ASSIGNMENT 1: SOLUTION

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

Written in XeLaTeX

CODING CONCEPTS

INTRODUCTION

In this tutorial we will introduce some useful MATLAB functions for directory organisation, and we will plot a close-packed plane.

GOOD CODING STYLE

In general, any written code should be easily understandable. If one does not adhere to writing code in good style, it is effectively useless to anyone else; this prevents them from using it for constructive purposes, regardless of how specialised their knowledge base is. Try to focus on the clarity of the code - instead of computation speed, attempt to make the code natural, and easy to follow. Do not worry about speed at this stage! It will always be the case that data handling scripts can be made faster, even after several versions have been created. Optimisation is a task we will eventually opt to achieve. However, we recommend that this is pursued as a secondary task, once a more basic working solution has been created. Optimising a script is much easier once there is something to base it on.

What can we consider "good coding style"? Some introductory comments on this matter include:

- Use meaningful and descriptive variable names. Do not use meaningless single character variable names (as physicists are notorious for doing), unless it is obvious, throughout the program, what the variable refers to. Place a comment on the line above the definition of any variable: allowing yourself, and others, to read the code with ease.
- Include comments that explain the logic and goal of each section of code. For example, when defining a function, describe what the function returns, and what input it takes.
- Include comments to justify the calculations used. This will make everyone happier.
- Write functions to represent sections of code used more than once, rather than repeating the code on numerous occassions. This will make the code easier to follow.

In the next section we will demonstrate some built-in MATLAB functions. Thse can be used to navigate file paths, and to call user-defined functions.

CREATING FOLDERS

When working with MATLAB, you may wish to "call" a function or a script which is not part of the default MATLAB installation. What does this mean? It provides us with a means to avoid the need to rewrite an existing section of code - we can save it as a function. For most practical purposes, it is useful to save an existing function in the same directory as the program, and then to use an in-line command to run the function. How do we specify what file to specify a function from? If it is stored in the same directory as the program being written, the task is simple: in the relevant block of code, it is only necessary to include the file name. Otherwise, should the function not be stored in the same directory, MATLAB must become aware of where it is stored. How do we add these directories to MATLAB's list of folders? The command "addpath" is used.

Example

The following general script checks whether a directory exists. If it does not exist, then the directory is created.

```
%'scripts_path' is a variable, storing the path as a string
%~ is the logical negation operator
%'mkdir' creates folder using the input string as the path
%'addpath' adds the path to the list that matlab searches
%'dir' tells the programme that the path points to a folder
if ~exist(scripts_path,'dir')
    mkdir(scripts_path);
end
addpath(script_path);
```

There are many more system functions available in the native MatLab installation. Try using the library documentation to explore some of them.

PROBLEMS

1. Write a script, similar to the one above, creating a folder containing a file named "auxCls.m".

PLOTTING LATTICES

PREREQUISITES

The chapter on *Crystal Structures* from the *Theory Handbook*.

INTRODUCTION

In our study of the structures of crystals, we will usually encounter the so-called "lattice points" which describe the positions of atoms. These are abstract entities, and require thought; plotting them is impractical, as they only consist of a single set of coordinates. In this tutorial, we will explore ways of making plots that act as useful graphics for the positions of atoms in a crystal lattice.

MATLAB'S MESHGRID

Later in this tutorial, we will plot lattice points. In order to plot these points, we must first generate the positions of the lattice points. How do we do this? Part of the task requires using the function `meshgrid`. This is a matrix manipulation function, taking two vectors $\mathbf{x}$ and $\mathbf{y}$, and returning two matrices $\mathbf{X}$ and $\mathbf{Y}$. These are given by:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x} & \rightarrow \\ \mathbf{x} & \rightarrow \\ \vdots & \vdots \\ \mathbf{x} & \rightarrow \end{pmatrix};$$

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y} & \mathbf{y} & \cdots & \mathbf{y} \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix},$$

which can be interpreted with ease. The number of rows in one matrix is equal to the number of rows in the other. In addition, each row in $\mathbf{X}$ is the vector $\mathbf{x}$, and each column in $\mathbf{Y}$ is the vector $\mathbf{y}$. From the equations above, it should be clear how each is constructed.

