Mslice extra utilities

This document summarises some useful functions that increase the flexibility of mslice beyond its standard useage. Some of these functions already exist in the official release of mslice (R.Coldea, available from http://www.isis.rl.ac.uk/excitations/ and follow links from the 'software' entry). These include, for example, ms_slice, slice_spe, ms_cut and cut_spe for performing slices and cuts from the command line. Others are extensions of functions that exist in the official release, for example ms_simulate, simulate_spe for performing simulations of spe files; these are backwards compatible and so can supersede the official versions. There are also a number of new functions that perform commonly required manipulations on spe files, such as correcting for the Bose factor, multiplying by energy transfer, and definitive versions of various symmetrisation routines that are floating about. Finally, there are a few miscellaneous routines that aid command line operation of mslice.

To use the enhanced functions and extra functions, put the \mslice_corrections folder on the path ahead of c:\mslice\mprogs., and \mslice_extras ahead or after c:\mslice\mprogs:

```
>> path('t:\matlab\mslice_extras\',path);
>> path('c:\mprogs\mslice\',path);
>> path('t:\matlab\mslice_enhancements\',path);
```

The official release of mslice is the sole work of R.Coldea; the other functions described here are mostly the work of T.G.Perirng, some of R.Coldea, and of others. I try to record the provenance of the various functions in the following and in the source code.

1. Operation of mslice from the command line

The common use of Mslice is as a simple GUI interface for examining a single data set. However, Mslice can be used in a much more flexible fashion by using it in conjunction with the Matlab command prompt to

- perform functional manipulations on the data, either by using various utilities provided in the installation, or by writing your own,
- simulate data,
- store multiple datasets in the Matlab workspace, and use the mslice GUI as a powerful data-browser,
- use the underlying functions of mslice to script the loading of data, slicing and cutting.

The most powerful use of mslice comes from a shift of emphasis from mslice being purely a GUI application to mslice being a collection of functions for manipulating data for which there is a powerful GUI interface to assist in slicing and dicing.

1.1 Functions to start and stop mslice, and get or set fields in the mslice GUI

```
>> mslice_start                          % Starts mslice if not already started
    :

>> ms_getfields                          % Write full list field names to the screen
    :
>> value = ms_getvalue(field_name)       % Get value of a field name
    :
>> ms_setvalue(field_name, value)        % Set value of a field name

    :
```

```
>> mslice_finish                                    % Closes mslice if not already closed
```

1.2 Slicing and cutting from the command line

All the following functions are in the official mslice release. For full details of the input arguments, type >> help <*function name*>.

1.2.1 Cuts:

- Perform a cut using the parameters in the mslice control window:

```
>> ms_cut
```

- Perform a slice using other parameters:

For single crystal data with PSD detectors:

```
>> data = fromwindow;
>> cut = cut_spe (data, x, vx_min, vx_max, bin_vx,…          % x=1,2 or 3 – defines the cut direction
                  vy_min, vy_max, vz_min, vz_max,…
                  i_min, i_max, out_type, out_file, tomfit) % these last 4 arguments are optional
```

For single crystal data with conventional detectors, or powder data:

```
>> data = fromwindow;
>> cut = cut_spe (data, x, vx_min, vx_max, bin_vx,…          % x=1,2 or 3 – defines the cut direction
                  vy_min, vy_max,…
                  i_min, i_max, out_type, out_file, tomfit) % these last 4 arguments are optional
```

1.2.1 Slices:

- Perform a slice using the parameters in the mslice control window:

```
>> slice_data = ms_slice
```

- Perform a slice using other parameters:

```
>> data = fromwindow;
>> slice_data = slice_spe (data, z, vz_min, vz_max, …        % z=1,2 or 3 – defines the slice normal
                           vx_min, vx_max, bin_vx, vy_min, vy_max, bin_vy,…
                           i_min,i_max,shad,noplot)          % these last 4 arguments are optional
```

- Plot a slice created with ms_slice or slice_spe:

```
>> plot_slice (slide_data)
>> plot_slice (slice_data, i_min)
>> plot_slice (slice_data, i_min, i_max shad ,map, linearlog)
```

- Smooth a slice:

```
>> slice_out  = smooth_slice(slice_data, n)
```

2. Simulating a cross-section model

The following describes an enhanced version of the release version of ms_simulate that enables an arbitrary function for S(**Q**,w) to be used, rather than one which is contained in a function with the specific name ms_sqw. The routine is backwards compatible with the official ms_simulate (see later in this section).

