ASSIGNMENT 10: THE W.I.M.

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

Written in XƎLaTeX

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 intoduced 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:

```
[lambda_i_Mat, lambda_f_Mat,wavelengthIntMat, tilt] = SEexpTools
    .calcMeasurementsParams(8, 0.5, 6, 30, 44.4);
[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.

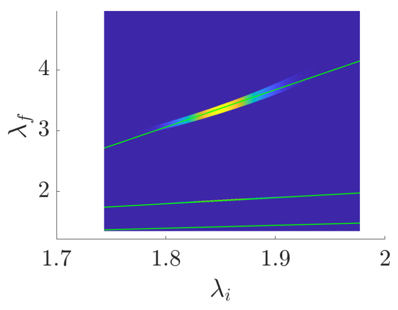

Figure 1 | The tilted projection axes for three features on a model wavelength intensity matrix.

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 $S(\lambda_1 \to \lambda_2)$ 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.

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
```

```
end




% calculate the energy in SI units
% change from eV to joules
E_SI = energy ./ 1000*SE_e;

% calculate wavelength in metres
% this is the output
% multiply by e10 to produce angstrom output
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.

```
%class spin echo experiment tools

classdef SEexpTools
    %UNTITLED2 Summary of this class goes here
    %   Detailed explanation goes here

    properties
        % there are no properties
    end


    % define the functions, one by one
    % This class contains
    % - calcMeasurementsParams
    % - projectByTilt
    % - OneD2finalLambda
    % - wavelength2energy
    % - calcMinWidthForModeInTilt
    % - phononModel
    % - plotModels



    methods(Static)


        %-------------------------------------------------------
        % define calcMeasurementsParams function
        %-------------------------------------------------------
        function [lambda_i_Mat, lambda_f_Mat,wavelengthIntMat,
            tilt] = calcMeasurementsParams(E0, FWHM,
            max_dEexchange, theta_i, theta_tot)
            %calcMeasurementsParams to calculate the
                measurements needed to map a specific mode.
            % modeModelHandle - handle to function which gets a
                dK, and gives dE.
            % E0 - the beams mean initial energy (in meV)

            % load in the constants
            load_chess_parameters;

            % define the number of points to use
            numOfPoints = 1e3;

            %--------------------------------------------------
            %% find beam params (energy, wave-vector, profile)
            %--------------------------------------------------
            % incident wavelength matrix - a uniform distribution, later converted to gaussian
            lambda_i_tmp = linspace(energy2wavelength(E0+2*FWHM,3), energy2wavelength(E0-2*FWHM,3), numOfPoints);

            % calculate the energy of a helium-3 particle with wavelength lambda_i_tmp
            Ei_vec = SEexpTools.wavelength2energy(lambda_i_tmp,3);

            % calculate the wavevector of a helium-3 particle with energy Ei_vec
            ki_vec = energy2wavevector(Ei_vec,3);

            % convert FWHM of particle energies to standard deviation
            c_param = FWHM/(2*sqrt(2*log(2)));

            % the energies of particles in the beam follow a normal distribution
            % calculate the relative intensity of a particle with energy E_i compared to unity, building a beam distribution
            beamIntensity = exp(-(E0-Ei_vec).^2./(2*c_param.^2));

            % scattered (final, denoted f) wavelength matrix for helium 3
            % maximum in the matrix is the wavelength of the sum of the maximum value in Ei_vec and the maximum energy transfer
            % 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
            % there are numOfPoints points in the matrix
            lambda_f_tmp = linspace(energy2wavelength(max(Ei_vec)+max_dEexchange, 3), energy2wavelength(max(min(Ei_vec)-max_dEexchange, 1e-1), 3), numOfPoints);

            % the energies of the particles in the scatted beam
            Ef_vec = SEexpTools.wavelength2energy(lambda_f_tmp,3);

            % the wavevectors of the particles in the scatted beam
            kf_vec = energy2wavevector(Ef_vec,3);

            % set up a mesh for the wavelengths using the incident and scattered waves
```

```
            [lambda_i_Mat, lambda_f_Mat] = meshgrid(lambda_i_tmp,lambda_f_tmp);