ADSORPTION SITES

When an atom adsorbs onto a surface, it can do so at many different sites. These can be easily summarised:

- Top sites: these are the positions directly above the atoms on the top layer of atoms. While they are not marked on the diagram, they would fall exactly above any of the atoms on the top layer.
- Bridge sites: these are the positions above the centre of the lines connecting two adjacent lattice points.
- Substitutional sites: when an atom is absent from the periodic arrangement of the surface layer, an adsorbate may take its place. These would be located in substitutional sites.
- Hollow sites: ordinarily, there is a periodic stacking arrangement of close-packed planes in a crystal structure. This stacking sequence is different for hexagonal close-packed (HCP) and face-centred cubic (FCC), which leads to their distinct bulk structures. Whilst FCC can be described with the stacking sequence "ABCABC", HCP is the more simple "ABAB" (the letters used here have no special meaning). Do not overthink this! It states that one stacking sequence is periodic with period 2, and the other has period 3. Consequently, the positions of atoms in the plane above the upmost plane, known as hollow sites, are in different positions for the two types of lattice. In both cases, hollow sites sit at the intersticial site where an atom in the next plane would be.

Top sites can be considered as the local maxima of potential well, while the hollow sites are the minima, and bridge sites the saddle points. This will be useful for future tutorials.

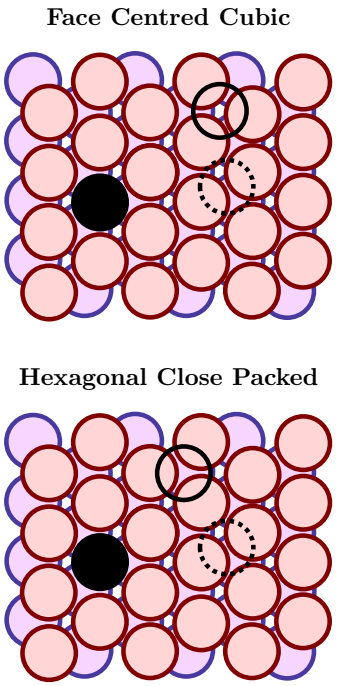

Figure 1 | The types of sites in FCC and HCP. Due to the different stacking sequence in the solids, the hollow sites are different for the type of lattice. The solid unfilled circles represent hollow sites. The dashed unfilled circles represent bridge sites. The filled circles represent substitutional sites.

AZIMUTHS

In this tutorial, we will only be considering the so-called "close-packed planes" of a crystal structure. In a close-packed plane, all adjacent atoms in a plane appear to touch each other, forming a hexagonal pattern. For convenience, it is very easy to label basis vectors in a plane: these are $\langle 01 \rangle$ and $\langle 10 \rangle$. These can be assigned arbitrarily to any of the axes in figure 2.

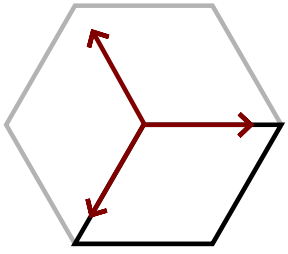

Figure 2 | Potential axes for the indexing of a close-packed surface. The two basis vectors can be assigned to any two of the above, as they form a complete basis.

TASKS

1. Derive the basis vectors for a close-packed plane. Give these in terms of the cartesian basis vectors.
2. Write a function that creates the basis vectors for a close packed plane.
3. Write a function that generates the reciprocal basis vectors for the

real space basis above.

4. Write a function that finds all lattice points in a certain interval (*e.g.* finding all lattice points between -10 and 10, or -5 and 5).
5. Consider the Ru(0001) surface. This is a close-packed plane. Draw a circle on each lattice point, and then draw the unit cell.
6. Change the colours of the circles. First plot them in black, then blue, then red. This is aimed to demonstrate how plotting structures with more than one type of atom can be easily done.
7. Plot triangles instead of circles. Then switch back to circles.
8. Derive the positions of the top sites, bridge sites and hollow sites on a face-centred cubic lattice. Plot each of these, on the same graph, using a different shape.
9. Attach a legend to your graph. Indicate the direction of the basis vectors on your graph.

