position and times of the detected neutrons. The diagonal streak actually corresponds to a single vertical tube.

Plot w\_sp2 and use run\_inspector on it to try to determine what caused the spurious scattering here.

Answer: In w\_sp2, the sample was stuck part-way through a set of rotation scans so towards the end of the scan, the actual sample rotation angle does not match the recorded angle.

Solution script:

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

% Spurious data

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

w\_sp1 = sqw('/mnt/ceph/auxiliary/excitations/edatc/data/spurious1.sqw')

cut1\_sp1 = cut(w\_sp1, [], [-0.1 0.1], []);

plot(cut1\_sp1)

% You should see an intense streak at the Bragg position.

% Lets look at a reciprocal space map of it

plot(cut(w\_sp1, [], [], [-2 2])); lz(0, 2000); keep\_figure;

plot(cut(w\_sp1, [], [], [8 12])); lz(0, 2000); keep\_figure;

% You should see that there are 3 streaks all in the same direction,

% all coming out of a Bragg peak.

run\_inspector(cut1\_sp1)

% Move through the runs – you should see around run 22 that there is a very

% intense diagonal streak which is present in several runs.

% The excitations are too intense and are not symmetric about the Bragg peak

% so they are not real dispersion, but because they are associated with the

% sample Bragg peak, it suggests they \_are\_ scattering from the sample.

% In fact they are a detector artefact. This happens because the crystal

% is aligned such that equivalent off-plane Bragg peaks hit a single detector

% tube at the same time causing the electronics to misrecord neutron events,

% because the peaks are so intense.

%% The second dataset

w\_sp2 = sqw('/mnt/ceph/auxiliary/excitations/edatc/data/spurious2.sqw')

plot(w\_sp2)

% You should see that there are Bragg peaks but they don’t seem to have the

% 6-fold symmetry you would expect from the (111) plane of a cubic crystal.

run\_inspector(w\_sp2, 'col', [0,1000])

% Move through the run\_inspector. You should see that the sqw file was formed

% of a set of 46 scans from 0 to 90 deg in 2 deg steps, and then another

% 45 scans from 1 to 89 deg in 2 deg steps.

% Comparing runs 22-27 and 69-74 (you can use run\_inspector twice to get 2 plots)

% you should see the scattering is similar but doesn't match up

% (e.g. run 22 looks like run 70 but are 5 degrees apart).

% This is because during the rotation from 90 deg to 1 deg for the second set

% of scans, the sample assembly became stuck and the motor lost its position

% So the second set was not actually measuring from 1 to 89 deg.

## Masking data

There are several different ways of masking out parts of a dataset. You may wish to do this, for example, to remove an obvious spurion from the data before you try to fit it. Read the manual for mask and mask\_points to understand the different ways of doing this: <https://pace-neutrons.github.io/horace-docs/3.5.0/Reshaping_etc.html#mask>

Using any 2D cut (from either the spurious dataset or the iron dataset):

1. Use the mask routine first. Create a mask array that is the same size as the intensity array in your Q-E slice that contains only ones. Apply this mask to the data. You should get back what you started with, because the ones everywhere mean you want to keep every bin.
2. Now mask out a large region of the data, by making the rows from half-way down the mask matrix contain zeros [hint: remember how we do logical indexing in Matlab]. Apply this new mask to the data, and see what it now looks like.
3. Now use mask\_points in its two different modes of operation. First keep all data within -1<Q<1 and 100<E<120, and plot the results. Then try the opposite, i.e. remove all data within the same limits, and plot the results.
4. Make a Q-E cut of the spurious dataset integrating between -0.6<Qh<-0.5 – you should see a spurious spot around Qk=0 and 18meV. Use either mask or mask\_points to mask this spurious spot.

Solution script:

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

% Masking data

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

% Mask parts of a dataset out, e.g. if there is a region with a spurion that

% you wish to remove before proceeding to fitting the data

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

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

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

mask\_arr = ones(size(my\_slice.data.npix)); % keeps everything

mask\_arr2 = mask\_arr;

mask\_arr2(61:121,:) = 0;

my\_slice\_masked1 = mask(my\_slice,mask\_arr); % should do nothing

my\_slice\_masked2 = mask(my\_slice,mask\_arr2);

plot(my\_slice\_masked1); keep\_figure;

plot(my\_slice\_masked2); keep\_figure;

