Example script fit MnF2 powder data using Horace and SpinW

Table of Contents

Fitting with SpinW - part 1, refining when you already have a good guess for the parameters 1
Fitting with Spinw - part 2, a first step in simplifying things 4
Fitting with SpinW - part 3 - applying constraints to the fits to aid convergence 6
Fitting with SpinW - part 4 - what to do if you have no idea of good parameter values? Brute force method 9
Fitting with SpinW - part 5 - alternative approaches to fitting - particle swarm optimisation 30

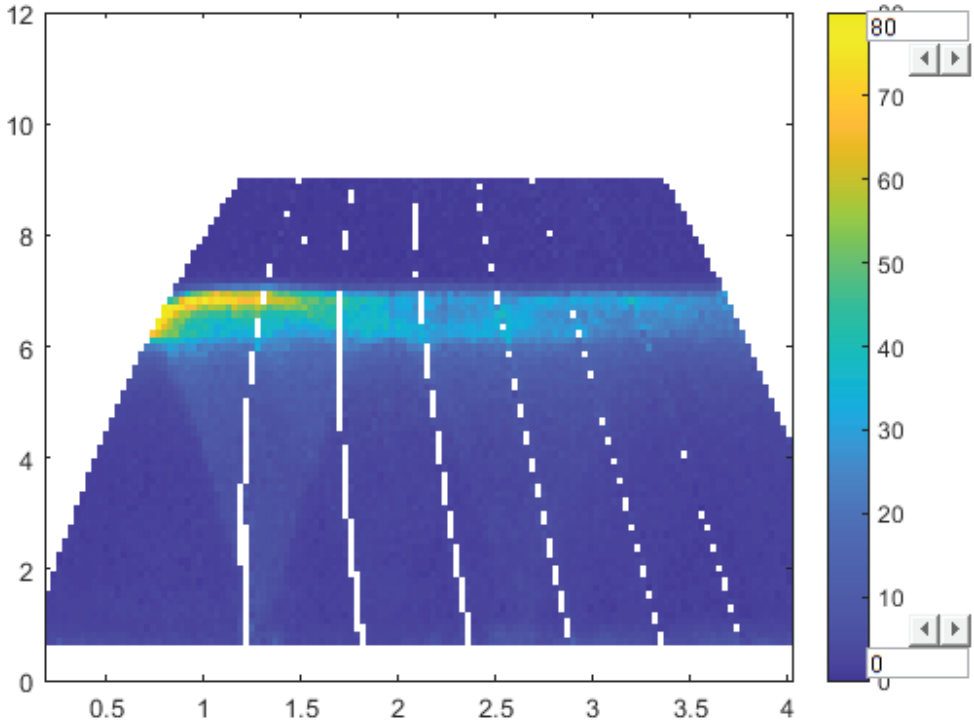
Assumes that you have already installed Horace and SpinW!

