# Excitations Data Analysis Training Course – Solutions 3

## 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).

## Horace Commandline

Yesterday we covered the GUI tools available with Horace – the Horace Planner and the Horace GUI. However, Horace is mainly a commandline (CLI) driven program, so to do more than the basics you will need to use the Matlab CLI. There is a hint-sheet / aide memoire if you’re not familiar with Matlab.

## Making a “fake” dataset

Yesterday we covered using the Horace Planner to determine the run parameters for an experiment. The Horace Planner is good for systems with orthogonal axes but is harder to use for lower symmetry crystals. Also, it may not fully show the full extent of the (***Q***,ω) coverage you can get for a given **Ei** and **psi** range. For these details, you can generate a fake dataset which mimics the dataset you would generate in your experiment. You can take cuts and slices from it just as you could the real data, so you can explore not just whether a particular reciprocal lattice point is covered, but also what is going on along off-symmetry directions nearby. The intensity scale is given by the average value of psi that contributed to a given bin.

The disadvantage of this approach is that if you wish to change the values of Ei and/or range of psi for your fake dataset, you have to spend a few minutes regenerating it. The advantage is that you get much more information about what you will measure in your experiment. Also, as described above, these routines are essential for planning measurement of systems where the crystallographic axes are non-orthogonal.

The following script will generate a fake dataset similar to the one used in the Horace Planner exercise yesterday:

% Name and folder for output "fake" generated file

sqw\_file = 'my\_fake\_file.sqw';

% Instrument parameter file (may be in another location to this)

par\_file = '4to1\_102.par';

% u and v vectors to define the crystal orientation

% u||ki, uv plane is horizontal but v does not need to be perp to u

u = [1, 0, 0];

v = [0, 1, 0];

% Range of rotation (psi) angles to cover in simulated dataset.

% (psi=0 when u||ki)

psi = [0:5:90];

% Incident energy in meV

efix = 401;

emode = 1; % This is for direct geometry (set to 2 for indirect)

% Range of energy transfer (in meV) for the dataset to cover

en = [0:5:360];

% Sample lattice parameters (in Angstrom) and angles (in degrees)

alatt = [2.87, 2.87, 2.87];

angdeg = [90, 90, 90];

% Sample misalignment angles ("gonios"). [More details tomorrow].

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

% This runs the command to generate the "fake" dataset.

fake\_sqw (en, par\_file, sqw\_file, efix, emode, alatt, angdeg,...

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

You can copy and paste the above code into a Matlab script window (if one is not open, right-click on the left panel and select New → Script and then double click on the file created; or type edit myscript.m in the Matlab command window) and the press Ctrl+Enter to run it.

Once the fake dataset has been created, you can use the cut\_sqw command to make cuts from it. This is exactly the same as with the Horace GUI yesterday. An example script to make the same 3D cut as the first example yesterday is:

% First we define the projection axes (the u, v and w vectors)

% Type is Q units for each axis and can be either 'r' for r.l.u. or

% 'a' for absolute (A^-1). E.g. 'rar' means u and w are

% normalissed to in r.l.u, v in A^-1.

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

% Now make a cut of the fake dataset.

% The four vectors indicate either the range and step (3-vector) or

% the integration range (2-vector), with units defined proj.type

% The following makes a 3D cut with axes u, v and energy

% (first, second and fourth vectors are 3-vectors),

% integrating over w between -0.1 and 0.1.

% '-nopix' indicates to discard the pixel information and create

% a dnd (d3d) object.

my\_cut = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-3,0.05,3], ...

[-0.1,0.1], [0,4,360], '-nopix');

% Now plot the 3D cut.

plot(my\_cut);

Make a **2D** Q-energy slice with a Q-axis of (h,0,0) around the (2,0,0) position, and repeat along other equivalent positions such as (0,2,0), (0,0,2), etc.

Hint: You don't have to use the projection in the blue example script - you can pick a different projection to make things easier...

Hint: projaxes accepts a keyword argument 'uoffset' which changes the centre of the cuts (check the help).

Which of the equivalent directions would be best to use here?

Tip: to keep multiple plots on the screen, use the ‘keep’ menu item on a plot.

Answer: (200) – the [H00] direction has the largest coverage.

Now try to make a 2D Q-energy slice with a Q-axis of (h,h,h) around the (2,0,0) position.

Hint: Use a different projection, and the **u** and **v** vectors do not have to be orthogonal.

Solution script:

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

% Make a fake data set to explore more thoroughly

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

% Name and folder for output "fake" generated file

sqw\_file = [pwd '/../aaa\_my\_work/my\_fake\_file.sqw'];

% Instrument parameter file (may be in another location to this)

par\_file = [pwd '/../data /4to1\_102.par'];

% u and v vectors to define the crystal orientation

% (u||ki, uv plane is horizontal but v does not need to be perp to u.

u = [1, 0, 0];

v = [0, 1, 0];

% Range of rotation (psi) angles to cover in simulated dataset.

