Obtaining the band structure of a 2D hexagonal lattice using the tight binding model with a MATLAB GUI

The following text is a description of the student project that has been done during the course "*molecular and solid state physics*" at the TU Graz. It is about the calculation of the band structure of a material in a 2 dimensional hexagonal lattice with two different kinds of atoms in the unit cell. As an example this kind of structure can be found at boron nitride. The calculation uses the tight binding model with the nearest neighbour approximation (only the nearest neighbours contribute to the solution, the terms of neighbours farer away are considered as very small and therefor neglected).

The text consists of two different parts, where in the first one the user interface of the MATLAB program is described and in the second one the structure of the program code behind the user interface is explained. It is not intended to give the reader an explanation of how to obtain the solution of the band structure in a mathematical way using the tight binding model nor will be an instruction of how to set up an MATLAB GUI. There are many different tutorial videos for MATLAB GUIs on You Tube (like [1]) and there are also some enlightening explanations of the tight binding model that can be found on the homepage of the earlier mentioned course (like [2] or [3]).

1.GUI Interface

The user interface consists of four different text fields where you can edit the parameters for your calculations (see Figure 1):

- Lattice constant of your material (for hexagonal Boron Nitride it is approx. 1.446 Å)
- Energy of the first kind of atom in the unit cell
- Energy of the second kind of atom in the unit cell
- Overlap integral

There are two different push-buttons:

- Calculation: Pushing this button starts the calculation with the pre-set parameters
- Cancel: This button ends the program and the user is returned to the command window

If the user gets lost in the interface just point the cursor at one of the mentioned fields and after a few seconds there will be a tooltip which explains what use this field (text field, push button) has.

The graphical presentation of your result is done by two different figures that can be found on the right side of the interface. The upper one is a three dimensional presentation of the band structure. The presentation can be rotated with the appropriate button in the tool bar and you can also zoom in and zoom out with the other buttons of the tool bar. In this figure there also can be the first Brillouin zone found, diagrammed as the black hexagon. The red triangle indicates the used k- and energy-values in the lower presentation, which is a 2 dimensional presentation of the obtained band structure.