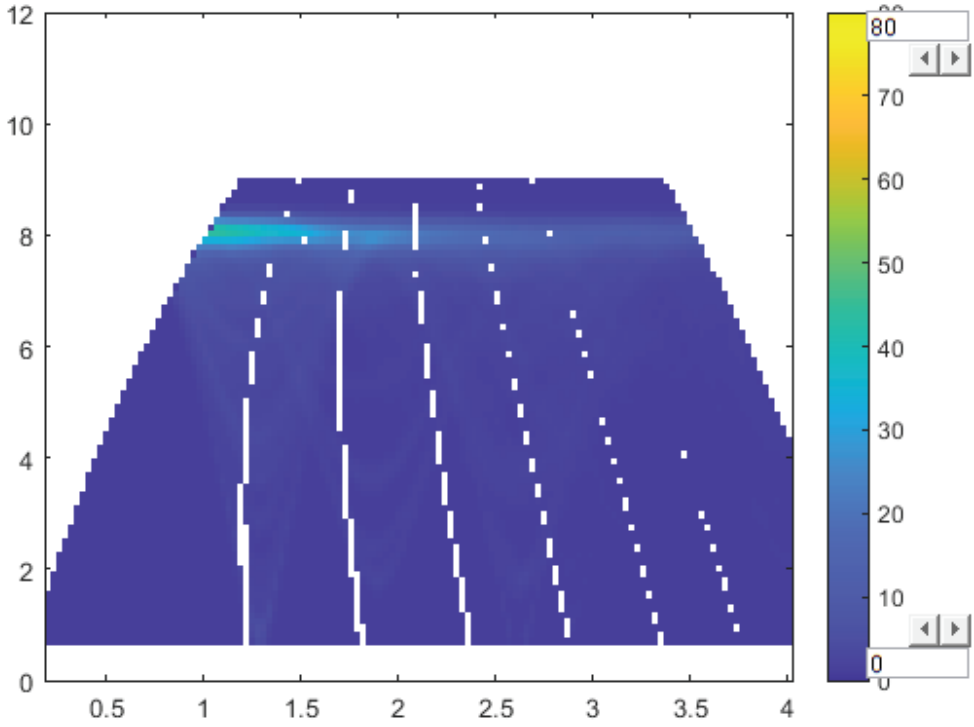
========== Russell Ewings - 20/5/2020

Fitting with SpinW - part 1, refining when you already have a good guess for the parameters

```
%For this tutorial only - reduce display level in Matlab window, to
 make
%the resulting pdf much smaller...
swpref.setpref('tid',0);
swpref.setpref('fid',0);


%What would happen if we did not know the answer in advance, and tried
 to
%fit using a sensible guess for J1, J2 and D? Let's try it:

%Remake the cut, to be only the region that is actually fittable (i.e.
%excluding the incoherent elastic line, and the higher energies):
mnf2_cut=cut_sqw(sqw_mnf2,[0.2,0.03,4],[0.7,0.12,9],'-nopix');
mnf2_IX=IX_dataset_2d(mnf2_cut.p{1},mnf2_cut.p{2},mnf2_cut.s,sqrt(mnf2_cut.e));
mnf2_IX.signal(mnf2_IX.error==0 & mnf2_IX.signal==0)=NaN;
ok=~isnan(mnf2_IX.signal);

% A sensible, but not correct, guess for initial parameters
scalefac=80;%intensity scale factor
J=[-0.02,0.4];%exchange guess values
D=0;%single ion anisotropy keep zero
bg=1;

%============================
% <REALLY IMPORTANT POINT>:
% In the ExcitationsPowder routines that you downloaded, you must copy
 the file "powspec_ran.m"
```

```
% to the folder in your SpinW installation .../swfiles/@spinw (this is
 where the regular powspec.m
% file lives - check by typing in Matlab "which powspec"). The routine
 ensures that the same random
% number seed is used every time - see above comment for details
%============================

%Be careful running this - even with a reduced number of q points
 (e.g.
%nRand = 400 inside the spinw_mnf2_fit code) it will still take a long
 time
%to execute. The list = 1 option allows you to see how the fit is
%progressing by giving a readback of chi^2 for each iteration.
[wfit2,fitdata2]=multifit(mnf2_IX,@spinw_mnf2_fit,
{[scalefac,J,D,bg],ok,Ei,dE,dQ,s},...
    [0,1,1,0,0],'list',1,'fit',[1e-3,30,1e-3]);

%You should find that the fit does not converge unless the initial
 guess is
%really very close the (in this case known) correct answer.

plot(mnf2_IX); ly 0 12; lz 0 80; keep_figure;
plot(wfit2); ly 0 12; lz 0 80; keep_figure

Taking cut from data in file C:\Russell\Software
\ExcitationPowderPublish\data_files\MnF2.sqw...
Step   1 of    1; Have read data for 40755 pixels -- now processing
 data... -----> retained 38589 pixels
-------------------------------------
Beginning fit (max 30 iterations)
-------------------------------------
Iteration  Time(s)  Reduced Chi^2
     0        9.010    564.0077
     1       36.050    566.1446
     1       44.736    561.7493
     2       71.376    559.0698
     3       97.631    561.1806
     3      106.582    561.3069
     3      115.683    560.6319
     3      124.493    559.0183
Fit converged
```