% (psi=0 when u||ki)

psi = [0:5:90];

% Incident energy in meV

efix = 401;

emode = 1; % This is for direct geometry (set to 2 for indirect)

% Range of energy transfer (in meV) for the dataset to cover

en = [0:4:360];

% Sample lattice parameters (in Angstrom) and angles (in degrees)

alatt = [2.87, 2.87, 2.87];

angdeg = [90, 90, 90];

% Sample misalignment angles ("gonios"). [More details in session 4].

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

% This runs the command to generate the "fake" dataset.

fake\_sqw (en, par\_file, sqw\_file, efix, emode, alatt, angdeg,...

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

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

% Once generated, you can use standard Horace plotting tools to explore

% this fake dataset, where the colour scale corresponds to the value of psi

% that contributed data to a given region of reciprocal space

% First define a view projection (these u and v do not need to be the same

% as the sample u and v above. They just define the first, second and third

% axes for making a cut (third axis w is implicit being perpendicular to the

% plane defined by u and v).

proj.u = [-1, -1, 1];

proj.v = [0, 1, 1];

% The 4th offset coordinate is energy transfer

proj.uoffset = [0, 0, 0, 0];

% Type is Q units for each axis and can be either 'r' for r.l.u. or 'a'

% for absolute (A^-1). E.g. 'rar' means u and w are normalissed to in r.l.u, v in A^-1.

proj.type = 'rrr';

% Actually, it is better to make a projection object with this information

% rather than a structure. Type: >> doc projaxes for more details.

% Note that the default for uoffset is [0,0,0,0] so it doesn't need to be set

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

% Now make a cut of the fake dataset.

% The four vectors indicate either the range and step (three-vector) or

% the integration range (2-vector), with units defined by the proj.type

% The following makes a 3D volume cut with axes u, v and energy

% (first, second and fourth vectors are 3-vectors),

% integrating over w between -0.1 and 0.1.

% '-nopix' indicates to discard the pixel information and create

% a dnd (d3d) object.

my\_vol = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-3,0.05,3], ...

[-0.1,0.1], [0,4,360], '-nopix');

% Plot the 3D cut - click on the graph to plot 2D projections of the volume

plot(my\_vol)

% The following makes a 2D cut with axes u and energy (first and fourth

% vectors are 3-vectors), integrating over v and w between -0.1 and 0.1

my\_cut = cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1], [-0.1,0.1], [0,10,400], '-nopix');

% Now plot the 2D cut.

plot(my\_cut);

% We set the offset to be centred on (200), (020) and (002) in turn

% Plotting the dispersion along the [h00] direction (note different u and v)

% Using keep\_figure to keep the figures on screen.

% Afterwards, you can check which figure gives the largest coverage.

proj = projaxes([1,0,0], [0,1,0], 'uoffset', [2,0,0,0], 'type', 'rrr');

plot(cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1], [-0.1,0.1], [0,4,360], '-nopix'));

keep\_figure;

proj = projaxes([1,0,0], [0,1,0], 'uoffset', [0,2,0,0], 'type', 'rrr');

plot(cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1], [-0.1,0.1], [0,4,360], '-nopix'));

keep\_figure;

proj = projaxes([1,0,0], [0,1,0], 'uoffset', [0,0,2,0], 'type', 'rrr');

plot(cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1], [-0.1,0.1], [0,4,360], '-nopix'));

keep\_figure;

% As an alternative, we could manually change the integration range in the relevant

% axes instead, but using uoffset is easier:

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

% w200 = cut\_sqw(sqw\_file, proj, [-1,0.05,1]+2, [-0.1,0.1], [-0.1,0.1], [0,4,360], '-nopix')

% w020 = cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1]+2, [-0.1,0.1], [0,4,360], '-nopix')

% w002 = cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1], [-0.1,0.1]+2, [0,4,360], '-nopix')

% Use a different projection to make a 2D slice along [hhh] centred at (200)

proj = projaxes([1,1,1], [0,1,0], 'uoffset', [2,0,0,0], 'type', 'rrr');

plot(cut\_sqw(sqw\_file, proj, [-1,0.05,1], [-0.1,0.1], [-0.1,0.1], [0,4,360], '-nopix'));

## Generating SQW files

Yesterday, we saw how to generate a 4D S(***Q***,ω) dataset using the GUI – now we will look at how to do it using the CLI. The command is gen\_sqw and it has a very similar syntax to the fake\_sqw example above.

Use help gen\_sqw and the above example to write a script to create the iron.sqw file from the nxspe files in the /home/dl11170/edatc/data folder.

Hint: You need to generate a cell array of file names. You can use a cellfun and an anonymous function to do it using:

runs = 15052:15097;

data\_path = '/home/dl11170/edatc/data';

spe\_file = cellfun(@(c) fullfile(data\_path, ...

['map' num2str(c) '\_ei400.nxspe']), num2cell(runs), ...

'UniformOutput', false)

Alternatively, you could also use a loop to do the same thing.

