

# Software to Aid the Study and Analysis of X-ray Fluorescence Data from Multilayer Structures DESY Summer Student Program 2010

Rebecca Lane  
Supervised by Dmitri Novikov

## Abstract

Several short programs were written as part of a larger project to evaluate the data in multilayer generated X-ray standing wave (XSW) experiments. The programs were intended to improve the efficiency with which fitting of intensity curves could be carried out and to study the secondary XSW signal. The experimental data in question was generated at HASYLAB, DESY and the theoretical data was taken from the server by Sergey Stepanov<sup>1</sup>. The programs are focussed on acquiring the data from the server, displaying the data in graphical form, and fitting with the experimental data. In addition a program to calculate the secondary fluorescence radiation intensity at a particular point for given input parameters was written, to allow for quick calculation of expected radiation yield for a given situation.

## 1 Introduction

### 1.1 Theory

X-ray diffraction patterns have been used for a long time as a method of studying the underlying structure of materials in a non-destructive way. The pieces of software written as part of my project were designed specifically for the study of such diffraction patterns in and around multilayer materials, consisting of a repeating pattern of layers made of different materials with different diffractive properties. The result of applying an incident beam to such a structure is to set up a standing wave field, consisting of maxima and minima where the beams reflected from the different layers interfere constructively and destructively respectively. Figure 1 shows how such a field is formed.

In order to study such interference patterns, the system is usually used in the total-reflection X-ray fluorescence (TXRF) regime. This means that at small angles of order a few degrees, X-rays are almost entirely reflected from boundaries between different materials, in analogy with total internal reflection in conventional optics. Hence the angle of incidence is greatly exaggerated for the purposes of the figures in this report, and typically would be between 0 and 3 degrees. The parameters of interest in order to study the properties of a particular structure are the reflectivity for particular angles of incidence, and the position-dependant intensity field produced throughout the material (again for different angles of incidence). This information is contained within two plots that are produced within the code, and throughout the project.

### 1.2 MATLAB

The programs making up this project were all written in MATLAB. MATLAB (short for matrix laboratory) is a numerical computing environment and fourth-generation programming language, developed by MathWorks. MATLAB was useful mainly for its ability to work with large matrices of

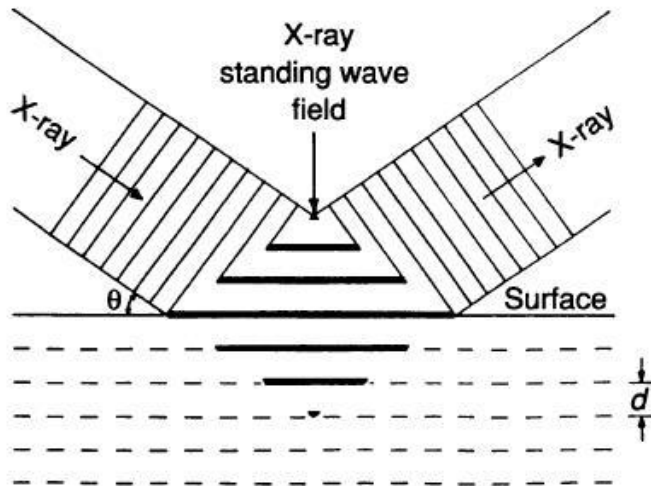


Figure 1: The X-ray standing wave field formed above the surface of a multilayer structure<sup>2</sup>

data, as well as being able to deal with and manipulate strings of text. In addition to this, graphs can be plotted and interfacing with programs written in other languages is possible, including C, C++ and Perl (elements of which were used in the few cases where MATLAB functions were not sufficient). MATLAB was particularly useful for the purposes of my project due to its functions designed for interpolation.

### 1.3 Sergey Stepanov's X-ray Server

The majority of the work for this project was concerned with linking together with the X-ray software based on a server created by Sergey Stepanov at the Argonne National Laboratory (ANL) in the US. The server contains software to calculate theoretical X-ray reflection and standing wave patterns depending on a number of parameters decided by the user, and then displays the data both in the form of raw data in a file and in graphical form. This can be achieved through use of a web based form or by manipulating and executing a Perl script provided by Stepanov that generates the data and automatically stores it in a file. Both versions of the same intrinsic software were used extensively during the course of my project.

### 1.4 The Sample

Whilst the software developed as part of this project was designed to be general for future use with different samples, during the course of my project I worked with data from a particular sample of interest which was being studied at the beamline at the time. The sample consisted of a silicon substrate, with 24 alternating layers of tungsten and silicon of 15 Å thickness, with one final layer of silicon (possibly partially oxidised) on the surface which is around 7-8 Å thick. The sample was produced for future use to study fluorescence patterns at double layer solid-liquid boundaries. This works because the sample could be placed in a salt solution, and the different pH conditions would cause the silicon oxide layer to be more or less charged relative to the solution, and hence attract ions in different ways. Later in my report the sample is used as a demonstration of the functions of my code and in particular in the final part of my project to make an improved estimate of the thickness of the top silicon layer by using my fitting script.

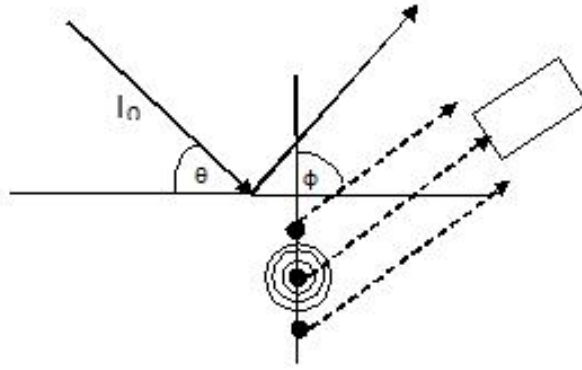


Figure 2: The secondary fluorescence intensity at a detector for a given orientation

## 2 My Project

I split the different elements of my project into subsections for convenience, starting with a piece of code to calculate the secondary intensity at a given point above the surface of a multilayer.

