# Horace Notes, Part 2

Used terminology and assumptions:

Keeping Horace interface would allow smooth transfer experience for all current Horace users and easy comparison of new and old well verified results, so it is suggested to keep Horace interface with only minor reasonable modifications.

Workspaces are equivalent to **dnd**(histo\_workspace) and **sqw**(MDEvent\_workspace) objects and are operated over in usual

Mantid way. One substantial difference is that **sqw** is a **dnd** object and **MDEventWorkspace** is not an **MDHistoWorkspace**.

Below we use

dnd == MDHistogram workspace

sqw == MDEvent workspace

win/wout – any of two workspaces

*Question*: Should we do something similar to Horace, explicitly distinguishing between MDEvent and MDHisto workspaces in the commands names?

Here we assume that all commands work on all object types except explicitly stated otherwise.

# Scattering models and dispersion.

Horace uses two types of scattering model functions. These functions used across whole workflow allowing their visualization, simulation on SQW(MDEvent) or DND(HistoWorkspace) datasets and fitting their parameters against datasets, containing experimental results.

Their general form is:

dndModel =dndFuncion(x1, x2, ..., xn, function\_parameters) Calculates model on DND mesh (x- arrays of the mesh coordinates of

sqwModel=sqwFunction(q1,q2,q3,en, function\_parameters) Calculates model on SQW mesh(any point of extended reciprocal space) q,en – points(arrays) in the reciprocal space

needed:

Numpy functions

*Color marking:*

*Red – missing in Mantid*

*Green – Present in Mantid (may be a bit different format but very close functionality)*

Some selection from Horace notes, part 1 provided here for completeness.

Detailed Horace command syntaxes are provided in the footnotes below the table.

|  |  |  |  |
| --- | --- | --- | --- |
| Function name & main Matlab call | Brief description | Notes on Mantid usage/Modification | Priority |
| Algorithms to Create and modify datasets: | | | |
| *Already discussed:* | |  |  |
| gen\_sqw[[1]](#endnote-1)  sqwout=gen\_sqw (spe\_file, par\_file, sqw\_file, efix, emode, alatt, angdeg, u, v, psi, omega, dpsi, gl, gs) | Read one or more spe files and a detector parameter file, and create an output sqw file | Two possibilities suggested for Mantid:   1. *Wout=buildMD ([spe\_file\_names,ws\_names], sqw\_file\_Name,\*\*kwargs)* – **Implicitly** requesting file based output workspace and works through mergeMDFiles   qwWout=gen\_sqw ([spe\_file\_names,ws\_names],\*\*kwargs) – **Implicitly** requesting memory based output workspace and go through mergeMD   1. Wout= buildMD ([spe\_file\_names,ws\_names], DoFileBased=[True,False],\*\*kwargs) –**explicitly** requesting file based/memory based workspace, kwargs contain file name or default file name is calculated   \*\*kwargs, if present, redefine UB matrix/Goniometer | Top |
| cut\_sqw[[2]](#endnote-2)  wout=cut\_sqw (data\_source, proj, p1\_bin, p2\_bin, p3\_bin, p4\_bin, '-nopix', filename) | Take a cut from a file(s) containing sqw or dnd object | Suggested:  *proj=GenerateSQWProj(arguments):* the proj object will be a slightly modified table workspace. The python binding would allow Horace syntax like proj.u= [1,1,1]  *cut = cut\_sqw(data\_source,proj,[-nopix][do\_filebased/orFilename]) -- nopix produces Histo workspace*   * Implicit or explicit filebased ws request like in gen\_sqw(…) * -**nopix** invokes binMD and returns HistoWS.   This would be a wrapper around BinMD/SliceMD Can work with HistoWS too but often useless | Top |
| *Requested:* | |  |  |
| section[[3]](#endnote-3)  wout = section (win, [ax\_1\_lo, ax\_1\_hi], [ax\_2\_lo, ax\_2\_hi], ...) | Takes a section out of an sqw/dnd object – like rebin but works along existing bin boundaries |  |  |
| [smoothing[[4]](#endnote-4)](http://horace.isis.rl.ac.uk/Plotting#Color)  wout = smooth (win, width, shape) | Smooth a dataset using selected smoothing function. |  |  |
| symmetrising[[5]](#endnote-5)  wout=symmetrise\_sqw(sqw,v1,v2, shift) | increases workspace statistics combining together equivalent areas of reciprocal space | Actually, it is reflection around a plain.  No Mantid equivalent; dnd(histo\_ws) may be equally useful but much more complex operation. |  |
| combining[[6]](#endnote-6)  wout=combine\_sqw(win)  Win – the list of input workspaces | Unlike plus allows different sizes dataset to be combined | MergeMD seems provides this functionality but I am not entirely sure – there are some constrains on bin type?. Should have exactly the same functionality.  Some options are close but numbers of others do not exist. See detailed command description for such options.  Do not exist:  combine\_equivalent\_zones |  |
| Rebinning[[7]](#endnote-7)  wout=rebin\_sqw(win,step1,step2,...) | Rebin data in an sqw object, either with the boundaries specified by another object, or with a specified set of ranges. | BinMD – behaves similarly to one of the rebin options. Ticket to provide step instead of nBins exists |  |
| compact[[8]](#endnote-8)  wout = compact(sqw) | Squeezes the data range in an sqw object to eliminate empty bins | Event mode would behave differently?  Is it equivalent to normalize by volume? |  |
| noisify[[9]](#endnote-9)  wout = noisify (w) | Adds random noise to an object or array of objects |  |  |
| replicate[[10]](#endnote-10)  wout = replicate (win, wref) | Make a higher dimensional dataset from n dimensional dataset by replicating the data along the extra dimensions of a reference dataset |  |  |
| change\_crystal[[11]](#endnote-11)  wout = change\_crystal (sqw, rlu\_corr) | Change the crystal lattice and orientation of an sqw object or array of objects | Behaves differently from Horace, as unlike Horace sqw pixels and dnd greed over the pixels MDEvents and MDBoxes have the same coordinate system |  |
| fake\_sqw[[12]](#endnote-12)  sqw=fake\_sqw (sqw, en, par\_file, sqw\_file, efix, emode, alatt, angdeg, u, v, psi, omega, dpsi, gl, gs)  similar to gen\_sqw format. | Create an output sqw file with dummy data using array(s) of energy bins instead spe file(s). |  |  |
| Binary operations (all there + boolean): | | | |
| w=w1+w2; w=w1\*w2 w=w1^w2 | One operand is either workspace with the same dimensions as another or a number or a vector of numbers if other argument is number of workspaces | PlusMD, MultiplyMD – Is python wrapper available? |  |
| w=w1-dnd; w=w1/dnd | One operand is either workspace with the same dimensions as another or a number or a vector of numbers if other argument is number of workspaces | MinusMD, DivideMD – Is python wrapper available?  Should it fail if RH operand is MDEventWS? |  |
| Auxiliary functions | | | |
| dispersion[[13]](#endnote-13)  wdisp = dispersion (win, @dispreln, parameters) | Calculate dispersion relation for dataset or array of datasets |  |  |
| disp2sqw\_plot [[14]](#endnote-14)  disp2sqw\_plot(rlp,dispreln,pars,ebins,fwhh) | Plot dispersion relation as colour map along a path in reciprocal space |  |  |
| calc\_sqw\_urange[[15]](#endnote-15): | Compute range of data for a collection of data files given the projection axes and crystal orientation | Almost there. Ticket in development |  |
| Simulation | | | |
| func\_eval[[16]](#endnote-16)  wout = func\_eval(win, @dndFunc, function\_parameters) | Evaluate a function at the plotting bin centres of datasets or array of objects | Should return histo workspace |  |
| sqw\_eval[[17]](#endnote-17)  wout = sqw(win, @sqwFunc or @dndFunc, function\_parameters); | Calculate sqw for a model scattering function for each event location if sqwFunction provided or as above if dndFunction | SimulateResolutionConvolvedModel without convolution? No python yet?  Should return MDEvent workspace if sqw function provided or MDHisto if DND function provided |  |
| Fitting | | | |
| Fit an sqw model[[18]](#endnote-18)  [wout, fitdata] = fit\_sqw(win, @sqwfunc, input\_parameters) | Fit a model for S(Q,w) to an sqw object, with an optional background function. | SimulateResolutionConvolvedModel with simple model?? |  |
| Fit an dnd model[[19]](#endnote-19)  fit & multifit | Find best fit of a parameterised function to data with an arbitrary number of dimensions | Mantid Fit  result = Fit(Function,InputWorkspace,[\*\*kwargs])  result is a tuple containing (OutputStatus,OutputChi2overDoF)  Missing python/numpy fit function |  |
| Export and comparison with other packages. | | | |
| save\_xye[[20]](#endnote-20)  save\_xye (w) | Save n-D.. sqw object to ascii file. | There is something like this in Mantid. Is XYE format supported? |  |
| save(w)[[21]](#endnote-21) | Save binary sqw/dnd object |  |  |
|  |  |  |  |
| Plotting (on Russell) | | | |
| plot[[22]](#endnote-22)  plot(win) | Polymorphic function which works on all dimension workspaces | h=plot(“ws\_name”) or h=plot(ws\_pointer) – introducing handle.  plot(1D ws)-> links to plotMD (expanded to MDEventWorkspace as it currently on )  plot(2-N\_Dim) -> links to sliceviwer. + **“-nice”** – working on 2D graphs in MantidPlot (*does not exist e.g. plotn(“ws2d”)*)  3-4D(currently) : Something to link to paraview ?  Suggested Simple Mantid overpolt command missing for cases 1 and 2-nice  **h=plot(”1D ws”,h)** |  |
|  |  |  |  |
|  |  |  |  |