The other parameters are the same as we used yesterday in the GUI:

**u** – 1 0 0

**v** – 0 1 0

**psi** – 0 to 90 in 2 degree steps

**lattice pars** – 2.87 2.87 2.87

**lattice angles** – 90 90 90

**Ei** – 401

**omega**, **dpsi**, **gl**, **gs** – all zero

To get help, type help gen\_sqw or doc gen\_sqw in Matlab. This will provide details of the syntax of the command that we will use to combine the files. Alternatively look on the Horace website: [http://horace.isis.rl.ac.uk/Getting\_started#Creating\_an\_SQW\_file](http://horace.isis.rl.ac.uk/Getting_started%23Creating_an_SQW_file%20)

Create variables that will be used as inputs for this function – e.g. ei=401. (note the variable emode=1 should be used – this specifies that the instrument (MAPS in this case) is a direct geometry machine). Run the command, and you should get an sqw file within a couple of minutes.

### Note for experts

During an experiment the number of data files will increase as time goes by. To avoid having to regenerate the entire sqw file each time you wish to look at the latest data, you can use the command accumulate\_sqw. The inputs to this are essentially the same as for gen\_sqw, but instead of a list of measured psi and spe files, you provide a list of psi that you plan to measure and a list of spe files that will eventually exist. The routine will then check which files do actually exist, and make an sqw file out of those. If you then run the routine again later only the new data will be processed, and then appended to the existing sqw file, saving time.

Solution script:

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

% Generating an sqw file

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

clear variables

close all

% Directory where data (spe or nxspe) files are:

data\_path = [pwd '/../data/'];

% Name of output sqw file (for the 4D combined dataset)

sqw\_file = [pwd '/../aaa\_my\_work/iron.sqw'];

% Instrument parameter file name (only needed for spe files - nxspe files

% have the par file embedded in them).

%par\_file = [data\_path, '4to1\_102.par'];

par\_file = '';

% u and v vectors to define the crystal orientation

% (u||ki when psi=0; uv plane is horizontal but v does not need to be perp to u).

u = [1, 0, 0];

v = [0, 1, 0];

% Range of rotation (psi) angles of the data files.

% (psi=0 when u||ki)

psi = [0:2:90];

% Data file run number corresponding to the psi angles declared above

% (must be the same size and order as psi)

runno = [15052:15097];

% Incident energy in meV

efix = 401;

emode = 1; % This is for direct geometry (set to 2 for indirect)

% Sample lattice parameters (in Angstrom) and angles (in degrees)

alatt = [2.87, 2.87, 2.87];

angdeg = [90, 90, 90];

% Sample misalignment angles ("gonios"). [More details in session 4].

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

% Construct the data file names from the run numbers (the data file names

% are actually what is required by the gen\_sqw function below, but we

% use the numbers as a convenience. This assumes that the data file names

% follow the standard convention of IIInnnnnn\_eiEEE.nxspe, where III is

% the instrument abbreviation (MAP, MER or LET), nnnnnn is the run number

% and EEE is the incident energy.

spefile = cellfun(@(c) fullfile(data\_path, ['map' num2str(c) '\_ei400.nxspe']), ...

num2cell(runno), 'UniformOutput', false);

% A loop also works:

%for i=1:numel(psi)

% spefile{i} = [data\_path, 'map', num2str(runno(i)), '\_ei400', '.nxspe'];

%end

% Now run the function to generate the sqw file.

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

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

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

% Accumulating data to an existing sqw file

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

%

% The above gen\_sqw file generates an sqw file from the list of input

% spe or nxspe files in one go, and deletes all temporary files after it

% finishes. If you are in the middle of a rotation scan, you can use

% accumulate\_sqw which does not delete the temporary files and so can

% append newly processed spe/nxspe files to an existing sqw file.

% This may save some time in processing, but is not now generally

% recommended since the implementation of parallelisation in gen\_sqw has

% made gen\_sqw much faster.

%

% This is because accumulate\_sqw needs to know \_all\_ the psi values

% (including those not yet measured) in order to construct coarse data

% grid that enables Horace to make fast cuts. If you then include

% measurements at psi values not in the original list, then it is possible

% that some data will lie outside this grid and it will be 'lost' to the

% sqw file. If the additional runs are ones that interleave between the

% original files, this will not be a problem, but if the additional runs

% extend the original angular range, then you must use the 'clean' option

% which is equivalent to gen\_sqw.

%

% The syntax for accumulate\_sqw is very similar to gen\_sqw:

%

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

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

%

% Or:

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

% u, v, psi, omega, dpsi, gl, gs, 'clean')

%

% This is a way of appending newly processed spe files to an existing

% dataset. The key point is that the psi and spe\_file arrays contain a list

% of PLANNED files and run-numbers - only those that actually exist will be

% included in the file.

%

% You can run this periodically, for example overnight.

## Making cuts and slices and then plotting them

Before we make a cut or slice, we need to define what will be our viewing axes. These do not have to be the same as (or even related to!) the **u** and **v** vectors that defined the scattering plane. We make this definition by creating a object called proj with properties u, v, uoffset, type, and nonorthogonal.

The first two define a set of viewing axes in Q (3-element vectors) – the 3rd viewing axis is the cross-product of u and v. uoffset is a 4-element vector that defines an offset for the viewing axes, and type is a string that defines whether you wish to use reciprocal lattice units or inverse angstroms for each of the axes. The option nonorthogonal is a special option, whose use will be explained in an extension exercise. You can ignore it for the moment as by default it will be assumed to be false. (uoffset is also assumed to be [0 0 0 0] so it could also be ignored).