            % mesh for associated wavevectors
            [ki_Mat, kf_Mat] = meshgrid(ki_vec, kf_vec);

            % mesh for associated energies
            [Ei_Mat, Ef_Mat] = meshgrid(Ei_vec,Ef_vec);

            % slightly randomise the incident angle - adds noise
            theta_i_Mat = theta_i + (rand(size(Ei_Mat))-0.5)*0.2;

            % outward angle = angle of scattering - angle of incidence
            theta_out_Mat = theta_tot - theta_i_Mat;

            % change in energy is final energy - initial energy
            dE_Mat = Ef_Mat-Ei_Mat;

            % change in wavevector is given by the formula for momentum transfer
            dK_Mat = kf_Mat.*sin(theta_out_Mat/180*pi)-ki_Mat.*sin(theta_i_Mat/180*pi);
            %----------------------------------------------------
            %----------------------------------------------------


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

            % 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.
            modeModelHandle = SEexpTools.phononModel('SpecBroad');

            % modeModelHandle is the variable defined on the line above. Thus find the value of the mode at each possible momentum transfer.
            modeDE_SpecBroad = modeModelHandle(dK_Mat);

            % load in a phonon model from a later function.
            modeModelHandle = SEexpTools.phononModel('RayleighWaveNi111');

            % modeModelHandle is the variable defined on the line above. Thus find the value of the mode at each possible momentum transfer.
            modeDE_RW = modeModelHandle(dK_Mat);

            % load in a phonon model from a later function.
            modeModelHandle = SEexpTools.phononModel('WaterOnNi6meV');

            % modeModelHandle is the variable defined on the line above. Thus find the value of the mode at each possible momentum transfer.
            modeDE_W6 = modeModelHandle(dK_Mat);



            % uncomment below to calculate more modes

            % modeModelHandle = SEexpTools.phononModel('
                WaterOnNi2meV');
            % modeDE_W2 = modeModelHandle(dK_Mat);

            % modeModelHandle = SEexpTools.phononModel('
                NiMagnons');
            % modeDE_NiMagnons = modeModelHandle(dK_Mat);

            % modeModelHandle = SEexpTools.phononModel('
                LongitudinalWaveNi111');
            % modeDE_LA = modeModelHandle(dK_Mat);

            % modeModelHandle = SEexpTools.phononModel('
                AcousticSurfacePlasmons');
            % modeDE_ASP = modeModelHandle(dK_Mat); %modeDE_ASP(
                find(abs(modeDE_ASP) > 15))=0;

            %-----------------------------------------------------
            %-----------------------------------------------------



            %-----------------------------------------------------
            % find the points in which the scan line crosses the
                 modes (modeDE and -modeDE)
            %-----------------------------------------------------

            % define a matrix of zeros the size of the matrix of
                 energy transfers
            % this is the precursor to the transfer function
            energyMatchMat = zeros(size(dE_Mat));



            % decice whether to set the entries of the matrix as
                 0 or 1
            energyMatchMat = energyMatchMat | abs(
                modeDE_SpecBroad-dE_Mat) < 1e-1 | abs(
                modeDE_SpecBroad+dE_Mat) < 1e-1;

            % and again, for a different mode
            energyMatchMat = energyMatchMat | abs(modeDE_RW-
                dE_Mat) < 1e-1 | abs(modeDE_RW+dE_Mat) < 1e-1;

            % and again, for a different mode
            energyMatchMat = energyMatchMat | abs(modeDE_W6-
                dE_Mat) < 1e-1 | abs(modeDE_W6+dE_Mat) < 1e-1;



            % make a matrix the size of energyMatchMat that
                contains copies of the beam intensity
            % this is the transfer function
            beamIntensityMat = repmat(beamIntensity, size(
                energyMatchMat,1), 1);

```

```
        % Using the WIM as the product of the possible scattering function and beam distribution
        wavelengthIntMat = energyMatchMat .* beamIntensityMat;
        %--------------------------------------------------
        %--------------------------------------------------