Tips

1. For task 2, try using the "`meshgrid`" function to speed up the calculations. This is given in the solutions to this task.
2. For task 5, use the functions that have been previously written. Do not write them again!
3. For task 5, the figure 3 is a comic resembling what your output may look like.

SUMMARY

In this tutorial, we have explored how to visually represent the close-packed planes from two lattice types. This skill of visualisation will be useful throughout your further studies.

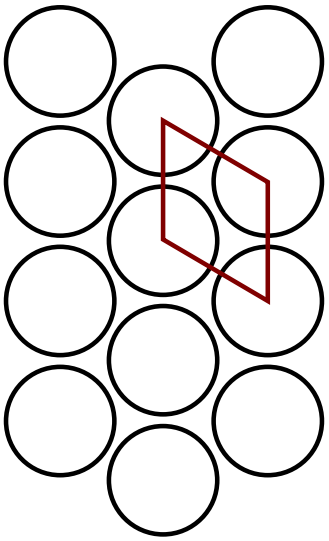

Figure 3 | Ru(0001) lattice structure. The unit cell is drawn on.

SOLUTIONS

INTRODUCTION: MAIN TASK

Task 1: Question

Write a script, similar to the one above, creating a folder containing a file named "`auxCls.m`".

Task 1: Solution

By using the tilde operator ("~"), we can determine whether a directory with a particular name exists. This allows us to proceed; the directory is created if it does not exist, and then the file with the appropriate name is created - regardless of whether the directory initially existed.

```
%evaluate the existence of the folder
if ~exist('./auxCls','dir')
        %create folder
    mkdir auxCls;
    %create file
    edit auxCls/auxCls.m;
end
```

Keeping the directory paths well-organised goes a long way. This is particularly important when you collaborate with others. It is also important to keep your work clear. The general guideline is that a person with little experience in your project should be able to understand what you are trying to achieve with your code by reading the comments. This tutorial series encourage the reader to value clarity over speed - writing clear code, and attempting to optimise it afterwards.

PLOTTING LATTICES

Task 1: Question

Derive the basis vectors for a close-packed plane, in terms of Cartesian basis vectors.

Task 1: Solution

Let us consider the hexagonal basis vectors in Cartesian coordinates as shown in figure 4.

We can define two cartesian basis vectors **x** and **y**, in the plane of our surface, that can we can use to elucidate the basis vectors for our close-packed plane of atoms. As we can see from figure 4, two of the basis vectors make a 60° angle with the **x** vector. Thus the normalised basis vectors for our surface are:

$$\mathbf{a}_1 = -\mathbf{x};$$
$$\mathbf{a}_2 = \frac{1}{2}\mathbf{x} + \frac{\sqrt{3}}{2}\mathbf{y};$$
$$\mathbf{a}_3 = \frac{1}{2}\mathbf{x} - \frac{\sqrt{3}}{2}\mathbf{y}.$$

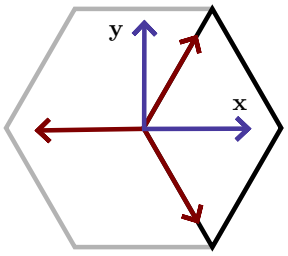

Figure 4 | The hexagonal basis vectors and Cartesian basis vectors on a close packed plane.

Task 2: Question

Write a function that creates the basis vectors for a close-packed plane.

Task 2: Solution

The first step in this task is to identify the inputs of the function. We aim to provide a function that computes the basis vectors of a close-packed plane. We achieve this by assigning numbers to label types of structures. Additionally, we use the lattice constant and degree of rotation to correct the scale of the basis vectors. Considering the above factors, we define the function "`closePackedPlane`" to produce the required basis vectors.

```
1  % Define a function that returns the lattice basis vectors
2  % Input:
3  %         lttcCnst – Lattice constant
4  %         rotation – either a 2x2 rotational Matrix, or the
       rotation in degrees.
5  % Output: {a1,a2}  – real space vectors
6  function [a1, a2] = closePackedPlane(lttcCnst, rotation)
```