Let us define a structure proj1 as follows:

u = [1, 0, 0];

v = [0, 1, 0];

proj1 = projaxes(u, v, 'uoffset', [0, 0, 0, 0], 'type' = 'rrr');

You can get information on the project axis class using help projaxes or doc projaxes.

1. Create another proj structure array, with zero offset and reciprocal lattice units, that has (1,1,0) and (-1,1,0) as the viewing axes.
2. Now make a volume cut using cut\_sqw (see [this web page](http://horace.isis.rl.ac.uk/Manipulating_and_extracting_data_from_SQW_files_and_objects#cut_sqw) for the command syntax or type help cut\_sqw) using the proj you just created. The range of Q along (1,1,0) should be -3 to 3 in steps of 0.05 (i.e. [-3,0.05,3]). Along (-1,1,0) the Q range should be the same, and along L we wish to integrate between -0.1 and 0.1. The energy axis should cover the range 0 to 360 meV in 4 meV steps. Select the option '-nopix' for now (this discards the pixel information and reduces memory but means you cannot re-cut from this cut).
3. Once you have created the volume, plot it using the [plot command](http://horace.isis.rl.ac.uk/Plotting#plot). This will pull up the sliceomatic window, which you can use to view orthogonal slices through the data: click on one of the slider bars to get a plane, and then drag the arrow up and down the slider bar.
4. Make a 2d slice from the data file using the same proj that has the same range and step sizes along (1,1,0) and energy. Integrate L between -0.1 and 0.1 again, and integrate along (-1,1,0) between -1.1 and -0.9. This time do not use the '-nopix' option. Plot the result.
5. Now make a 1d cut from the data file using the same proj. Keep the same range and step as before along (1,1,0), and integrate energy between 130 and 150 meV. Keep the other integrations the same as for your 2d slice. Plot the result once again.
6. Re-run the command above, this time timing it using the Matlab tic and toc commands i.e.

tic;

<Horace commands>;

toc

1. Now try taking a cut from a cut, i.e. rather than taking a cut from the file you can extract data if they already form part of another slice or cut. See [this page](http://horace.isis.rl.ac.uk/Manipulating_and_extracting_data_from_SQW_files_and_objects#cut) for details of the syntax. In our case we want to take a cut from the slice you took in the exercise 4, keeping the binning the same along the first axis and integrating from 130 to 150 along the second axis.
2. Repeat the above using the tic; <command>; toc syntax as in exercise 6. Plot this cut and compare it to that which you generated in exercise 6. Are they the same? How long did it take to execute the command in exercise 6 compared to this exercise?

Answer: The plots should be identical. Running the cut from an existing cut should be much faster because the second cut (step 7) is done in memory (I get 0.05s for the in-memory cut and 0.5s for the cut from file).

Solution script:

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

% Making cuts and slices

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

% Before making a cut, we have to define viewing (projection) axes, and

% these u and v do not need to be the same as the sample orientation which

% is defined by u and v above (where u||ki at psi=0).

% These u and v just define the Q axes for cut\_sqw. Generally you only need

% to define the first two axes, u and v. The third axis w is implicitly

% constructed as being perpendicular to the plane defined by u and v.

% The units of the Q axes are specified by the 'type', which can be 'r'

% for r.l.u. or 'a' for absolute units (A^-1).

% E.g. 'rar' means u and w are in r.l.u, v in A^-1.

% The offset gives a offset for the zero of that axis, with the fourth

% coordinate being the energy transfer in meV.

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

% The syntax for cut\_sqw is:

%

% cut = cut\_sqw(sqw\_file, proj, u\_axis\_limits, v\_axis\_limits, w\_axis\_limits, ...

% en\_axis\_limits, keywords)

%

% The \*\_axis\_limits are either:

% 1. a single number, [0.05], which means that this axis will be plotted

% with the number being the bin size and limits being the limits of

% the data.

% 2. two numbers, [-1, 1], which means that this axis will be integrated

% over between the specified limits.

% 3. three numbers, [-1, 0.05, 1], which means that this axis will be

% plotted between the first value to the last value with the bin size

% specified by the middle value.

% In the following we make 3d volume plots along u, v and energy and

% integrating over the w direction. The '-nopix' at the end means that

% cut\_sqw will discard all pixel information - that is it will only retain

% the counts and errors for each bin rather than keep the counts of each

% neutron event which is enclosed by each bin. This saves a lot of memory

% and is good enough for plotting but would not be good enough for fitting,

% or for re-cutting as shown below.

my\_vol = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-3,0.05,3], [-0.1,0.1], [0,4,360], '-nopix');

plot(my\_vol);

% Now we make 2D slices integrating over both v and w in Q.

