Visualisation of concepts in condensed matter physics

Nikolai Plambech Nielsen, LPK331

2018-06-13

Contents

1 General implementation

All of the programs in this project will be implemented in the open source programming language Python, using the standard library, along with the NumPy and Matplotlib packages. Numpy adds the necessary tools for handling arrays and linear algebra operations, whilst Matplotlib is used for plotting the results.

One thing to note is that doing numerical calculations will lead to some numerical error. As such testing for numerical equality will often time yield the "wrong" result. Because of this, we employ the `isclose` function, which takes 4 parameters: `a`, `b`, `atol` and `rtol`. `a` and `b` are the numerical values to be tested for equality, `atol` is the absolute tolerance and `rtol` is the relative tolerance. The function then tests if $|\texttt{a}-\texttt{b}| \leq (\texttt{atol}+\texttt{rtol}\cdot|\texttt{b}|)$, with default values for `atol` and `rtol` being $1\cdot10^{-8}$ and $1\cdot10^{-5}$ respectively.

2 Lattices and crystal structure

There are three parts to the description of a crystal structure. First is the lattice. This is the mathematical "framework" upon which the physical part of the crystal lies. It can be defined in a number of ways, however the one we will use here is the following:

A lattice is defined as the infinite set of points produces by a linear combination of independent *primitive lattice vectors*, with integer coefficients.

Usually, the primitive lattice vectors are labelled $\mathbf{a}_i$, and the coefficients n_i, so for a d-dimensional lattice, the lattice points $\mathbf{R}$ are given by

$$R = \sum_{i=1}^{d} n_i \mathbf{a}_i \tag{2.1}$$

For this thesis we will mainly focus on the cases of $d = 3$ (visualization of lattices, families of lattice planes and scattering) and $d = 2$ (visualization of the Fermi surface).

A thing to note, is that the choice of primitive lattice vectors is not unique. A new set of primitive lattice vectors can be created by taking a linear combination of the original primitive lattice vectors, with integer coefficients. If the original set is ordered in a matrix $A = \begin{pmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n\end{pmatrix}$, and the new set in a matrix $B = \begin{pmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n\end{pmatrix}$, then $B = MA$, where M is the matrix containing the coefficients. This matrix must have integer entries, and its inverse likewise.

The integer entries of the direct matrix makes sure that any lattice point generated with the new set of primitive lattice vectors will also have integer coefficients when expressed in the old set of primitive lattice vectors. The integer entries of the inverse matrix then makes sure that this process will also happen in reverse.

This will come into play when trying to detect the lattice based on the users input of primitive lattice vectors.

Simple cubic	base centred cubic (bcc)
face centred cubic (fcc)	tetragonal
body centred tetragonal	orthorhombic
body centred orthorhombic	face centred orthorhombic
base centred orthorhombic	simple monoclinic
base centred monoclinic	hexaonal
triclinic	rhombohedral

Table 1: The 14 Bravais lattices

fcc, conventional	bcc, conventional
zincblende	wurtzite
diamond (zincblende with 1 atom)	

Table 2: Other available crystal presets

The second part is the unit cell. This is the building block of the lattice. It is a region of space which, when stacked will completely tile the space. Like with the choice of primitive lattice vectors, the choice of unit cell is not unique. In particular we distinguish between two type of unit cells: the smallest possible unit cell and everything else. The smallest possible unit cell is called a *primitive* unit cell, and must contain only one lattice point (it cannot contain no lattice points, as then it would not recreate the lattice when tiled). Any unit cell containing more than one lattice point is called a *conventional unit cell*. Usually a conventional unit cell is chosen for ease of calculation (as we will see in the scattering simulation), where the primitive lattice vectors constitute an orthogonal set.

The third part of the crystal structure is the basis. This is a description of the physical objects that make up the structure, and their position in relation to the lattice. In our case the objects are of course atoms.

The basis is specified as a list of vectors that are to be added to the lattice points, specifying the position of the atoms in the crystal.

The user can create any type of crystal they want by specifying any set of primitive lattice vector and supplying any desired basis. However a small selection of crystals will be available as presets. These include the 14 Bravais lattices with a 1 atom basis (at $(0, 0, 0)$):

Each of these will specify the primitive lattice vectors for a corresponding primitive unit cell. Furthermore the following crystal presets will also be available: Specifications of all of these presets are available in the appendix. (**LINK TO APPENDIX**)

Mathematically speaking, a lattice is infinite. A physical crystal is, of course, is not. However, even though a real crystal is finite, plotting all of the atoms would be infeasible, both due to the number of atoms, each atom would be way too small (if it was feasible, what would be the point of this project then?). So for the purposes of this project, only a couple of unit cells will be plotted. A good amount seems to be 8 unit cells. 2 in each of the directions specified by the primitive lattice vectors. This keeps the size of the plot relatively small, whilst still showing the important parts of the crystal structure.