One of the basis vectors in a close packed plane form an angle of 60° to another. This leads to the typical sines given in the listing. Thus our basis vectors are:

```
7      % the basis vectors of hexagonal lattice
8      a1s=[1 0]*lttcCnst;
9      a2s=[-0.5 sqrt(3)/2]*lttcCnst;
```

We also add functionality to rotate the basis vectors by a particular angle. This rotation can either be expressed as an angle, or as a matrix: it is then trivial to implement its effect.

```
10     % rotation can either be an angle or rotational matrix
11     % assume rotation angle in degrees
```

```
    if length(rotation)==1
        Rmat = [cosd(rotation) sind(rotation);-sind(rotation)
            cosd(rotation)];
        a1 = [Rmat*a1s];
        a2 = [Rmat*a2s];
    % Adsorbate vectors

    % else its a matrix
    else
        a1 = rotation(1,1)*a1s + rotation(1,2)*a2s;
        a2 = rotation(2,1)*a1s + rotation(2,2)*a2s;
    end
end
```

Running this using an appropriate lattice constant and a corresponding rotation will generate the vectors describing a lattice. The rotation can usually be set as 0.

Task 3: Question

Write a function that generates the reciprocal basis vectors, from the output above.

Task 3: Solution

The equations relating the real space basis vectors $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{a}_3$ to those in reciprocal space $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ are defined as:

$$\mathbf{b}_1 = 2\pi \frac{\mathbf{a}_2 \times \mathbf{a}_3}{\mathbf{a}_1 \cdot (\mathbf{a}_2 \times \mathbf{a}_3)};$$
$$\mathbf{b}_2 = 2\pi \frac{\mathbf{a}_3 \times \mathbf{a}_1}{\mathbf{a}_2 \cdot (\mathbf{a}_3 \times \mathbf{a}_1)};$$
$$\mathbf{b}_3 = 2\pi \frac{\mathbf{a}_1 \times \mathbf{a}_2}{\mathbf{a}_3 \cdot (\mathbf{a}_1 \times \mathbf{a}_2)}.$$

To provide a complete set of basis vectors to span all of space, we pick two of the in plane basis vectors, and in addition, a third basis vector $\mathbf{a}_4$. This third vector points out of the plane. We define this in MatLab to have the direction `[0 0 1]`. Given that we can use the cross product to find the reciprocal vectors, the listing to achieve this is given below:

```
% define a function that takes a pair of basis vectors, and
    returns the corresponding reciprocal basis vectors
function [b1, b2] = unitCellVecsReciprocal(a1, a2)
    % define the out of plane vector
    a4 = [0 0 1]

    % define the scalar triple product
    tripleProd = ([a1 0]*[  cross( [a2 0], a4 )  ]')

    % reciprocal vector is the cross product
    b1=2*pi*cross([a2 0],[0 0 1])/tripleProd;

    % do not include the third component: keep only the in plane
        components
```

```
13      b1=b1(1:2);
14
15      % reciprocal vector is the cross product
16      b2=2*pi*cross(a4, [a1 0])/tripleProd
17
18      % do not include the third component: keep only the in plane
            components
19      b2=b2(1:2);
20  end
```

Task 4: Question

Write a function that finds all lattice points in a certain interval (*e.g.* plot all lattice points between -10 and 10, or -5 and 5).

Task 4: Solution

To solve this problem, we use a "mesh" to create a grid. How do we do this? The function `[X,Y]=meshgrid(x,y)` returns the matrices `X` and `Y`, where each row of `X` is `x`, and each column of `Y` is `y`. This grid represents the coefficients of each of the basis vectors in a given range.

```
1  % define a function that returns lattice points/top sites
2  function G = spanVecs(b1, b2, nVec1, nVec2)
3      % meshgrid generates the grid of coefficients to be used
4      [N_g, M_g] = meshgrid(nVec2, nVec1);
```

These coefficients can be then used to calculate the coordinates from the basis vectors.

```
5      % use the grid of coefficients to calculate the lattice
           points
6      Gx = N_g * b1(1) + M_g * b2(1);
7      Gy = N_g * b1(2) + M_g * b2(2);
8      G = [Gx(:) Gy(:)];
9  end
```

Task 5: Question

Consider the Ru(0001) surface. This is a close-packed plane. Draw a circle on each lattice point, and draw the unit cell.