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);

% Now we make a 1D cut along u, timing how long it takes.

tic

my\_cut = cut\_sqw(sqw\_file, proj, [-3,0.05,3], [-1.1,-0.9], [-0.1,0.1], [130,150]);

toc

plot(my\_cut);

% In addition to cutting from an sqw file, you can also cut from a previous

% cut. Note that if the previous cut had used '-nopix', the cut bins must

% be aligned with the old cut. Thus if you want to re-cut a cut, do not

% use '-nopix'.

tic

my\_cut2 = cut(my\_slice, [], [130,150]);

% The [] above is to keep 1st axes as it is, [130,150] is integration range

% for 2nd axis note because the object we are cutting from is 2d, we only

% need 2 binning arguments, rather than the 4 that are needed when taking a

% cut form the 4-dimensional dataset in the file

toc

plot(my\_cut2);

% This plot is identical to my\_cut, but was much faster to create.

% Imagine if you were running a script to take many cuts from the data - it

% is probably quicker to take them from existing data objects, where

% possible!

% For use with later example scripts, save a cut and slice

save(my\_slice,'../aaa\_my\_work/iron\_slice.sqw')

save(my\_cut,'../aaa\_my\_work/iron\_cut.sqw')

## Basic customisation of plots

You will have seen in the plots you created in the previous section that the Q axes we chose went beyond the extent of the data in some cases. You can squeeze the axes to just enclose the data using the compact command: http://horace.isis.rl.ac.uk/Reshaping\_etc#compact

Run this command on your 2d slice from exercise 4 above, and then plot the result.

Use the smooth command (http://horace.isis.rl.ac.uk/Reshaping\_etc#smooth) to apply some smoothing to your 2d slice, and plot the result.

Answer: You could get an error if you made the cut you want to smooth without the '-nopix' option – this is because smoothing only affects the rebinned data, there is no consistent way to smooth the pixel data. Hence smoothing is only allowed on dnd type objects (without pixel information).

Try this with the default options, and then try using a Gaussian smoothing with varying widths, to see what this does.

Use the lx, ly, lz commands to modify the length of the axes of your plots – specifically put the x-limits as -1.5 to 1.5, the y-limits as 50 to 250 and the colour limits as 0 to 0.5 for your 2d slice plot.

(Note that you can reset the limits back to their original values by just typing lx, ly or lz with no additional arguments).

You will have noticed that every time you make a new plot of an object with the same dimensionality as that of a previous plot on-screen, the old plot is replaced by the new one.

Use the keep\_figure command to retain a figure that will not be overwritten.

Try using the xycursor command on one of your plots.

What does it do?

Answer: Creates a cursor on the screen which lets you put text or coordinates on the plot.

A variant on this command is xyselect. Try it.

Solution script

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

% Basic customisation of plots

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

% Make axes tight:

plot(compact(my\_slice));

% Smoothing:

plot(smooth(my\_slice)); % this gives an error - think about why!

%

plot(smooth(d2d(my\_slice)));

% d2d command (for 2d objects) converts from sqw type data, with detector pixel retained

% to d2d / dnd object that is smaller in memory and without detector pixel info

% Smoothing options:

plot(smooth(d2d(my\_slice),[2,2],'gaussian'));

% Set colour scale and other axes scales in script:

lz 0 0.5

ly 50 250

lx -1.5 1.5

% Reset a limit

lx

% Retain a figure, so it is not replaced next time you make a plot (of the

% same dimensionality)

keep\_figure;

plot(my\_slice);

% Cursor to find a particular data point value

plot(my\_cut2);

xycursor

## Plotting data with non-orthogonal axes

The option proj.nonorthogonal (set to be true or false; default false) is for optional use only when the lattice angles of the material are not all 90 degrees. If it is set to be false for such a system, the projection axes will be constructed to be orthogonal, with the first lying along u, the second in the plane of u and v, and the third perpendicular to this plane. If instead you wish to use the correct non orthogonal reciprocal lattice the option is set to be true. In this case the axes used will be those you define, however the images of plots will still be plotted as if these axes were orthogonal, so there will be a shear transformation of the image.

1. Take a cut from /home/dl11170/edatc/data/upd3\_elastic.sqw, with constant energy in the range -2 to 2meV, and with L=-0.1 to 0.1. Note the proj.nonorthogonal option should be set to false by default. The remainder of the proj array should specify the (H,K)-plane for viewing data.

Hint: Use an empty vector [] for the H and K axes to get the full range.

1. Make the same cut, with the nonorthogonal option set to be true for the (H,K)-plane. Note the differences between the two images – in particular the anisotropy of the peaks in both.
2. Take 1d cuts through the peaks along H and K in both projection bases. Compare and contrast the widths, and confirm the differences between the images make physical sense.