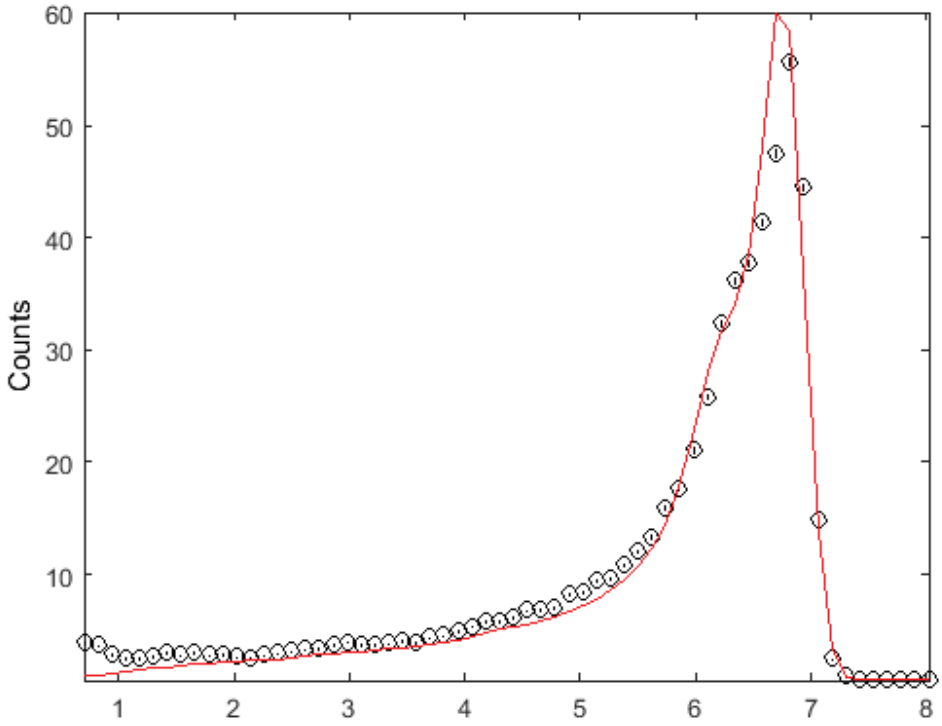

12
10
8
6
4
2
0
0.5
1
1.5
2
2.5
3
3.5
4
80
70
60
50
40
30
20
10
0


12
10
8
6
4
2
0
0.5
1
1.5
2
2.5
3
3.5
4
80
70
60
50
40
30
20
10
0

Fitting with Spinw - part 2, a first step in simplifying things

```
%In the example above we tried fitting a 2d slice. This took a long
 time
%and did not converge unless we pretty much already knew the answer. A
%better strategy might then be to find a judicious 1d cut that could
 tell
%us the answer, and converge more quickly.

%Take a 1d cut along the energy axis, integrating 1<|Q|<2
mnf2_1dcut=cut_sqw(sqw_mnf2,[1,2],[0.7,0.12,8],'-nopix');
mnf2_IX1d=IX_dataset_1d(mnf2_1dcut.p{1},mnf2_1dcut.s,sqrt(mnf2_1dcut.e));

%determine if there are any "bad" data points
ok1d=~isnan(mnf2_IX1d.signal);
Qrange=[1,2];%note down the Q range for integration, as we need to
 pass this to the simulating / fitting function

scalefac=9;%intensity scale factor
J=[-0.06,0.32];%exchange guess values (remember correct vals are
 -0.0575 and 0.3161)
D=0;%single ion anisotropy keep zero
bg=0.6;%keep this fixed for now.

[wfit3,fitdata3]=multifit(mnf2_IX1d,@spinw_mnf2_1dfit,
{[scalefac,J,D,bg],Qrange,ok1d,Ei,dE,dQ,s},...
    [1,1,1,0,0],'list',1);

%Plot data with fit over the top (pl means plot line in Horace)
acolor black
plot(mnf2_IX1d)
acolor red
pl(wfit3)

fitdata3

%We can see that now we have got quite close to the correct answer,
 but
%only if our initial guess was still quite good. We are afforded a bit
 more
%slack than if we try to fit the 2d dataset, but still run into
 trouble if
%our starting parameters are off. For example, if we start with
 J=[-0.02
%0.25] then the fit fails miserably!

Taking cut from data in file C:\Russell\Software
\ExcitationPowderPublish\data_files\MnF2.sqw...
Step   1 of    1; Have read data for 17499 pixels -- now processing
 data... -----> retained  8705 pixels
-------------------------------------
```

```
Beginning fit (max 30 iterations)
-------------------------------------
Iteration  Time(s)  Reduced Chi^2
     0         6.237   1118.3297
     1        31.591    111.3105
     2        56.819     98.9734
     3        82.677     94.0201
     4       108.819     90.1491
     5       134.317     84.8789
     6       159.610     84.2445
     7       184.895    173.5412
     7       191.336    173.5412
     7       197.796    173.5412
     7       204.049    173.4283
     7       210.296    160.0509
     7       216.844     82.6968
     8       242.910     83.8376
     8       250.086     82.9225
     8       257.185     82.6863
     9       285.668     82.4826
    10       311.030     82.5671
    10       317.274     82.5440
    10       323.480     82.4898
    10       329.667     82.4838
    10       335.863     82.4860
    10       342.120     82.4825
Fit converged

fitdata3 =

  struct with fields:

           p: [6.2875 -0.0734 0.3100 0 0.6000]
         sig: [0.1544 0.0018 0.0012 0 0]
        corr: [3×3 double]
       chisq: 82.4825
   converged: 1
      pnames: {'p1'  'p2'  'p3'  'p4'  'p5'}
```



