ASSIGNMENT 10: SOLUTION

E. ARNOLD, M-S. LIU, R. PRABHU & C.S. RICHARDS
THE UNIVERSITY OF CAMBRIDGE

Written in XeLaTeX

THE W.I.M.

PREREQUISITES

The chapter *Spin Echo* from the *Theory Handbook*.

INTRODUCTION

Helium spin echo spectroscopy is a modern surface technique, allowing measurements of surface processes to be taken to greater depth than ever before. This tutorial will introduce how to simulate some experimental results that this technique can generate. Simulations are important for understanding the physics; they allow us to check our models, to see how well the model fits with the experimental data. We will focus on simulating the "wavelength intensity matrix" for a few systems.

THE WAVELENGTH INTENSITY MATRIX

One of the quantities that the helium spin echo technique can measure is the so-called "wavelength intensity matrix". If this is not familar to you, read the prerequisites to this tutorial. While the *Theory Handbook* covers this in full, we will only briefly discuss it here; a lengthy discussion is not appropriate for this document. The measured polarisation $P(\kappa_1, \kappa_2)$ in helium spin-echo spectroscopy can be related to the wavelength intensity matrix $I(\lambda_1, \lambda_2)$ by

$$P(\kappa_1, \kappa_2) \propto \iint I(\lambda_1, \lambda_2) \exp\left(2\pi i \kappa_1 \lambda_1 + 2\pi i \kappa_2 \lambda_2\right) \mathrm{d}\lambda_1 \mathrm{d}\lambda_2 + C,$$

where:

1. κ_1 and κ_2 are experimentally controllable parameters, proportional to the current through the machine's coils;
2. λ_1 and λ_2 are the de Broglie wavelengths of the incident and scattered beams,
3. C is a constant term (effectively noise); this is measured separately, and removed.

The object of interest in this document is the wavelength intensity matrix $I(\lambda_1, \lambda_2)$. This is reconstructed from the Fourier transform of the measured polarisation; the techniques for processing it are introduced in the *Theory Handbook*, but we will discuss it in this document - allowing the reader to develop a useful skill set for helium spin-echo spectroscopy.

How do we make sense of the wavelength intensity matrix? It is very fortunate that we can interpret it as the relative intensity of the beam that has a wavelength of λ_1 in the first arm and λ_2 in the second arm. This interpretation allows us to quantify the matrix as the product of two distributions: first the wavelength distribution $\rho(\lambda_1)$ in the incident beam, and then the probability of the de Broglie wavelength changing from λ_1 to λ_2 during scattering $S(\lambda_1 \rightarrow \lambda_2)$, given the incident beam has a wavelength of λ_1. Thus we can write

$$I(\lambda_1, \lambda_2) = \rho(\lambda_1) S(\lambda_1 \rightarrow \lambda_2).$$

How do we use this? Simulating both of these quantities, we can construct an estimate of the appearance of the wavelength intensity matrix. The matrix can be simultaneously plotted as a function of both the incident de Broglie wavelength λ_1, and the scattered de Broglie wavelength λ_2. Information from this plot can be compared to experiment. The code given with this document generates the transition function S from the energy of a mode, and then plots the wavelength intensity matrix using the same information.

The Tilted Projection Theorem

We have included this section as a reminder that the tilted projection theorem exists. It is detailed in significant detail in the *Theory Handbook*. We will use it here!

THE CODE

This assignment is accompanied by a set of code files. The use of this code is simple, using only two lines of code. We will list this code, then discuss how the program works. Run:

```
1 [lambda_i_Mat, lambda_f_Mat,wavelengthIntMat, tilt] = SEexpTools
      .calcMeasurementsParams(8, 0.5, 6, 30, 44.4);
2 [lambda_1D, lambda_axis_shifted, wavelengthInt_proj,energy,
      spectrum_in_energy] = SEexpTools.projectByTilt(lambda_i_Mat,
       lambda_f_Mat, wavelengthIntMat,tilt, 8);
```

which will prompt the program to ask for a small number of user inputs. Following the on-screen guidance, a set of figures are generated. These are:

1. A plot of each phonon mode as a function of the parallel momentum transfer from the helium-3 particle.
2. A plot of the wavelength intensity matrix that these phonon modes generate.
3. Plots of the projected wavelength intensity matrix in both the energy domain, and in the wavelength domain.

Part of the task below will be to investigate how changing the surface modes leads to a change in the projected matrix.

It is important to know what the variables used in the code above are. In order, in line 1, the arguments are:

1. The mean beam energy,
2. The beam energy full width at half maximum (FWHM),
3. The maximum energy transfer of the incident helium-3 particle,
4. The maximum angle of incidence for the helium-3 particle,
5. The total scattering angle.

All of the energies used are in meV. Varying the input parameters (along with the mean beam energy as inputted into line 2) will reveal the effect of a slightly different setup.

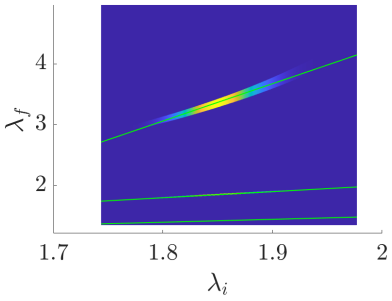

Figure 1 | The tilted projection axes for three features on a model wavelength intensity matrix.

TASKS

In these tasks, we will individually explore how to create the wavelength intensity matrix for a small number of events. We will then discuss the tilted projection theorem.

Typical beam wavevectors for the Cambridge helium-3 spin-echo spectrometer are between 3.4 and 4.2 inverse angstroms. Assume that the wavevector for a particlular setup has a mean of 4 angstroms. Assume that the standard deviation of the velocities of the particles in the beam is 3% (this is not the real value).