### 2.1 Isecondary

The purpose of this piece of code was to calculate the secondary fluorescence radiation intensity at a detector at a given angle to the surface of the diffracting material, for radiation with a particular angle of incidence. Figure 2 shows schematically the situation being considered. The incoming beam is incident on the surface of the material, and mostly reflected as shown. Then the material can be considered as a series of points which reradiate the absorbed radiation in the form of secondary fluorescence radiation. This radiation is emitted in all directions, and measured by a detector at a specific orientation as shown. The total intensity at the detector can be found by summing (or integrating) the radiation from all points in the material as shown. Due to the homogeneity of the sample it was adequate to consider just one dimension as shown and to use discrete depth points at intervals selected by the user.

Then the total intensity at the detector is calculated from the initial intensity  $I_0$ , which is then normalised, and then scaled for the effects of absorption in the material. Hence the secondary intensity emitted from each depth point is given by:

$$I_{out} = A \cdot I_0 \cdot e^{-\mu_{abs} \cdot t} \quad (1)$$

where  $t$  in Equation 1 is the depth of the point in question below the surface. This assumes that absorption only takes place beneath the surface and not in the air above it. If the detector is orientated at an angle  $\phi$  to the surface normal, then, by simple geometry,  $t = z \cdot \sin \phi$ , where  $z$  is the depth coordinate of the point with respect to the surface.

Hence, for the purposes of the code, the final expression to be evaluated is given by:

$$I_{tot} = \sum_z I_0 \cdot e^{-\mu_{abs} \cdot \sin \phi \cdot z} \quad (2)$$

At this point all that remains is to know the values of the depth points  $z$  the user wishes to consider and the incoming intensity  $I_0$  at each of those points. These values are simply the data provided from Stepanov's server for a particular angle of incidence. The code assumes that a file containing the intensity profile for a range of angles of incidence ( $\theta$ ) and depth points ( $z$ ) has already been created on the server and downloaded.

Thus the code was required to implement the following stages:

- Ask the user for the angle parameters of the incident radiation and the detector. In addition to this, the user must supply the absorption coefficient of the material being studied.
- Search for the correct line of intensity values within the previously generated data file for the specified angle of incidence.
- Read the appropriate data into vectors and matrices to be used by MATLAB.
- Evaluate Equation 2 for the range of depth points in the file, using the intensity values of  $I_0$  also from the file.

The full code with comments can be seen in the appendix in Section 6.

My next section of code is focussed around how the data used previously could be automatically acquired without the use of the web-based interface.

## 2.2 Xraydata

The main body of my project was concerned with improving the efficiency with which a user could compare theoretical data as used in the above code, and experimental data taken at the beamline. Sergey Stepanov provided a script written in the Perl programming language which was executable, and could automatically download the data obtained from the server and place it in the current working directory. The parameters used to calculate the data were set within the file. My job was to write a program to interface the Perl file with the user, to allow the parameters to easily be altered without having to do it manually.

The Perl file was in the form of a simple text file, and can be seen in the appendix Section 7. The main challenge involved was getting MATLAB to automatically search the file for the variable the user wished to alter, then altering the line and resaving with the rest of the file intact. Hence the various stages of the program were as follows:

- The Perl text file is read into the MATLAB workspace.
- The user is asked if they would like to change a variable. Answering 'y' continues to the variable altering subscript, answering 'n' ends the process. This question loops in order to change as many variables as the user wishes.
- Within the subscript, the user must choose which variable they wish to change, and it must be written in the correct format. To aid this process a list of the possible variables is displayed on the screen. The user also specifies the value they wish to assign to the variable.
- The program then searches for the appropriate line within the Perl file, which is a string within the MATLAB workspace. This string is then separated into parts which remain and the part to be altered (the part where the value is stated), then put back together again. Then the new Perl script is resaved.

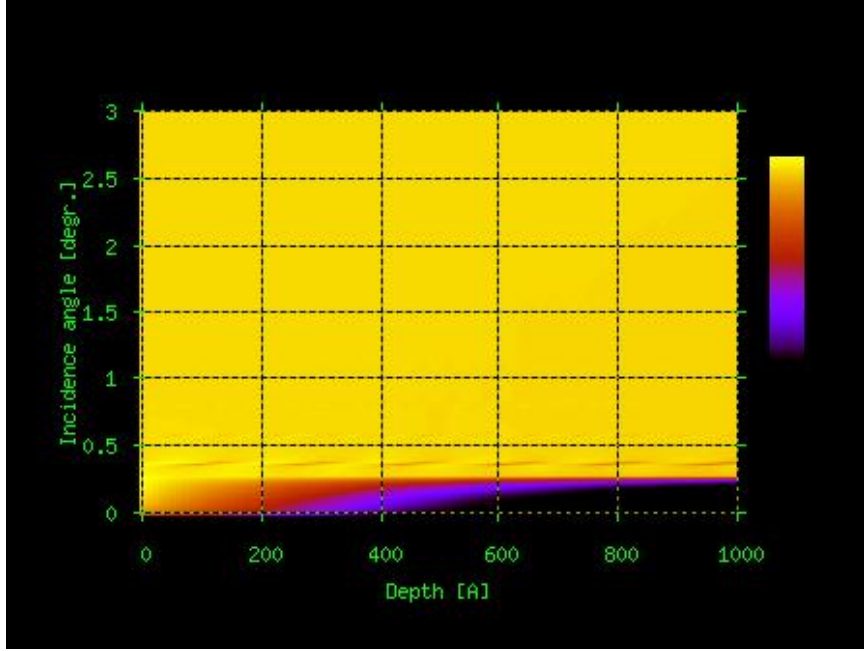


Figure 3: Example contour intensity plot using Gnuplot

- Back in the main script, once all the required variables have been altered, the program executes the Perl file which automatically collects the required data from Stepanov's server and saves it in the current working directory. The files can then be automatically read in, making use of the fact that the data is assigned a unique file ID number which is stated when the Perl file is executed.
- Finally the data can be displayed in the 2 graphs described previously.

Examples of the 2 graphs produced can be seen in Figures 4 and 5. As mentioned in the script, the contour intensity plot can also be produced via a gnuplot file provided by Stepanov. An example of this plot is also shown in Figure 3.