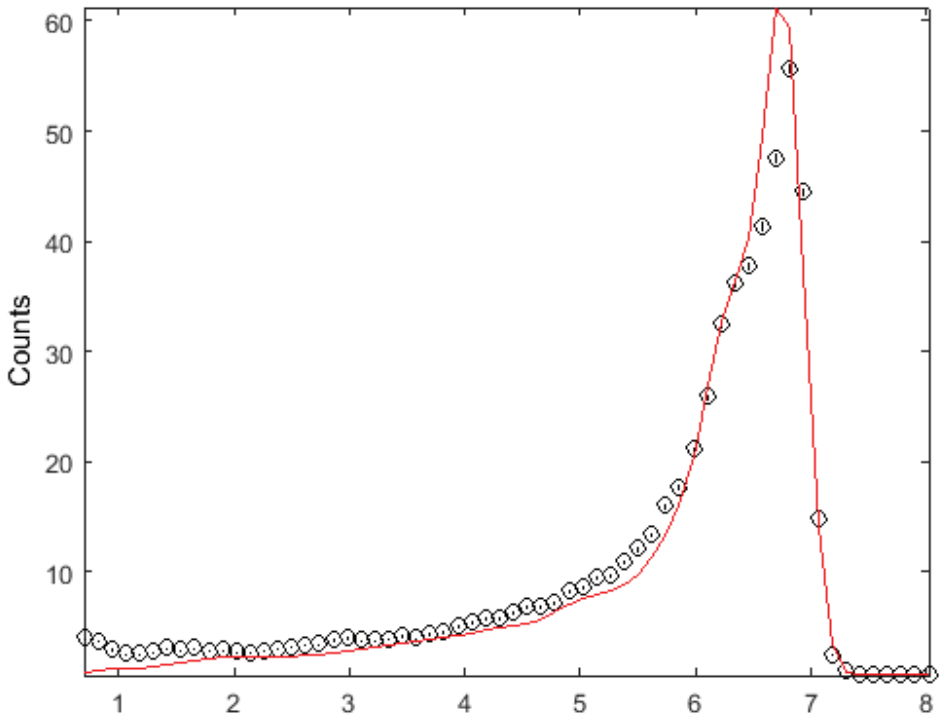
Fitting with SpinW - part 3 - applying constraints to the fits to aid convergence

```
%Here we will use a new fitting function that allows us to impose
%constraints (upper and lower limits) on what are considered valid fit
%parameters.

scalefac=9;%intensity scale factor
J=[-0.06,0.32];%exchange guess values (remember correct vals are
 -0.0575 and 0.3161)
D=0;%single ion anisotropy keep zero
bg=0.6;%keep this fixed for now.

%Specify lower and upper limits for a parameter:
plims={[5,9],[-0.07,-0.04],[0.25,0.4],[],[]};%number of elements of
 this cell array is the same
%as the number of input parameters. If the nth variable is to have
 limits
%imposed on its values during the fit, we put 2 element array in the
 nth
%element of the plims cell array. If a parameter's value is to be
%unbounded, we put an empty array in the nth element of plims
```

```
%So in the above we have specified limits for the scale factor and the
 two
%Js, but not for the single ion anisotropy or the background.

%Now need to be careful about how we specify the inputs. For any
 parameter
%that we use bounds for, it is converted into a phase factor for the
%expression
% par = lower_bound + (0.5 * (sin(phase) + 1)) * (upper_bound -
 lower_bound)

%Initialise the values:
pars_in=[scalefac,J,D,bg];

%Convert those parameters with bounds to phase factors:
for i=1:numel(plims)
    if ~isempty(plims{i})
        sp=(pars_in(i) - plims{i}(1))./(0.5*(plims{i}(2) - plims{i}
(1))) -1;
        p=asin(sp);
        pars(i)=p;
    else
        pars(i)=pars_in(i);
    end
end

%run the fit, with a special fitting function that is expecting the
%parameters to be input in this new form
[wfit4,fitdata4]=multifit(mnf2_IX1d,@spinw_mnf2_1dfit_limits,
{pars,plims,Qrange,ok1d,Ei,dE,dQ,s},...
    [1,1,1,0,0],'list',1);

%Plot data with fit over the top (pl means plot line in Horace)
acolor black
plot(mnf2_IX1d)
acolor red
pl(wfit4)

%We can see that by constraining values we end up with a better fit
 than we
%did with unbounded parameters for the same initial guess.

%Convert the fit parameters and errors back to physically meaningful
 values:
for i=1:numel(pars)
    if ~isempty(plims{i})
        p(i)=plims{i}(1) + (0.5.*(sin(fitdata4.p(i)) + 1)).*(plims{i}
(2) - plims{i}(1));
        sig(i)=(0.5.*cos(fitdata4.p(i))).*(plims{i}(2) - plims{i}
(1)).*fitdata4.sig(i);
    else
        p(i)=fitdata4.p(i);
        sig(i)=fitdata4.sig(i);
    end
```

```
end

p
sig


%This has worked remarkably well, and we have recovered the correct
%answer for J.
```

```
-------------------------------------
Beginning fit (max 30 iterations)
-------------------------------------
Iteration  Time(s)  Reduced Chi^2
     0         3.000   1167.5846
     1        14.953   1003.0511
     2        27.937    210.7352
     3        43.213    174.7240
     4        57.413    129.5926
     5        70.749    119.7311
     6        83.735    118.8349
     7        96.422    122.6167
     7        99.758    122.6166
     7       103.146    122.6055
     7       106.465    122.4598
     7       109.461    118.6456
     8       121.700    112.2680
     9       134.458    112.9873
     9       138.113    112.9854
     9       141.807    112.7807
     9       145.443    111.4459
    10       159.669    107.8986
    11       174.130    106.3149
    12       186.194    105.9214
    13       198.199    101.2032
    14       210.245    101.6400
    14       213.227    101.6385
    14       216.190    101.6296
    14       219.163    101.4091
    14       222.177    101.2081
    14       225.179    101.2058
 *** No improvement in chi-squared over previous iteration ***
    13       225.179    101.2032
Fit converged

p =

    6.2508   -0.0688    0.3121         0    0.6000


sig =

    0.2375    0.0024    0.0007         0         0
```