1. gen\_sqw

   Read one or more spe files and a detector parameter file, and create an output sqw file.

   Normal use:

   >> gen\_sqw (spe\_file, par\_file, sqw\_file, efix, emode, alatt, angdeg, u, v, psi, omega, dpsi, gl, gs)

   To allow an spe file to appear more than once:

   >> gen\_sqw (spe\_file, par\_file, sqw\_file, efix, emode, alatt, angdeg, u, v, psi, omega, dpsi, gl, gs, 'replicate')

   Optionally (before the keyword 'replicate' if present):

   >> gen\_sqw (..., instrument, sample,...) % instrument and sample information

   >> gen\_sqw (..., grid\_size\_in, urange\_in,...) % grid size and range of data to retain

   >> gen\_sqw (..., grid\_size\_in, urange\_in, instrument, sample,...)

   If want output diagnostics:

   >> [tmp\_file,grid\_size,urange] = gen\_sqw (...)

   Input: (in the following, nfile = number of spe files)

   ------

   spe\_file Full file name of spe file - character string or cell array of character strings for more than one file

   par\_file Full file name of detector parameter file (Tobyfit format)

   sqw\_file Full file name of output sqw file

   efix Fixed energy (meV) [scalar or vector length nfile]

   emode Direct geometry=1, indirect geometry=2, elastic=0 [scalar]

   alatt Lattice parameters (Ang) [row or column vector]

   angdeg Lattice angles (deg) [row or column vector]

   u First vector (1x3) defining scattering plane (r.l.u.)

   v Second vector (1x3) defining scattering plane (r.l.u.)

   psi Angle of u w.r.t. ki (deg) [scalar or vector length nfile]

   omega Angle of axis of small goniometer arc w.r.t. notional u (deg) [scalar or vector length nfile]

   dpsi Correction to psi (deg) [scalar or vector length nfile]

   gl Large goniometer arc angle (deg) [scalar or vector length nfile]

   gs Small goniometer arc angle (deg) [scalar or vector length nfile]

   Optional arguments:

   grid\_size\_in [Optional] Scalar or row vector of grid dimensions. Default if not given or [] is is [50,50,50,50]

   urange\_in [Optional] Range of data grid for output as a 2x4 matrix:

   [x1\_lo,x2\_lo,x3\_lo,x4\_lo;x1\_hi,x2\_hi,x3\_hi,x4\_hi]

   Default if not given or [] is the smallest hypercuboid that encloses the whole data range.

   instrument Structure or object containing instrument information [scalar or array length nfile]

   sample Structure or object containing sample geometry information [scalar or array length nfile]

   Optional keyword argument:

   'replicate' Normally the function forbids an spe file from appearing more than once.

   This is to trap common typing errors. However, sometimes you might want to to create an sqw file using, for example, just one spe file as the source of data for all crystal orientations in order to construct a background from an empty piece of sample environment. In this case, use the keyword 'replicate' to override the uniqueness check.

   Output:

   --------

   tmp\_file List of temporary files created by this call to gen\_sqw (can be empty e.g. if a single spe file, when no temporary file is created)

   grid\_size Actual size of grid used (size is unity along dimensions where there is zero range of the data points)

   urange Actual range of grid

   Overloaded methods:

   sqw/gen\_sqw [↑](#endnote-ref-1)
2. ### sqw/cut

   Take a cut from an sqw object by integrating over one or more of the momentum and energy axes.

   >> w = cut (data\_source, p1\_bin, p2\_bin...) % cut plot axes, keeping existing integration ranges % (as many binning arguments as there are plot axes)

   >> w = cut (data\_source, proj, p1\_bin, p2\_bin, p3\_bin, p4\_bin) % cut with new projection axes

   >> w = cut (..., '-nopix') % output cut is dnd structure

   >> w = cut (..., '-save') % Save cut to file (prompt for output file)

   >> w = cut (..., filename) % save cut to named file

   For very large output, can avoid out-of-memory errors by writing file without making output to workspace:

   >> cut(...) % save cut to file without making output to workspace

   Input:

   ------

   data\_source Data source: sqw file name or sqw-type object

   proj Data structure containing details of projection axes:

   Defines two vectors u and v that give the direction of u1 (parallel to u) and u2 (in the plane of u1 and u2, with u2

   having positive component along v); also defines the normalisation of u1,u2,u3

   Required arguments:

   proj.u [1x3] Vector of first axis (r.l.u.)

   proj.v [1x3] Vector of second axis (r.l.u.)

   Optional arguments:

   proj.w [1x3] Vector of third axis (r.l.u.) - only needed if third character of type is 'p'

   Will otherwise be ignored.

   proj.type [1x3] Char. string defining normalisation:

   Each character indicates if u1, u2, u3 normalised as follows:

   - if 'a': unit length is one inverse Angstrom

   - if 'r': then if (h,k,l) in r.l.u., is normalised so max(abs(h,k,l))=1

   - if 'p': then normalised so that if the orthogonal set created from u and v is u1, u2, u3:

   |u1|=|u|, (u x u2)=(u x v), (u x u3)=(u x w) i.e. the projections of u,v,w along u1,u2,u3 match the lengths of u1,u2,u3

   Default:

   'ppr' if w not given

   'ppp' if w is given

   proj.uoffset Row or column vector of offset of origin of projection axes (r.l.u.)

   proj.lab Short labels for u1,u2,u3,u4 as cell array (e.g. {'Q\_h', 'Q\_k', 'Q\_l', 'En'})

   \*OR\*

   proj.lab1 Short label for u1 axis (e.g. 'Q\_h' or 'Q\_{kk}')

   proj.lab2 Short label for u2 axis

   proj.lab3 Short label for u3 axis

   proj.lab4 Short label for u4 axis (e.g. 'E' or 'En')

   p1\_bin Binning along first Q axis

   p2\_bin Binning along second Q axis

   p3\_bin Binning along third Q axis

   - [] or '' Plot axis: use bin boundaries of input data

   - [pstep] Plot axis: sets step size; plot limits taken from extent of the data

   - [plo, phi] Integration axis: range of integration

   - [plo, pstep, phi] Plot axis: minimum and maximum bin centres and step size

   p4\_bin Binning along the energy axis:

   - [] or '' Plot axis: use bin boundaries of input data

   - [pstep] Plot axis: sets step size; plot limits taken from extent of the data If pstep=0 then use bin size of the first spe file and synchronise the output bin boundaries with those boundaries. The overall range is chosen to ensure that the energy range of the input data is contained within the bin boundaries.