### 2.3 Alterurl

This script was written as a slightly alternative method to achieve the same ultimate goal as the previous program. It is designed to possibly be implemented later as part of a larger piece of code should fast iterations of the theoretical data from Stepanov's server ever be needed. It is a method of reaching the same data in raw data and graphical form, by simply visiting Stepanov's server over the internet, but with a more efficient user interface allowing a variable to be changed at a time as with the Perl file. In order to ever be used for fast iterations, the code must eventually be combined with something from Stepanov allowing the data to be extracted from the server automatically and quickly. This is a possible problem for the future.

The code works in a similar way to the method used to alter one line of the Perl file, in that it constructs the url as a string. The various stages are described below:

- The 'default' url is read into the workspace, having previously been saved into a file for reference. The url is constructed out of 3 sections. The first section contains the basic directions

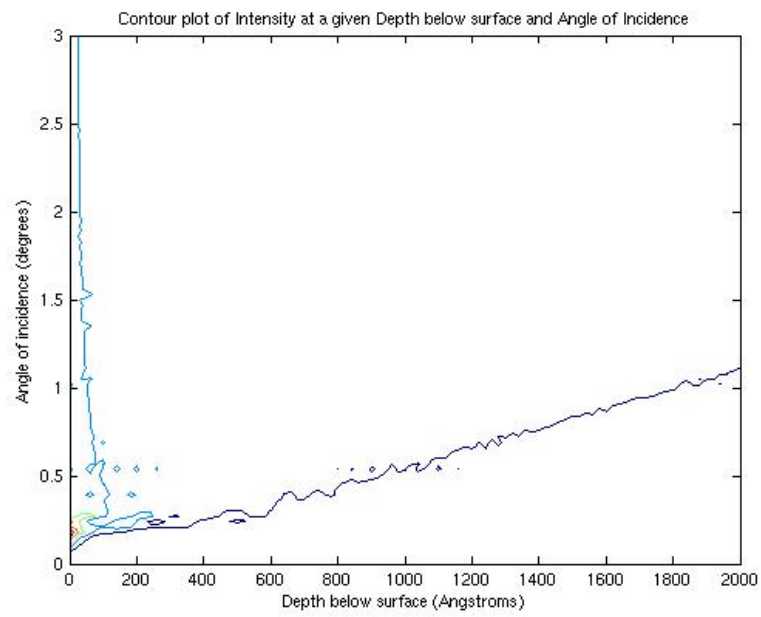


Figure 4: Example intensity contour plot using MATLAB

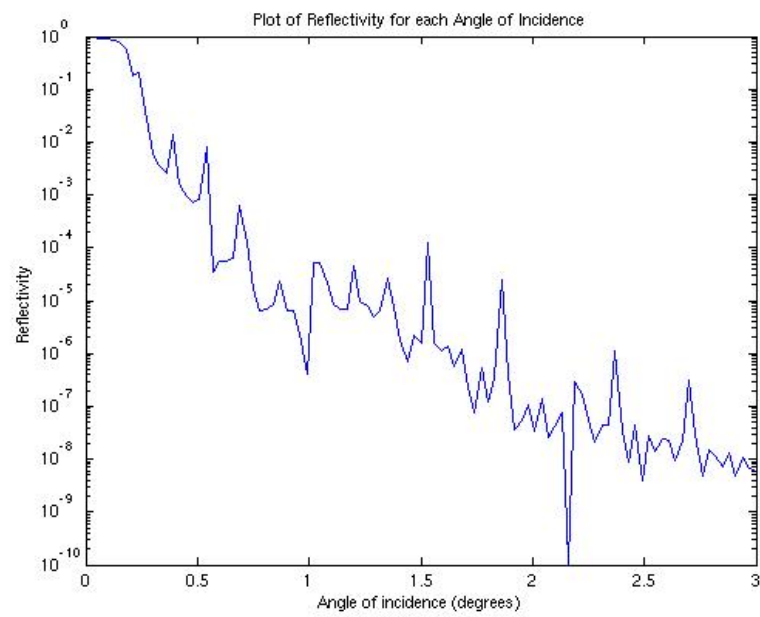


Figure 5: Example reflectivity graph using MATLAB

to the web page, and is the same for every set of input parameters. The second section contains all the parameter properties of the set up and beam, such as range of angle of incidence, energy and number and range of depth points to consider. Finally the last section contains all the specific information about the particular profile of the multilayer structure. For the purposes of this program I have assumed that this does not need altering, and it is set as the settings for the Si/W structure described in the introduction. However, a few lines of code added later could easily allow this to be extended to include changes by the user.

- Next a loop is implemented to allow the user to alter as many variables as they want, as with the previous program. The code then leads to a subroutine.
- In the subroutine, the user chooses a variable from the list and a value as previously. Then the url string is searched to find the variable's position within it, and the appropriate part is altered, before the url is put back together. This was a fairly complicated procedure since all the variable names were different lengths, and so altering the correct part of the string required a loop searching for the '&' signs which appeared either side of the variable. For more details, see the comments within the script.
- Finally, returning to the main routine the code opens a webpage containing the data for the chosen parameters, and displays the appropriate graphs.

For reference purposes, the 'default' url as stored in the file in 3 sections is as follows:

```
http://sergey.gmca.aps.anl.gov/cgi/TER_form.exe?comment1=Template%3A+TER_sl_multilay_
sw.htm

&xway=1&wave=1&line=Cu-Ka1&ipol=1&subway=1&code=Si&df1df2=-1&chem=&rho=&x0=&w0=1.&sigma=
4.&tr=0.&scanmin=1&scanmax=2&unis=0&nscan=201&swflag=1&swref=0&swmin=20&swmax=-20&swpts=
21

&profile=t%3D8+code%3DSi+sigma%3D3+%0D%0A%3D15+code%3DW+sigma%3D3%0D%0A%0D%0Aperiodz%
3D24%0D%0A%3D15+code%3DSi++sigma%3D2%0D%0A%3D15+code3DW+sigma%3D2%0D%0Aend+period%
0D%0A
```

The full code with comments can be seen in the appendix Section 8.