Solution script:

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

% Plotting data with non-orthogonal axes

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

% To demonstrate plotting a non-orthogonal axes system we will use data

% from a crystal with hexagonal symmetry

sqw\_nonorth = [data\_path, 'upd3\_elastic.sqw'];

proj\_nonorth.u = [1, 0, 0];

proj\_nonorth.v = [0, 1, 0];

proj\_nonorth.type = 'rrr';

proj\_nonorth.nonorthogonal = false; % <--- Default sets to false

% If proj.nonorthogonal is false, u, v and w will be reconstructed to be

% orthogonal. The plots will have correct aspect ratio but it will be

% harder to tell the right reciprocal lattice coordinates by eye.

ws\_orth = cut\_sqw(sqw\_nonorth, proj\_nonorth, [-7,0.02,3], [-2,0.02,2], [-0.1,0.1], [-1,1]);

plot(ws\_orth)

keep\_figure()

% Now set the projection axes to non-orthogonal

proj\_nonorth.nonorthogonal = true;

ws\_nonorth = cut\_sqw(sqw\_nonorth, proj\_nonorth, [-7,0.02,3], [-2,0.02,2], [-0.1,0.1], [-1,1]);

plot(ws\_nonorth); keep\_figure();

## Correcting for sample misalignment

It is quite common to find after you have accumulated sufficient data that your sample was not perfectly aligned in the instrument. This can be seen by making slices or cuts at the elastic line in your Horace data. Horace provides a set of utilities to characterise and correct for such sample misalignment, and the following set of exercises provide a walk-through of how to use them.

1. Make a series of constant energy slices centred at 0 meV (+/-5 meV is a good integration range) to find out what Bragg peaks our example dataset covers. Make a list of 5 Bragg peaks (not all parallel to each other!) in a 5-by-3 matrix. Make sure you keep these slice plots for later.
2. Use the routine bragg\_positions to make and then fit transverse and radial cuts through these peaks. Note that the input parameters to this routine are rather complex, so read the help quite carefully before starting. You should use a 'gauss' lineshape, and the 'bin\_absolute' option in pretty much all cases.
3. You can examine the results of fitting the radial and transverse peaks using the routine bragg\_positions\_view. Before progressing to the next step it is imperative that all of the fits are good, otherwise your correction may end up making things less correct! Particular attention should be paid to the length and step size of the cuts to ensure reasonable statistics and a good fit.
4. Use the routine refine\_crystal to obtain the alignment correction matrix (type help refine\_crystal to get the syntax). In order to preserve the known cubic crystal structure, use the options fix\_angdeg (to keep all lattice angles 90 degrees) and fix\_alatt\_ratio (to ensure the cubic lattice parameters a=b=c). This should give you a matrix called rlu\_corr, that tells you how to modify the axes u and v (and their cross-product) to get the correct alignment.
5. Use the routine change\_crystal\_horace to apply this change to your sqw file (so you do not have to regenerate the whole dataset again).
6. Remake your constant energy slices, and compare what you have now with the plots you saved in part 1. Hopefully the Bragg peaks will be better aligned than they were before!
7. You can also use the results obtained above to work out the values for the goniometer offsets to be used for a correctly aligned sample in a regeneration of the sqw file using gen\_sqw. The routine you need is called crystal\_pars\_correct

Extension question: Can you think of any circumstances in which it would actually be beneficial to deliberately misalign your sample and then apply the above post-hoc corrections? [Hint: Consider the symmetry of both the sample and the spectrometer]

Solution script:

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

% Correcting for sample misalignment

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

clear variables

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

% Name of output sqw file (for the 4D combined dataset)

sqw\_file = [pwd '/../aaa\_my\_work/iron.sqw'];

% Make a series of hk-slices at different l, in order to work out what Bragg

% positions we have. Step sizes and energy integration should be customised for your data

% Step sizes should be as small as possible, and energy integration tight.

proj.u = [1,0,0];

proj.v = [0,1,0];

proj.uoffset = [0,0,0];

proj.type = 'rrr';

alignment\_slice1=cut\_sqw(sqw\_file,proj,[-5,0.03,8],[-5,0.03,8],[-0.05,0.05],[-10,10],'-nopix');

alignment\_slice2=cut\_sqw(sqw\_file,proj,[0.95,1.05],[-5,0.03,8],[-3,0.03,3],[-10,10],'-nopix');

alignment\_slice3=cut\_sqw(sqw\_file,proj,[-5,0.03,8],[-0.05,0.05],[-3,0.03,3],[-10,10],'-nopix');

% Look at the 3 orthogonal slices to figure out what bragg peaks are visible

plot(compact(alignment\_slice1)); keep\_figure;

plot(compact(alignment\_slice2)); keep\_figure;

plot(compact(alignment\_slice3)); keep\_figure;

% Our notional Bragg peaks - a list of accessible Bragg peaks (in data they

% may be off from these notional positions)

bragg\_peaks=[4,0,0; 2,0,0; 1,1,0; 4,4,0; 1,0,1];

% Get the actual Bragg peak positions with the current crystal alignment

% This routine takes radial and transverse cuts around the Bragg peaks listed

% above. See the help for further information about how the routine works -

% you will in general have to adjust some of the inputs here, especially the

% energy window

[rlu0,width,wcut,wpeak]=bragg\_positions(sqw\_file, bragg\_peaks, 1.5, 0.06, 0.4,...