   - [plo, phi] Integration axis: range of integration

   - [plo, pstep, phi] Plot axis: minimum and maximum bin centres and step size;

   If pstep=0 then use bin size of the first spe file and synchronise the output bin boundaries with the reference boundaries. The overall range is chosen to ensure that the energy range plo to phi is contained within the bin boundaries.

   Output:

   -------

   w Output data object:

   - sqw-type object with full pixel information

   - dnd-type object if option '-nopix' given [↑](#endnote-ref-2)
3. ### section

   Takes a section out of an sqw object

   Syntax:

   >> wout = section (win, [ax\_1\_lo, ax\_1\_hi], [ax\_2\_lo, ax\_2\_hi], ...)

   Input:

   ------

   win Input sqw object

   [ax\_1\_lo, ax\_1\_hi] Lower and upper limits for the first axis. Bins are retained whose centres lie in this range.

   To retain the limits of the input structure, type '', [], or the scalar '0'

   [ax\_2\_lo, ax\_2\_hi] Lower and upper limits for the second axis for as many axes as there are plot axes

   Output:

   -------

   wout Output dataset.

   Example: to alter the limits of the first and third axes of a 3D sqw object:

   >> wout = section (win, [1.9,2.1], 0, [-0.55,-0.45]) [↑](#endnote-ref-3)
4. ### smoothing

   Smooths a dataset

   Syntax:

   >> wout = smooth (win, width, shape)

   Input:

   ------

   win Input dataset structure

   width Vector that sets the extent of the smoothing along each dimension.

   The interpretation of width depends on the argument 'shape' described below.

   If width is scalar, then the value is applied to all dimensions

   shape [Optional] Shape of smoothing function [Default: 'hat']

   'hat' hat function

   - width gives FWHH along each dimension in pixels

   - width = 1,3,5,...; n=0 or 1 => no smoothing

   'gaussian' Gaussian

   - width gives FWHH along each dimension in pixels

   - elements where more than 2% of peak intensity are retained

   Output:

   -------

   wout Smoothed data structure [↑](#endnote-ref-4)
5. ### Symmetrise:

   wout=symmetrise\_sqw(win,v1,v2,v3)

   WORKS ONLY FOR DATA OBJECTS OF SQW-TYPE (I.E. WITH PIXEL INFO RETAINED).

   Symmetriese sqw dataset in the plane specified by the vectors v1, v2, and v3. v1 and v2 are two vectors which lie in the plane of the reflection plane. v3 is a vector connecting the plane to the origin (i.e. specifies an offset).

   e.g. wout=symmetrise\_sqw(win,[0,1,0],[0,0,1],[1,0,0])

   The object win is symmetrised in the plane specified by [0,1,0] and [0,1,0] (i.e a mirror plane which reflects [-1,0,0] on to [1,0,0]). v3 is [1,0,0], so the plane is offset from the origin. This means that [-1,0,0] --> [3,0,0] etc. [↑](#endnote-ref-5)
6. ### combining

   wout=combine\_sqw(w1,w2)

   Combine two sqw objects (w1 and w2) of the same dimensionality into a single sqw object in order to improve statistics. Note that w1 AND w2 must be "true" sqw object, for which pixel information has been retained.

   The output object will have a combined value for the integration range e.g. combining two 2d slices taken at L=1 and L=2 will result in an output for which the stated value of L is L=1.5

   Two objects which use different projection axes can be combined. The output object will have the projection axes of w1.

   wout=combine\_equivalent\_zones(data\_source,proj,pos,qstep,erange,outfile)

   wout=combine\_equivalent\_zones(data\_source,proj,pos,qstep,erange,outfile,keyword)

   wout=combine\_equivalent\_zones(data\_source,proj,pos,qstep,erange,outfile,zonelist)

   or as above with no output argument, so that final 4-dimensional object is not retained in memory

   Necessary input Arguments:

   data\_source -- input sqw file

   proj -- the projection plane [proj.u, proj.v ] in the Horace meaning (see cut\_sqw or gen\_sqw) describing the target sqw object

   pos -- three integer numbers, (e.g [0,0,0]) specifying the initial point in reciprocal space (h,k,l) which is a reference point for the transformation

   qstep -- 3-vector or float describing delta Q in all 3 q-directions of the reciprocal space. If one number is specified, the steps are equal in all 3 directions

   egange -- 3-vector describing energy range and energy step of the cut [e\_min,e\_step,e\_max]

   Output argument:

   outfile -- the file with target output data

   Additional input arguments describe the symmetrization operation.

   Create a new sqw file which corresponds to just one Brillouin zone, but with data from equivalent positions. Default choice is all equivalent wavevectors, but can also manually specify which zones are to be combined.

   This is done either by using the keywords:

   '-cyclic' : only cyclic permutations (with no sign changes) of the chosen zone are included

   '-cycwithneg' : cyclic permutations AND the negatives (e.g. (2,1,0),

   (-2,1,0) etc.

   '-ab' : equivalent positions in the ab plane

   '-ac' : equivalent positions in the ac plane

   '-bc' : equivalent positions in the bc plane

   Alternatively one can explicitly provide a list of wavevectors to be combined by providing a cell array [↑](#endnote-ref-6)
7. ### Rebin

   Rebin data in an sqw object, either with the boundaries specified by another object, or with a specified set of [lo,step,hi].

   Because a rebinnable sqw can have dimensionality 1-3, we have to do tests for quite a large number of scenarios.

   When working with sqw objects we can use the "MSlice approximation", i.e. that the size of the pixels is much smaller than the size of the bins, so that we just put all of the spectral weight from a given pixel into then bin where we find it's centre.

   wout=rebin\_sqw(win,[lo1,step1,hi1],[lo2,step2,hi2],...) - rebin between specified limits with a given step size.

   wout=rebin\_sqw(win,step1,step2,...) - rebin with a given step size.

   wout=rebin\_sqw(win,w2) - rebin using the bin boundaries of object w2, which is either an sqw of the same dimensionality, or a dnd. [↑](#endnote-ref-7)
8. ### compact

   Squeezes the data range in an sqw object to eliminate empty bins

   Syntax:

   >> wout = compact(win)

   Input:

   ------

   win Input object

   Output:

   -------

   wout Output object, with length of axes reduced to yield the smallest cuboid that contains the non-empty bins. [↑](#endnote-ref-8)
9. ### noisify

   Adds random noise to an sqw object or array of sqw objects

   Syntax:

   >> wout = noisify (w)

   Add noise with Gaussian distribution, with standard deviation = 0.1\*(maximum pixel signal value)

   >> wout = noisify (w,factor)

   Add noise with Gaussian distribution, with standard deviation = factor\*(maximum pixel signal value)

   >> wout = noisify (w,'poisson')

   Add noise with Poisson distribution, where the mean value at a point is the value of pixel signal. [↑](#endnote-ref-9)
10. ### replicate

    Make a higher dimensional dataset from a n-dimensional dataset by replicating the data along the extra dimensions of a reference dataset.

    Syntax:

    >> wout = replicate (win, wref)

    Input:

    ------

    win n-dimensional dataset.

    wref Reference dataset structure to use as template for expanding the input structure.

    - The plot axes of win must also be plot axes of wref, and the number of points along these common axes must be the same, although the numerical values of the coordinates need not be the same.

    - The data is expanded along the plot axes of wref that are integration axes of win.

    - The annotations etc. are taken from the reference dataset.

    Output:

    -------

    wout Output dataset structure. [↑](#endnote-ref-10)
11. ### change\_crystal

    Change the crystal lattice and orientation of an sqw object or array of objects

    Most commonly:

    >> wout = change\_crystal (w, rlu\_corr) change lattice parameters and orientation

    OR

    >> wout = change\_crystal (w, alatt) change just length of lattice vectors

    >> wout = change\_crystal (w, alatt, angdeg) change all lattice parameters

    >> wout = change\_crystal (w, alatt, angdeg, rotmat) change lattice parameters and orientation

    >> wout = change\_crystal (w, alatt, angdeg, u, v) change lattice parameters and redefine u, v

    Input:

    -----

    w Input sqw object

    rlu\_corr Matrix to convert notional rlu in the current crystal lattice to the rlu in the the new crystal lattice together with any re-orientation of the crystal. The matrix is defined by the matrix: qhkl(i) = rlu\_corr(i,j) \* qhkl\_0(j) This matrix can be obtained from refining the lattice and orientation with the function refine\_crystal (type

    >>help refine\_crystal for more details).