## 2.4 Fitdata

This part of my project was concerned with the real experimental data taken at a beamline at HASYLAB and its fit to theory. Specifically my program was to be used to compare a reflectivity profile (rocking curve) with the reflectivity/angle of incidence curve calculated using Stepanov's software. At this stage the sample described in the introduction was used, and it's properties as described were fed into the software to generate the theoretical curve. The first element of the fitting process was to find the shift in the x-axis of the data between the two sets and the scale, or normalization, in the y direction. This required use of MATLAB fitting functions to iterate and find the optimum solution for the two parameters.

This was probably the most complex part of my program, although the MATLAB fitting functions made the job a lot easier. The main program consisted of the following stages:

- User manually selects values for the best guess of shift and normalization factor. These can be judged by eye. In fact, the initial normalization factor should be close to the peak value of the experimental curve, since the theoretical data is normalized to close to one. It is essential

that these values chosen are close to the actual values, because if the program is forced outside of the range of the data it is trying to interpolate then it will not function.

- Use the function "bothfit". This function was created by myself, and it evaluates the difference between the theoretical and experimental data, taking into account the effects of the normalization factor and shift. This is achieved by using a sub function "myfun3" to interpolate the theoretical data to find the reflectivity at the theta values used in the experiment (after applying the shift). Thus the function depends on the two possible parameters, and returns a value for each theta point in the experimental data file. Hence for the best fit, the size of the values the function returns should be minimized, or in fact the sum of the square of all the values for each theta needs to be minimized.
- The minimization is achieved using the MATLAB function "lsqnonlin", which utilises the Levenberg-Marquardt algorithm. This takes a function dependant on multiple variables and minimizes the sum of the square of the function values subject to the parameters, starting from the initial values chosen by the user.
- The program then returns the optimum parameter values found and displays them for the user.
- Finally, the program corrects the theta values for the shift found, and saves the values to a file.

The full code with comments can be seen within the appendix in Section 9 and the functions can also be seen.

Once the data has been corrected for the shift, it is possible to display the experimental and theoretical curve on the same graph to compare and see how good the fit is. An example of such a graph can be seen in Figure 6.

Clearly the data is not a perfect fit, but the fact that the greatest part of the curves coincides demonstrates that the program has successfully found the best parameters for the normalization factor and shift. Any differences beyond this are a matter for further analysis.

The final stage of my project was to apply the above method to the other form of data used throughout my project-the intensity/depth profile.

## 2.5 Fitfluo

The above fitting method could also be used for fitting the intensity profile. Since the shift has already been calculated in the previous case, and the angle values saved to a file, these values can be used and the shift is no longer a fitting parameter. Instead, now we have again a normalization factor, and also a value of depth below the surface which gives the closest fit. It is this depth value that is of particular interest to the user. This is because as described in the introduction, the exact thickness of the top silicon layer of the sample is unknown.

The program is similar to that previously, with the following differences:

- The function, as before, calculates the difference between the experimental and theoretical data, and minimizing the square of this function finds the best fit. As before the function is varied with different values of normalization constant, but this time the second parameter is the column index (x-coordinate) of the data within the intensity profile data. The data is stored in a matrix where the x-coordinate represents the different depth points and the y-coordinate represents the different angles of incidence.



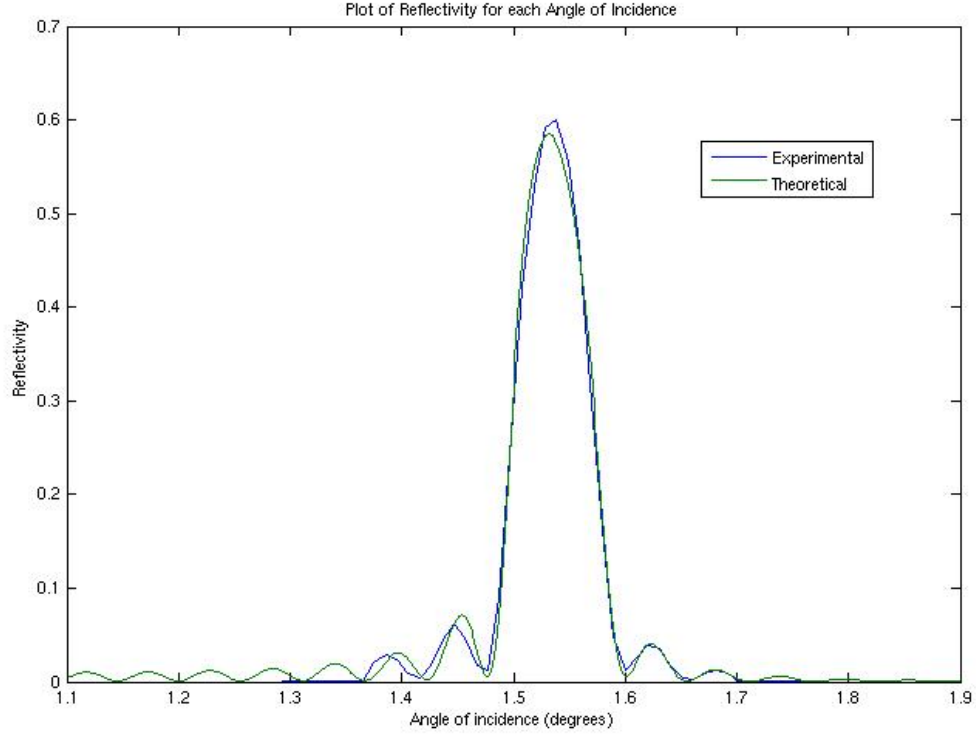


Figure 6: Example fit between a theoretical and experimental reflectivity curve

- In order to select a column, it is necessary for the user to input a first "best guess" value, and then within the function the intensity data is interpolated to produce the required column of intensities dependant on angle of incidence. It is this column that is then used in the function as the theoretical data, subject to the normalization factor.
- Then, as before, we have a function to be minimized subject to 2 parameters.

Once again, the full code with comments and the associated function are shown in the appendix, Section 10.