1. State the standard deviation for the wavevectors of the particles in the beam.
2. Calculate the range of wavelengths of the particles in the beam. How can the wavelength of the particles in the beam be changed?
3. The helium beam is incident on a sample. Assume that, for this task, that all of the scattering events are elastic.
   a) Consider a single particle in the beam. Upon collision, what is the emergent de Broglie wavelength of the particle? Give your answer relative to the incident wavelength.
   b) Assume that, regardless of wavelength, the particles are equally likely to emerge in the correct direction to be detected. Suggest the form of the scattering distribution $S(\lambda_1 \rightarrow \lambda_2)$ for these events.
   c) Plot the wavelength density matrix for this system as a colour map in 2D. Use the code file given to do this. Consider how

to define a new mode in the correct place, along with the code that must accompany it.

4. Assume that, instead of always scattering elastically, that the scattering distribution $S(\lambda_1 \rightarrow \lambda_2)$ takes a different form. Describe how the inclusion of inelastic processes affects the shape of the wavelength intensity matrix.
5. Modify the code to simulate the wavelength intensity matrix for a mode with an energy distribution independent of momentum transfer. Produce this plot. Describe why it is likely to be an oversimplification of the system.

EXTENSIONS

Include and exclude various modes, and comment on the consequence this has on the wavelength intensity matrix. Repeat the same task for varying the system parameters. Then study the effect of varying the modes used, the system constants, and the mode constants on the tilted projection measurements.

When a spin echo measurement is made, it is done so along a tilted axis. Therefore it is important to understand how the behaviour of a scattering event affects the tilted projection of the wavelength intensity matrix!

SUMMARY

In this tutorial, you should have learned how to modify an existing piece of code that simulates the wavelength intensity matrix. You should have become more comfortable with the tilted projections produced by wavelength intensity matrices.

SOLUTIONS

THE W.I.M.: MAIN TASKS

Task 1: Question

State the standard deviation for the wavevectors of the particles in the beam.

Task 1: Solution

The wavevector of each particle in the beam is inversely proportional to its velocity. Therefore given the standard deviation in the velocities is 3%, the standard deviation in the wavevectors is also 3%.

Task 2: Question

Calculate the range of wavelengths of the particles in the beam. How can the wavelength of the particles in the beam be changed?

Task 2: Solution

Irrespective of whether the limits of 3.4 and 4.2 inverse angstroms hold as the true values for the cut-off points of the wavevectors, we will use them to model the limits of the distribution. Noting simply the wavevector to be defined as $2\pi/\lambda$, where λ is the wavelength of the particle in question, the range of the wavelengths can be calculated to be 1.5 to 1.8 angstroms.

Task 3: Question

The helium beam is incident on a sample. Assume that, for this task, that all of the scattering events are elastic.

a) Consider a single particle in the beam. Upon collision, what is the emergent de Broglie wavelength of the particle? Give your answer relative to the incident wavelength.

b) Assume that, regardless of wavelength, the particles are equally likely to emerge in the correct direction to be detected. Suggest the form of the scattering distribution $S(\lambda_1 \rightarrow \lambda_2)$ for these events.

c) Plot the wavelength density matrix for this system as a colour map in 2D. Use the code file given to do this. Consider how to define a new mode in the correct place, along with the code that must accompany it.

Task 3: Solution

For the elastic event, the incoming and outgoing particles have the same de Broglie wavelength. If they did not, then there would have been an energy transfer. When interpreting this using the wavelength intensity matrix, we must consider the transition function $S(\lambda_1 \rightarrow \lambda_2)$. This is a delta function, centred on the incoming wavelength.

How do we plot the elastic feature? We must modify the code given with the tutorial to achieve what we desire it to. It is trivial to modify the small number of steps required to generate the output.

It is important not to overcomplicate this task. It is very simple. All that is needed to be changed in the code file is a small part of the function called `calcMeasurementsParams`. The code below, which can be seen as the original version in the appendices, calculates the scattering function for a particular system. A studious reading of the phonon models late in the provided class will reveal the mode denoted `SpecBroad` represents an elastic mode. Therefore it is the only mode we require to represent a scattering event!

```
1            %------------------------------------------------
2            %% find (dk, de) points of phonon
3            %------------------------------------------------
4
5            % SpecBroad is the elastic mode: include only this
                 mode
6            modeModelHandle = SEexpTools.phononModel('SpecBroad'
                 );
7            modeDE_SpecBroad = modeModelHandle(dK_Mat);
8
9            % find the scattering function
10           energyMatchMat = zeros(size(dE_Mat));
11           energyMatchMat = energyMatchMat | abs(
                 modeDE_SpecBroad-dE_Mat) < 1e-1 | abs(
                 modeDE_SpecBroad+dE_Mat) < 1e-1;
12
13           beamIntensityMat = repmat(beamIntensity,size(
                 energyMatchMat,1),1);
14
15           % find the wavelength intensity matrix
16           wavelengthIntMat = energyMatchMat .*
                 beamIntensityMat;
17
18           % plot the wavelength intensity matrix as a colour
                 map
19           figure; hold on
20           pcolor(lambda_i_Mat, lambda_f_Mat,wavelengthIntMat);
                  shading flat
21           ...
```

Following through with plotting the optimal tilted measurement axis gives the figure in the margin.

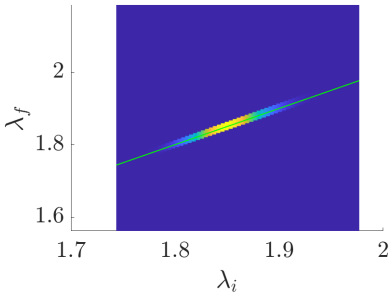

Figure 2 | The wavelength intensity matrix generated for a system where only elastic scattering events occur.