        %--------------------------------------------------
        % plot each of the mode energies as a function of momentum transfer
        %--------------------------------------------------

        % initialise figure
        figure;

        % hold on
        hold on

        % plot dK against modeDE_SpecBroad
        plot(reshape(dK_Mat,1,[]), reshape(modeDE_SpecBroad,1,[]), 'b.', reshape(dK_Mat,1,[]), -reshape(modeDE_SpecBroad,1,[]) ,'b.', 'MarkerSize', 2)

        % plot dK against modeDE_RW
        plot(reshape(dK_Mat,1,[]),reshape(modeDE_RW,1,[]),'b.',reshape(dK_Mat,1,[]),-reshape(modeDE_RW,1,[]),'b.','MarkerSize',2)

        % plot dK against modeDE_W6
        plot(reshape(dK_Mat,1,[]),reshape(modeDE_W6,1,[]),'b.',reshape(dK_Mat,1,[]),-reshape(modeDE_W6,1,[]),'b.','MarkerSize',2)

        % plot dK against dE_Mat (which is the scan curve)
        plot(reshape(dK_Mat,1,[]),reshape(dE_Mat,1,[]),'g.','MarkerSize',2)


        % initialise a second figure
        figure; hold on

        % produce a pseudocolour plot of the wavelength intensity matrix
        pcolor(lambda_i_Mat, lambda_f_Mat, wavelengthIntMat);

        % specify the colouring on the plot
        shading flat;

        % find the indices of all positions where the wavelength intensity matrix is non-zero
        tmpIndx = find(wavelengthIntMat~=0);

        % find the maximum value of lambda_f where the wavelength intensity matrix is non-zero
        maxNonZeroLambdaFinal = max(lambda_f_Mat(tmpIndx));

        % find the minimum value of lambda_f where the wavelength intensity matrix is non-zero
        minNonZeroLambdaFinal = min(lambda_f_Mat(tmpIndx));

        % find the maximum value of lambda_i where the wavelength intensity matrix is non-zero
        maxNonZeroLambdaIn = max(lambda_i_Mat(tmpIndx));

        % find the minimum value of lambda_i where the wavelength intensity matrix is non-zero
        minNonZeroLambdaIn = min(lambda_i_Mat(tmpIndx));

        % define the axes to be used: the ranges of lambda specified above, with 10% leeway
        axis([minNonZeroLambdaIn*0.9 maxNonZeroLambdaIn*1.1 minNonZeroLambdaFinal*0.9 maxNonZeroLambdaFinal*1.1])



        % user input for how many distinct features there are displayed in the wavelength intensity matrix plot on the screen
        featureNum = inputdlg('how many features would you like to analyse?'); featureNum = str2num(featureNum{:});

        % guidance for the user input
        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')

        % find the coordinates of all points picked by the user
        [l1, l2] = ginput(featureNum + 1);
        %---------------------------------------------------
        %---------------------------------------------------




        %---------------------------------------------------
        % pick out and plot each feature. Calculate the angle of tilt the feature makes to the lambda_i axis
        %---------------------------------------------------

        % iterate through the second to last entry in the set of coordinates above
        for i=2:length(l2)

            % 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.
            indx = find(lambda_f_Mat < l2(i-1) & lambda_f_Mat > l2(i));

            % save the wavelengths of the indices just described
            l1current = lambda_i_Mat(indx);
            l2current = lambda_f_Mat(indx);

            % find the values of the wavelength intensity matrix where the final wavelength crosses the scan curve
            intensityCurrent = wavelengthIntMat(indx);

            % collate the indices of all positions in the slice of the wavelength intensity matrix described above where the intensity is greater than 0.05
            indx = find(intensityCurrent > 0.05);

            % fit a polynomial to the currents above
            p = polyfit(l1current(indx),l2current(indx),1);


            hold on;

            % plot each feature
            plot([min(l1current) max(l1current)], p(1)*[min(l1current) max(l1current)]+p(2), 'g')

            % calculate angle of tilt of each feature
            tilt(i -1) = atan(p(1))*180/pi + 90;
        end
        %----------------------------------------------------
        %----------------------------------------------------

    end
    %--------------------------------------------------------
    %--------------------------------------------------------