Fitting with SpinW - part 4 - what to do if you have no idea of good parameter values? Brute force method

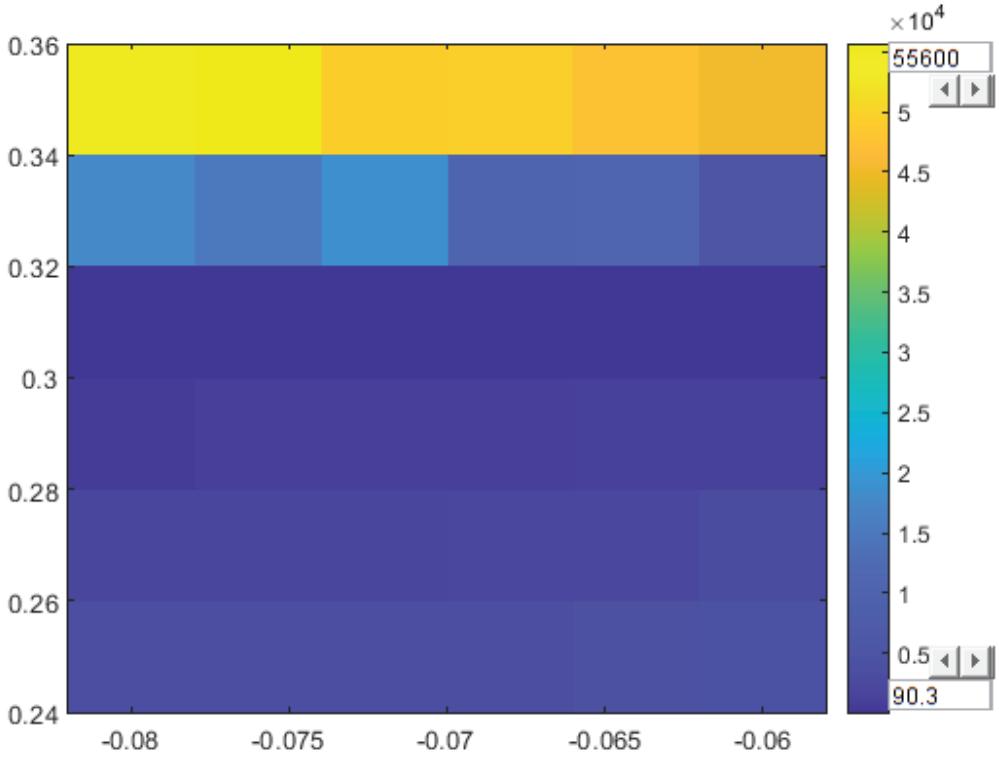
```
%Because we only have 2 exchange parameters that we're interested in,
 we
%can try a brute force approach of trying a grid of values for J1 and
 J2,
%and then look at a colour map of chi^2 plotted against these two to
 see if
%we can identify approximately the best fit. This can be useful if one
 has
%local minima in chi^2 space. However this approach is hard if you
 have
%more than 2 or 3 exchange parameters.

%Remember our initial guess from last time:
scalefac=9;%intensity scale factor
J=[-0.06,0.32];%exchange guess values (remember correct vals are
 -0.0575 and 0.3161)
D=0;%single ion anisotropy keep zero
bg=0.6;%keep this fixed for now.
```

```
%Specify lower and upper limits for a parameter:
plims={[5,9],[],[],[],[]};

%Now we will fix J1 and J2 to be a mesh of values, and for each case
 we
%will fit the scale factor (within our constraints), and then log the
 value
%of chi^2. Be careful here though - if your mesh is too fine (lots of
%points) even fitting a single parameter (the scale factor) could be
 very
%time consuming. One option is to start with a coarse mesh to check
 the lay
%of the land, and then maybe go finer in a more focused region if that
 is
%appropriate. But if you have several local minima you may need to
 have a
%fine mesh over a wide range anyway, and accept the time penalty.

%Start coarse (NB on a moderate spec laptop this takes NNNN seconds)
J1mesh=[-0.07:0.01:-0.04];
J2mesh=[0.1:0.1:0.5];

%Then go finer later if desired (takes NNNN seconds on same laptop)
% J1mesh=[-0.07:0.003:-0.04];
% J2mesh=[0.1:0.025:0.5];

[JJ1,JJ2]=ndgrid(J1mesh,J2mesh);

for n=1:numel(J1mesh)
    for m=1:numel(J2mesh)
        %Initialise the values:
        pars_in=[scalefac,J1mesh(n),J2mesh(m),D,bg];

        %Convert those parameters with bounds to phase factors:
        for i=1:numel(plims)
            if ~isempty(plims{i})