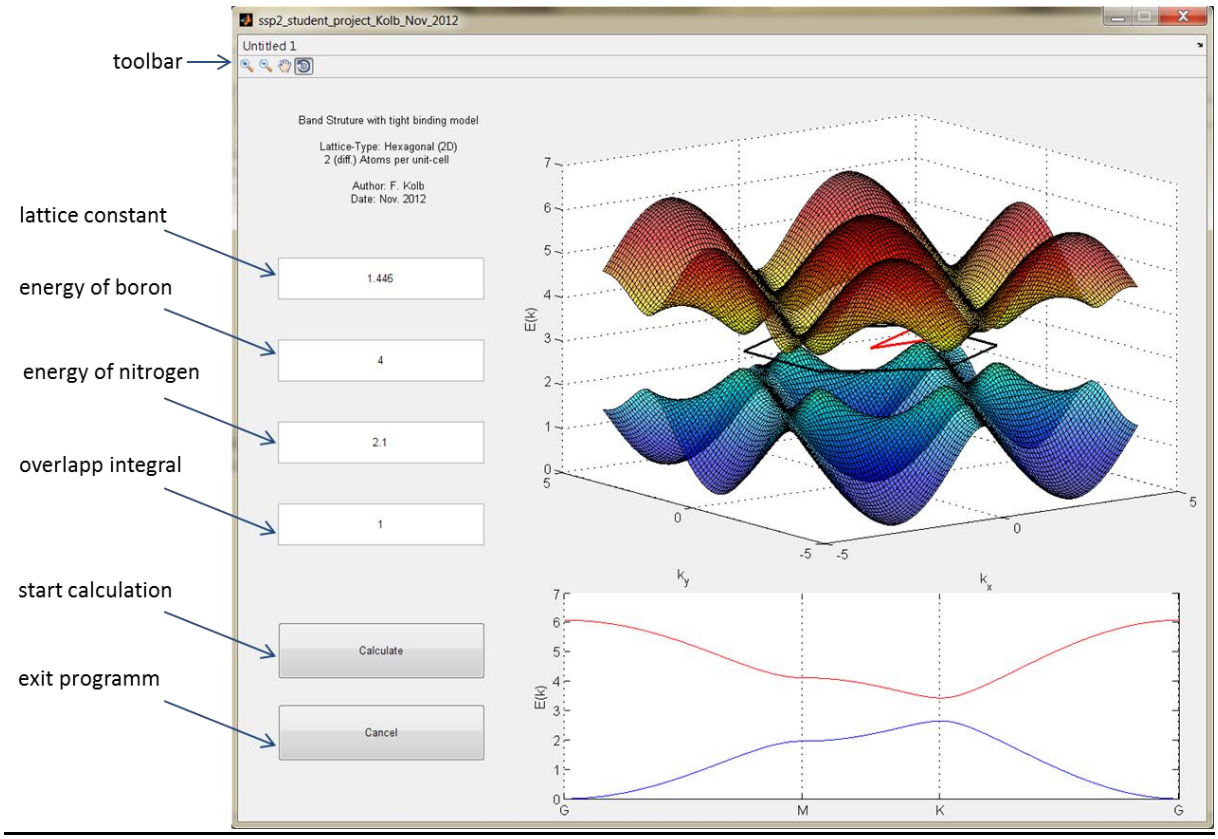

Figure 1 Description of the user interface

2. Structure of the program code

The program code can be distinguished between two main parts. Each part is for one of the two presentations (2D and 3D presentation of the energy dispersion relation). To make the code more comprehensible the two parts are build up the same way. At the end of this chapter the reader can find the whole program code that is executed when pressing the *Calculate* button.

Each part consists of the following sections:

- Parameter: At this section the program gets the values of the text fields like energy, overlap integral etc. and converts them from a string format into a double format. After that the k-vectors are created for the area at which the calculation should be executed.

- Calculation of the energy values: Like earlier mentioned there will be no description of how to get to the energy dispersion relation of this problem. It works quite the same way as obtaining the energy dispersion relation of graphene (See [2] and scroll down to the very bottom of the page). However, the tight binding wave function of h-BN is:

$$\psi_k(\vec{r}) = \frac{1}{\sqrt{N}} \sum_{h,j} e^{i(h\vec{k}\hat{a}_1 + j\vec{k}\hat{a}_2)} \left(c_B \emptyset_B(\vec{r} - h\hat{a}_1 - j\hat{a}_2) + c_N \emptyset_N(\vec{r} - h\hat{a}_1 - j\hat{a}_2)\right)$$

Legend:

$\psi_k(\vec{r})$…Tight binding wave function

$\hat{a}_1 = \frac{\sqrt{3}}{2} a\,\hat{x} + \frac{a}{2}\hat{y}\,;\ \hat{x} = \begin{pmatrix}1\\0\\0\end{pmatrix}$ …First primitive vector

$\hat{a}_2 = \frac{\sqrt{3}}{2} a\,\hat{x} - \frac{a}{2}\hat{y};\ \hat{y} = \begin{pmatrix}0\\1\\0\end{pmatrix}$…Second primitive vector

$\emptyset_B, \emptyset_N$…Orbital functions of boron and nitrogen respectively

c_B, c_N…Coefficients of the orbital functions

If you multiply the time independent Schrödinger equation from the left by the orbital functions of the two different atoms you will get two different equations. Written in matrix form they will look like:

$$\begin{pmatrix} \varepsilon_B - E & -t\left(1 + e^{-i\vec{k}\hat{a}_1} + e^{-i\vec{k}\hat{a}_2}\right) \\ -t\left(1 + e^{i\vec{k}\hat{a}_1} + e^{i\vec{k}\hat{a}_2}\right) & \varepsilon_N - E \end{pmatrix} \begin{pmatrix} c_B \\ c_N \end{pmatrix} = 0$$