    \*OR\*

    alatt New lattice parameters [a,b,c] (Angstroms)

    angdeg New lattice angles [alf,bet,gam] (degrees)

    rotmat Rotation matrix that relates crystal Cartesian coordinate frame of the new lattice as a rotation of the current crystal frame. Orthonormal coordinates in the two frames are related by v\_new(i)= rotmat(i,j)\*v\_current(j)

    u, v Redefine the two vectors that were used to determine the scattering plane These are the vectors at whatever misorientation angles dpsi, gl, gs (which cannot be changed).

    Output:

    -------

    wout Output sqw object with changed crystal lattice parameters and orientation

    NOTE

    The input data set(s) can be reset to their original orientation by inverting the input data e.g.

    - call with inv(rlu\_corr)

    - call with the original alatt, angdeg, u and v [↑](#endnote-ref-11)
12. ### fake\_sqw

    Create an output sqw file with dummy data using array(s) of energy bins instead spe file(s).

    >> fake\_sqw (sqw, en, par\_file, sqw\_file, efix, emode, alatt, angdeg u, v, psi, omega, dpsi, gl, gs)

    >> fake\_sqw (sqw, en, par\_file, sqw\_file, efix, emode, alatt, angdeg u, v, psi, omega, dpsi, gl, gs, grid\_size\_in, urange\_in)

    >> [tmp\_file, grid\_size, urange] = fake\_sqw (...) [↑](#endnote-ref-12)
13. ### Dispersion

    Calculate dispersion relation for dataset or array of datasets.

    >> wdisp = dispersion (win, dispreln, p) % dispersion only

    >> [wdisp,weight] = dispersion (win, dispreln, p) % dispersion and spectral weight

    The output dataset (or array of data sets), wdisp, will retain only the Q axes, and the signal array(s) will contain the values of energy along the Q axes. If the dispersion relation returns the spectral weight, this will be placed in the error array (actually the square of the spectral weight is put in the error array). In the case when the dispersion has been calculated on a plane in momentum (i.e. wdisp is IX\_datset\_2d) then the plot function ps2 (for plot\_surface2)

    >> ps2(wdisp)

    will plot a surface with the z axis as energy and coloured according to the spectral weight.

    The dispersion relation is calculated at the bin centres (that is, the individual pixel information in a sqw input object is not used).

    If the function that calculates dispersion relations produces more than one branch, then in the case of a single input dataset the output will be an array of datasets, one for each branch. If the input is an array of datasets, then only the first dispersion branch will be returned, so there is one output dataset per input dataset.

    Input:

    ======

    win Dataset that provides the axes and points for the calculation If one of the plot axes is energy transfer, then the output dataset will have dimensionality one less than the input dataset

    dispreln Handle to function that calculates the dispersion relation w(Q)

    Must have form:

    w = dispreln (qh,qk,ql,p)

    where

    qh,qk,ql Arrays containing the coordinates of a set of points in reciprocal lattice units

    p Vector of parameters needed by dispersion function e.g. [A,js,gam] as intensity, exchange, lifetime

    w Array of corresponding energies, or, if more than one dispersion relation, a cell array of arrays.

    More general form is:

    w = dispreln (qh,qk,ql,p,c1,c2,..)

    where

    p Typically a vector of parameters that we might want to fit in a least-squares algorithm

    c1,c2,... Other constant parameters e.g. file name for look-up table

    p Arguments needed by the function. Most commonly, a vector of parameter values e.g. [A,js,gam] as intensity, exchange, lifetime. If a more general set of parameters is required by the function, then package these into a cell array and pass that as pars. In the example above then pars = {p, c1, c2, ...}

    Output:

    =======

    wdisp Output dataset or array of datasets. Output is always dnd-type. The output dataset (or array of data sets) will retain only the Q axes, the signal array(s) will contain the values of energy along the Q axes, and the error array will contain the square of the spectral weight. If the function that calculates dispersion relations produces more than one branch, then in the case of a single input dataset the output will be an array of datasets, one for each branch. If the input is an array of datasets, then only the first dispersion branch will be returned, so there is one output dataset per input dataset.

    weight Mirror output: the signal is the spectral weight, and the error array contains the square of the frequency.

    e.g. If win is a 2D dataset with Q and E axes, then wdisp is a 1D dataset with just the Q axis [↑](#endnote-ref-13)
14. ### disp2sqw\_plot

    Plot dispersion relation as colour map along a path in reciprocal space

    >> disp2sqw\_plot(rlp,dispreln,pars,ecent,fwhh)

    >> disp2sqw\_plot(lattice,rlp,dispreln,pars,ecent,fwhh)

    >> disp2sqw\_plot(...,'labels',{'G','X',...}) % customised labels at the positions of the rlp

    >> disp2sqw\_plot(...,'ndiv',n) % plot with number of points per interval other than the default

    >> weight=disp2sqw\_plot(...) % output IX\_dataset\_2d with spectral weight

    >> weight=disp2sqw\_plot(...,'noplot') % output IX\_dataset\_2d with spectral weight, no plot

    Input:

    --------

    lattice [optional] Lattice parameters [a,b,c,alpha,beta,gamma] (Angstrom, degrees)

    Default is [2\*pi,2\*pi,2\*pi,90,90,90]

    rlp Array of r.l.p. e.g. [0,0,0; 0,0,1; 0,-1,1; 1,-1,1; 1,0,1; 1,0,0];

    dispreln Handle to function that calculates the dispersion relation w(Q) and spectrl weight, s(Q)

    Most commonly used form is:

    [w,s] = dispreln (qh,qk,ql,p)

    where

    qh,qk,ql Arrays containing the coordinates of a set of points in reciprocal lattice units

    p Vector of parameters needed by dispersion function e.g. [A,js,gam] as intensity, exchange, lifetime

    w Array of corresponding energies, or, if more than one dispersion relation, a cell array of arrays.

    More general form is:

    [w,s] = dispreln (qh,qk,ql,p,c1,c2,..)

    where

    p Typically a vector of parameters that we might want to fit in a least-squares algorithm

    c1,c2,... Other constant parameters e.g. file name for look-up table.

    pars Arguments needed by the function. Most commonly, a vector of parameters

    values e.g. [A,js,gam] as intensity, exchange, lifetime. If a more general set of parameters is required by the function, then package these into a cell array and pass that as pars. In the example above then pars = {p, c1, c2, ...}

    ecent Energy bin centres: [ecent\_lo, step, ecent\_hi]

    fwhh Full-width half-height of Gaussian broadening to dispersion relation(s)

    Keyword options (can be abbreviated to single letter):

    'labels' Tick labels to place at the positions of the Q points in argument rlp.

    e.g. {'G','X','M','R'}

    By default the labels are character representations of rlp

    e.g. {0,0,0; 0.5,0,0; 0.5,0.5,0; 0.5,0.5,0.5}

    becomes

    {'0,0,0', '0.5,0,0', '0.5,0.5,0', '0.5,0.5,0.5'}

    'ndiv' Number of points into which to divide the interval between two r.l.p. (default=100)

    'noplot' Do not plot, just return the output IX\_dataset\_2d (see below)

    Ouptut:

    --------

    weight IX\_dataset\_2d with spectral weight [↑](#endnote-ref-14)
15. ### calc\_sqw\_urange

    Compute range of data for a collection of data files given the projection axes and crystal orientation

    urange = calc\_grid (efix, emode, eps\_lo, eps\_hi, det, alatt, angdeg, u, v, psi, omega, dpsi, gl, gs)

    Input: (in the following, nfile = no. spe files)

    ------

    dummy Dummy sqw object - used only to ensure that this service routine was called

    efix Fixed energy (meV) [scalar or vector length nfile]

    emode Direct geometry=1, indirect geometry=2 [scalar]

    eps\_lo Lower energy transfer (meV) [scalar or vector length nfile]

    eps\_hi Upper energy transfer (meV) [scalar or vector length nfile]

    det Name of detector .par file, or detector structure as read by get\_par

    alatt Lattice parameters (Ang^-1) [vector length 3 or array size [nfile,3]] [↑](#endnote-ref-15)
16. ### func\_eval

    Evaluate a function at the plotting bin centres of sqw object or array of sqw object

    Syntax:

    >> wout = func\_eval (win, func\_handle, pars)

    >> wout = func\_eval (win, func\_handle, pars, 'all')

    Input:

    ======

    win Dataset or array of datasets; the function will be evaluated at the bin centres along the plot axes

    func\_handle Handle to the function to be evaluated at the bin centres

    Must have form:

    y = my\_function (x1,x2,... ,xn,pars)

    or, more generally:

    y = my\_function (x1,x2,... ,xn,pars,c1,c2,...)