Task 4: Question

Assume that, instead of always scattering elastically, that the scattering distribution $S(\lambda_1 \rightarrow \lambda_2)$ takes a different form. Describe how the inclusion of inelastic processes affects the shape of the wavelength intensity matrix.

Task 4: Solution

If the scattering distribution is no longer elastic, then it is no longer a Dirac delta function. It is now the case that the wavelength intensity matrix displays the three regions in figure 1: one elastic region, surrounded by two (or more) inelastic regions.

Task 5: Question

Modify the code to simulate the wavelength intensity matrix for a mode with an energy distribution independent of momentum transfer. Produce this plot. Describe why it is likely to be an oversimplification of the system.

Task 5: Solution

The final task is to introduce a constant energy mode. Reading the function `phononModel`, it should be evident that the mode denoted `WaterOnNi2meV` is appropriate. Therefore again modifying the same section of code as in question 3, running the code below gives the figure in the margin

```
%-----------------------------------------------------
%% find (dk, de) points of phonon
%-----------------------------------------------------

% SpecBroad is the elastic mode: include only this mode
modeModelHandle = SEexpTools.phononModel('SpecBroad');
modeDE_SpecBroad = modeModelHandle(dK_Mat);

% initialise the constant energy mode
modeModelHandle = SEexpTools.phononModel('WaterOnNi2meV');
modewater = modeModelHandle(dK_Mat);

% find the scattering function
energyMatchMat = zeros(size(dE_Mat));
energyMatchMat = energyMatchMat | abs(modeDE_SpecBroad-dE_Mat) < 1e-1 | abs(modeDE_SpecBroad+dE_Mat) < 1e-1;
```

```
16              energyMatchMat = energyMatchMat | abs(modewater-
                    dE_Mat) < 1e-1 | abs(modewater+dE_Mat) < 1e-1;
17
18
19              beamIntensityMat = repmat(beamIntensity,size(
                    energyMatchMat,1),1);
20
21              % find the wavelength intensity matrix
22              wavelengthIntMat = energyMatchMat .*
                    beamIntensityMat;
23
24              % plot
25              figure; hold on
26              pcolor(lambda_i_Mat, lambda_f_Mat,wavelengthIntMat);
                     shading flat
27              ...
```

There are likely to be many modes appearing during scattering. In addition, we have only included zeroth order modes. Thus it is expected our system is an oversimplification.

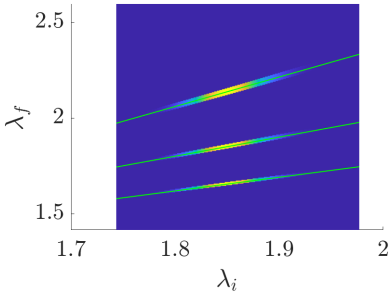

Figure 3 | The wavelength intensity matrix produced when a mode with energy independent of momentum transfer is included.

APPENDICES

The files for use in this problem are listed below.

A: ENERGY2WAVEVECTOR

This takes an energy distribution, and returns the wavevectors to which it corresponds.

```
function [k] = energy2wavevector(E, mass)
% function [k] = energy2wavevector(E, mass)
%
% function to convert the energy of atoms in a helium beam to the associated wavevector
%
% k is in 1/Angstrom
% E is in meV
% mass is either 3 or 4 amu



% load in parameters
if ~exist('SE_amu','var'), load_chess_parameters;

% end condition
end



% if mass is 3 amu i.e. if particle is helium 3, then find mass in SI
if mass==3
    m = SE_amu * 3.01603;

% if mass is 4 amu i.e. if particle is helium 4, then find mass in SI
elseif mass==4
    m = SE_amu * 4.00260;

end



%find energy in SI
Ei_SI = E/1000*SE_e;

% find wavevector in SI
% in inverse angstroms
```

```
ki_SI = (2*m*Ei_SI).^0.5/SE_hbar;

% convert to inverse angstroms
k = ki_SI/1e10;
```

B: ENERGY2WAVELENGTH

Similarly to the above, this function takes an energy distribution, and returns the corresponding wavelengths.

```
% function lambda = energy2wavelength(energy,mass)
%
% function to convert energy (in meV) to wavelength (in Angstrom)
% wavelength is returned in Angstroms



function lambda = energy2wavelength(energy, mass)

% h is Planck's constant
h = 6.62608e-34;

% e is the charge on the electron
e = 1.60218e-19;

%c is the speed of light
c = 2.99792e8;

% load in a file of constants
if ~exist('SE_amu','var'), load_chess_parameters;

% end condition
end




% if mass is 3 amu i.e. if particle is helium 3
% converts mass to kg
if mass==3
    m = SE_amu * 3.01603;


% if mass is 4 amu i.e. if particle is helium 4
% converts mass to kg
elseif mass==4
    m = SE_amu * 4.00260;


% helium only has two isotopes - anything else is not allowed
else
    disp('Only masses 3 and 4 allowed')

    %return nothing
    return

% end conditions
```

```
49  end
50
51
52
53
54  % calculate the energy in SI units
55  % change from eV to joules
56  E_SI = energy ./ 1000*SE_e;
57
58  % calculate wavelength in metres
59  % this is the output
60  % multiply by e10 to produce angstrom output
61  lambda = SE_h ./ sqrt(2*m*E_SI) * 1e10;
```