Task 5: Solution

To start this problem, we note that we need to plot a figure. Thus our first step is to initialise the `figure` environment. We then declare the parameters of the lattice.

```
1  % initialise the figure
2  fig1 = figure;
3  title('Figure 1');
4
5  % define the lattice constant
6  a = 2.71;
```

```
7
8  % define the first basis vector
9  a1 = [a 0]';
10
11 % define the angle to rotate the first basis vector by to obtain
       the other
12 theta = 120;
13
14 % use this angle in a rotational matrix to perform the rotation
15 RotM = [cosd(theta) -sind(theta);sind(theta) cosd(theta)];
16
17 % produce the second basis
18 a2 = RotM*a1;
```

To draw a unit cell, we first need to identify the unit cell lattice points. We note that these are all possible permutations of the basis vectors, where the coefficients of each are assigned either 1 or 0. Note that when `end` is used as an index it returns the index of the last entry. Thus our task is achieved with:

```
19 % span four lattice points and plot the unit cell
20 % n and m are the coefficients of the lattice vectors at each of
       the positions on the unit cell, where n is the coefficient
      of a1, and m is the coefficient of a2
21 n = [0 1 1 0]';
22 m = [0 0 1 1]';
23
24 % calculate all possible positions of the vertices on a unit
      cell in one line, denoted coords
25 coords = n*a1'+m*a2';
26
27 % plot each of these positions, and connect them using a line.
      This traces the unit cell.
28 plot(coords([1:end 1],1), coords([1:end 1],2), '-', 'LineWidth',
       5);
29
30 % set the axes to be equal in ratio
31 axis equal
```

To find the lattice points (top sites), we use the `spanVecs` function we have written above. As an example, we have chosen to plot the points between -5 to 5 on both axes.

```
32 %calling function in the auxCls class
33 top_sites = auxCls.spanVecs(a1, a2, -5:5, -5:5);
```

Plot the top sites using the plot function. The points are plotted as circles using the argument `'o'`.

```
34 % do not overide the plot of the unit-cell
35 hold on;
36
37 %plot each site
38 plot(top_sites(:,1),top_sites(:,2),'o');
39
40 %set range of axes
41 axis([-2 6.5 -2 4]);
42
43 % set axes to be equal in ratio
44 axis equal
```

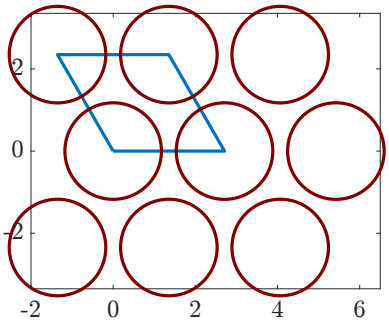

Figure 5 | Top sites and unit cell of a lattice.

Task 6: Question

Change the colours of the circles. First plot them in black, then blue, then red.

Task 6: Solution

The MatLab `plot` environment is versatile with many properties available for customisation. To name a few, `MarkerSize` specifies the size of the marker, `MarkerFaceColor` and `MarkerEdgeColor` specifies the colour of fill-in and the edge. The following example draws larger red circles at the lattice points.

```
1 %using the marker attributes to set the style
2 plot(top_sites(:,1),top_sites(:,2),'o','MarkerSize',97,'
      MarkerEdgeColor','red');
```

Figure 5 shows an example of the output.

Task 7: Question

Plot triangles instead of circles.

Task 7: Solution

The MatLab `plot` environment also includes different marker shapes, check the documentation for more information. The following example uses a triangle instead of a circle.

```
1 %'^' produces triangular markers
2 plot(top_sites(:,1),top_sites(:,2),'^','MarkerSize',97,'
      MarkerEdgeColor','red');
```

Task 8: Question

Derive the positions of the top sites, bridge sites and hollow sites on a face-centred cubic lattice. Plot each of these, on the same graph, using a different shape.

