Quantum Transport in Nanoporous Graphene

Rasmus Kronborg Finnemann Wiuff (s163977)[*]

and Christoffer Vendelbo Sørensen (163965)[†]

Technical University of Denmark[‡]

(Dated: May 1st 2019)

Abstract: Abstract...

CONTENTS

* E-mail at rwiuff@dtu.dk

† E-mail at chves@dtu.dk

‡ Homepage of the Technical University of Denmark http://www.dtu.dk/english/;

Project Repository: https://github.com/rwiuff/QuantumTransport

I. INTRODUCTION

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

[1]

II. QUANTUM TRANSPORT

A. Ballistic quantum transport

As graphene is a two dimensional material that consists of carbon atoms arranged in a hexagonal pattern, features in such a material can approach nanometer and sub nanometer scales. Because of the small scale the electrical properties and the electrical nature of the material is greatly changed. Normal drift-diffusion current models describe electric charges per area and current per area, but because the conductor is graphene, it can be considered one dimensional. This makes drift-diffusion models insufficient to describe the electrical transport and properties of graphene because the drift-diffusion model is based on scattering of multiple electrons and the mean free path between scattering. Because of the strong binding between atoms in graphene hardly any phonons are present in the material, even at room temperature. In the ballistic model, this means that electrons are not affected by phonons, i.e. no electrons will be excited by phonons, and that they move through the material as waves. Furthermore the model looks at only one electron at a time in the presence of an electron gas. This model has been used with big success for regular graphene and it seems the model also gives a good approximation for NPG.

B. π-orbitals and π-electrons

The main scope of this paper is dealing with electron transport in novel nanoporous graphene devices. When modeling such transport one needs to address the orbital structure of carbon lattices and later this will motivate the use of tight-binding and Green's functions. In its basic form graphene can be divided into rings of carbon atoms as shown in Fig. 1. In the (x, y)-plane the carbon atoms are bound in sp^2 orbitals as shown in Fig. 2.

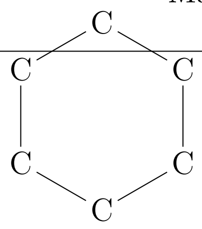


Figure 1: *Graphene lattices consists of hexagonal arrangements of carbon atoms.*

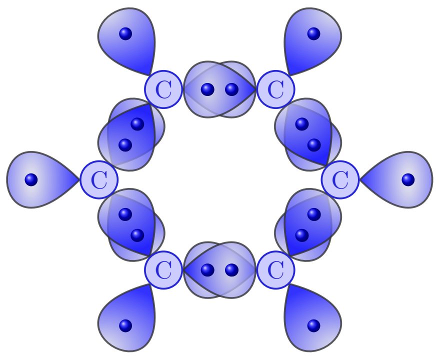


Figure 2: *Carbon atoms in a hexagonal lattice are sp^2 hybridised in the (x, y)-plane.*

This hybridisation lock all but one valence electron for the carbon atoms. These electrons exists in a p-orbital in the z-direction. Fig. 3 shows the valence orbitals of carbon.

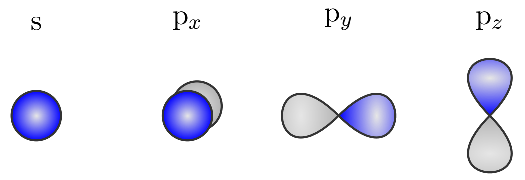


Figure 3: *The valence orbitals of carbon.*

The last electron in the p_z orbital does not mix with the tightly bound s, p_x and p_y electrons and moves more freely. Thus these electrons have higher energies compared to the sp^2 electrons and occupy states at the Fermi level. These electrons dominates transport in the graphene lattice. The p_z orbital is also known as the π-orbital and as such the electron lying there is called a π-electron. Through a carbon lattice the π-electrons will travel through π-orbitals, switching sign as they go as shown in Fig. 4.