However, plotting just the atoms will quickly become confusing. Because of this we plot grid lines. For the crystals that can be expressed as a lattice with orthogonal primitive lattice vectors and a basis (cubic, tetragonal and orthorhombic), we usually want to plot orthogonal grid lines, whilst for other crystals plotting grid lines along the lattice vectors will be more useful.

2.1 Implementation

In creating a program that plots crystal structures, the thought should always be on how the end product looks. Mainly we want the plot to be as clear and instructive as possible.

To achieve this, we have to be aware of how Matplotlib handles its 3-dimensional plots. One example of this is that the graphics engine will not hide objects plotted outside of the limits of the figure. This may lead to unpredicted artefacts like atoms "outside" of the crystal (atoms plotted on the edge of the crystal in a unit cell that will not be fully populated).

Another problem may be unfilled unit cells, where we expect them to be filled. Say we have specified an fcc lattice, and we want to view the conventional (cubic) unit cell. Plotting just the atoms at the lattice points with coefficients $n_i \in [0, 1, 2]$ for all i will give a parallelepiped and not the anticipated

cube. To do this we need to extend the range of coefficients to fill out any incomplete unit cells. In practise this is done by adding the maximum magnitude of coefficients (in this case 2) to the upper range of coefficients, and subtracting it from the lower range. In this case the resulting range of coefficients is $n_i \in [-2, -1, 0, 1, 2, 3, 4]$.

2.2 Step-by-step

Initially the program will either load the chosen crystal preset in such a way as to make the resulting plot as informative as possible (eg. place lattice vectors along cardinal axes for an orthogonal lattice, to make plotting grid lines easier). If the user manually specifies the lattice and basis, the program will try to classify the lattice according to the specifications in the appendix (**LINK TO APPENDIX**). Next the program checks whether or not the crystal should be rotated to make plotting prettier.

In general the rotation algorithm tries to align one lattice vector with the x-axis. $\mathbf{a}_1$ is preferred, but is only chosen to lie along the x-axis if it forms an orthogonal pair with at least one other primitive lattice vector. The second primitive lattice vector of the pair ($\mathbf{a}_2$ being preferred) is then aligned along the y-axis. If the three primitive lattice vectors form an orthogonal set, then the last vector of the set will now be aligned along the z-axis.

The actual rotation is done by rotating the whole crystal (each primitive lattice vector and all vectors in the basis) along the cross product between the initial vector and the destination vector. Rotating the crystal such that $\mathbf{a}_1$ lies along the x-axis is done by rotating along $\mathbf{a}_1 \times \hat{\mathbf{x}}$, with an angle of $\sin\theta = |\mathbf{a}_1 \times \hat{\mathbf{x}}|/|\mathbf{a}_1|$.

However, this might rotate the crystal the wrong way, depending on the orientation between the two vectors. Because of this the program checks whether or not the rotated initial vector and the destination vector is parallel (that is, if the rotated $\mathbf{a}_1$ is parallel to $\hat{\mathbf{x}}$). If this is not the case, the whole crystal is rotated about the same vector, with an angle -2θ.

Five of the Bravais lattices have specialised rotation functions: hexagonal, base centred monoclinic and the three face centred lattices.

For the hexagonal, the program detects which primitive lattice vectors constitute the triangular lattice and orients them such that one is along the x-axis and the other is in the xy-plane (easily done by rotating the third primitive lattice vector such that it is parallel with the z-axis). The same approach is used for the base centred monoclinic, but here the program always aligns $\mathbf{a}_1$ along the x-axis, and $\mathbf{a}_2$ in the xy-plane. Here however, the easy option of rotating $\mathbf{a}_3$ is not available. Instead the program uses the fact that the vector rejection of $\mathbf{a}_2$ with $\mathbf{a}_1$ is orthogonal to $\mathbf{a}_1$ (the vector rejection of $\mathbf{a}_2$ with $\mathbf{a}_1$ being $\mathbf{a}_2$ minus the projection of $\mathbf{a}_2$ along $\mathbf{a}_1$). This vector rejection is then in the yz-plane, and the crystal can be rotated along $\mathbf{a}_1$ with the angle the rejection makes with $\hat{\mathbf{y}}$: $\cos\theta = \mathbf{a}_{2,rej} \cdot \hat{\mathbf{y}}/|\mathbf{a}_{2,rej}|$.