1.5, 0.06, 0.4, 20, 'gauss','bin\_ab');

% Check how well the function did (note the command line prompts to allow you

% to scan through the cuts made above)

bragg\_positions\_view(wcut,wpeak)

% Determine corrections to lattice and orientation (in this example we choose

% to keep the lattice angles fixed, but allow the lattice parameters to be

% refined, keeping a cubic structure by keeping ratios of lattice pars to be same):

alatt = [2.87,2.87,2.87]; % original lattice parameters

angdeg = [90,90,90];

[rlu\_corr,alatt,angdeg,~,~,rotangle] = refine\_crystal(rlu0, alatt, angdeg,...

bragg\_peaks,'fix\_angdeg','fix\_alatt\_ratio');

% Apply changes to sqw file. For the purposes of this examples sheet you might

% want to copy the file in case you have made a mistake. In practice, you shouldn't

% make a copy as the sqw file could many hundreds of gigabytes and could take

% along time to copy.

sqw\_file\_new = [pwd '/../aaa\_my\_work/iron\_aligned.sqw'];

copyfile(sqw\_file,sqw\_file\_new)

change\_crystal\_horace(sqw\_file\_new, rlu\_corr);

% Check the outcome: Get Bragg peak positions and look at output: should be much better

[rlu0,width,wcut,wpeak]=bragg\_positions(sqw\_file\_new, bragg\_peaks, 1.5, 0.06, 0.4,...

1.5, 0.06, 0.4, 20, 'gauss','bin\_ab');

bragg\_positions\_view(wcut,wpeak)

%=========

% Generally you only want to figure out the misorientation once, then apply

% some correction to subsequent data. You can do this by finding the values

% of the notional goniometers gl, gs, dpsi that are used in gen\_sqw:

u = [1,0,0];

v = [0,1,0];

alatt = [2.87,2.87,2.87]; % original lattice parameters

angdeg = [90,90,90];

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

[alatt, angdeg, dpsi, gl, gs] = crystal\_pars\_correct...

(u, v, alatt, angdeg, omega, dpsi, gl, gs, rlu\_corr);

% u and v are the notional scattering plane, alatt0, angdeg0, etc are the

% original values for those parameters you used in gen\_sqw, rlu\_corr is the

% misalignment correction matrix determined above. The routine outputs the

% corrected lattic parameters (if these were refined) and the values of

% dpsi, gl and gs to use in future regenerations of the sqw file.

## Advanced Plotting

### Dispersions (a.k.a. “Spaghetti”) plots

You will probably have seen band structure or phonon dispersion calculations that show the dispersion along several high symmetry directions. Horace allows you to make equivalent plots from your data using a routine called spaghetti\_plot. Use the sqw file that you created for iron previously.

1. Make a basic dispersion plot that goes along the trajectory (1,-1,0) -> (2,0,0) -> (1,1,0) -> (1,-1,0).
2. Play with the qbin and ebin options to match the step sizes approximately to those used earlier.
3. Add labels (of your choice!) to the plot.
4. Save the slice along each trajectory into an sqw object, then plot these separately.

### Useful Horace Plotting Commands

The commands in the table below can be used to adjust features of plots to make them more presentable.

|  |  |
| --- | --- |
| lx, ly, lz | These are Horace-specific limit commands (similar to Matlab-native xlim, ylim, zlim and caxis) – because of customizations with Horace 3D, 2D and 1D plots, the Matlab standard commands don’t work well and we recommend to use these instead. E.g. lx(0, 2) will set the x-limit to between 0 and 2. |
| amark, acolor | Horace-specific commands to change the marker type and color for the *next* 1D plot. These are similar to arguments to the Matlab-standard plot command, but again because of Horace-specific customizations of the 1D plots, it is recommended to use these instead. |
| pp, pl, etc | Please see: <http://horace.isis.rl.ac.uk/Plotting#One_dimensional_plots> for a list of Horace-specific 1D plot commands. These should be used in preference to the standard Matlab plot. E.g. to plot a data in square red marker and a fit line in blue use: amark('s'); acolor('r'); plot(w1d); acolor('b'); pl(wfit); |
| title | This sets the title of the current active plot, e.g. title('My Plot') |
| xlabel, ylabel | Sets the title of the x- or y-axis. E.g. xlabel('Energy Transfer') Additional parameters can be added to this, for example to change the font size, xlabel('Qh', 'FontSize', 16). Type doc xlabel for more help. LaTeX can also be used in these and in the title commands – but you may have to set xlabel(…, 'interpreter', 'latex'). |
| colorslider, colorbar | colorslider is the Horace-specific widget which includes a standard Matlab colorbar with two editable text boxes to allow you to type the limits in – however, it does not look good in a plot output. colorslider('delete'); colorbar will replace the colorslider with a colorbar. |
| text, annotation | Matlab-standard commands which can add text or annotations to a plot. Type doc text or doc annotation to get help. |
| print | This saves a graph/figure to a file. E.g. print('fig1.pdf', '-dpdf') will save the current active figure to a pdf. Type doc print to get help. |
| gcf, gca | Matlab-standard commands to return the handle to the current figure (window, gcf) or axes (gca) |

Use the above commands to make a nice 2D and 1D Horace plot.