If the determinant is set to zero the equation will have solutions and you are able to determine the energy dispersion relation:

$$E(k_x, k_y) = \frac{(\varepsilon_B + \varepsilon_N)}{2} \pm \sqrt{\frac{(\varepsilon_B - \varepsilon_N)^2}{2} + 4t^2\left(\left(\cos\frac{k_y}{2}a\right)^2 + \cos\frac{\sqrt{3}k_x}{2}a\,\cos\frac{k_y}{2}a + \frac{1}{4}\right)}$$

Legend:

$E(k_x, k_y)$…Energy of the system depending on the k-vectors
$\varepsilon_B, \varepsilon_N$…Energy of the two kinds of atoms of the unit cell (e.g boron and nitrogen)
t…Overlap integral
k_x, k_y…k-vectors in x or y direction

Note: If the same energy value for both kinds of atoms is chosen the solution will be the energy dispersion relation of graphene.

- Graphical presentation of the solution for the 2 dimensional and the 3 dimensional presentation of the energy dispersion relation

```
% --- Executes on button press in pushbutton_calc.
function pushbutton_calc_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_calc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%-------General parameters--------
t=str2double(get(handles.edit_t,'string'));
lattice=str2double(get(handles.edit_lattice,'String'));
eps_en1=str2double(get(handles.edit_eps1,'String'));
eps_en2=str2double(get(handles.edit_eps2,'String'));


%Starting with the 3D-presentation....

%Creating the necessary k-vectors for the calculation
k_vec_x = linspace(-2*pi/(lattice),2*pi/(lattice),100);
k_vec_y = linspace(-2*pi/lattice,2*pi/lattice,100);
[k_meshx,k_meshy]=meshgrid(k_vec_x,k_vec_y);


%Energy values with the preset parameters in the 3D presentation
energy_mesh=NaN([size(k_meshx,1),size(k_meshx,2),2]);
for a = 1 : size(k_meshx,1)
    energy_mesh(:,a,1) = ((eps_en1+eps_en2))/2 + sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_meshy(:,a)/2*lattice).^2)+cos
(sqrt(3)/2*k_meshx(:,a)*lattice).*cos(k_meshy(:,a)/2*lattice))+1/4);
    energy_mesh(:,a,2) = ((eps_en1+eps_en2))/2 - sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_meshy(:,a)/2*lattice).^2)+cos
(sqrt(3)/2*k_meshx(:,a)*lattice).*cos(k_meshy(:,a)/2*lattice))+1/4);
end


%Graphic presentation
set(handles.axes_mesh,'Color','w')
set(handles.axes_mesh,'NextPlot','replace')
surf(k_meshx,k_meshy,real(energy_mesh(:,:,1)),'parent',handles.axes_mesh,'FaceAlpha',0.6)
set(handles.axes_mesh,'NextPlot','add')
hold on
surf(k_meshx,k_meshy,real(energy_mesh(:,:,2)),'parent',handles.axes_mesh,'FaceAlpha',0.6)
colormap('jet')
shading interp
%Plot of the brillouin zone (black) and the triangular path (red) of the
%dispersion plot
plot3(2*pi/(sqrt(3)*lattice)*[1,0,-1,-1,0,1,1],2*pi/(3*lattice)*[1,2,1,-1,-2,-1,1],(eps_en1+eps_en2)/2*ones
(1,7),'k-','LineWidth',2,'parent',handles.axes_mesh)
plot3(2*pi/(lattice*sqrt(3))*[0,1,1,0],2*pi/(3*lattice)*[0,0,1,0],(eps_en1+eps_en2)/2*ones
(1,4),'r-','LineWidth',2,'parent',handles.axes_mesh)
hx = xlabel('k_x','parent',handles.axes_mesh);
hy = ylabel('k_y','parent',handles.axes_mesh);
hz = zlabel('E(k)','parent',handles.axes_mesh);
axis equal
grid off
hold off
```