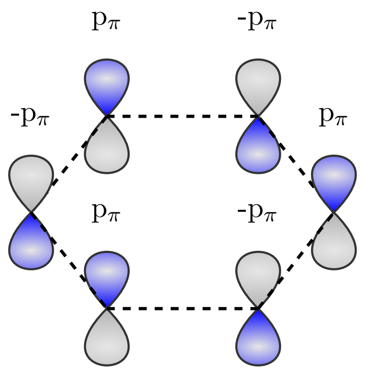


Figure 4: *When jumping from one carbon atom to another, the π-electron goes between p_π and -p_π. Such a jump is described by two matrix elements in the system's Hamiltonian.*

C. Tight-binding

Now that the transport carrying electrons are defined, one must choose a formalism for the transport itself. Introducing: **"The Tight-Binding approximation"**. In this approximation the electrons are considered being tightly bound to the atoms. Contrary to a free electron gas approximation, the electrons does not spend time in between orbitals, but jump from orbital in atom a to orbital in atom b. In this world view the Hamiltonian contains a matrix of hopping elements for a collection of neighbouring atomic orbitals, i.e. molecular orbitals, as well as the energy contained within each orbital (which will be addressed later on). This can be done by describing the orbitals as a Linear Combination of Atomic Orbitals (LCAO). The solution to the Schrödinger equation is then:

$$\Psi_{\text{MO}} = \sum_{\alpha,R} c_{\alpha,R}\phi_\alpha(R) \tag{C.1}$$

where $\phi_\alpha(R)$ is some atomic orbital at position R, with α denoting the valence of the orbital $(2s, 2p_x, 2p_y, 2p_z)$. In electron transport the states close to the Fermi level is of interest. These are namely the highest occupied molecular orbitals (HOMO), or the lowest unoccupied molecular orbitals (LUMO). As stated earlier only the π-electrons is then of interest. The electrons' motion can be described with the hopping matrix of elements:

$$V_{pp\pi} = \langle\phi_\pi(1)|\,\hat{H}\,|\phi_\pi(2)\rangle \tag{C.2}$$

Physically this means that there is a potential between the π orbitals of neighbouring atoms 1 and 2. The element

$$\epsilon_0 = \langle\phi_\pi(1)|\,\hat{H}\,|\phi_\pi(1)\rangle \tag{C.3}$$

is the average energy of the electron on atom 1 and, it is normal to define the hopping energy relative to this:

$$\epsilon_0 = 0 \tag{C.4}$$

If the atoms or their environment differs, so does the on-site potential.

D. The benzene molecule

As an example the Hamiltonian of benzene is considered. In Fig. 5 one can see the indices of a benzene molecule. Remember that $\langle\phi_\pi(1)|\,\hat{H}\,|\phi_\pi(1)\rangle = 0$ and Eq. (C.2), the Hamiltonian reads:

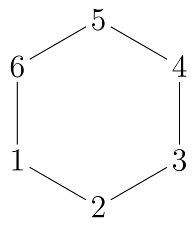


Figure 5: *Indices of a benzene molecule*

$$\mathbf{H} = V_{pp\pi} \begin{array}{c c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \tag{D.1}$$

As a helping aid, Eq. (D.1) shows the atomic indices of the atom on the top and to the left of the matrix. This will give an understanding of how to work with such matrices. The structure of the benzene molecule is rotationally symmetric and rotating the indices one sixth must yield the same Hamiltonian. Consider the energy eigenvector:

$$\phi = \begin{pmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{pmatrix} \tag{D.2}$$

There must exist an operator that rotates the indices as such:

$$C_6\phi = \begin{pmatrix} c_2 & c_3 & c_4 & c_5 & c_6 & c_1 \end{pmatrix} \tag{D.3}$$

The rotated Hamiltonian is the same, and thus C_6 and $\mathbf{H}$ commutes. The rotated vector must be an eigenvector with the same energy and it should be possible to find simultaneous eigenvectors to C_6 and $\mathbf{H}$.

$$C_6\phi = \begin{pmatrix} c_2 & c_3 & c_4 & c_5 & c_6 & c_1 \end{pmatrix} = \lambda \begin{pmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{pmatrix} \tag{D.4}$$