Task 8: Solution

We start by constructing a function finding the two hollow sites and three bridge sites. For convenience, we have converted all the coordinates used in this solution into row vectors.

```
1 % Define a function that finds the other sites using only top
      site coordinates
2 function [bridge_sites,hollow_sites] = construct_bridge_hollow(
      top_sites,a1,a2)
3
4     % If required, fix the dimensions by converting the basis
          vectors to row vectors
```

```
5       if iscolumn(a1), a1 = a1'; end
6       if iscolumn(a2), a2 = a2'; end
7
8       % if the number of columns is not 2, transpose (i.e. convert
            to be consistent with the number of rows above)
9       if size(top_sites, 2) ~= 2, top_sites = top_sites'; end
10
11      % three different bridge sites
12      bridge_sites(1) = {top_sites + a1/2};
13      bridge_sites(2) = {top_sites + a1+a2/2};
14      bridge_sites(3) = {top_sites + a1/2+a2/2};
15
16      % two different hollow sites due to fcc and hcp
17      hollow_sites(1) = {top_sites + (2/3)*(2*a1+a2)/2};
18      hollow_sites(2) = {top_sites + (2/3)*(a2+0.5*a1)};
19  end
```

Equipped with the coordinates of the different sites, we now attempt to plot them, differentiating each site using colours and shapes.

```
20  function plot_unit_cell_sites(top_sites,bridge_sites,
        hollow_sites)
21      % suppress refreshing the figure
22      hold on
23
24      % top sites marked by circles
25      plot(top_sites(:,1),top_sites(:,2),'o','MarkerFaceColor','y'
            ,"MarkerSize",15);
26
27      % bridge sites parallel to top sites also marked by circles,
            but with different colour
28      plot(bridge_sites{1}(:,1),bridge_sites{1}(:,2),'o','
            MarkerFaceColor','b',"MarkerSize",15);
29
30      % bridge sites with another substrate arrangement,
            represented by squares
31      plot(bridge_sites{2}(:,1),bridge_sites{2}(:,2),'s','
            MarkerFaceColor','b',"MarkerSize",15);
32
33      % bridge sites with another substrate arrangement,
            represented by diamonds
34      plot(bridge_sites{3}(:,1),bridge_sites{3}(:,2),'d','
            MarkerFaceColor','b',"MarkerSize",15);
35
36      % hollow sites of fcc and hcp represented with upward and
            downward triangles
37      plot(hollow_sites{1}(:,1),hollow_sites{1}(:,2),'v','
            MarkerFaceColor','g',"MarkerSize",15);
38      plot(hollow_sites{2}(:,1),hollow_sites{2}(:,2),'^','
            MarkerFaceColor','g',"MarkerSize",15);
39  end
```

This is a good point to pause and run the two functions you have written and check it produces the correct graphs.

Task 9: Question

Attach a legend to your graph.

Task 9: Question

Look up the MatLab documentation on `legend`. The following code adds a legend to the graph.

```
1      % Name the sites
2      %'Box' tells MatLab whether to show the box enclosing the
           legend
3      legend({'Top', 'Bridge', 'Bridge',   'Bridge', 'Hollow', '
           Hollow'}, 'Location', 'southoutside', 'Orientation', '
           horizontal', 'Box', 'off');
4      end
```

Figure 6 shows an example of the plot.

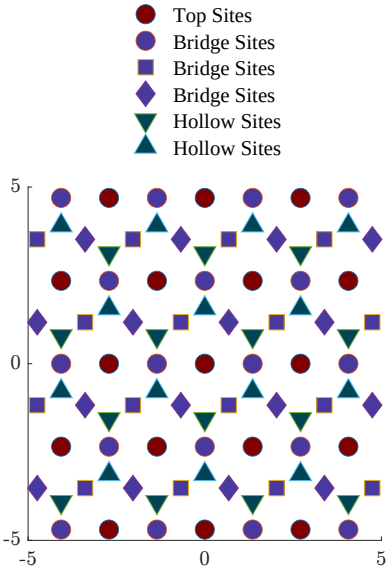

Figure 6 | Top, hollow, and bridge sites of a lattice.