    - x1,x2,.xn Arrays of x coordinates along each of the n dimensions

    - pars Parameters needed by the function

    - c1,c2,... Any further arguments needed by the function e.g. they could be the filenames of lookup tables for resolution effects)

    e.g. y=gauss2d(x1,x2,[ht,x0,sig])

    y=gauss4d(x1,x2,x3,x4,[ht,x1\_0,x2\_0,x3\_0,x4\_0,sig1,sig2,sig3,sig4])

    pars Arguments needed by the function.

    - Most commonly just a numeric array of parameters

    - If a more general set of parameters is needed by the function, then wrap as a cell array {pars, c1, c2, ...}

    'all' [option] Requests that the calculated function be returned over the whole of the domain of the input dataset. If not given, then the function will be returned only at those points of the dataset that contain data. Applies only to input with no pixel information - this option is ignored if the input is a full sqw object.

    Output:

    =======

    wout Output objects or array of objects

    e.g.

    >> wout = func\_eval (w, @gauss4d, [ht,x1\_0,x2\_0,x3\_0,x4\_0,sig1,sig2,sig3,sig4])

    where the function gauss appears on the matlab path

    function y = gauss4d (x1, x2, x3, x4, pars)

    y = (pars(1)/(sig\*sqrt(2\*pi))) \* ... [↑](#endnote-ref-16)
17. ### sqw\_eval

    Calculate sqw for a model scattering function

    >> wout=sqw\_eval(win,sqwfunc,p)

    Input:

    ------

    win Dataset (or array of datasets) that provides the axes and points for the calculation

    sqwfunc Handle to function that calculates S(Q,w)

    Most commonly used form is:

    weight = sqwfunc (qh,qk,ql,en,p)

    where

    qh,qk,ql,en Arrays containing the coordinates of a set of points

    p Vector of parameters needed by dispersion function e.g. [A,js,gam] as intensity, exchange, lifetime weight Array containing calculated energies; if more than one dispersion relation, then a cell array of arrays

    More general form is:

    weight = sqwfunc (qh,qk,ql,en,p,c1,c2,..)

    where

    p Typically a vector of parameters that we might want to fit in a least-squares algorithm

    c1,c2, Other constant parameters e.g. file name for look-up table

    pars Arguments needed by the function. Most commonly, a vector of parameter values e.g. [A,js,gam] as intensity, exchange, lifetime. If a more general set of parameters is required by the function, then package these into a cell array and pass that as pars. In the example above then pars = {p, c1, c2, ...}

    'all' [option] Requests that the calculated sqw be returned over the whole of the domain of the input dataset. If not given, then the function will be returned only at those points of the dataset that contain data.

    Applies only to input with no pixel information - it is ignored if full sqw object.

    'ave' [option] Requests that the calculated sqw be computed for the average values of h,k,l of the pixels in a bin, not for each pixel individually. Reduces cost of expensive calculations. Applies only to the case of sqw object with pixel information - it is ignored if dnd type object.

    Output:

    -------

    wout Output dataset or array of datasets [↑](#endnote-ref-17)
18. ### fit\_sqw

    Fit a model for S(Q,w) to an sqw object, with an optional background function. If passed an array of sqw objects, then each object is fitted independently.

    Differs from multifit\_sqw, which fits all objects in the array simultaneously but with independent backgrounds.

    Fit several objects in succession to a given function:

    >> [wout, fitdata] = fit\_sqw (w, func, pin) % all parameters free

    >> [wout, fitdata] = fit\_sqw (w, func, pin, pfree) % selected parameters free to fit

    >> [wout, fitdata] = fit\_sqw (w, func, pin, pfree, pbind) % binding of various parameters in fixed ratios

    With optional 'background' function added to the function

    >> [wout, fitdata] = fit\_sqw (..., bkdfunc, bpin)

    >> [wout, fitdata] = fit\_sqw (..., bkdfunc, bpin, bpfree)

    >> [wout, fitdata] = fit\_sqw (..., bkdfunc, bpin, bpfree, bpbind)

    If unable to fit, then the program will halt and display an error message. To return if unable to fit, call with additional arguments that return status and error message:

    >> [wout, fitdata, ok, mess] = fit\_sqw (...)

    Additional keywords controlling which ranges to keep, remove from objects, control fitting algorithm etc.

    >> [wout, fitdata] = fit\_sqw (..., keyword, value, ...)

    Keywords are:

    'keep' range of x values to keep

    'remove' range of x values to remove

    'mask' logical mask array (true for those points to keep)

    'select' if present, calculate output function only at the points retained for fitting

    'list' indicates verbosity of output during fitting

    'fit' alter convergence critera for the fit etc.

    'evaluate' evaluate function at initial parameter values only, with argument check as well

    'chisqr' evaluate chi-squared at the initial parameter values (ignored if 'evaluate' not set)

    'average' compute the function at the average h,k,l,e of the pixels in a bin

    Example:

    >> [wout, fitdata] = fit\_sqw (..., 'keep', xkeep, 'list', 0)

    Input:

    ======

    win sqw object or array of sqw objects to be fitted

    sqwfunc Handle to function that calculates S(Q,w)

    Most commonly used form is:

    weight = sqwfunc (qh,qk,ql,en,p)

    where

    qh,qk,ql,en Arrays containing the coordinates of a set of points

    p Vector of parameters needed by dispersion function e.g. [A,js,gam] as intensity, exchange, lifetime

    weight Array containing calculated energies; if more than one dispersion relation, then a cell array of arrays

    More general form is:

    weight = sqwfunc (qh,qk,ql,en,p,c1,c2,..)

    where

    p Typically a vector of parameters that we might want to fit in a least-squares algorithm

    c1,c2,... Other constant parameters e.g. file name for look-up table

    pin Initial function parameter values

    - If the function my\_function takes just a numeric array of parameters, p, then this contains the initial values [pin(1), pin(2)...]

    - If further parameters are needed by the function, then wrap as a cell array {[pin(1), pin(2)...], c1, c2, ...}

    pfree [Optional] Indicates which are the free parameters in the fit e.g. [1,0,1,0,0] indicates first and third are free

    Default: all are free

    pbind [Optional] Cell array that indicates which of the free parameters are bound to other parameters in a fixed ratio determined by the initial parameter values contained in pin:

    pbind={1,3} parameter 1 is bound to parameter 3.

    pbind={{1,3},{4,3},{5,6}} parameter 1 bound to 3, 4 bound to 3, and 5 bound to 6. In this case, parameters 1,3,4,5,6 must all be free in pfree.

    To explicity give the ratio, ignoring that determined from pin:

    pbind=(1,3,0,7.4) parameter 1 is bound to parameter 3, ratio 7.4 (the extra '0' is required)

    pbind={{1,3,0,7.4},{4,3,0,0.023},{5,6}}

    To bind to background parameters (see below), use the function index unity:

    pbind={1,3,1} Parameter 1 bound to background parameter 3

    pbind={1,3,1,3.14} Give explicit binding ratio.

    Optional background function:

    --------------------------------

    bkdfunc Handle to background function

    The background function is assumed to be defined by the axes x1,x2,...xn (n=number of dimensions).