C: SPIN ECHO EXPERIMENT TOOLS

This file brings together a large quantity of theory. For a given set of surface modes, it calculates, and plots, the projected wavelength intensity matrix along the optimal axis for measurement. Do not worry about understanding it line-by-line, but appreciate the output it produces using the method detailed previously.

```
1   %class spin echo experiment tools
2
3   classdef SEexpTools
4       %UNTITLED2 Summary of this class goes here
5       %   Detailed explanation goes here
6
7       properties
8           % there are no properties
9       end
10
11
12      % define the functions, one by one
13      % This class contains
14      % - calcMeasurementsParams
15      % - projectByTilt
16      % - OneD2finalLambda
17      % - wavelength2energy
18      % - calcMinWidthForModeInTilt
19      % - phononModel
20      % - plotModels
21
22
23
24      methods(Static)
25
26
27          %-------------------------------------------------------
28          % define calcMeasurementsParams function
29          %-------------------------------------------------------
30          function [lambda_i_Mat, lambda_f_Mat,wavelengthIntMat,
                tilt] = calcMeasurementsParams(E0, FWHM,
                max_dEexchange, theta_i, theta_tot)
31              %calcMeasurementsParams to calculate the
                    measurements needed to map a specific mode.
32              % modeModelHandle - handle to function which gets a
                    dK, and gives dE.
```

```
33              % E0 - the beams mean initial energy (in meV)
34
35              % load in the constants
36              load_chess_parameters;
37
38              % define the number of points to use
39              numOfPoints = 1e3;
40
41              %-------------------------------------------------
42              %% find beam params (energy, wave-vector, profile)
43              %-------------------------------------------------
44              % incident wavelength matrix - a uniform
                    distribution, later converted to gaussian
45              lambda_i_tmp = linspace(energy2wavelength(E0+2*FWHM
                    ,3), energy2wavelength(E0-2*FWHM,3), numOfPoints
                    );
46
47              % calculate the energy of a helium-3 particle with
                    wavelength lambda_i_tmp
48              Ei_vec = SEexpTools.wavelength2energy(lambda_i_tmp,
                    3);
49
50              % calculate the wavevector of a helium-3 particle
                    with energy Ei_vec
51              ki_vec = energy2wavevector(Ei_vec,3);
52
53              % convert FWHM of particle energies to standard
                    deviation
54              c_param = FWHM/(2*sqrt(2*log(2)));
55
56              % the energies of particles in the beam follow a
                    normal distribution
57              % calculate the relative intensity of a particle
                    with energy E_i compared to unity, building a
                    beam distribution
58              beamIntensity = exp(-(E0-Ei_vec).^2./(2*c_param.^2))
                    ;
59
60              % scattered (final, denoted f) wavelength matrix for
                     helium 3
61              % maximum in the matrix is the wavelength of the sum
                     of the maximum value in Ei_vec and the maximum
                    energy transfer
62              % minimum in the matrix is the wavelength of the
                    maximum of either the minimum value of Ei_vec
                    minus the maximum energy transfer, or 0.1
63              % there are numOfPoints points in the matrix
64              lambda_f_tmp = linspace(energy2wavelength(max(Ei_vec
                    )+max_dEexchange, 3), energy2wavelength(max(min(
                    Ei_vec)-max_dEexchange, 1e-1), 3), numOfPoints);
65
66              % the energies of the particles in the scatted beam
67              Ef_vec = SEexpTools.wavelength2energy(lambda_f_tmp,
                    3);
68
69              % the wavevectors of the particles in the scatted
                    beam
70              kf_vec = energy2wavevector(Ef_vec,3);
71
72              % set up a mesh for the wavelengths using the
                    incident and scattered waves
```

```
73              [lambda_i_Mat, lambda_f_Mat] = meshgrid(lambda_i_tmp
                    ,lambda_f_tmp);
74
75              % mesh for associated wavevectors
76              [ki_Mat, kf_Mat] = meshgrid(ki_vec, kf_vec);
77
78              % mesh for associated energies
79              [Ei_Mat, Ef_Mat] = meshgrid(Ei_vec,Ef_vec);
80
81              % slightly randomise the incident angle - adds noise
82              theta_i_Mat = theta_i + (rand(size(Ei_Mat))-0.5)
                    *0.2;
83
84              % outward angle = angle of scattering - angle of
                    incidence
85              theta_out_Mat = theta_tot - theta_i_Mat;
86
87              % change in energy is final energy - initial energy
88              dE_Mat = Ef_Mat-Ei_Mat;
89
90              % change in wavevector is given by the formula for
                    momentum transfer
91              dK_Mat = kf_Mat.*sin(theta_out_Mat/180*pi)-ki_Mat.*
                    sin(theta_i_Mat/180*pi);
92              %-------------------------------------------------
93              %-------------------------------------------------
94
95
96              %-------------------------------------------------
97              %% find (dk, de) points of phonon
98              %-------------------------------------------------
99
100             % load in a phonon model from a later function. This
                     is an anonymous function, describing the energy
                     transfer of the phonon as a function of
                    momentum transfer.
101             modeModelHandle = SEexpTools.phononModel('SpecBroad'
                    );
102
103             % modeModelHandle is the variable defined on the
                    line above. Thus find the value of the mode at
                    each possible momentum transfer.
104             modeDE_SpecBroad = modeModelHandle(dK_Mat);
105
106             % load in a phonon model from a later function.
107             modeModelHandle = SEexpTools.phononModel('
                    RayleighWaveNi111');
108
109             % modeModelHandle is the variable defined on the
                    line above. Thus find the value of the mode at
                    each possible momentum transfer.
110             modeDE_RW = modeModelHandle(dK_Mat);
111
112             % load in a phonon model from a later function.
113             modeModelHandle = SEexpTools.phononModel('
                    WaterOnNi6meV');
114