This operator C_6 is represented with the matrix:

$$\mathbf{C}_6 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{D.5}$$

It can quickly be shown that the normalised eigenvectors to C_6 are

$$\phi_n = \frac{1}{\sqrt{6}} \begin{pmatrix} \lambda_n^0 & \lambda_n^1 & \lambda_n^2 & \lambda_n^3 & \lambda_n^4 & \lambda_n^5 \end{pmatrix}, \quad \lambda_n = \exp\{-i2\pi n/6\}, \quad n = 0, 1, 2, 3, 4, 5 \tag{D.6}$$

These eigenvectors are also eigenvectors for $\mathbf{H}$ with the eigenvalues:

$$\varepsilon_n = \lambda_n + \lambda_{n-1} = 2\cos n\pi/3 \tag{D.7}$$

Thus thanks to the rotational symmetry it was possible to find the eigenvectors and eigenenergies for the Hamiltonian.

III. BAND STRUCTURE OF NPG

A. Creating on-site Hamiltonian and hopping matrices for the system

In order to calculate and visualise the band structure of the NPG within a unit cell, the on-site Hamiltonian along with its hopping matrices must be defined. The on-site Hamiltonian lies within the unit cell of the system. As the unit cell is repeated periodically the on-site Hamiltonian fills out one full period of the system. The unit cell is defined with respect given unit vectors which determine the periodicity of the system. By these definitions the elements of the hopping matrices represents hops between the repeated unit cells with respect to the periodicity. See Fig. 8 for a visual representation of the structure with its periodicity.

On the premise of the Tight Binding Model, the next step to find all the nearest neighbouring atoms within the given set of atom coordinates to define the on-site Hamiltonian $\mathbf{h_0}$ and then define the hopping matrices $\mathbf{V}$ and its conjugate $\mathbf{V}^{\dagger}$. This is because the Tight Binding model only allow for hopping of electrons between nearest neighbours which in this case is the periodic boundary conditions set for the system.

Maybe implement code pieces or explain code terms that executes these operations. i.e. "By using two for loops and the numpy function np.subtract(), we get the all the combinations of... etc. This should be considered throughout the section. This can be done simply by taking what is similar to an outer product of the coordinate matrix with itself only here every combination of coordinates are subtracted not multiplied, hereafter taking the norm of all those combinations. Such an outer product in python is done using numpy as shown in Listing 1.

```
33 def Onsite(xyz, Vppi):
34     h = np.zeros((xyz.shape[0], xyz.shape[0]))
35     for i in range(xyz.shape[0]):
36         for j in range(xyz.shape[0]):
37             h[i, j] = LA.norm(np.subtract(xyz[i], xyz[j]))
38     h = np.where(h < 1.6, Vppi, 0)
39     h = np.subtract(h, Vppi * np.identity(xyz.shape[0]))
40     return h
```

Listing 1: *The outer operator in numpy is manifested as two nested loops. On lines 35-37 each atomic distance is calculated. Line 38 replaces all nearest neighbour distances with an input potential, leaving the rest as zero. Lastly the diagonal is subtracted from the matrix.*

This will produce a matrix which contain all distances between all atoms in the NPG. Once obtained a threshold will be applied to sort out the nearest neighbours within the matrix. In this specific case the coordinates given was with respect to unit vectors with a lattice constant of 1.00 Å, which means that the threshold in this case should be 1.60 Å which is the inter-atomic distance for nearest neighbours in a graphene mesh. All distances above 1.60 Å will be changed to a 0-element in the on-site Hamiltonian as no hopping of electrons is possible between atoms that are more than one inter-atomic distance apart.

All distances below the threshold will be changed to 1 to represent a hop between to atoms. The on-site Hamiltonian is then multiplied by at scalar which is the on-site potential V_{ppi} which in this case is -1. Now the on-site Hamiltonian is complete and the construction of the hopping matrices can begin.