    Must have form:

    y = my\_function (x1,x2,... ,xn,p)

    or, more generally:

    y = my\_function (x1,x2,... ,xn,p,c1,c2,...)

    - x1,x2,.xn Arrays of x coordinates along each of the n dimensions

    - p a vector of numeric parameters that can be fitted

    - c1,c2,... any further arguments needed by the function e.g. they could be the filenames of lookup tables for resolution effects)

    e.g. Two dimensional Gaussian:

    function y = gauss2d(x1,x2,p)

    y = p(1).\*exp(-0.5\*(((x1 - p(2))/p(4)).^2+((x2 - p(3))/p(5)).^2);

    bpin Initial function parameter values

    - If the function my\_function takes just a numeric array of parameters, bp, then this contains the initial values [bpin(1), bpin(2)...]

    - If further parameters are needed by the function, then wrap as a cell array {[bpin(1), bpin(2)...], c1, c2, …}

    bpfree [Optional] Indicates which are the free parameters in the fit e.g. [1,0,1,0,0] indicates first and third are free

    Default: all are free

    bpbind [Optional] Cell array that indicates which of the free parameters are bound to other parameters

    Examples of a single binding description:

    {1,4} Background parameter (bp) 1 is bound to bp 3, with the fixed ratio determined by the initial values

    {5,11,0} Bp 5 bound to parameter 11 of the foreground fitting function, func

    {5,11,1} Bp 5 bound to parameter 11 of the background function

    {5,11,0,0.013} Explicit ratio for binding bp 5 to parameter 11 of the foreground fitting function

    {1,4,1,14.15} Explicit ratio for binding bp 1 to bp 4 of background function

    Several binding descriptions:

    {{1,3},{2,4,0,1.2},{5,11,1}}

    Optional keywords:

    ------------------

    'list' Numeric code to control output to Matlab command window to monitor status of fit =0 for no printing to command window

    =1 prints iteration summary to command window

    =2 additionally prints parameter values at each iteration

    'fit' Array of fit control parameters

    fcp(1) relative step length for calculation of partial derivatives

    fcp(2) maximum number of iterations

    fcp(3) Stopping criterion: relative change in chi-squared i.e. stops if chisqr\_new-chisqr\_old < fcp(3)\*chisqr\_old

    ‘keep' Ranges of data to retain for fitting. A range is specified by two pairs of numbers which define a rectangle:

    [xlo, xhi, ylo, yhi]

    Several ranges can be defined by making an (m x 4) array:

    [xlo(1), xhi(1), ylo(1), yhi(1); xlo(2), xhi(2), ylo(2), yhi(2); ...]

    'remove' Ranges to remove from fitting. Follows the same format as 'keep'.

    'mask' Array of ones and zeros, with the same number of elements as the data array, that indicates which of the data points are to be retained for fitting

    'select' Calculates the returned function values, yout, only at the points that were selected for fitting by 'keep' and 'remove'; all other points are set to NaN. This is useful for plotting the output, as only those points that contributed to the fit will be plotted.

    A final useful set of keyword is:

    'evaluate' Evaluate the fitting function at the initial parameter values only. Useful for checking the validity of starting parameters.

    'chisqr' If 'evaulate' is set, then if this option keyword is present the reduced chi-squared is evaluated. Otherwise, chi-squared is set to zero.

    'average' if sqw object, then compute the function at the average h,k,l,e of the pixels contributing to each bin, rather than for each pixel. This can save a lot of computation

    Output:

    =======

    wout Array or cell array of the objects evaluated at the fitted parameter values

    If there was a problem for ith data set i.e. ok(i)==false, then wout(i)==w(i) (or wout{i}

    =[] if cell array input).

    If there was a fundamental problem e.g. incorrect input argument syntax, then fitdata=[].

    fitdata Result of fit for each dataset

    fitdata.p - parameter values

    fitdata.sig - estimated errors of foreground parameters (=0 for fixed parameters)

    fitdata.bp - background parameter values

    fitdata.bsig - estimated errors of background (=0 for fixed parameters)

    fitdata.corr - correlation matrix for free parameters

    fitdata.chisq - reduced Chi^2 of fit (i.e. divided by (no. of data points) - (no. free parameters))

    fitdata.pnames - parameter names

    fitdata.bpnames- background parameter names

    If there was a problem for ith data set i.e. ok(i)==false, then fitdata(i) will be dummy.

    If there was a fundamental problem e.g. incorrect input argumnet syntax, then

    fitdata=[].

    ok True if all ok, false if problem fitting.

    If an array of input datasets was given, then ok is an array with the size of the input data array. If the error was fundamental e.g. wrong argument syntax, then ok will be a scalar.

    mess Character string contaoning error message if ~ok; '' if ok

    If an array of datasets was given, then mess is a cell array of strings with the same size as the input data array.

    If the error was fundamental e.g. wrong argument syntax, then mess will be a simple character string.

    EXAMPLES:

    Fit a spin waves to a collection of sqw objects, allowing only intensity and coupling constant to vary:

    >> weight=100; SJ; gamma=3;

    >> [wout, fdata] = fit\_sqw (w, @bcc\_damped\_spinwaves, [ht,SJ,gamma], [1,1,0])

    If an array of 1D cuts: allow all parameters to vary, only keep data in restricted range, and allow independent linear background for each cut in the units of the x axis:

    >> weight=100; SJ; gamma=3;

    >> const=0; slope=0;

    >> [wout, fdata] = fit\_sqw (w, @bcc\_damped\_spinwaves, [ht,SJ,gamma], @linear, [const,slope],'keep',[-1.5,0.5]) [↑](#endnote-ref-18)
19. ### multifit

    Contents of multifit:

    fit - Find best fit of a parametrised function to data. Works for arbitrary

    multifit - Find best fit of a parametrised function to data with an arbitrary number of dimensions.

    multifit\_bind\_local\_pars\_as\_global - Bind parameters of a function that is used across all datasets as if they were local

    multifit\_gateway - Interface function so that Horace multifit functions can work with latest multifit.

    multifit\_gateway\_get\_state - Get the most recently set datset index and type of function evaluation request

    multifit\_gateway\_main - Gateway function to multifit least-squares fitting algorithm

    multifit\_gateway\_parameter\_get - Get the numeric array of parameters in a valid parameter list

    multifit\_gateway\_parameter\_set - Set the numeric array of parameters in a valid parameter list

    multifit\_gateway\_parsefunc - Gateway function to argument testing capability of multifit\_main

    multifit\_gateway\_pbind\_struct\_to\_cell - Convert structure output of pbind\_parse into cell array of binding descriptions

    multifit\_gateway\_wrap\_functions - Wrap the functions for multifit with another function and arguments

    multifit is both a directory and a function.

    Find best fit of a parameterised function to data with an arbitrary number of dimensions.

    The data can be x,y,e arrays or objects of a class.

    Simultaneously fit several objects to a given function:

    >> [wout, fitdata] = multifit (w, func, pin) all parameters free

    >> [wout, fitdata] = multifit (w, func, pin, pfree) selected parameters free to fit

    >> [wout, fitdata] = multifit (w, func, pin, pfree, pbind) binding of various parameters in fixed ratios

    With optional 'background' functions added to the global function, one per object

    >> [wout, fitdata] = multifit (..., bkdfunc, bpin)

    >> [wout, fitdata] = multifit (..., bkdfunc, bpin, bpfree)

    >> [wout, fitdata] = multifit (..., bkdfunc, bpin, bpfree, bpbind)

    If unable to fit, then the program will halt and display an error message.

    To return if unable to fit, call with additional arguments that return status and error message:

    >> [wout, fitdata, ok, mess] = multifit (...)

    Additional keywords controlling which ranges to keep, remove from objects, control fitting algorithm etc.

    >> [wout, fitdata] = multifit (..., keyword, value, ...)

    Keywords are:

    'keep' range of x values to keep

    'remove' range of x values to remove

    'mask' logical mask array (true for those points to keep)

    'select' if present, calculate output function only at the points retained for fitting

    'list' indicates verbosity of output during fitting

    'fit' alter convergence critera for the fit etc.