% Mask out specific points, if the mask you need for the above is more

% complex:

sel1 = mask\_points(my\_slice, 'keep', [-1,1,100,120]); % specify limits to keep

sel2 = mask\_points(my\_slice, 'remove', [-1,1,100,120]); % specify limits to remove

my\_slice\_masked3 = mask(my\_slice, sel1);

my\_slice\_masked4 = mask(my\_slice, sel2);

plot(my\_slice\_masked3); keep\_figure;

plot(my\_slice\_masked4); keep\_figure;

%% Masking spurious data

cut\_sp1 = cut(w\_sp1, [-0.6 -0.5], [], [])

plot(cut\_sp1); keep\_figure

% Determine the mask – best way is to plot the actual picture with pcolor

%figure; pcolor(w\_sp1.data.s); caxis([0,1000])

% Then determine the coordinates from this (remember that pcolor transposes the matrix)

mask\_arr\_sp = ones(size(cut\_sp1.data.npix));

mask\_arr\_sp(40:42, 57:59) = 0;

wmasked = mask(cut\_sp1, mask\_arr\_sp)

plot(wmasked);

## Miscellaneous functions

Below are a list of miscellaneous functions which might be useful in future. You can always type help <function\_name> in the Matlab command window to get help on them.

|  |  |
| --- | --- |
| Bose | This function converts a sqw or dnd object from S(***Q***,ω) to χ’’(***Q***,ω) by multiplying by the factor (1 – exp(-E/kT)). |
| Signal | Creates a new sqw (not dnd) object from an existing sqw object but with the signal (intensity) set to the value of a given coordinate. For example to multiply a workspace by the Q-magnitude, use: w1 \* signal(w1, 'Q'). Instead of 'Q', 'E', 'h', 'k', or 'l' can be used. |
| Section | Extracts a portion of the rebinned data of a dnd **or** sqw object and returns another object – this is much faster than doing a cut, but does not rebin the data (bin sizes stay the same). E.g. ws = section(w1, [0, 2.5], [100, 250]) will extract the segment between 0 and 2.5 along axis 1 and 100 and 250 along axis 2 of object w1. |
| Split | Splits a sqw object (not dnd) into an array of sqw each corresponding to a single rotation angle scan (a single input spe or nxspe datafile). This is useful if there was an error or spurion in a particular run which should be removed. See also run\_inspector. |

Solution script:

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

% Rescaling data

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

% Bose correction function.

% NB it does not do much at high energies, or course!

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

plot(my\_slice);

lz 0 2

keep\_figure;

my\_slice\_bose = bose(my\_slice, 300); % pretend the data was taken at 300K...

plot(my\_slice\_bose); % you can still see what this does

lz 0 2

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

% Miscellaneous

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

% If you want to see how a certain parameter varies across a dataset:

w\_sig = signal(my\_slice, 'Q'); % mod Q in this case

plot(w\_sig)

% You can use this now to apply a scale factor to the data. Suppose you wish

% to multiply signal by energy:

w\_sig = signal(my\_slice, 'E');

my\_slice2 = my\_slice \* w\_sig;

plot(my\_slice2)

lz 0 100

% Take a section out of a dataset:

w\_sec = section(my\_slice, [0, 2.5], [100, 250]); % just 0 to 2.5 in Q, 100 to 250 in energy

plot(w\_sec);

% Split a dataset up into its contributing runs

w\_split = split(my\_slice);

% w\_split is an array of objects (recall indexing of arrays in Matlab)

% each element of the array corresponds to the data from a single

% contributing spe file

plot(w\_split(1)); keep\_figure;

plot(w\_split(10)); % etc.

% Allows you to determine if a spurious or strange signal is coming from a

% single run, or if it is from a collection of runs.

## Model Simulation

We will use the slice and cuts previously made:

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

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

% Usual 2D slice

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

[-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], [0,4,360]);

% New array of 1D constant energy cuts along [110]

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

### Simulating data using a template function

We have created a template S(Q,w) function that we will use for our first go at simulating. The function is called /home/dl11170/edatc/matlab/sr122\_xsec.m. This particular cross-section has nothing in particular to do with the iron dataset we are using, so do not expect the simulations to look much like the data at this stage!

1. To simulate a dataset with an S(Q,w) model, use the routine sqw\_eval. ([http://horace.isis.rl.ac.uk/Simulation#sqw\_eval](http://horace.isis.rl.ac.uk/Simulation%23sqw_eval) for details). In this case the input parameters should be the following vector: [1, 0, 0, 35, -5, 15, 10, 0.1]. Run the routine on the slice and cuts, for both sqw and dnd objects. You should notice when you plot the results that the sqw and dnd simulations of the same thing do not look exactly the same. Think about what this might be.

[hint: what extra information is there in an sqw object?]