As the on-site Hamiltonian represents a (three)two-dimensional structure one has to make sure that hopping between the nearest neighbour atoms relative to that of a repeated unit cell (on-site Hamiltonians) can described in all directions in the plane. Effectively this means that six hopping matrices should be created. One in the x-direction, one in the y-direction, one in the xy-direction and their hermitian conjugates. Graphically this corresponds to a structure of this kind:

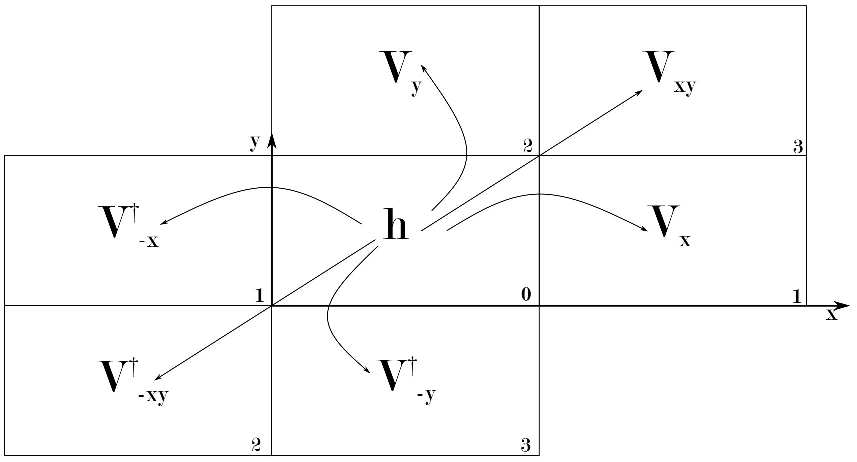

$\mathbf{V}_y$
$\mathbf{V}_{xy}$
y
2
3
$\mathbf{V}^\dagger_{-x}$
h
$\mathbf{V}_x$
1
0
1
x
$\mathbf{V}^\dagger_{-xy}$
$\mathbf{V}^\dagger_{-y}$
2
3

Figure 6: *Representative figure of how the on-site Hamiltonian along with its hopping matrices are structured*

In practice this is done by shifting the original coordinates by the given unit vectors of the system in each direction, adding the unit vector to the coordinate matrix making three new matrices. Now that three shifted matrices has been obtained, the same routine to find the distances as with the on-site Hamiltonian can be utilised, the only difference being that the outer subtraction will be between the on-site Hamiltonian and the shifted matrices respectively to make sure it is the distance between the on site Hamiltonian and the shifted matrices that is calculated. Again by replacing the distances in the shifted matrices with 1 and 0 according to the inter-atomic threshold, the hopping matrices are obtained. The three hopping matrices denoted $\mathbf{V}_{1x}$, $\mathbf{V}_{2y}$ and $\mathbf{V}_{3xy}$ represent hopping in the "forward" direction (left-to-right). To create the hopping matrices hopping in the "backwards" (right-to-left) direction one simply has to transpose the hopping matrices. These matrices are denoted with a dagger: $\mathbf{V}^\dagger_{1x}$, $\mathbf{V}^\dagger_{2y}$ and $\mathbf{V}^\dagger_{3xy}$. Fig. 7 is a figure of the resulting matrix-maps from the calculation, stitched together like in Fig. 6.