115             % modeModelHandle is the variable defined on the
                    line above. Thus find the value of the mode at
                    each possible momentum transfer.
116             modeDE_W6 = modeModelHandle(dK_Mat);
```

```
117
118
119
120             % uncomment below to calculate more modes
121
122             % modeModelHandle = SEexpTools.phononModel('
                    WaterOnNi2meV');
123             % modeDE_W2 = modeModelHandle(dK_Mat);
124
125             % modeModelHandle = SEexpTools.phononModel('
                    NiMagnons');
126             % modeDE_NiMagnons = modeModelHandle(dK_Mat);
127
128             % modeModelHandle = SEexpTools.phononModel('
                    LongitudinalWaveNi111');
129             % modeDE_LA = modeModelHandle(dK_Mat);
130
131             % modeModelHandle = SEexpTools.phononModel('
                    AcousticSurfacePlasmons');
132             % modeDE_ASP = modeModelHandle(dK_Mat); %modeDE_ASP(
                    find(abs(modeDE_ASP) > 15))=0;
133
134             %-------------------------------------------------
135             %-------------------------------------------------
136
137
138
139             %-------------------------------------------------
140             % find the points in which the scan line crosses the
                     modes (modeDE and -modeDE)
141             %-------------------------------------------------
142
143             % define a matrix of zeros the size of the matrix of
                     energy transfers
144             % this is the precursor to the transfer function
145             energyMatchMat = zeros(size(dE_Mat));
146
147
148
149             % decice whether to set the entries of the matrix as
                     0 or 1
150             energyMatchMat = energyMatchMat | abs(
                    modeDE_SpecBroad-dE_Mat) < 1e-1 | abs(
                    modeDE_SpecBroad+dE_Mat) < 1e-1;
151
152             % and again, for a different mode
153             energyMatchMat = energyMatchMat | abs(modeDE_RW-
                    dE_Mat) < 1e-1 | abs(modeDE_RW+dE_Mat) < 1e-1;
154
155             % and again, for a different mode
156             energyMatchMat = energyMatchMat | abs(modeDE_W6-
                    dE_Mat) < 1e-1 | abs(modeDE_W6+dE_Mat) < 1e-1;
157
158
159
160             % make a matrix the size of energyMatchMat that
                    contains copies of the beam intensity
161             % this is the transfer function
162             beamIntensityMat = repmat(beamIntensity, size(
                    energyMatchMat,1), 1);
163
```

```
164             % Using the WIM as the product of the possible
                    scattering function and beam distribution
165             wavelengthIntMat = energyMatchMat .*
                    beamIntensityMat;
166             %-------------------------------------------------
167             %-------------------------------------------------
168
169
170
171
172             %-------------------------------------------------
173             % plot each of the mode energies as a function of
                    momentum transfer
174             %-------------------------------------------------
175
176             % initialise figure
177             figure;
178
179             % hold on
180             hold on
181
182             % plot dK against modeDE_SpecBroad
183             plot(reshape(dK_Mat,1,[]), reshape(modeDE_SpecBroad
                    ,1,[]), 'b.', reshape(dK_Mat,1,[]), -reshape(
                    modeDE_SpecBroad,1,[]) ,'b.', 'MarkerSize', 2)
184
185             % plot dK against modeDE_RW
186             plot(reshape(dK_Mat,1,[]),reshape(modeDE_RW,1,[]),'b
                    .',reshape(dK_Mat,1,[]),-reshape(modeDE_RW,1,[])
                    ,'b.','MarkerSize',2)
187
188             % plot dK against modeDE_W6
189             plot(reshape(dK_Mat,1,[]),reshape(modeDE_W6,1,[]),'b
                    .',reshape(dK_Mat,1,[]),-reshape(modeDE_W6,1,[])
                    ,'b.','MarkerSize',2)
190
191             % plot dK against dE_Mat (which is the scan curve)
192             plot(reshape(dK_Mat,1,[]),reshape(dE_Mat,1,[]),'g.',
                    'MarkerSize',2)
193
194
195             % initialise a second figure
196             figure; hold on
197
198             % produce a pseudocolour plot of the wavelength
                    intensity matrix
199             pcolor(lambda_i_Mat, lambda_f_Mat, wavelengthIntMat)
                    ;
200
201             % specify the colouring on the plot
202             shading flat;
203
204             % find the indices of all positions where the
                    wavelength intensity matrix is non-zero
205             tmpIndx = find(wavelengthIntMat~=0);
206
207             % find the maximum value of lambda_f where the
                    wavelength intensity matrix is non-zero
208             maxNonZeroLambdaFinal = max(lambda_f_Mat(tmpIndx));
209
```

```
210             % find the minimum value of lambda_f where the
                    wavelength intensity matrix is non-zero
211             minNonZeroLambdaFinal = min(lambda_f_Mat(tmpIndx));
212
213             % find the maximum value of lambda_i where the
                    wavelength intensity matrix is non-zero
214             maxNonZeroLambdaIn = max(lambda_i_Mat(tmpIndx));
215
216             % find the minimum value of lambda_i where the
                    wavelength intensity matrix is non-zero
217             minNonZeroLambdaIn = min(lambda_i_Mat(tmpIndx));
218
219             % define the axes to be used: the ranges of lambda
                    specified above, with 10% leeway
220             axis([minNonZeroLambdaIn*0.9 maxNonZeroLambdaIn*1.1
                    minNonZeroLambdaFinal*0.9 maxNonZeroLambdaFinal
                    *1.1])
221
222
223
224             % user input for how many distinct features there
                    are displayed in the wavelength intensity matrix
                     plot on the screen
225             featureNum = inputdlg('how many features would you
                    like to analyse?'); featureNum = str2num(
                    featureNum{:});
226
227             % guidance for the user input
228             zvl = questdlg('Choose a point above all of the
                    features, then one point vertically between each
                     of the features you wish to analyse, then one
                    point below all of the features you wish to
                    analyse')
229
230             % find the coordinates of all points picked by the
                    user
231             [l1, l2] = ginput(featureNum + 1);
232             %-------------------------------------------------
