Excitations Data Analysis Training Course – Worksheet 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 $S(\boldsymbol{Q},\omega)$ 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].

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 = '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?)

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.

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('/home/dl11170/edatc/data/spurious1.sqw')
w_sp2 = sqw('/home/dl11170/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?

Plot w_sp2 and use run_inspector on it to try to determine what caused the spurious scattering here.

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: http://horace.isis.rl.ac.uk/Reshaping_etc#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.

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`.

Simulation and Fitting I

We will use the slice and cuts previously made:

```
sqw_file = '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/scripts/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 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?]

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.

2. Use `func_eval` (http://horace.isis.rl.ac.uk/Simulation#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.

3. 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!

Fitting a single 1d cut with some peaks

We will now try to fit the `mgauss` function to the 1d cut you used above. An introduction of how to do this is documented here: http://horace.isis.rl.ac.uk/Fitting#fit_func. 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.

4. Subtract a scalar from your 1d cut, so that the background level is approximately zero (we will deal with accounting for background later).

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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.

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: http://horace.isis.rl.ac.uk/Simulation#disp2sqw_plot.

12. 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:
http://horace.isis.rl.ac.uk/Simulation#dispersion_plot.

References

[1] R.A. Ewings, A. Buts, M.D. Lee, 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.