    'evaluate' evaluate at the initial parameter values (convenient to test starting values)

    'chisqr' evaluate chi-squared at the initial parameter values (ignored if 'evaluate' not set)

    Example:

    >> [wout, fitdata] = multifit (..., 'keep', xkeep, 'list', 0)

    Input:

    ======

    Data to be fitted:

    x Coordinates of the data points:

    - An array of any size whose outer dimension gives the coordinate dimension i.e. x(:,:,...:,1) is the array of x values along axis 1, x(:,:,...:,2 along axis 2) .. to x(:,:,...:,n) along the nth axis.

    The exception is if size(x) matches size(y), then the outer dimension is taken as unity and the data is considered to be one dimensional e.g. x=[1.1, 2.3, 4.3 & y=[110, 121, 131 1.7, 5.4, 7.0] 141, 343, 89]

    or - A cell array of length n, where x{i} gives the coordinates in the ith dimension for all the data points. The arrays can have any size, but they must all have the same size.

    y Array of the of data values at the points defined by x. Must have the same same size as x(:,:,...:,i) if x is an array, or of x{i} if x is a cell array.

    e Array of the corresponding error bars. Must have same size as y.

    Alternatively:

    w - Structure with fields w.x, w.y, w.e with form described above (this is a single dataset)

    - Array of structures w(i).x, w(i).y, w(i).e with form above (this defines several datasets)

    - Cell array of structures, each element a single dataset

    - Array of objects to be fitted.

    - Cell array of objects to be fitted.

    If a cell array, not all the objects need to be of the same class, so long as the function to be fitted is defined as a method for each of the class types.

    Notes on required methods if adding objects other than x-y-e triples:

    (1) The global function and background function (if given) must be methods, with input argument form as described below in detail; the general format is

    >> wcalc = my\_function (w,p,c1,c2,...)

    (2) A method that returns the intensity and variance arrays from the objects, along with a mask array that indicates which elements are to be ignored:

    >> [y,var,msk] = sigvar\_get(w)

    (3) A method that masks data points from further calculation:

    >> wout = mask (win, mask\_array)

    (elements of mask\_array: 1 to keep a point, 0 to remove from fitting)

    ($) \*EITHER\*

    A method that returns and array of the x values (one dimensional data) or

    a cell array of arrays of the the x1, x2, x3... values for every point in the object

    >> x = sigvar\_getx (w)

    (The size and shape of each of the x arrays must match those of the y and e arrays returned by sigvar\_get, in point (2).)

    \*OR\*

    Create a mask array given ranges of x-coordinates to keep &/or remove &/or &/or mask array Must output a logical flag ok, with message string if ~ok rather than terminate. (Can have it terminate if ok and mess are not given as return arguments; it is the advanced syntax that is required within multifit)

    >> [sel, ok, mess] = mask\_points (win, 'keep', xkeep, 'xremove', xremove, 'mask', mask)

    (Normally this method would not be supplied sigvat\_getx is easy to code, but special knowledge of the class could make mask\_points more efficient.)

    (5) If a background function is provided, addition of objects must be defined as

    >> wsum = w1 + w2

    (requires overloading of the addition operator with a method named plus.m)

    func Function handle to function to be fitted to each of the objects.

    If x,y,e or structure(s) with fields w.x,w.y,w.e, must have form:

    ycalc = my\_function (x,p)

    or, more generally:

    ycalc = my\_function (x,p,c1,c2,...)

    If objects, then:

    wcalc = my\_function (w,p)

    or, more generally:

    wcalc = my\_function (w,p,c1,c2,...)

    - p a vector of numeric parameters that can be fitted

    - c1,c2,... any further arguments needed by the function e.g. they could be the filenames of lookup tables for resolution effects)

    The functions can be nested. The examples below illustrate why this might be necessary. The convention that is adopted is

    func1 (w, @func2, {p}, c1, c2, ...) => func1 (func2(w,p{:}), c1, c2,...)

    EXAMPLE: Fit a model for S(Q,w) to an sqw object:

    Will have a function to compute S(Q,w) with the standard form:

    weight = my\_sqwfunc (qh, qk, ql, en, p, c1, c2,..)

    and there is a method of sqw to evaluate this function:

    wcalc = sqw\_eval (w, @my\_sqwfunc, {p, c1, c2, ...})

    Consequently, because we have chosen the convention for nesting functions:

    func1 (w, @func2, {p}, c1, c2, ...) => func1 (func2(w,p{:}), c1, c2,...)

    then the model for S(q,w) can be fitted by the call:

    multifit(w, @sqw\_eval, {@my\_sqwfunc, {p, c1, c2,...}})

    EXAMPLE: Resolution convolution of S(Q,w):

    Will have a method of sqw class that takes a model for S(Q,w) and convolutes with the resolution function:

    wres = resconv (w, @my\_sqwfunc, {p,c1,c2,...}, res\_p1, res\_p2,...)

    In this case, the function call will be:

    multifit (w, @resconv, {@my\_sqwfunc, {p, c1, c2,...}, res\_p1, res\_p2,...})

    pin Initial function parameter values

    - If the function my\_function takes just a numeric array of parameters, p, then this contains the initial values [pin(1), pin(2)...]

    - If further parameters are needed by the function, then wrap as a cell array {[pin(1), pin(2)...], c1, c2, ...}

    pfree [Optional] Indicates which are the free parameters in the fit.

    e.g. if length(p)=5, then pfree=[1,0,1,0,0] indicates first and third are free

    Default: all are free. Similarly, pfree=[] is interpreted as all being free.

    pbind [Optional] Cell array that indicates which of the free parameters are bound to other parameters in a fixed ratio determined by the initial parameter values contained in pin:

    pbind={1,3} parameter 1 is bound to parameter 3.

    pbind={{1,3},{4,3},{5,6}} parameter 1 bound to 3, 4 bound to 3, and 5 bound to 6

    In this case, parmaeters 1,3,4,5,6 must all be free in pfree.

    To explicitly give the ratio, ignoring that determined from pin:

    pbind=(1,3,0,7.4) parameter 1 is bound to parameter 3, ratio 7.4 (the extra '0' is required)

    pbind={{1,3,0,7.4},{4,3,0,0.023},{5,6}}

    To bind to background parameters (see below), give the index of the background function handle in the cell array bkdfunc defined below.

    bind={1,3,[7,2]} parameter 1 bound to background parameter 3 of background

    function handle bkdfunc{7,2}.

    pbind={1,3,[7,2],3.14} Give explicit binding ratio.

    Optional background function(s):

    --------------------------------

    It is sometimes convenient to fit a global function, func, to the collection of objects w,but have a function that might be used to provide an object-specific background. For example, we may want to fit S(Q,w) to several one-dimensional cuts, but have an independent linear background for each cut. These are defined similarly to the above

    bkdfunc Cell array of background function handles

    - If contains a single handle, then the same function applies to every object in the collection of objects, w, that is to be fitted. (Internally, is expanded into a cell array of the same size as w. If wish to bind background parameters to a particular background handle, then refer to the corresponding index of that expanded array).

    - Otherwise, the size of the cell array must match the size of w, and there will be a one-to-one correspondence of the background function handles to the elements of w.

    The form required for the functions is identical to that for func above.

    bpin Cell array of initial parameter values for the background function(s), following the same definitions and conventions as pin

    - If a single element in the cell array, then will be used for every background function;

    - Otherwise, the size of the cell array must match the size of w, and there will be a one-to-one correspondence of the elements to the background initial parameters

    NOTE\*\*\*

    The insistence on bpin being a cell array resolves an ambiguity. Otherwise, for exanple, if length(w)==4, and length(bkdfunc)=1, then if iscell(bpin) and length(bpin)==4, then this could mean that the cell array is to be passed to bkdfunc as the list of parameters, and is the same for each element of w. Equally, it could mean that we have four different initial parameter sets for the same function, one per element of w. With the syntax demanded here, only the latter interpretation is possible.

    bpfree Array, or cell array of arrays indicating which parameters are free to vary.