2.1 Simple use of ms_simulate: return results directly to mslice control window

In the mslice control window, supply .phx file, lattice parameters etc. as required for 'load data' and 'calculate projections'. You can choose not to give an .spe file, in which case the simulation will be performed for all the detectors; if you give an spe file then the calculation will be performed only for the detectors unmasked in the .spe file.

To simulate the data for S(**Q**,w) defined in the function `my_cross_section` (or any other function name):

Simulation using the energy bins in the spe file:

```
>> ms_simulate(@my_cross_section, pars)
```

To simulate on a different set of energy bins:

```
>> ms_simulate(emin, emax, de, @my_cross_section, pars)
```

The function my_cross_section (or whatever you have chosen to call it) has arguments:

```
>>[s,wdisp]=my_cross_section(parameters,Qhkl,en,data_pars)
```

Input:

pars	Parameters needed by the cross-section - Typically vector of scalar parameters [p1 p2 ...] - but can be data structure, cell array...
Qhkl	Q points (h,k,l) (size(Q)=[n,3])
en	Energy values at each point in Q (size(w)=[n,1])
data_pars	[Optional]. Structure with various useful parameters passed from mslice including length of reciprocal lattice vectors, scattering angles... See the description of the function get_data_pars elsewhere in this document for a full list.

Output:

s	Calculated signal strength (size(s)=[n,1])
wdisp	[Optional]. Structure of calculated dispersion relation(s) at the points Q: wdisp.w1 size(wdisp.w1)=[n,1] wdisp.w2 size(wdisp.w2)=[n,1] : :

An example function, called my_cross_section.m, is included in the \mslice_extras folder. This returns S(Q,w) for a spin ½ Heisenberg antiferromagnetic chain and the dispersion of the lower and upper bounds.

Utility functions to help write cross-section models:

If have load data and calculate projections, then after

```
data = from_window;
```

- Calculate h,k,l,e for each pixel

```
>> [Qhkl,e] = get_hkle (data)
```

- Get components of q in inverse Angstrom in an orthonormal frame

```
>> Qangstrom=rlu2q(Qhkl,data.ar,data.br,data.cr)
```

- Get structure containing various parameters e.g. scattering angles, length of a*, b*,c* from data

```
>> dd = get_data_pars (data)
```

dd.theta	Scattering angle for each pixel
dd.psi	Azimuthal angle for each pixel
dd.arlu	[1x3] length of a*,b*,c* (Ang^-1)

dd.en	[1xne] energy bin centres
dd.det_theta	[ndetx1] scattering angle for each detector
dd.det_psi	[ndetx1] azimuthal angle for each detector
dd.emode	=1 direct geometry; =2 indirect geometry
dd.efixed	Fixed energy value
dd.ar	-| a*, b*, c* (Ang^-1) in orthonormal frame
dd.br	|- with x-axis || u, z-axis || u x v
dd.cr	-|
dd.psi_samp	Psi anlge of sample (rad)
dd.uv	[2x3] scattering plane vectors u and v in r.l.u.

Backwards compatibility with the original ms_simulate:

The function ms_simulate accepts the same syntax as the original mslice version: instead of a function handle (i.e. the @muyfunc), provide the integer value that selects the desired cross-section model in the function ms_sqw. The function ms_sqw.m must be somewhere on the Matlab path. There is one instance of ms_sqw.m in the mslice root directory. Type >> help ms_sqw for the arguments of this function, and how to add additional cross-sections.

2.2 Perform simulation within a script or function from the matlab command window

After performing 'load data' and 'calculate projections'

```
>> data = fromwindow;
     :
>> data.S = simulate_spe(@my_cross_section,pars,data)
     :
>> towindow(data)
```

2.3 For the greatest flexibility of use:

```
>> data = fromwindow;
     :
>> [Qhkl,en] = get_hkle (data)% calculate h,k,l,e for each pixel
>> dd = get_data_pars (data)  % structure containing various useful parameters e.g. scattering
                              % angles, emode, length of a*, b*,c* from data (see description
                              % of get_data_pars later on for full details)
     :
>> data.S = my_cross_section (parameters,Qhkl,en,dd)
>> data.S = reshape(data.S,size(data.ERR));      % Shape required by towindow
     :
>> towindow(data)
```