Solution script:

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

% Simulation and Fitting

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

clear variables

add\_path('/mnt/ceph/auxiliary/excitations/edatc/scripts');

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

% Simulating a pre-prepared S(Q,w) function

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

% 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

% Simulate on sqw objects

parameter\_vector = [1,0,0,35,-5,15,10,0.1];

sim\_slice = sqw\_eval(my\_slice, @sr122\_xsec, parameter\_vector);

sim\_cut = sqw\_eval(my\_cuts, @sr122\_xsec, parameter\_vector);

% Repeat on dnd objects

sim\_slice\_dnd = sqw\_eval(d2d(my\_slice), @sr122\_xsec, parameter\_vector);

sim\_cut\_dnd = sqw\_eval(d1d(my\_cuts), @sr122\_xsec, parameter\_vector);

plot(sim\_slice); keep\_figure;

plot(sim\_slice\_dnd); keep\_figure;

acolor blue

dl(sim\_cut(1));

acolor red

pl(sim\_cut\_dnd(1));

keep\_figure;

% Note the differences between simulations of notionally the same data.

% This is because dnd just takes the centre point of the integration range,

% whereas sqw takes all of the contributing detector pixels. This is

% imperative if the dispersion varies significantly in a direction

% perpendicular to your cut/slice, as it introduces broadening that the dnd

% simulation fails to capture.

### Simulating peaks

Horace (in fact the Herbert sub-libraries) contain a generic 1d peak function called mgauss, whose input parameters specify the height, centre and width of an arbitrary number of Gaussians. i.e. the input parameters are [height1,centre1,width1,height2,centre2,width2,…., heightN,centreN,widthN]. Horace can evaluate / simulate either S(Q,w) models, which require h,k,l,e plus parameters as inputs, or dimension-specific models, e.g. 1d Gaussians, 2d Lorentzians, etc.

1. Use func\_eval (<https://pace-neutrons.github.io/horace-docs/3.5.0/Simulation.html#func-eval>) to simulate the mgauss function on one of the 1d cuts. Choose the input parameters to put peaks in the places where they exist in the data.
2. Plot the 1d cut, and then overplot the simulation in a different colour by using acolor, and then using the pl command.

You can see a list of the other functions available in Horace and Herbert by finding the folder in which mgauss is kept (type “which mgauss”) and then using dir [e.g. dir(fileparts(which('mgauss')))]to find out what else in the folder. There are quite a few!

Solution script:

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

% Simulate a peak function with a cut

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

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];

peak\_cut = func\_eval(my\_cuts(1), @mgauss, pars\_in);

acolor black

plot(my\_cuts(1))

acolor b

pl(peak\_cut);

### Making dispersion plots

Horace allows you to make dispersion relation plots (including spectral weight) of your model. These can be useful during both initial planning, interpretation, and analysis of results. The model function to calculate the dispersion is similar to the S(Q,w) model – it accepts h, k, l and input parameters, but not energy. It outputs spectral weight, and the energy of the dispersion at the input Q. The plotting function disp2sqw\_plot , its syntax, and the form of the disperslation relation function that should be passed to it is described here: <https://pace-neutrons.github.io/horace-docs/3.5.0/Simulation.html#disp2sqw-plot>.

1. Use the lattice parameters (including angles) that we have been using all along, and make a dispersion plot for the following trajectory – (0,0,0) --> (0,0,1) --> (0,0,0) --> (1,0,0) --> (0,0,0) --> (1,1,0) --> (0,0,0) --> (1,1,1). Use the function sr122\_disp, with input parameters [1,0,0,35,-5,15,10,0.1]. An appropriate energy range over which to make the plot is 0 to 200 meV in 0.1 meV steps.

If you just want to plot the dispersion relation, but not worry about colour-coding the spectral weight, you can use the closely related function dispersion\_plot, which will be much faster. The syntax is almost identical and is described here: <https://pace-neutrons.github.io/horace-docs/3.5.0/Simulation.html#dispersion-plot>

Solution script:

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

% Make dispersion plots

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

alatt = [2.87, 2.87, 2.87];

angdeg = [90,90,90];

lattice = [alatt, angdeg];

% Reciprocal lattice points to draw dispersion between:

rlp = [0,0,0; 0,0,1; 0,0,0; 1,0,0; 0,0,0; 1,1,0; 0,0,0; 1,1,1];

% Input parameters

pars = [1, 0.05, 0.05, 35, -5, 15, 10, 0.1];

% Energy grid

ecent = [0,0.1,200];

% Energy broadening term

fwhh = 2;

disp2sqw\_plot(lattice, rlp, @sr122\_disp, pars, ecent, fwhh);

## 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).