233             %-------------------------------------------------
234
235
236
237
238             %-------------------------------------------------
239             % pick out and plot each feature. Calculate the
                    angle of tilt the feature makes to the lambda_i
                    axis
240             %-------------------------------------------------
241
242             % iterate through the second to last entry in the
                    set of coordinates above
243             for i=2:length(l2)
244
245                 % find the indices of all positions where the
                        final wavelength lies between the two y
                        coordinates picked by the user. This picks
                        out the all points from a single mode in the
                         wavelength intensity matrix.
246                 indx = find(lambda_f_Mat < l2(i-1) &
                        lambda_f_Mat > l2(i));
247
```

```
248                 % save the wavelengths of the indices just
                        described
249                 l1current = lambda_i_Mat(indx);
250                 l2current = lambda_f_Mat(indx);
251
252                 % find the values of the wavelength intensity
                        matrix where the final wavelength crosses
                        the scan curve
253                 intensityCurrent = wavelengthIntMat(indx);
254
255                 % collate the indices of all positions in the
                        slice of the wavelength intensity matrix
                        described above where the intensity is
                        greater than 0.05
256                 indx = find(intensityCurrent > 0.05);
257
258                 % fit a polynomial to the currents above
259                 p = polyfit(l1current(indx),l2current(indx),1);
260
261
262                 hold on;
263
264                 % plot each feature
265                 plot([min(l1current) max(l1current)], p(1)*[min(
                        l1current) max(l1current)]+p(2), 'g')
266
267                 % calculate angle of tilt of each feature
268                 tilt(i -1) = atan(p(1))*180/pi + 90;
269             end
270             %-------------------------------------------------
271             %-------------------------------------------------
272
273         end
274         %-----------------------------------------------------
275         %-----------------------------------------------------
276
277
278
279
280
281
282         %-----------------------------------------------------
283         % define projectByTilt function
284         %-----------------------------------------------------
285
286         % define a function to project a wavelength intensity
              matrix onto an angle
287         function [lambda_1D, lambda_axis_shifted,
              wavelengthInt_proj,energy,spectrum_in_energy] =
              projectByTilt(lambda_i_Mat, lambda_f_Mat,
              wavelengthIntMat_orig, tilt, E0)
288             % lambda_i_Mat - matrix of values for the wavelength
                    of the incident beam
289             % lambda_f_Mat - matrix of values for the wavelength
                    of the final beam
290             % wavelengthIntMat_orig - original wavelength
                    intensity matrix
291             % tilt - vector of tilt angles for each feature
292             % E0 - mean energy of helium beam
293
294
```

```
295             % initialise a vector for the projection variable on
                     each projection axis
296             lambda_axis_shifted=zeros(size(lambda_i_Mat, 2),
                    length(tilt));
297
298             % do the same for the energy
299             energy = zeros(size(lambda_i_Mat,2), length(tilt));
300
301             % initialise the vectors for the energies of the
                    incident beam
302             spectrum_in_energy = zeros(size(lambda_i_Mat, 2),
                    length(tilt));
303
304             % if the SE class variables exist, load them
305             if ~exist('SE_h','var'), load_chess_parameters; end
306
307             % initialise a new figure
308             h=figure;
309
310             % go through each possible tilt and project the WIM
311             for i=1:length(tilt)
312
313                 %-----------------------------------------------
314                 % project the wavelength intensity matrix onto a
                        line
315                 %-----------------------------------------------
316                 % rotation matrix
317                 R=[cosd(tilt(i)-90) -sind(tilt(i)-90) ; sind(
                        tilt(i)-90) cosd(tilt(i)-90)]
318
319                 % find the rotated version of the incident
                        wavelength matrix
320                 lambda_i_Mat_rotated = R(1)*lambda_i_Mat + R(2)*
                        lambda_f_Mat;
321
322                 % find the rotated version of the final
                        wavelength matrix
323                 lambda_f_Mat_rotated = R(3)*lambda_i_Mat + R(4)*
                        lambda_f_Mat;
324
325                 % interpolate to fit data to the wavelengths
326                 F = TriScatteredInterp(lambda_i_Mat_rotated(:),
                        lambda_f_Mat_rotated(:),
                        wavelengthIntMat_orig(:));
327
328                 % find indices of all wavelengths in the
                        original wavelength intensity matrix where
                        the intensity is larger than 0.05
329                 indx = find(wavelengthIntMat_orig > 0.05);
330
331                 % find, as a vector: the minimum and maximum of
                        the incident and scattered wavelengths along
                         the projection
332                 minMax = [min(min(lambda_i_Mat_rotated(indx)))-1
                         max(max(lambda_i_Mat_rotated(indx)))+1 min(
                        min(lambda_f_Mat_rotated(indx)))-1 max(max(
                        lambda_f_Mat_rotated(indx)))+1];
333
334                 % calculate the new wavelength distributions
                        projected onto the axis
```

```
335                 [lambda_i_Mat_new, lambda_f_Mat_new] = meshgrid(
                        linspace(minMax(1), minMax(2), size(
                        lambda_i_Mat, 1)), linspace(minMax(3),
                        minMax(4), size(lambda_i_Mat, 2)));
336
337                 % obtain rotated version of the wavelength
                        intensity matrix, using the interpolated
                        version of the wavelengths
338                 wavelengthIntMat_rotated = F(lambda_i_Mat_new,
                        lambda_f_Mat_new);
339