The fit obtained for our sample is shown in Figure 7. The fit is not as good as that from the previous section, for several possible reasons. As could be seen also in Figure 6, the two curves are slightly out of phase to the left of the maximum. In addition to this, parameters selected during the obtaining of the theoretical data such as the roughness factor can contribute slightly to the amplitude fitting, making it impossible to fit the theoretical data from the 'perfect' scenario to the experimental data. Possible improvements to the fit are discussed later.

### 3 Results

As I have mentioned several times above, my scripts were written in general form for future use by colleagues at HASYLAB. However, during the course of the project we were able to make some conclusions based on the sample I have been using to demonstrate the purposes of my code.

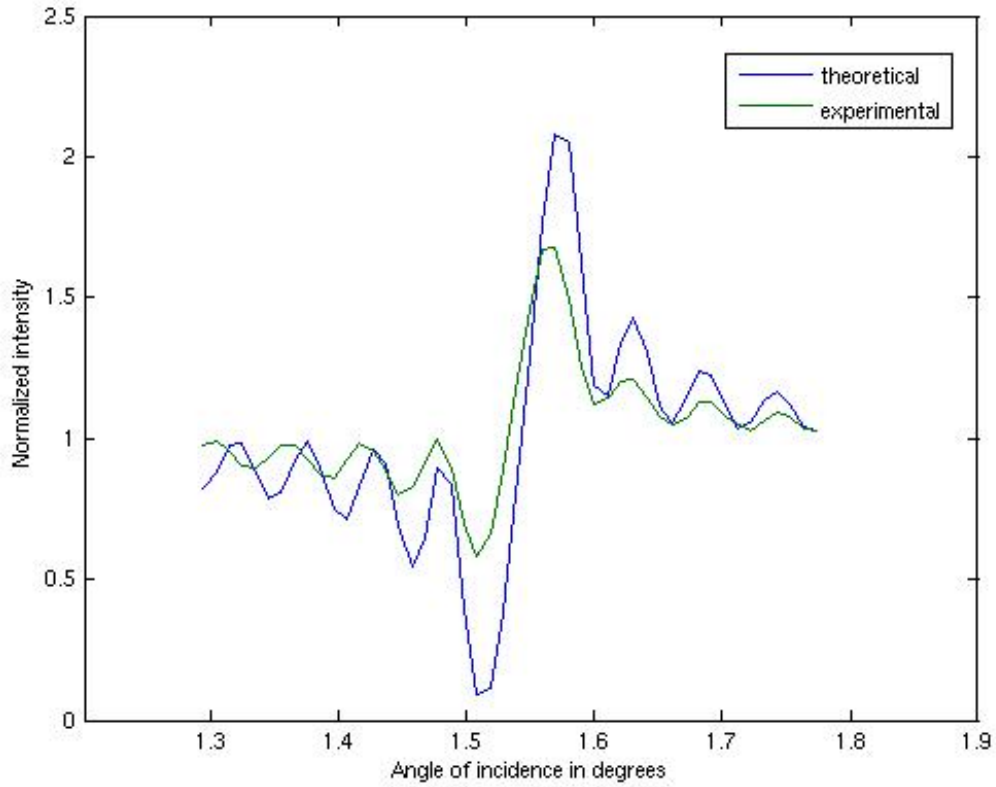


Figure 7: Example of the intensity curve fit using the example described previously

The final outcome of the fitting process resulted in a normalization factor of  $1.54 \cdot 10^{-6}$  and a matrix position of 73.68 providing the closest fit. These are the parameters used to produce the graph in Figure 7. The real value of interest for this particular sample is the depth position implied by the matrix index of 74. This corresponds to a depth of 18 Å. The situation is as follows in Figure 8.

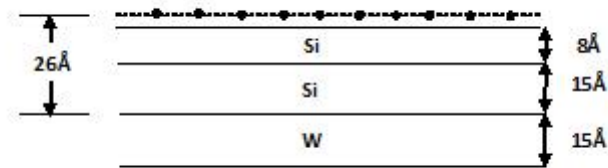


Figure 8: The profile of the sample according to the measured depth value

Combining the 18 Å with the 8 Å surface layer gives a total of the 26 Å shown on the diagram. This as a result was entirely unexpected, since a layer of silicon 26 Å thick was not part of the original design in which the surface layer of silicon was placed on a layer of tungsten. This points to a very surprising result—that the actual profile is more like that in Figure 8 in which the final layer of silicon

is placed on another layer of silicon. This means that the manufacturers must have made a mistake when creating the sample! Hence the 26 Å is made up as shown, with a 2-3 Å deficit which could indicate the level to which the silicon layer has been oxidised at the surface, indicated on the diagram.

## 4 Discussion

Reaching the conclusion of my part in the work is by no means the end of the story. All my programs have been produced for future use by people involved in studying the properties of materials such as the multilayer structure used in this project, in particular to study double-layer fluorescence phenomena as described in my introduction. There is much room for improvement in the further implementation of my programs, and they need to be made more unique to the user's needs.

More specifically, before progress can be made in the study of intensity profiles such as that in Figure 7, some improvements to the fitting process need to be introduced. One possible solution is to perform a weighted fit, in which the part of the curve to the left hand side where there is a phase difference is less weighted in the fitting process compared to the rest of the curve. From a little research into this it seems that there is no straightforward way of telling the "lsqnonlin" function to weight parts of the data differently, but it can be achieved by simply including a weight vector into the function to be minimized, such that each residual is weighted before the function is passed to the Levenberg-Marquardt algorithm.

## 5 References

- [1] Sergey Stepanov's X-ray server. Found at: <http://sergey.gmca.aps.anl.gov/>.
- [2] Diagram taken from *X-ray standing waves at a reflecting mirror surface* by M.J Bedzyk et al. Phys. Rev. Lett. 62, 1376-1379 (1989)

For background information I also made use of:

*Total-Reflection X-ray Fluorescence Analysis* by Reinhold Klockenkämper.

*Crash course in MATLAB* by T.A. Driscoll.

*Scientific Computing with MATLAB* by A Quarteroni and F Saleri.