3. Writing a slice to an ASCII file

Modified version of ms_slice, to write slice to file if single input dataset.

e.g. >> ms_slice_write % write slice; prompts for filename

>> ms_slice_write ('c:\myfile.slc') % writes to named file

Full syntax:
>> slice_d = ms_slice_write (flname, cmd)

flname	= ''	prompts for output filename
	= 'myfile.slc'	saves slice to ASCII file named myfile.slc
cmd	= '', [] or absent	for normal slice plot
	= 'noplot'	for calculation only
	= 'surf'	for calculation and surface plot
	= 'clear'	clear slice data stored in ControlWindow object 'tag','ms_slice_data'

4. Symmetrisation of data

- About horizontal and/or vertical axes:

>> sym(n1)
>> sym(n1,n2)

n1, n2 are the indices of the two orthogonal viewing axes about which the data is symmetrised. The symmetrisation algorithm works as, for example: sym(1) sets u1 to |u1|.

- About diagonals (assumes unit length of the horizontal and vertical axes are the same)

>> symd(n1,n2,ntype)

n1, n2 are the two viewing axes that define the plane in which symmetrisation takes place.
ntype = 1 fold the data into the lower right triangle
ntype = 2 fold the data into the upper right triangle
ntype = 3 fold the data about both diagonals, into the quadrant centred on the positive n1 axis

- About general axis: one or more symmetrisation directions

>> symgen(n1,n2,theta_array)

n1, n2 are the two viewing axes that define the plane in which symmetrisation takes place.
theta_array: angle (or row of angles) in degrees of the symmetrisation axis w.r.t . n1.

The angle theta defines the direction of the new x-axis; the data is folded onto positive component along the new y axes. For example
>> symgen(1,2,-45) is equivalent to symd(1,2,1)
>> symgen(1,2,[-45,-135]) is equivalent to symd(1,2,3)

5. Functions to operate on Mslice data sets

- Multiply dataset by scale factor:

>> msfun_scale(scale)

- Multiply dataset by eps^(p):

>> msfun_mult_eps % multiply by eps

>> msfun_mult_eps(p) % multiply by eps^p

- Convert S(Q,w) to X''(Q,w):

X''(Q,w) = S(Q,w)*(1-exp(-hbar.w/kB.T)))

>> msfun_s2chi(T) % T = temperature in Kelvin

6. Constructing a background from a slice

The following example illustrates how to construct a background from the data that lies outside lower and upper bounds defined by two functions.

```
>> p = ms_slice     % calculates slice from data in mslice window
                    % Alternatively, use slice_spe
>> [bkgd,ps,pb,pm,q,wlo,whi]=slice_bkgd(p,0.05,30,0.2,10,...
                                @muller_lower,240,@muller_upper,240);
>> plot_slice(pm)   % plot mask array
>> hold on;
>> plot(q,wlo)      % overplot lower bound
>> plot(q,whi)      % overplot upper bound

>> data = fromwindow;
>> data_sub = sub_slice(data, bkgd); % subtract the background from the spe file
>> towindow(data_sub);
```

Appendix

ms_setvalue(‘sample’,1)	ms_getvalue(‘sample’)	=>’single crystal’
ms_setvalue(‘sample’,2)		=> ‘powder’
ms_setvalue(‘analysis_mode’,1)	ms_getvalue(‘analysis_mode’)	=>’single crystal’
ms_setvalue(‘analysis_mode’,2)		=> ‘powder’
ms_setvalue(‘det_type’,1)	ms_getvalue(‘det_type’)	=>’PSD’
ms_setvalue(‘det_type’,2)		=> ‘conventional (non-PSD)’

PSD mode:

ms_slice and slice_spe

ms_cut and cut_spe

powder and non-PSD mode:

ms_disp and disp_spe

ms_cut and cut_spe

Some things to know about output cut fields

‘none’ and ‘xye’ equal

‘cut’ and ‘mfit’ have same initial fields, but mfit has extra fields:
efixed,emode,MspDir,Mspfile,sample,abc,uv,psi_samp

‘hkl’ has it own format again

If plot with a range, then this gets added as extra fields.

Essentially, the structures differ depending on the cut, and can have extra information to do with plotting, not the data itself.

If the cuts are not written to file, then they all have the same fields.

However, with mslice_1d and slice_1d, the fields in the output cut are dependent solely on the type, but not on whether or not the cut is written to file.