340                 % find the incides of all nan  (not a number) in
                         the rotated wavelength intensity matrix
341                 indx = isnan(wavelengthIntMat_rotated);
342
343                 % set all nan entries in the wavelength
                        intensity matrix as 0
344                 wavelengthIntMat_rotated(indx) = 0;
345
346                 % project the wavelength intensity matrix onto
                        the rotated line
347                 wavelengthInt_proj = sum(
                        wavelengthIntMat_rotated, 2);
348
349                 % calculate the one dimensional wavelength
                        distribution
350                 lambda_1D = lambda_f_Mat_new(:, 1);
351                 %-----------------------------------------------
352                 %-----------------------------------------------
353
354
355
356                 %-----------------------------------------------
357                 % plot the wavelength intensity matrix
358                 %-----------------------------------------------
359                 % initialise a new figure
360                 figure;
361
362                 % plot the projected version of the wavelength
                        intensity matrix against the projected
                        wavelength distribution
363                 plot(lambda_1D,wavelengthInt_proj);
364
365                 % give it a title
366                 title(['tilted projection peasurement for'
                        num2str(tilt(i))])
367
368
369                 % convert the 1D version of the data into
                        something else
370                 [lambda_axis_shifted_tmp, energy_tmp,
                        spectrum_in_energy_tmp] = SExpTools.
                        OneD2finalLambda(lambda_1D,
                        wavelengthInt_proj, E0, tilt(i));
371
372                 % slice the variable defined above
373                 lambda_axis_shifted(i,1:length(
                        lambda_axis_shifted_tmp)) =
                        lambda_axis_shifted_tmp';
374
375                 % do so again
```

```
376                 energy(i, 1:length(energy_tmp)) = energy_tmp';
377
378                 % do so again
379                 spectrum_in_energy(i,1:length(
                        spectrum_in_energy_tmp)) =
                        spectrum_in_energy_tmp';
380
381
382
383                 % add to the main figure for each item in the
                        loop
384                 figure(h)
385
386                 % go to the right subplot
387                 subplot(1+ceil(length(tilt)/2),2,i)
388
389                 % plot the graph
390                 plot(energy(i,:)-E0,spectrum_in_energy(i,:))
391
392                 % label the axes
393                 xlabel('dE [meV]');
394                 ylabel('Intensity');
395
396                 % put a title on the plot
397                 title(['tilt=' num2str(tilt(i))]);
398
399                 % set the axes to suitably display the plot
400                 axis([-10 10 -1e-22 max(spectrum_in_energy(i,:))
                        *1.2])
401                 %-----------------------------------------------
402                 %-----------------------------------------------
403             end
404
405
406             figure(h)
407             subplot(1+ceil(length(tilt)/2),2,(1+ceil(length(tilt
                    )/2))*2)
408             subplot(1+ceil(length(tilt)/2),2,(1+ceil(length(tilt
                    )/2))*2-1)
409
410         end
411         %-------------------------------------------------------
412         %-------------------------------------------------------
413
414
415
416
417         %-------------------------------------------------------
418         % define OneD2finalLambda function
419         %-------------------------------------------------------
420         function [lambda_axis_shifted, energy,
                spectrum_in_energy] = OneD2finalLambda(lambda_1D,
                wavelengthInt_proj, E0, tilt)
421
422             % load the constants if they don't exist
423             if ~exist('SE_h','var'), load_chess_parameters;
424             end
425
426
427             % find the energy of a helium 3 particle with energy
                      E0
```

```
428             lambda0=energy2wavelength(E0, 3);
429
430             % tilted projection angle from a different axis
431             t1=tilt-90;
432
433             % calculate the shifted axis for the wavelength
434             lambda_axis_shifted=lambda0*tand(t1)+lambda_1D/cosd(
                    t1);
435
436             % find the indices of all positions where the
                    wavelengths on the shifted axis are positive
437             lambda_pos=lambda_axis_shifted(lambda_axis_shifted >
                     0);
438
439             % calculate the right spectrum
440             spectrum_in_lambda=wavelengthInt_proj(
                    lambda_axis_shifted>0);
441
442             % the energy of the beam can be calculated from de
                    broglie relation
443             energy=SEexpTools.wavelength2energy(lambda_pos, 3);
444
445             % calculate the jacobian
446             jacobian=(SE_h/SE_3hemass^2)*(2*energy/SE_3hemass)
                    .^(-3/2);
447
448             % the energy is the product of the spectrum and the
                    jacobian
449             spectrum_in_energy=spectrum_in_lambda.*jacobian;
450         end
451         %-------------------------------------------------------
452         %-------------------------------------------------------
453
454
455
456
457         %-------------------------------------------------------
458         % define wavelength2energy function
459         %-------------------------------------------------------
460         function E = wavelength2energy(lambdaInAnrstrem, mass)
461             % calculate the energy E of a particle with mass and
                     wavelength
462
463
464             % load in the basic parameters and select the
                    correct atomic mass
465             if ~exist('SE_amu','var'), load_chess_parameters;
                    end
466
467             % if mass is 3 amu i.e. if particle is helium 3
468             % then define the mass in kg
469             if mass==3
470                 m = SE_3hemass;
471
472             % if mass is 4 amu i.e. if particle is helium 4
473             % then define mass in kg
474             elseif mass==4
475                 m = SE_amu * 4.00260;
476
477             % other masses aren't allowed
478             else
```

```
479                 disp('Only masses 3 and 4 allowed')
480                 return
481
482             end
483
484