Figure 2 Program code for the 3D presentation of the energy dispersion relation (1st main part)

```
%2nd main part of the program code....2D presentation
%k-values along the red coloured triangle
k_vec_k=nan(2,1000);
ener_k=k_vec_k;
k_vec_m=nan(2,ceil(1000/sqrt(3)));
ener_m=k_vec_m;
k_vec_g=k_vec_k;
ener_g=k_vec_g;

k_vec_k(1,:)=linspace(0,2*pi/(lattice*sqrt(3)),1000);
k_vec_k(2,:)=0;

k_vec_m(1,:)=k_vec_k(1,end);
k_vec_m(2,:)=linspace(0,2*pi/(lattice*3),ceil(1000/sqrt(3)));

k_vec_g(1,:)=linspace(2*pi/(lattice*sqrt(3)),0,1000);
k_vec_g(2,:)=linspace(2*pi/(lattice*3),0,1000);

%Energy values with the preset parameters and the k-values along the red
%triangle in the 2D presentation (3 energy vectors of the 3 different sides
%of the triangle)
ener_k(1,:)=((eps_en1+eps_en2))/2 + sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_vec_k(2,:)/2*lattice).^2)+cos(sqrt
(3)/2*k_vec_k(1,:)*lattice).*cos(k_vec_k(2,:)/2*lattice))+1/4);
ener_k(2,:)=((eps_en1+eps_en2))/2 - sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_vec_k(2,:)/2*lattice).^2)+cos(sqrt
(3)/2*k_vec_k(1,:)*lattice).*cos(k_vec_k(2,:)/2*lattice))+1/4);

ener_m(1,:)=((eps_en1+eps_en2))/2 + sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_vec_m(2,:)/2*lattice).^2)+cos(sqrt
(3)/2*k_vec_m(1,:)*lattice).*cos(k_vec_m(2,:)/2*lattice))+1/4);
ener_m(2,:)=((eps_en1+eps_en2))/2 - sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_vec_m(2,:)/2*lattice).^2)+cos(sqrt
(3)/2*k_vec_m(1,:)*lattice).*cos(k_vec_m(2,:)/2*lattice))+1/4);

ener_g(1,:)=((eps_en1+eps_en2))/2 + sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_vec_g(2,:)/2*lattice).^2)+cos(sqrt
(3)/2*k_vec_g(1,:)*lattice).*cos(k_vec_g(2,:)/2*lattice))+1/4);
ener_g(2,:)=((eps_en1+eps_en2))/2 - sqrt(((eps_en1-eps_en2)^2)/4 + 4*t^2*((cos(k_vec_g(2,:)/2*lattice).^2)+cos(sqrt
(3)/2*k_vec_g(1,:)*lattice).*cos(k_vec_g(2,:)/2*lattice))+1/4);


%Graphic presentation (3 different plots for
%every 3 sides of the triangle)
plot_ind=(1:(length(k_vec_m)+length(k_vec_k)+length(k_vec_g)))/(length(k_vec_m)+length(k_vec_k)+length(k_vec_g));%x-axes of the
dispersion plot
plot(plot_ind,real([ener_k(1,:),ener_m(1,:),ener_g(1,:)]),'r-','parent',handles.axes_enerDisp)
hold on
plot(plot_ind,real([ener_k(2,:),ener_m(2,:),ener_g(2,:)]),'b-','parent',handles.axes_enerDisp)
xlabel('k')
ylabel('E(k)')
hold off
set(handles.axes_enerDisp,'XGrid','on')
set(handles.axes_enerDisp,'XTickMode','manual')
set(handles.axes_enerDisp,'XTickLabelMode','manual')
set(handles.axes_enerDisp,'XTick',[0;0.3878;0.6121;1])
set(handles.axes_enerDisp,'XTickLabel',['G';'M';'K';'G'])
guidata(hObject,handles)
```

Figure 3 Program code for the 2D presentation of the energy dispersion relation (2nd main part)