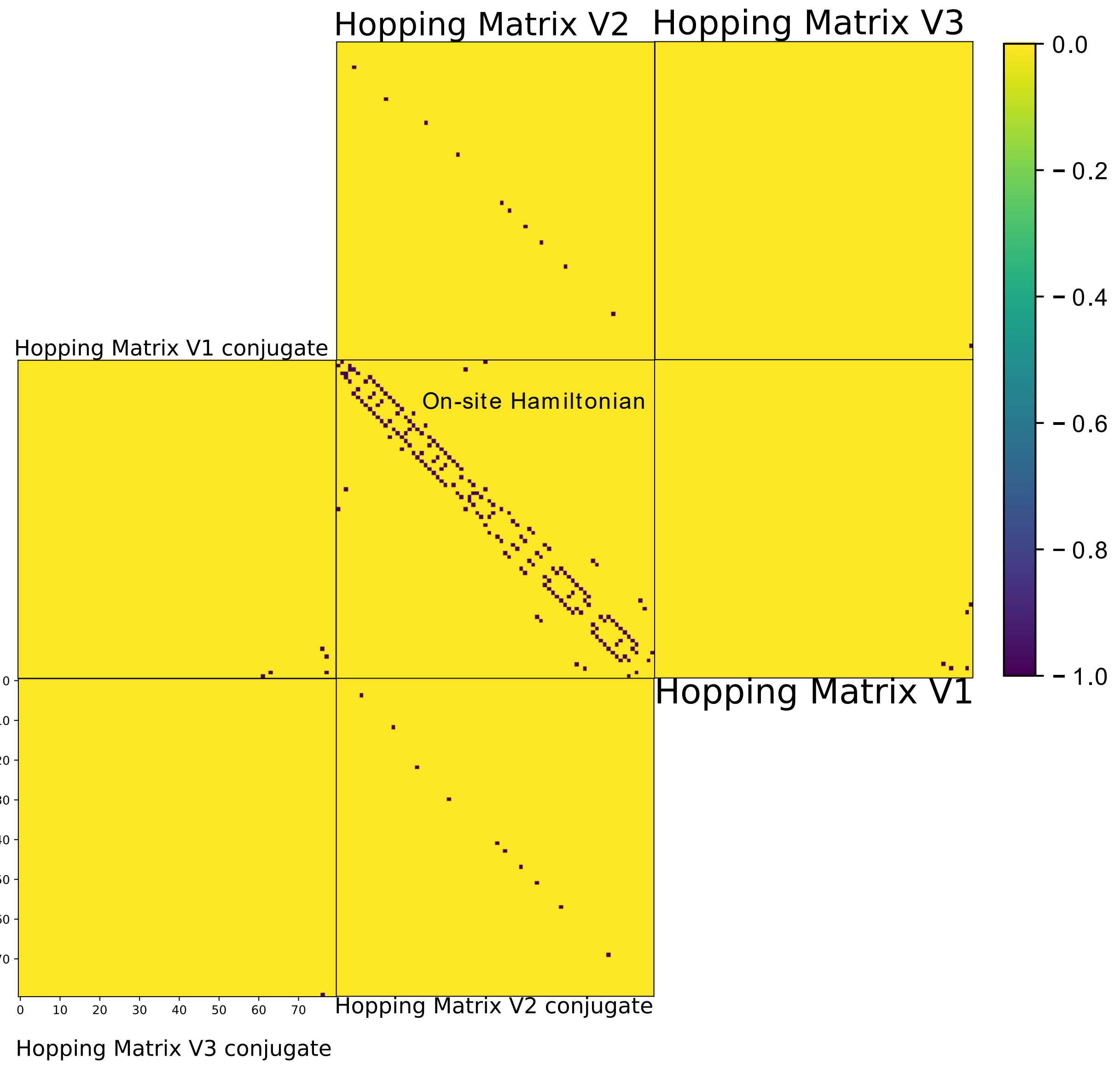


Figure 7: *Matrix maps from calculation on the NPG. The on-site Hamiltonian along with all its hopping matrices are stitched together like the representative figure Fig. 6. All the dark spots represents a hopping of an electron to its nearest neighbour.*