    %--------------------------------------------------------
    % define projectByTilt function
    %--------------------------------------------------------

    % define a function to project a wavelength intensity matrix onto an angle
    function [lambda_1D, lambda_axis_shifted, wavelengthInt_proj,energy,spectrum_in_energy] = projectByTilt(lambda_i_Mat, lambda_f_Mat, wavelengthIntMat_orig, tilt, E0)
        % lambda_i_Mat - matrix of values for the wavelength of the incident beam
        % lambda_f_Mat - matrix of values for the wavelength of the final beam
        % wavelengthIntMat_orig - original wavelength intensity matrix
        % tilt - vector of tilt angles for each feature
        % E0 - mean energy of helium beam


        % initialise a vector for the projection variable on each projection axis
        lambda_axis_shifted=zeros(size(lambda_i_Mat, 2), length(tilt));

        % do the same for the energy
        energy = zeros(size(lambda_i_Mat,2), length(tilt));

        % initialise the vectors for the energies of the incident beam
        spectrum_in_energy = zeros(size(lambda_i_Mat, 2), length(tilt));

        % if the SE class variables exist, load them
        if ~exist('SE_h','var'), load_chess_parameters; end

        % initialise a new figure
        h=figure;

        % go through each possible tilt and project the WIM
        for i=1:length(tilt)

            %------------------------------------------------
            % project the wavelength intensity matrix onto a line
            %------------------------------------------------
            % rotation matrix
            R=[cosd(tilt(i)-90) -sind(tilt(i)-90) ; sind(tilt(i)-90) cosd(tilt(i)-90)]

            % find the rotated version of the incident wavelength matrix
            lambda_i_Mat_rotated = R(1)*lambda_i_Mat + R(2)*lambda_f_Mat;

            % find the rotated version of the final wavelength matrix
            lambda_f_Mat_rotated = R(3)*lambda_i_Mat + R(4)*lambda_f_Mat;

            % interpolate to fit data to the wavelengths
            F = TriScatteredInterp(lambda_i_Mat_rotated(:), lambda_f_Mat_rotated(:), wavelengthIntMat_orig(:));

            % find indices of all wavelengths in the original wavelength intensity matrix where the intensity is larger than 0.05
            indx = find(wavelengthIntMat_orig > 0.05);

            % find, as a vector: the minimum and maximum of the incident and scattered wavelengths along the projection
            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];

            % calculate the new wavelength distributions projected onto the axis
            [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)));

            % obtain rotated version of the wavelength
                intensity matrix, using the interpolated
                version of the wavelengths
            wavelengthIntMat_rotated = F(lambda_i_Mat_new,
                lambda_f_Mat_new);

            % find the incides of all nan  (not a number) in
                 the rotated wavelength intensity matrix
            indx = isnan(wavelengthIntMat_rotated);

            % set all nan entries in the wavelength
                intensity matrix as 0
            wavelengthIntMat_rotated(indx) = 0;

            % project the wavelength intensity matrix onto
                the rotated line
            wavelengthInt_proj = sum(
                wavelengthIntMat_rotated, 2);

            % calculate the one dimensional wavelength
                distribution
            lambda_1D = lambda_f_Mat_new(:, 1);
            %-----------------------------------------------
            %-----------------------------------------------



            %-----------------------------------------------
            % plot the wavelength intensity matrix
            %-----------------------------------------------
            % initialise a new figure
            figure;

            % plot the projected version of the wavelength
                intensity matrix against the projected
                wavelength distribution
            plot(lambda_1D,wavelengthInt_proj);

            % give it a title
            title(['tilted projection peasurement for'
                num2str(tilt(i))])


            % convert the 1D version of the data into
                something else
            [lambda_axis_shifted_tmp, energy_tmp,
                spectrum_in_energy_tmp] = SEexpTools.
                OneD2finalLambda(lambda_1D,
                wavelengthInt_proj, E0, tilt(i));