The product support at: <http://www.mathworks.com>

## 6 Appendix A

Below is the code for the program "Isecondary".

```
% This is a script to calculate the secondary fluorescence radiation
intensity
% at a fixed angle of incidence ("alpha", zero is at the surface!),
fixed
% energy and a semi-infinite bulk material. Secondary radiation yield
is
% calculated at a given detector angle (from the surface normal, "phi"
angle)

% At the moment:
% - the script uses a previously generated data file with a very
special
% format, for comments see Stepanov's web pages
% - the script asks the user for the angle parameters in addition to
the
% absorption coefficient of the material being studied.
% ( in principle, one can also use sergey.gmca.aps.anl.gov to get this
% information)

% - Then the nearest angle is found within the data generated using
% the server by Sergey Stepanov at http://sergey.gmca.aps.anl.gov/,
and
% the (primary) intensity distribution inside the bulk
% is read in from the file.
% - The final intensity at the detector is found from this primary
% intensity by scaling for the effects of absorption within the
% material, and summed for each depth point in the file.

% The user must take care to put the maximal depth value high enough
% to let the field decrease to zero

% Written by Rebecca Lane, Summer Student 2010.

clear % beware of losing all information in the workspace !!!
% User input parameters.

alpha=input('Angle of incidence (from surface) in degrees: ');
% Defines angle of incident beam.
phi=input('Detector angle (from normal) in degrees: ');
% Defines position of detector.
mu=input('Absorption coefficient in 1/cm: ');
% Dependant on material in question
file=input('Filename of data: ');
% User inputs the name of the file with the appropriate data in ' '
% (e.g. 'TER26337_sw.grd').
```

```

k=1.00; % the scattering cross-section coefficient etc
        % to be replaced with correct value later (constant of
        % proportionality between secondary radiation and intensity
value in
        % file.)

phi=phi*(pi)/180; % Convert to radians.

B=importdata(file, ' ');
B=B.data; % 'importdata' automatically stores data in B.data even if
you
% specify B, this step redefines B as B.data to make it easier % to
manipulate.

% The previously generated file contains information about the maximum
% and minimum incident angle, and the maximum and minimum depth point
% used to generate the data. It also contains the number of data
points used,
% allowing us to extract information about the range of angles and
depth
% points used.

Anglemin=B(3,1); %Finds the minimum angle used in generating the file.
Anglemax=B(3,2); %Finds the maximum angle used in generating the file.
Nangles=B(1,2); %Finds the number of angle points used in generating
the file.
Angleincrim=(Anglemax-Anglemin)/(Nangles-1); %Finds the angle
increment
% used in generating the file data.

Depthmin=B(2,1); %Finds the minimum depth used in generating the file.
Depthmax=B(2,2); %Finds the maximum depth used in generating the file.
Ndepths=B(1,1); %Finds the number of depth points used in generating
the file.
Depthincrim=(Depthmax-Depthmin)/(Ndepths-1); %Finds the depth
increment
% used in generating the file data.

if xor(Anglemin>alpha, Anglemax<alpha)
    fprintf(1, 'Value of alpha is outside of range\nTry it again\n')
    return
end

Angleoptions=(Anglemin:Angleincrim:Anglemax)';
Depthoptions=(Depthmin:Depthincrim:Depthmax)';
% These vectors contain all the values of incident angle and depth in
the
% previously generated file, and will be useful for the rest of the
code.

```

```

% Read in intensity data.
A=importdata(file, ' ', 5); % This step is necessary to form the
correct
% table of intensity data ignoring the first 5 lines which contain the
% data used above.
A=A.data;

for f=1:Ndepths; % Loop includes all the depth points chosen.
    I(f)=interp1(Angleoptions, A(:, f), alpha); %Interpolates the data
    % using a linear interpolation to give the best intensity value
    % for user's input "alpha". For more information on the possible
    % types of interpolation see 'help interp1'.

    Iout(f)=k*I(f)*exp(Depthoptions(f)*sin(phi)*1e-2)*exp(-mu*1e-6);
    % Calculates the intensity at the detector from each depth point.
    % If changing the wavelength please take care to adjust the above
    % exponential scaling mu and depth units. Includes factor of
attenuation
    % due to its depth in the material, multiplied by the size of the
angle
    % the detector subtends.
end

Itot=sum(Iout) % Sum over all depth points to give the total intensity
% at the detector.

```

## 7 Appendix B

Below is the code for the program "xraydata".

```

% This is a program to generate and manipulate the Xray data from
% the server by Sergey Stepanov at http://sergey.gmca.aps.anl.gov/.
% One parameter is altered at a time within the perl file and then
% it is executed. This is achieved by finding a parameter's position
% within the file, changing its value and then resaving the file.
% After the file is executed, graphs are plotted of reflectivity
% against incident angle and a contour plot showing intensity
% variation with depth and incident angle.
% Written by Rebecca Lane, Summer Student 2010.

clear %% Beware of losing all information in the workspace !!!

% First read in the original perl file to make the appropriate
% alteration.
fid=fopen('test3.pl');
C=textscan(fid, '%s', 'delimiter', '\n');
C=C{1,1}; % Makes the matrix easier to manipulate.

```

```

% The user is asked if they would like to change a variable, and if
% they choose 'y' then the variable is changed within a sub script
% xraydata_variable. The virtually endless "while 1" loop is repeated
% until the user has changed all the variables they would like, then
% the program continues. If the user selects 'no' then the program
% continues using the unaltered perl file.

while 1
fprintf('Would you like to change a variable (y/n)?\n')

    if yesno
    xraydata_variable

    else break
    end

end

fprintf('Now it will run perl\n')

% Now the new perl script can be executed from within matlab:
result=perl('test3.pl')

%These lines are to find the unique job ID number generated when the
perl
%file is executed. It can be used later to automatically read in the
data.
IDindex=strfind(result, 'job ID');
ID=result(IDindex+7: IDindex+14);

% Now the data is automatically generated in the user directory, and
% can be manipulated in the required way.

file=strcat(ID, '_sw.grd'); % Generate .grd filename in the standard
% form from file ID.
datfile=strcat(ID, '.dat'); % Generate .dat filename in the standard
% form from file ID.

% First need to read in the data for the 2 required graphs.
D=importdata(datfile); % For the reflectivity data
Angleoptions=D(:,1); % Seperate the x and y coordinates for the graph.
Reflec=D(:,2);

Intensity=importdata(file, ' ', 5); %for the intensity data. The data
% is contained in the file excluding the first 5 lines.
Intensity=Intensity.data; %is the required matrix

% Need to generate the depth points for plotting. The information
about
% the range of depth points chosen is stored within the file.