For the face centred lattices, the ideal, rotated lattice is created from the magnitude of the primitive lattice vectors: if $|\mathbf{a}_1| = a, |\mathbf{a}_2| = b, |\mathbf{a}_3| = c$, then $\mathbf{a}'_1 = (a/2, b/2, 0), \mathbf{a}'_2 = (a/2, 0, c/2), \mathbf{a}'_3 = (0, b/2, c/2)$. First the crystal is rotated such that $\mathbf{a}_1$ aligns with $\mathbf{a}'_1$, by using their cross product. Then the crystal is rotated such that the now rotated $\mathbf{a}_2$ aligns with $\mathbf{a}'_2$, via the vector rejection of $\mathbf{a}_2$ with $\mathbf{a}'_2$. (**THIS METHOD AND THE ONE FOR THE HEXAGONAL LATTICE ASSUMES THAT THE PRIMITIVE LATTICE VECTORS ARE SPECIFIED AS IN THE APPENDIX**).

Next, the program calculates the limits of the plot box. This is done by calculating the 8 possible vectors arising from linear combinations of the primitive lattice vectors, with coefficients from the specified minimum and maximum coefficients and taking the limits of the plot box as the minimum and maximum values for these 8 vectors. For example, say that the specified minimum and maximum coefficients are $[0, 0, 0]$ and $[2, 2, 2]$ respectively, then the 8 vectors are $\mathbf{v}_1 = \mathbf{0}, \mathbf{v}_2 = 2\mathbf{a}_1, \mathbf{v}_3 = 2\mathbf{a}_2, \mathbf{v}_4 = 2\mathbf{a}_3, \mathbf{v}_5 = 2(\mathbf{a}_1 + \mathbf{a}_2), \mathbf{v}_6 = 2(\mathbf{a}_1 + \mathbf{a}_3), \mathbf{v}_7 = 2(\mathbf{a}_2 + \mathbf{a}_3), \mathbf{v}_8 = 2(\mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3)$. These form a parallelepiped from $2\mathbf{a}_1, 2\mathbf{a}_2, 2\mathbf{a}_3$, where the plot box is a cuboid, with edges along the cardinal axes, that just contain the aforementioned parallelepiped.

With the lattice and basis rotated, and the limits of the plot box found, the crystal is generated. This is done by looping over the three ranges specified by the minimum and maximum coefficients, creating each lattice point $\mathbf{R}$ by Eq. (2.1). For each lattice point n atomic positions are created by adding one of the n vectors in the basis to the lattice point. Furthermore lists of the colours and sizes associated with each atom are also created at this point. After creating all the atoms, the program deletes any that may lie outside the limits of the plot box.

The only thing missing now is to create the grid lines and plot everything. The program has two ways of creating grid lines: along the primitive lattice vectors and along the cardinal axes.

Creating grid lines along the primitive lattice vectors works by taking each lattice point $\mathbf{R}_n$, and

finding the lattice point $\mathbf{R}_m$ the furthest away from it, in the (postive) direction of these lattice vectors, such that $\mathbf{R}_m = R_n + \alpha \mathbf{a}_i$, where $\alpha > 0$, for all $i \in [1, 2, 3]$, and creating lines between $\mathbf{R}_n$ and $\mathbf{R}_m$. This does create duplicate grid lines, but these do not show on the final plot, since they are all plotted with the same width and colour.

Creating grid lines along the cardinal axes works by finding the minimum spacing between lattice points on these axes (called a_x, a_y and a_z), and using these as the spacing between grid lines. The program then finds the maximum and minimum coordinates of lattice points along the cardinal axes (called x_{min}, x_{max}, etc.). This is used to create ranges corresponding to each lattice points on the cardinal axes: The range for the x-axis starting at x_{min} and ending at x_{max} with steps of a_x.

These ranges then specify a grid of lattice points on the xy, xz and yz-planes. The program then creates lines orthogonal to these planes, stretching from z_{min} to z_{max} for the points in the xy-plane, and similarly for the other two planes.

Next a blank figure is created with an orthogonal projection and limits as calculated above. The atoms are plotted with colours and sizes specified by the user. The grid lines are plotted with a uniform size and colour and lastly the primitive lattice vectors are plotted with corresponding labels.