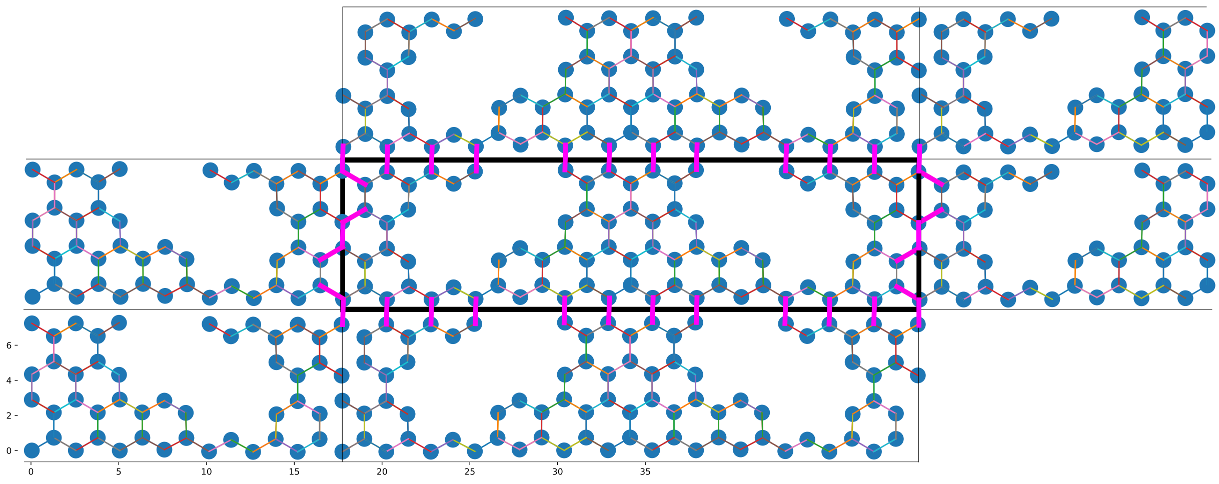


Figure 8: *Visual representation of the periodic NPG-structure. The atoms surrounded by the black box in the centre represents the unit cell. The neighbouring boxes are unit cells repeated periodically. Note that the two cells left and right with respect to the centre cell has been cut in half for figure space. The pink lines crossing the black box represents the link between the nearest neighbours in the adjacent cell. Note how this representation corresponds to that of Fig. 7. There is more 'hopping' when going up and down, than left or right and the two cells at the diagonal only have one atom which is nearest neighbour to the centre cell which corresponds to the one hopping elements in Fig. 7.*

B. Defining the full Hamiltonian and solving the Schrodinger equation

Now that the on-site Hamiltonian along with its hopping matrices has been defined, the next step is to create the full Hamiltonian in order to solve the Schrodinger equation for system, which is a eigen-value/vector problem. In essence the full Hamiltonian denoted **H** is a sum of the on-site Hamiltonian and its corresponding hopping matrices multiplied by a complex exponential function that has the appropriate phase relative to the hopping matrix:

$$\begin{aligned}\mathbf{H}(k_x,k_y) =&\mathbf{h}_0 + (\mathbf{V}_{1x}e^{-ik_x} + \mathbf{V}_{1x}^\dagger e^{ik_x} + \mathbf{V}_{2x}e^{-ik_y} + \mathbf{V}_{2x}^\dagger e^{-ik_y} \\ &+ \mathbf{V}_{3xy}e^{-ik_x}e^{-ik_y} + \mathbf{V}_{3xy}^\dagger e^{ik_x}e^{ik_y})\end{aligned} \tag{B.1}$$

Here k represents a continuous variable between 0 and π along the x- and y-axis in inverse space. Using the full Hamiltonian, the Schrodinger equation can be solved

$$\mathbf{H}(k_x,k_y)\boldsymbol{\phi}_k = \boldsymbol{\epsilon}_n(k_x,k_y)\boldsymbol{\phi}_k \tag{B.2}$$

In practice this is all done by defining a function that takes the on-site Hamiltonian, the hopping matrices, and k (x, y) as inputs and outputs the eigenvalues, using numpy's *numpy.linalg.eigh*. The number of eigenvalues in the output corresponds to the dimension of the full Hamiltonian. In Listing 2 the code for the function is shown.

```python
def Hkay(Ham, V1, V2, V3, x, y):
    Ham = Ham + (V1 * np.exp(-1.0j * x)
                 + np.transpose(V1) * np.exp(1.0j * x)
                 + V2 * np.exp(-1.0j * y)
                 + np.transpose(V2) * np.exp(1.0j * y)
                 + V3 * np.exp(-1.0j * x) * np.exp(-1.0j * y)
                 + np.transpose(V3) * np.exp(1.0j * x) * np.exp(1.0j * y))
    e = LA.eigh(Ham)[0]
    v = LA.eigh(Ham)[1]
    return e, v
```

Listing 2: *Function producing the full hamiltonian, corresponding to* **??**

As the eigenvalues corresponds to the eigenenergies of the system, the band structures can be plotted.