            % slice the variable defined above
            lambda_axis_shifted(i,1:length(
                lambda_axis_shifted_tmp)) =
                lambda_axis_shifted_tmp';

            % do so again
```

```
        energy(i, 1:length(energy_tmp)) = energy_tmp';

        % do so again
        spectrum_in_energy(i,1:length(
            spectrum_in_energy_tmp)) =
            spectrum_in_energy_tmp';



        % add to the main figure for each item in the
            loop
        figure(h)

        % go to the right subplot
        subplot(1+ceil(length(tilt)/2),2,i)

        % plot the graph
        plot(energy(i,:)-E0,spectrum_in_energy(i,:))

        % label the axes
        xlabel('dE [meV]');
        ylabel('Intensity');

        % put a title on the plot
        title(['tilt=' num2str(tilt(i))]);

        % set the axes to suitably display the plot
        axis([-10 10 -1e-22 max(spectrum_in_energy(i,:))
            *1.2])
        %------------------------------------------------
        %------------------------------------------------
    end


    figure(h)
    subplot(1+ceil(length(tilt)/2),2,(1+ceil(length(tilt
        )/2))*2)
    subplot(1+ceil(length(tilt)/2),2,(1+ceil(length(tilt
        )/2))*2-1)

end
%--------------------------------------------------------
%--------------------------------------------------------




%--------------------------------------------------------
% define OneD2finalLambda function
%--------------------------------------------------------
function [lambda_axis_shifted, energy,
    spectrum_in_energy] = OneD2finalLambda(lambda_1D,
    wavelengthInt_proj, E0, tilt)

    % load the constants if they don't exist
    if ~exist('SE_h','var'), load_chess_parameters;
    end


    % find the energy of a helium 3 particle with energy
         E0
```

```
            lambda0=energy2wavelength(E0, 3);

            % tilted projection angle from a different axis
            t1=tilt-90;

            % calculate the shifted axis for the wavelength
            lambda_axis_shifted=lambda0*tand(t1)+lambda_1D/cosd(
                t1);

            % find the indices of all positions where the
                wavelengths on the shifted axis are positive
            lambda_pos=lambda_axis_shifted(lambda_axis_shifted >
                 0);

            % calculate the right spectrum
            spectrum_in_lambda=wavelengthInt_proj(
                lambda_axis_shifted>0);

            % the energy of the beam can be calculated from de
                broglie relation
            energy=SEexpTools.wavelength2energy(lambda_pos, 3);

            % calculate the jacobian
            jacobian=(SE_h/SE_3hemass^2)*(2*energy/SE_3hemass)
                .^(-3/2);

            % the energy is the product of the spectrum and the
                jacobian
            spectrum_in_energy=spectrum_in_lambda.*jacobian;
        end
        %--------------------------------------------------------
        %--------------------------------------------------------




        %--------------------------------------------------------
        % define wavelength2energy function
        %--------------------------------------------------------
        function E = wavelength2energy(lambdaInAnrstrem, mass)
            % calculate the energy E of a particle with mass and
                 wavelength


            % load in the basic parameters and select the
                correct atomic mass
            if ~exist('SE_amu','var'), load_chess_parameters;
                end

            % if mass is 3 amu i.e. if particle is helium 3
            % then define the mass in kg
            if mass==3
                m = SE_3hemass;

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

            % other masses aren't allowed
            else
                disp('Only masses 3 and 4 allowed')
                return

            end


            % wavelength of particle in SI units: e-10 due to angstroms
            lambda_SI = lambdaInAnrstrem*1e-10;

            % energy of the particle from Planck's equation
            E_SI = (SE_h./lambda_SI).^2/(2*m);

            %convert to joules
            E = E_SI/SE_e*1000;
        end
        %-------------------------------------------------------
        %-------------------------------------------------------




        %-------------------------------------------------------
        % define calcMinWidthForModeInTilt function
        %-------------------------------------------------------
        function [dK, dE, modeWidthDue2beam] = calcMinWidthForModeInTilt(modeModelHandle, E0, FWHM, max_dK, tiltDeg)
            % in 1/Angstrem, TODO: implement angle resolution of instrument
            dK = -max_dK:0.001:max_dK;