485             % wavelength of particle in SI units: e-10 due to
                    angstroms
486             lambda_SI = lambdaInAnrstrem*1e-10;
487
488             % energy of the particle from Planck's equation
489             E_SI = (SE_h./lambda_SI).^2/(2*m);
490
491             %convert to joules
492             E = E_SI/SE_e*1000;
493         end
494         %-----------------------------------------------------------
495         %-----------------------------------------------------------
496
497
498
499
500         %-----------------------------------------------------------
501         % define calcMinWidthForModeInTilt function
502         %-----------------------------------------------------------
503         function [dK, dE, modeWidthDue2beam] =
                calcMinWidthForModeInTilt(modeModelHandle, E0, FWHM,
                 max_dK, tiltDeg)
504             % in 1/Angstrem, TODO: implement angle resolution of
                     instrument
505             dK = -max_dK:0.001:max_dK;
506
507             % calculate the energy transfer of the given model
                    using the vector of momentum transfers
508             dE = modeModelHandle(dK);
509
510             % reverse the effect of the tilt
511             transMat = [cosd(180-tiltDeg), sind(180-tiltDeg)];
512
513             % for each momemtum transfer
514             for i=1:length(dK)
515                 % find a vector: first entry is the wavelength
                        of the incident beam, and the second entry
                        is the wavelength of the scattered beam
516                 lambda_0 = [energy2wavelength(E0, 3),
                        energy2wavelength(E0+dE(i), 3)];
517
518                 % find a vector: first the wavelength at half
                        maximum (the maximum scanning wavelength),
                        then the wavelength after the energy change
                        at this wavelength
519                 lambda_1 = [energy2wavelength(E0 + FWHM/2, 3),
                        energy2wavelength(E0+FWHM/2+dE(i),3)];
520
521                 % the mode width is the product of the change in
                         wavelength from the mean to half maximum
                        with the transition matrix
522                 modeWidthDue2beam(i) = lambda_0*transMat' -
                        lambda_1*transMat';
523             end
524
```

```
525             % plot, in 3D, the mode width as a function of the
                    momentum transfer and energy transfer
526             plot3(dK, dE, modeWidthDue2beam, 'o')
527
528         end
529         %-----------------------------------------------------------
530         %-----------------------------------------------------------
531
532
533
534
535         %-----------------------------------------------------------
536         % define phononModel function
537         %-----------------------------------------------------------
538
539         % one by one, define anonymous functions that describe
                the energy transfer for a particular surface mode as
                 a function of momentum transfer. The output of the
                function is the anonymous function; thus setting a
                variable equal to the function sets the variable as
                an anonymous function
540
541         % these models can be found in the literature
542
543         function modeModelHandle = phononModel(modelName)
544             switch modelName
545                 case 'RayleighWaveNi111'
546                     modeModelHandle=@(x) 16.88*sin(pi/2/(2.91/2)
                            .*x)-0.5192*sin(pi/2/(2.91/2).*x).^3;
547
548                 % case 'LongitudinalWaveNi111'
549                     % modeModelHandle=@(x) *sin(.*x)-*sin(.*x)
                            .^3;
550
551                 case 'WaterOnNi6meV'
552                      modeModelHandle=@(x) (14.53.*(abs(x)<0.6).*
                             abs(x).^2+5.559).*sign(x);
553
554                 case 'specular'
555                     modeModelHandle=@(x) zeros(size(x));
556
557                 case 'WaterOnNi2meV'
558                     modeModelHandle= @(x) repmat(2,size(x));
559
560                 case 'NiMagnons'
561                     modeModelHandle= @(x) (364.*(abs(x)<0.2).*x
                            .^2).*sign(x);
562
563                 % case 'AcousticSurfacePlasmons'
564                     % modeModelHandle=@(x) 4250*x.*(abs(4250*x)
                            <20)+20*(abs(4250*x)>=20);
565
566                 case 'SpecBroad'
567                     modeModelHandle= @(x) zeros(size(x));
568
569                 otherwise
570             end
571
572         end
573         %-----------------------------------------------------------
574         %-----------------------------------------------------------
```

```
575
576
577
578
579         %-------------------------------------------------------
580         % define plotModels function
581         %-------------------------------------------------------
582         function plotModels()
583
584             % the range of momentum transfers used
585             dK = -10:0.05:10;
586
587             % assign labels to the different modes, one by one
                    as defined above
588
589             modeModelHandle1 = SEexpTools.phononModel('
                    RayleighWaveAu111');
590
591
592             modeModelHandle2 = SEexpTools.phononModel('
                    WaterOnAu6meV');
593
594
595             modeModelHandle3 = SEexpTools.phononModel('
                    WaterOnAu2meV');
596
597
598             modeModelHandle4 = SEexpTools.phononModel('
                    LongitudinalWaveAu111');
599
600
601             modeModelHandle5 = SEexpTools.phononModel('
                    AcousticSurfacePlasmons');
602
603
604
605             % one by one plot the different modes against their
                    momentum transfers
606
607
608             plot(dK,modeModelHandle1(dK),'b',dK,-
                    modeModelHandle1(dK),'b'); hold on
609
610
611             plot(dK,modeModelHandle2(dK),'b',dK,-
                    modeModelHandle2(dK),'b'); hold on
612
613
614             plot(dK,modeModelHandle3(dK),'b',dK,-
                    modeModelHandle3(dK),'b'); hold on
615
616
617             plot(dK,modeModelHandle4(dK),'b',dK,-
                    modeModelHandle4(dK),'b'); hold on
618
619
620             plot(dK,modeModelHandle5(dK),'b',dK,-
                    modeModelHandle5(dK),'b'); hold on
621         end
622         %-------------------------------------------------------
623         %-------------------------------------------------------
```

```
624
625         end
626
627     end
```