                sp=(pars_in(i) - plims{i}(1))./(0.5*(plims{i}(2) -
 plims{i}(1))) -1;
                p=asin(sp);
                pars(i)=p;
            else
                pars(i)=pars_in(i);
            end
        end


 [wfit5(n,m),fitdata5(n,m)]=multifit(mnf2_IX1d,@spinw_mnf2_1dfit_limits,
{pars,plims,Qrange,ok1d,Ei,dE,dQ,s},...
            [1,0,0,0,0],'list',1);
        chisq(n,m)=fitdata5(n,m).chisq;
    end
end

%Plot the result:
```

```
chisq_IX=IX_dataset_2d(J1mesh,J2mesh,chisq);
plot(chisq_IX)

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

%Then go finer later if desired (takes NNNN seconds on same laptop)
J1mesh=[-0.08:0.004:-0.06];
J2mesh=[0.25:0.02:0.35];
[JJ1,JJ2]=ndgrid(J1mesh,J2mesh);

clear wfit5 fitdata5 chisq
for n=1:numel(J1mesh)
    for m=1:numel(J2mesh)
        %Initialise the values:
        pars_in=[scalefac,J1mesh(n),J2mesh(m),D,bg];

        %Convert those parameters with bounds to phase factors:
        for i=1:numel(plims)
            if ~isempty(plims{i})
                sp=(pars_in(i) - plims{i}(1))./(0.5*(plims{i}(2) -
 plims{i}(1))) -1;
                p=asin(sp);
                pars(i)=p;
            else
                pars(i)=pars_in(i);
            end
        end


 [wfit5(n,m),fitdata5(n,m)]=multifit(mnf2_IX1d,@spinw_mnf2_1dfit_limits,
{pars,plims,Qrange,ok1d,Ei,dE,dQ,s},...
            [1,0,0,0,0],'list',1);
        chisq(n,m)=fitdata5(n,m).chisq;
    end
end

%Plot the result:
chisq_IX=IX_dataset_2d(J1mesh,J2mesh,chisq);
plot(chisq_IX)

%With a longer time one could of course plot a finer mesh, but this is
%enough right now for us to make some progress.
```

```
-------------------------------------
Beginning fit (max 30 iterations)
-------------------------------------
Iteration  Time(s)  Reduced Chi^2
     0        3.252   55820.1701
     1        9.467   47830.4961
     2       16.058   25602.6042
     3       23.713   46157.9739
     3       27.368   48348.9743
     3       30.944   54266.5995
     3       34.947   55561.4959
```

```
Fit converged
```