    - If single array, or cell array with a single array, then applies to every background function

    - Otherwise, the size of the cell array must match the size of w, and there will be a one-to-one correspondence of the elements to indicate the free background parameters

    Pbind Cell array of binding cell arrays of the form defined for pbind. Indicates how background parameters are bound. Take care here: as the object that defined the binding is itself a cell array of cell arrays, it is easy to get confused as to whether the binding description applies to all background functions, or if it is a set of distinct binding descriptions, one for each background function. The size of the outermost cell array dictates which:

    - If a single element in the cell array, then will be used for every background function;

    - Otherwise, the size of the cell array must match the size of w, and there will be a one-to-one correspondence of the elements to the background functions.

    Examples of a single binding description:

    {1,4} Background parameter (bp) 1 is bound to bp 3, with the fixed ratio determined by the initial values

    {2,3,[7,2]} Bp 2 bound to bp 3 of background function handle bkdfunc{7,2}

    {5,11,0} Bp 5 bound to parameter 11 of the global fitting function, func

    {{1,4}, {2,3,[7,2]}, {5,11,0}} Several bindings defined together

    {5,11,0,0.013} Explicit ratio for binding bp 5 to parameter 11 of the global fitting function

    {1,4,[7,2],14.15} Explicit ratio for binding bp 1 to bp 4 of background function [7,2]

    In a call to multifit: as an example of the need to take care:

    bpbind = {{1,4}, {2,3,[7,2]}, {5,11,0}} binding description for three separate backgrounds

    bpbind = { {{1,4}, {2,3,[7,2]}, {5,11,0}} } Binding description for all background functions

    NOTE\*\*\*

    The insistence on bpin being a cell array resolves the same type of ambiguity as was solved for bpin

    Optional keywords:

    ------------------

    'list' Numeric code to control output to Matlab command window to monitor status of fit

    =0 for no printing to command window

    =1 prints iteration summary to command window

    =2 additionally prints parameter values at each iteration

    ‘fit' Array of fit control parameters

    fcp(1) relative step length for calculation of partial derivatives

    fcp(2) maximum number of iterations

    fcp(3) Stopping criterion: relative change in chi-squared i.e. stops if chisqr\_new-chisqr\_old < fcp(3)\*chisqr\_old

    'keep' Array or cell array of arrays giving ranges of x to retain for fitting.

    - single array: applies to all elements of w

    - a cell array of arrays must have length the same as w, and describes the keep ranges for those elements one-by-one.

    A range is specified by an array of numbers which define a hypercube.

    For example in case of two dimensions:

    [xlo, xhi, ylo, yhi]

    or in the case of n-dimensions:

    [x1\_lo, x1\_hi, x2\_lo, x2\_hi,..., xn\_lo, xn\_hi]

    More than one range can be defined in rows,

    [Range\_1; Range\_2; Range\_3;...; Range\_m]

    where each of the ranges are given in the format above.

    'remove' Ranges to remove from fitting. Follows the same format as 'keep'.

    If a point appears within both xkeep and xremove, then it will be removed from the fit i.e. xremove takes precendence over xkeep.

    'mask' Array, or cell array of arrays, of ones and zeros, with the same number of elements as the data arrays in the input object(s) in w. Indicates which of the data points are to be retained for fitting (1=keep, 0=remove).

    'select' Calculates the returned function values, wout, only at the points that were selected for fitting by 'keep', 'remove', 'mask' and were not eliminated for having zero error bar etc; this is useful for plotting the output, as only those points that contributed to the fit will be plotted.

    A final useful pair of keyword is:

    'evaluate' Evaluate the fitting function at the initial parameter values only. Useful for checking the validity of starting parameters.

    'chisqr' If 'evaulate' is set, then if this option keyword is present the reduced chi-squared is evaluated. Otherwise, chi-squared is set to zero.

    Output:

    =======

    wout Array or cell array of the objects evaluated at the fitted parameter values

    Has the same form as the input data. The only exception is if x,y,e were given as three separate arrays, only ycalc is returned.

    If there was a problem i.e. ok==false, wout=[]

    fitdata Result of fit for each dataset

    fitdata.p - parameter values

    fitdata.sig - estimated errors of global parameters (=0 for fixed parameters)

    fitdata.bp - background parameter values

    fitdata.bsig - estimated errors of background (=0 for fixed parameters)

    fitdata.corr - correlation matrix for free parameters

    fitdata.chisq - reduced Chi^2 of fit (i.e. divided by (no. of data points) - (no. free parameters))

    fitdata.pnames - parameter names

    fitdata.bpnames - background parameter names

    If there was a problem i.e. ok==false, fitdata=[]

    ok True if all ok, false if problem fitting.

    mess Character string contaoning error message if ~ok; '' if ok

    Examples:

    Fit five one-dimensional sqw objects to a model for S(Q,w) broadened by the instrument resolution function. The resolution model depends on two parameters

    r1, r2, and the S(Q,w) model uses two large arrays c1, c2 in addition to the fitting parameters p. The 4th and 5th parameters of p are constraint to vary with a fixed ratio; parameters 2 and 3 are fixed. Each sqw object has its own independent quadratic background, but all have the same initial parameter values.

    Two of them are constrained to a linear background.

    >> [wout,fitdata] = multifit (w,...@resconv, {@my\_sqwfunc,{p,c1,c2},r1,r2}, [1,0,0,1,1], {4,5},...

    @func\_eval, {{@quad,[1.1,0.1,0.02]}}, {[],[1,1,0],[],[],[1,1,0]} )

    Overloaded methods:

    sqw/multifit

    d4d/multifit

    d3d/multifit

    d2d/multifit

    d1d/multifit

    d0d/multifit [↑](#endnote-ref-19)
20. ### save\_xye

    Save 1D,2D,.. sqw object to ascii file.

    Syntax:

    >> save\_xye (w) Prompts for file to write to

    >> save\_xye (w, null\_value) Substitute intensity of empty cells with the numerical value empty (default: NaN)

    >> save\_xye (w, file) Write to named file

    >> save\_xye (w, null\_value, file)

    Unless otherwise specified, bins where there is no data are written as having NaN (i.e. not-a-number) for the signal and zero for the standard deviation.

    You can always substitue a different value e.g. -10^30 or 0 by assigning a value to the optional parameter null\_value.

    Note that if w is an array of objects, then "file" must be a cell array of filenames.

    The data is saved in the format:

    1D dataset:

    x(1) y(1) e(1)

    x(2) y(2) e(2)

    : : :

    x(n) y(n) e(n)

    2D dataset:

    x1(1) x2(1) y(1,1) e(1,1)

    x1(2) x2(1) y(2,1) e(2,1)

    : : : :

    x1(n1) x2(1) y(n1,1) e(n1,1)

    x1(1) x2(2) y(1,2) e(1,2)

    x1(2) x2(2) y(2,2) e(2,2)

    : : : :

    x1(n1) x2(2) y(n1,2) e(n1,2)

    x1(1) x2(3) y(1,3) e(1,3)

    : : : :

    x1(n1) x2(n2) y(n1,n2) e(n1,n2)

    3D dataset:

    x1(1) x2(1) x3(1) y(1,1) e(1,1)

    x1(2) x2(1) x3(1) y(2,1) e(2,1)

    : : : : :

    x1(n1) x2(1) x3(1) y(n1,1) e(n1,1)

    x1(1) x2(2) x3(1) y(1,2) e(1,2)

    x1(2) x2(2) x3(1) y(2,2) e(2,2)

    : : : : : [↑](#endnote-ref-20)
21. ### save

    Save a sqw object or array of sqw objects to file

    >> save (w) % prompt for file

    >> save (w, file) % give file

    Input:

    w sqw object

    file [optional] File for output. if none given, then prompted for a file

    Note that if w is an array of sqw objects then file must be a cell array of filenames of the same size.

    Output: [↑](#endnote-ref-21)
22. ### Plot 1D, 2D or 3D sqw object or array of objects

    >> plot(w)

    Equivalent to:

    >> dp(w) % 1D dataset

    >> da(w) % 2D dataset

    >> sliceomatic(w) % 3D dataset [↑](#endnote-ref-22)