```

```

B=importdata(file); % The first few lines contain the information
about
% the input range chosen by the user.
B=B.data; % 'importdata' automatically stores data in B.data even if
you
% specify B, this step redefines B as B.data to make it easier to
manipulate.

Depthmin=B(2,1); %Finds the minimum depth used in generating the file.
Depthmax=B(2,2); %Finds the maximum depth used in generating the file.
Ndepths=B(1,1); %Finds the number of depth points used in generating
the file.
Depthincrm=(Depthmax-Depthmin)/(Ndepths-1); %Finds the depth
increment
% used in generating the file data.

Depthoptions=(Depthmin:Depthincrm:Depthmax)'; % Contains all the
depth points used.

semilogy(Angleoptions,Reflec) % Reflectivity/Angle of incidence graph
(x, y).
xlabel('Angle of incidence (degrees)')
ylabel('Reflectivity')
title('Plot of Reflectivity for each Angle of Incidence')

figure % Opens a new figure for the second graph
contour(Depthoptions, Angleoptions, Intensity) % Intensity at
different depths
% and angles graph (x, y, z).
xlabel('Depth below surface (Angstroms)')
ylabel('Angle of incidence (degrees)')
title('Contour plot of Intensity at a given Depth below surface and
Angle of
% Incidence')

% Note the above graph can also be plotted in gnuplot using the .plt
file.

% This program is a subscript to the program "xraydata". It contains
the
% part of the program which allows the user to change a variable
within the
% perl file.
% Written by Rebecca Lane, Summer Student 2010.

fprintf('%s\n', 'Possible choices of variable are:', ...
'$wave=Wavelength of radiation', '$sigma= Roughness in Angstroms',
...
'$scanmin= Minimum scan angle', ...
'$scanmax= Maximum scan angle', '$nscan= Number of scan points',

```



```

...
'$swmin= Minimum depth point', ...
'$swmax= Maximum depth point', '$swpts= Number of depth points')
V=input('Variable to alter: ');
val=input('Value: '); % Size of variable the user requires.

v=num2str(val); % Converts number to string, necessary to generate
the line.

str = sprintf('%s%c%s', 'my', ' ', V);
I=strmatch(str, C); % Finds the index of the line that needs to be
% altered for the user's requirement.
str2=strcat(str, '=', v, ';'); % Builds the line necessary for the
% perl file out of the user's entered value.

% Once the appropriate line and value are defined, use the
following
% to create a set of text identical to the original perl
% file except for the new line:
C(I,1)={str2}; % Alters the line containing the changed variable.

% Generate a new perl file with the new line:
fid=fopen('test3.pl', 'w+');
fprintf(fid, '%s\n', C{:});

```

## 8 Appendix C

Below is the code for the program "alterurl".

```

% This script is to change parameter values within the url linking
% to the server by Sergey Stepanov at http://sergey.gmca.aps.anl.gov/.
% The user can select which parameter they wish to alter, and the
value
% they want to change it to, and then the url is constructed by
altering
% the appropriate part of the string. The url can then be looked at
with a
% web browser.
% Written by Rebecca Lane, Summer Student 2010.

clear % Beware of losing all the data in the workspace!

b=importdata('url2'); % The default url is arranged in the file
% in 3 sections.
b1=b{1,1}; % This is the part that needs no alteration.
b2=b{2,1}; % This part contains the basic parameters that may
% be altered by the user.
b3=b{3,1}; % This part contains the detail about the top layer
profile.

```

```

% Currently the program assumes that this is not in need of alteration
% (it is set for the details of the W/Si multilayer sample),
% but a similar program to that of b2 could be created to serve this
purpose
% later if required.

% The "while 1" loop ensures that the user can change as many
variables
% as they want while they select "y" in answer to the question. The
code
% for changing the variable within the file is stored in the subscript
% "alterurl_variable". If the user selects "n" then the loop
terminates
% and the program continues.

while 1
fprintf('Would you like to change a variable (y/n)?\n')

    if yesno
        alterurl_variable

    else break
    end

end

% Execute the appropriate command to go to url, if necessary use
system()
% to go to the shell

if exist('c') % Ensures that the web page is only opened if the url
has
% been altered and so "c" exists.
web(c)
end

% This program is a subscript to the program "alterurl". It contains
the
% part of the program which allows the user to change a variable
within
% the url.
% Written by Rebecca Lane, Summer Student 2010.

% The user must choose a variable to alter from the printed list.

fprintf('%s\n', 'Possible choices of variable are:', ...
    'wave= Wavelength of radiation', 'sigma= Roughness in Angstroms',
    ...
    'scanmin= Minimum scan angle', ...)

```

```

'scanmax= Maximum scan angle', 'nscan= Number of scan points', ...
'swmin= Minimum depth point', ...
'swmax= Maximum depth point', 'swpts= Number of depth points')

variable=input('\nWhich variable? '); % Enter in ' ' marks.
value=input('Value: '); % Enter the value of the variable.
v=num2str(value); % Converts number to string.

index=findstr(b2, variable); % Finds the starting index of the
% variable name.

% Now the string needs to be altered only around the variable name. It
is
% convenient to split the string into 3 parts, alter the one
containing and
% variable leaving the other 2 unchanged, then put the string back
together
% again.

% The loop below finds the index of the first "&" after the variable,
in order
% to know where to split the string into pieces to implement the new
variable value.
% e.g. if the variable to be changed is "wave" the appropriate part of
the string
% could be "...&wave=1&line=..." and the string needs to be split
before the first "&"
% at "index-1", and at the second "&", and the middle part altered.

for i=index:20; % 20 is an arbitrary value. None of the "&" signs are
more than 20
% characters away from the start.
    h=b2(i);
    if h=='&'
        break
    else continue
end
end

% The value of i at which the loop terminates is the index of the
nearest
% "&" character, which is where the string needs to be broken for the
third
% part "c3" as below.

c1=b2(1:index-1); % The position of the unaltered part before.
c2=strcat(variable, '=', v); % The part that's altered.
c3=b2(i: end); % The position of the unaltered part after.

c=strcat(b1, c1, c2, c3, b3); % Put the whole url back together.