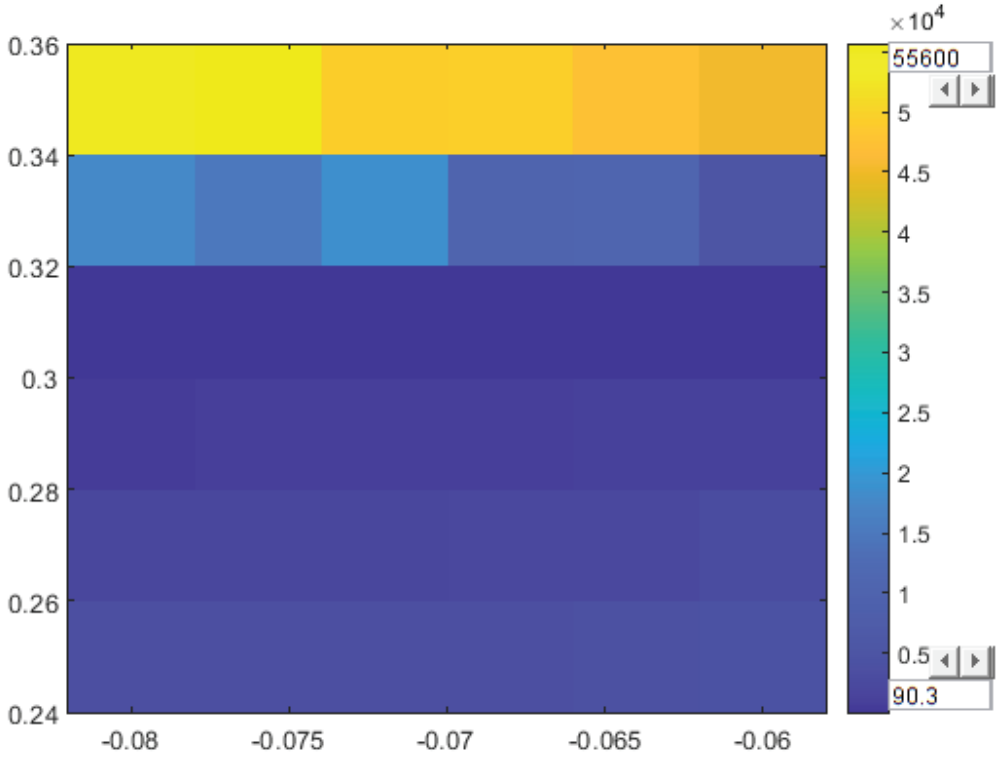
Fitting with SpinW - part 5 - alternative approaches to fitting - particle swarm optimisation