            % calculate the energy transfer of the given model using the vector of momentum transfers
            dE = modeModelHandle(dK);

            % reverse the effect of the tilt
            transMat = [cosd(180-tiltDeg), sind(180-tiltDeg)];

            % for each momemtum transfer
            for i=1:length(dK)
                % find a vector: first entry is the wavelength of the incident beam, and the second entry is the wavelength of the scattered beam
                lambda_0 = [energy2wavelength(E0, 3), energy2wavelength(E0+dE(i), 3)];

                % find a vector: first the wavelength at half maximum (the maximum scanning wavelength), then the wavelength after the energy change at this wavelength
                lambda_1 = [energy2wavelength(E0 + FWHM/2, 3), energy2wavelength(E0+FWHM/2+dE(i),3)];

                % the mode width is the product of the change in wavelength from the mean to half maximum with the transition matrix
                modeWidthDue2beam(i) = lambda_0*transMat' - lambda_1*transMat';
            end

```

```
        % plot, in 3D, the mode width as a function of the momentum transfer and energy transfer
        plot3(dK, dE, modeWidthDue2beam, 'o')

    end
    %-------------------------------------------------------
    %-------------------------------------------------------




    %-------------------------------------------------------
    % define phononModel function
    %-------------------------------------------------------

    % 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

    % these models can be found in the literature

    function modeModelHandle = phononModel(modelName)
        switch modelName
            case 'RayleighWaveNi111'
                modeModelHandle=@(x) 16.88*sin(pi/2/(2.91/2).*x)-0.5192*sin(pi/2/(2.91/2).*x).^3;

            % case 'LongitudinalWaveNi111'
                % modeModelHandle=@(x) *sin(.*x)-*sin(.*x).^3;

            case 'WaterOnNi6meV'
                 modeModelHandle=@(x) (14.53.*(abs(x)<0.6).*abs(x).^2+5.559).*sign(x);

            case 'specular'
                modeModelHandle=@(x) zeros(size(x));

            case 'WaterOnNi2meV'
                modeModelHandle= @(x) repmat(2,size(x));

            case 'NiMagnons'
                modeModelHandle= @(x) (364.*(abs(x)<0.2).*x.^2).*sign(x);

            % case 'AcousticSurfacePlasmons'
                % modeModelHandle=@(x) 4250*x.*(abs(4250*x)<20)+20*(abs(4250*x)>=20);

            case 'SpecBroad'
                modeModelHandle= @(x) zeros(size(x));

            otherwise
        end

    end
    %-------------------------------------------------------
    %-------------------------------------------------------
```

```matlab




%-------------------------------------------------------
% define plotModels function
%-------------------------------------------------------
function plotModels()

    % the range of momentum transfers used
    dK = -10:0.05:10;

    % assign labels to the different modes, one by one
        as defined above

    modeModelHandle1 = SEexpTools.phononModel('
        RayleighWaveAu111');


    modeModelHandle2 = SEexpTools.phononModel('
        WaterOnAu6meV');


    modeModelHandle3 = SEexpTools.phononModel('
        WaterOnAu2meV');


    modeModelHandle4 = SEexpTools.phononModel('
        LongitudinalWaveAu111');


    modeModelHandle5 = SEexpTools.phononModel('
        AcousticSurfacePlasmons');



    % one by one plot the different modes against their
        momentum transfers


    plot(dK,modeModelHandle1(dK),'b',dK,-
        modeModelHandle1(dK),'b'); hold on


    plot(dK,modeModelHandle2(dK),'b',dK,-
        modeModelHandle2(dK),'b'); hold on


    plot(dK,modeModelHandle3(dK),'b',dK,-
        modeModelHandle3(dK),'b'); hold on


    plot(dK,modeModelHandle4(dK),'b',dK,-
        modeModelHandle4(dK),'b'); hold on


    plot(dK,modeModelHandle5(dK),'b',dK,-
        modeModelHandle5(dK),'b'); hold on
end
%-------------------------------------------------------
%-------------------------------------------------------
```

```

        end

end
```