```

## 9 Appendix D

Below is the code for the program "fitboth".

```
% This is a script to compare theoretical and experimental data, and
% find the relative normalization factor and shift between them.
% Written by Rebecca Lane, Summer Student 2010.

% Read in the experimental and theoretical curve raw data.

exptheta=importdata('fix_positions.dat');
exptheta=exptheta(:, 2);
expreflec=importdata('rock_fix.dat');
expreflec=expreflec(:, 2);

theo=importdata('rock_theory.dat');
theotheta=theo(:, 1);
thereflec=theo(:, 2);

% Note that the data is also read in again within the function
% environments.

% Manually select the approximate values of the normalization constant
% and the shift between the experimental and theoretical data.
% NOTE: it is necessary to get these values both fairly close for the
% code to function.

initialnorm=input('Initial normalization value: ');
initialshift=input('Approximate shift: ');

% Apply the approximate shift to the experimental data in order to
% ensure that the experimental theta range lies inside the range of
the
% theoretical theta range. This is necessary for the interpolation
% function to work correctly.

exptheta_shifted=exptheta+initialshift;

% Now it is necessary to interpolate the theoretical data for the
% experimental theta values. This is performed using "myfun3".

thereflec_interp=myfun3(exptheta_shifted);

% Now we need to find the values of the normalization factor and shift
% which give the best fit.

% The function "bothfun" calculates the difference between the
% experimental and theoretical data subject to the normalization
factor
% and the shift.
```

```

% Then "lsqnonlin" finds the point where the square of "bothfun" is
% minimized, starting from the initial normalisation defined above and
% using a small initial shift. A small initial shift is used since we
% have already manually shifted the data, so the point should be
close.

x0=[initialnorm, 0.1];

f2 = @(x)bothfun(x, initialshift); % Anonymous function, see manual
for
% details.
x=lsqnonlin(f2, x0);

fprintf('%s%f\n', 'The value of the normalization factor is: ', x(1))

fprintf('%s%f\n', 'The value of the additional shift is: ', x(2))

totalshift=initialshift+x(2);
fprintf('%s%f\n', 'The value of the total shift is: ', totalshift)

% Save the corrected theta values for future use.

exptheta_corr=exptheta_shifted+x(2);
save 'fix_positions_corr.dat' exptheta_corr

```

Below is the code for the functions used in the above script, "bothfun" and "myfun3".

```

% This function calculates the difference between experimental and
% theoretical data, subject to a possible shift along the x direction
and a
% possible normalization difference.

function g = bothfun(x, initialshift)
% Read in the required data within the function environment.
exptheta=importdata('fix_positions.dat');
exptheta=exptheta(:, 2);
expreflec=importdata('rock_fix.dat');
expreflec=expreflec(:, 2);
% Apply the initial shift as defined in the main script "fit_both"
exptheta_shifted=exptheta+initialshift;

g = (x(1)*(myfun3(exptheta_shifted+x(2)))-expreflec);

% This function interpolates the theoretical data to return the
theoretical
% reflectivities at given angle points (for example the angle points
used in
% an experiment).

```

```

function [interpolate]=myfun3(x)
theo=importdata('rock_theory.dat');
theotheta=theo(:, 1);
theorelec=theo(:, 2);
interpolate=interp1(theotheta, theorelec, x);

```

## 10 Appendix E

Below is the code for the program "fitfluo".

```

% This script fits an experimental intensity curve against the
theoretical
% intensity curve generated from Stepanov's server. The fit is
optimised
% relative to the parameters of depth position and normalization
factor.
% Written by Rebecca Lane, Summer Student 2010.

% Read in the experimental data
cr_fluo_exp=importdata('cr_fluo_yield.dat');
cr_fluo_exp=cr_fluo_exp(:, 2);

exptheta_corr=importdata('fix_positions_corr.dat');
% Correct for the effect of slope.
cr_fluo_exp_corr=cr_fluo_exp.*exptheta_corr;

% Read in the theoretical data
cr_fluo_theo=importdata('cr_fluo_yield_theory.dat');
% The rows (y-coordinate) contain the different angles, and the
columns
% (x-coordinate) contain the different depth points. The angle values
% should exactly coincide with the experimental values as they have
been
% chosen that way.

% It is necessary to find values of the two parameters which give the
% closest fit to the experimental data. The two parameters are the
depth
% and the normalization factor.

initialN=input('Initial depth position within file: ');
initialA=input('Initial normalization factor: ');

% User selects the initial best guess.

```

```

y0=[initialN, initialA];

f5 = @(y)fitfluofun2(y, exptheta_corr); % Anonymous function, see
manual for details.
y=lsqnonlin(f5, y0);

fprintf('%s%f\n', 'The value of the position within the file is: ',
y(1))

fprintf('%s%d\n', 'The value of the normalization factor is: ', y(2))


% This function calculates the difference between experimental and
% theoretical data, subject to a normalization factor and a depth
position,
% which is defined by its position within the matrix "cr_fluo_theo".
% Written by Rebecca Lane, Summer Student 2010.

function h = fitfluofun2(y, exptheta_corr)

% Read in the required data within the function environment.
cr_fluo_exp=importdata('cr_fluo_yield.dat');
cr_fluo_exp=cr_fluo_exp(:, 2);

% Read in the theoretical data
cr_fluo_theo=importdata('cr_fluo_yield_theory.dat');

% For the chosen matrix index value, which may or may not be an
integer,
% the appropriate line of theoretical data is found by interpolation.

index=[1:121]; % Must contain the number of depth data points.
cr_fluo_theo_column=interp1(index, cr_fluo_theo', y(1));

h = ((y(2)*cr_fluo_exp.*exptheta_corr))-(cr_fluo_theo_column');

```