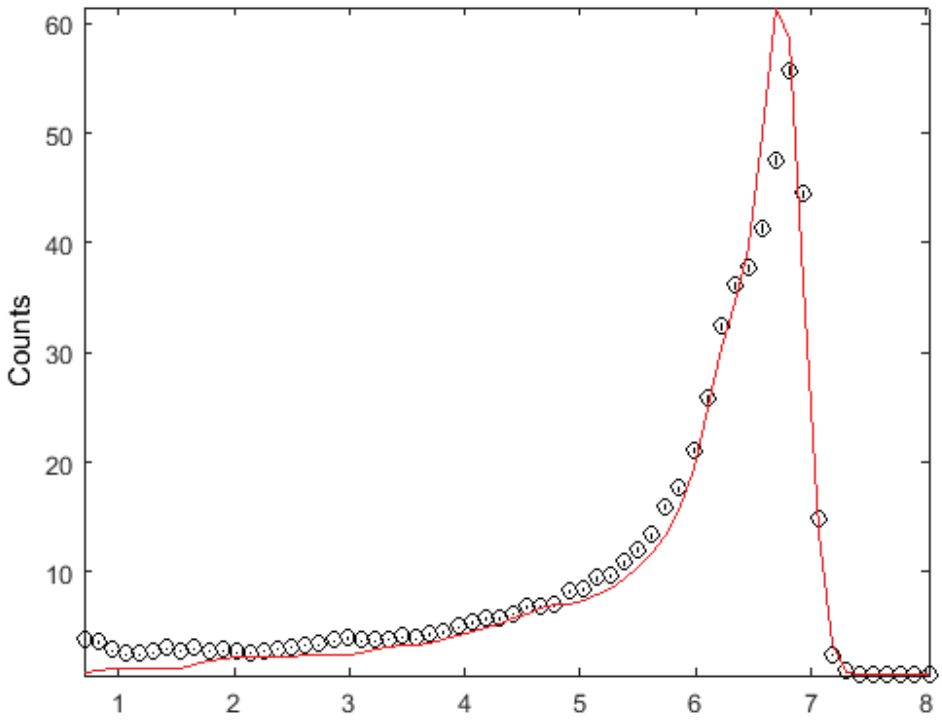
```
%Here we will use particle swarm optimisation - this is useful for
 cases
%with multiple local minima but one global minimum. See the following
%webpage for a basic introduction to the method:
%https://en.wikipedia.org/wiki/Particle_swarm_optimization

%SpinW comes with a version of a particle swarm optimiser already
%installed. It is used when searching for the magnetic ground state of
 a
%system with known exchange couplings. In order to access these
 routines
%separately we must call them in a specific way. To check that you
 have
%these routines available to you, type:

which ndbase.pso
```

```
%if you get a return message of a directory on your computer, you are
 good
%to go. If not, you may need to install a more recent version of SpinW

dat.x=mnf2_IX1d.x(1:end-1)'; dat.y=mnf2_IX1d.signal;
 dat.e=mnf2_IX1d.error;

func=@spinw_mnf2_1dfit_pso;

scalefac=9;%intensity scale factor
%J=[-0.06,0.32];%exchange guess values (remember correct vals are
 -0.0575 and 0.3161)
J=[-0.03,0.5];%guess that is further from the mark...
D=0;%single ion anisotropy keep zero
bg=0.6;%keep this fixed for now by setting upper and lower bounds to
 be identical

ok1d=~isnan(mnf2_IX1d.signal);
Qrange=[1,2];
Ei=12;
dE=0.36;
dQ=0.05;
rnseed=1;

%pars=[scalefac J D bg Qrange Ei dE dQ rnseed];
%pars=[6.2410 -0.0677 0.3086 D bg Qrange Ei dE dQ rnseed];%partial
 results after 50 iterations from above starting params
pars=[6.2567 -0.0695 0.3085 D bg Qrange Ei dE dQ rnseed];
upbound=[9 -0.04 0.4 0 0.6 Qrange Ei dE dQ rnseed];%NB if we do not
 set bounds then the default is 1e5
lowbound=[5 -0.07 0.25 0 0.6 Qrange Ei dE dQ rnseed];%default is -1e5

popsize=ceil(25+1.4*3);%optimal population size of swarm = 25 +
 1.4*(No. free params) - round up

[pOpt,fVal,stat] =
 ndbase.pso(dat,func,pars,'lb',lowbound,'ub',upbound, 'PopulationSize',
 popsize,...
    'MaxIter',5, 'TolX',1e-2, 'TolFun', 1);
%set the maximum number of iterations = 5 here, to test the routine's
 correct operation
%Do not expect convergence with so few iterations - if this is not set
 then
%the default is 100 * number input parameters.
%Similarly we make the tolerance on the parameter changes to be 1%, to
 get
%faster convergence.
%
%A bit of hand-crafting is needed on the function tolerance, which is
 defined as
%the difference between best and worst function evaluation in the
 simplex is
%is smaller than TolFun.
%This is the *absolute* difference in chi^2 between the best and worst
```

```
%particle in the population. i.e. we only have convergence if all of
 the
%particles have found minima in chi^2 within the value specified by
 TolF
%

%We find that with only 5 iterations convergence is not achieved,
 however
%we do find that from a distant parameter guess we have found
 something
%quite close to the correct set of parameters already.

%Repeat, with more iterations allowed:
[pOpt,fVal,stat] =
 ndbase.pso(dat,func,pars,'lb',lowbound,'ub',upbound, 'PopulationSize',
 popsize,...
    'MaxIter',50, 'TolX',1e-2, 'TolFun',10);

% stat =
%
%   struct with fields:
%
%           msg: 'Maximum Number of iterations is reached without
 convergence!'
%       warning: 1
%             p: [6.2567 -0.0695 0.3085 0 0.6000 1 2 12 0.3600 0.0500
 1]
%          sigP: []
%         redX2: 48.1374
%

%In this example we did not actually reach convergence. However if we
 plot
%the result we see that the solution looks quite good, and if nothing
 else
%we can say that we have found a good starting point for doing a
 "normal"
%least squares fit as described previously.

wfit6=IX_dataset_1d(mnf2_IX1d.x,fVal);
acolor black
plot(mnf2_IX1d);
acolor red
pl(wfit6);

%As a final comment:
%A key consideration for more complex datasets
%is the possibility that there will be many more than 3 parameters to
%optimize, in which case the time taken to complete the fit may be
 much
%longer. Absolutely critical in getting to a valid result is a
 sensible
%choice of upper and lower bounds on each parameter's value.
```

```
C:\mprogs\SpinW\spinw-master\spinw-master\swfiles\+ndbase\pso.m  %
 static method or package function
Warning: Convergence is not reached!
Warning: Convergence is not reached!
```





Published with MATLAB® R2019b