Solution script:

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

% Advanced plotting and publication quality figures

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

clear variables

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

% Two dimensional plot

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

sqw\_file = [pwd '/../aaa\_my\_work/iron.sqw'];

rlp = [1,-1,0; 2,0,0; 1,1,0; 1,-1,0];

wspag = spaghetti\_plot(rlp,sqw\_file,'qbin',0.1,'qwidth',0.3,'ebin',[0,4,250]);

lz 0 3

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

% Two dimensional plot

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

% Recreate the Q-E slice from earlier, this time without saving the pixel

% information

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

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

% Plot the 2d slice first:

plot(smooth(compact(my\_slice)));

% Set limits

lx -2 2

ly 40 250

lz 0 0.5

% Make a nicer title

title('My QE slice');

% Label the axes with something nicer

xlabel('(1+h,-1+h,0) (r.l.u.)');

ylabel('Energy (meV)');

% Get rid of the colour slider

colorslider('delete');

colorbar

% If we want to set the font sizes to be bigger, then we have to re-do the

% above:

title('My QE slice', 'FontSize', 16);

xlabel('(1+h,-1+h,0) (r.l.u.)', 'FontSize', 16);

ylabel('Energy (meV)', 'FontSize', 16);

% To set the font size of the ticks, we need to access the figure's axes.

my\_handles = get(gca)

% there are many things you can adjust! To set the font size, or any of the

% other properties, do the following:

set(gca, 'FontSize', 16);

% Suppose we want to change what tick marks are used on the x-axis

set(gca, 'XTick', -2:0.5:2);

set(gca, 'XTickLabel', arrayfun(@num2str, -2:0.5:2, 'UniformOutput', false));

%Put some text on the figure:

text(-0.5, 220, 'Ei = 400 meV', 'FontSize', 16);

% Some fancier text to label the colour bar:

tt = text(3.2, 240, 'Intensity (mb sr^{-1} meV^{-1} f.u.^{-1})', 'FontSize', 16);

set(tt, 'Rotation', -90)

%Save as jpg and eps

print('-djpeg', '../aaa\_my\_work/figure.jpg');

print('-depsc', '../aaa\_my\_work/figure.eps');

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

% One dimensional plots

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

% Make an 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

% plot them individually, to see what they look like first

for i = 1:numel(energy\_range)

plot(my\_cuts(i)); keep\_figure;

end

% We want to plot them all on the same axes, with different colours and

% markers.

my\_col={'black','red','blue','green','yellow'};

my\_mark={'+', 'o', '\*', '.', 'x', 's', 'd', '^', 'v', '>', '<', 'p', 'h'};

% note the above are all the possible choices!

for i = 1:numel(my\_cuts)

acolor(my\_col{i})

amark(my\_mark{i});

if i==1

plot(my\_cuts(i));

else

% The pp command overplots (markers and errorbars) on existing 1d axes

pp(my\_cuts(i));

end

end

% This is a bit messy. Let's add a constant offset between each cut, and make

% the markers bigger

my\_offset=[0:0.3:1.2];

for i = 1:numel(my\_cuts)

acolor(my\_col{i})

amark(my\_mark{i},6);

if i==1

plot(my\_cuts(i) + my\_offset(i));

else

pp(my\_cuts(i) + my\_offset(i));

end

end

% But we could have done this much more cleanly using the vectorised capabilities

% of Horace functions

acolor({'black','red','blue','green','yellow'})

amark({'+', 'o', '\*', '.', 'x', 's'},6)

my\_cut\_offset = my\_cuts + [0:0.3:1.2];

dp(my\_cut\_offset)

% Now need to extend axes to see everything:

lx -2 2

ly 0 1.8

% Use the same settings as before to get nice font sizes

title('Q cuts', 'FontSize', 16);

xlabel('(1+h,-1+h,0) (r.l.u.)', 'FontSize', 16);

ylabel('Intensity (mb sr^{-1} meV ^{-1} f.u.^{-1})', 'FontSize', 16);

set(gca, 'FontSize', 16);

set(gca, 'XTick', -2:0.5:2);

set(gca, 'XTickLabel', arrayfun(@num2str, -2:0.5:2, 'UniformOutput', false));

% Insert a figure legend

legend('80 meV','100 meV','120 meV', '140 meV','160 meV');

% But this is wrong!!! This is a peculiarity of Horace, in that it plots the

% markers then the errorbars, and Matlab doesn't keep track of this. Luckily

% there is a workaround, by getting a "handle" to each plot and then

% attaching the legend to that.

for i = 1:numel(my\_cuts)

acolor(my\_col{i})

amark(my\_mark{i},8);

if i==1

[fig\_handle, axes\_handle, plot\_handle] = plot(my\_cuts(i) + my\_offset(i));

else

[fig\_handle, axes\_handle, plot\_handle] = pp(my\_cuts(i) + my\_offset(i));

end

end

lx -2 2

ly 0 1.8

%legend(plot\_handle([10,8,6,4,2]), ...

% {'80 meV','100 meV','120 meV', '140 meV','160 meV'}, ...

% 'Location','NorthWest');

% You can also manually edit the plot, using the arrow tool to highlight

% part of the plot you want to change. e.g. you can remove the box around

% the legend by setting its colour to be white

## 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](https://doi.org/10.1016/j.nima.2016.07.036).