C. Plotting the band structures

The continuous variable k extends in two directions $(-k_x)$ and (k_y) which corresponds to lengths between the symmetry points X, Γ and Z in the Brillouin zone, with Γ being the zero-point. It is therefore necessary to make two plots, one for each pair of symmetry points. The y-values in each plot corresponds to the eigenvalues obtained by the function described in Section III B. Each x-value between Γ, X and Γ, Z therefore has associated with it the amount of y-values which the dimension of full Hamiltonian dictates. F.ex. if the full Hamiltonian has dimension 80×80 there is 80 eigenvalues associated with it and the amount of y-values for each x-value between Γ, X and Γ, Z is 80. In this specific case, the full Hamiltonian for obtaining the eigenvalues that corresponds to X and Z are:

$$X: \ \mathbf{H}_X = \mathbf{h}_0 + (\mathbf{V}_{1x}e^{ik_x} + \mathbf{V}_{1x}^\dagger e^{-ik_x} + \mathbf{V}_{2x} + \mathbf{V}_{2x}^\dagger + \mathbf{V}_{3xy}e^{ik_x} + \mathbf{V}_{3xy}^\dagger e^{-ik_x}) \qquad (C.1)$$

$$Z: \ \mathbf{H}_Z = \mathbf{h}_0 + (\mathbf{V}_{1x} + \mathbf{V}_{1x}^\dagger + \mathbf{V}_{2x}e^{-ik_y} + \mathbf{V}_{2x}^\dagger e^{ik_y} + \mathbf{V}_{3xy}e^{-ik_y} + \mathbf{V}_{3xy}^\dagger e^{ik_y}) \qquad (C.2)$$

Using the eigenvalues as y-values in the two plots, putting the two plots together will yield a full plot of the band structure shown in Fig. 9.

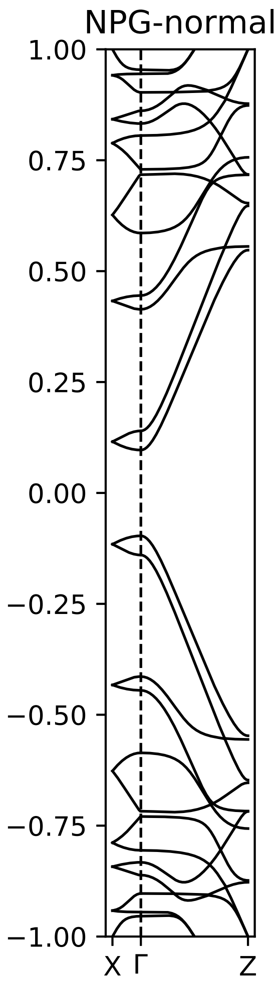
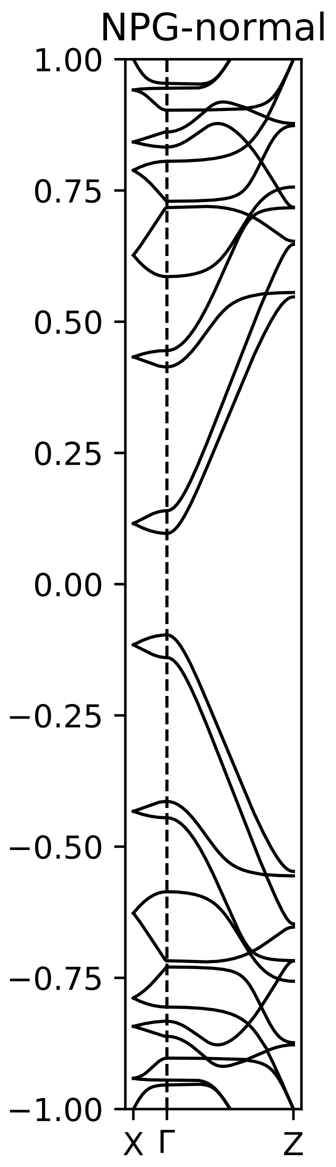


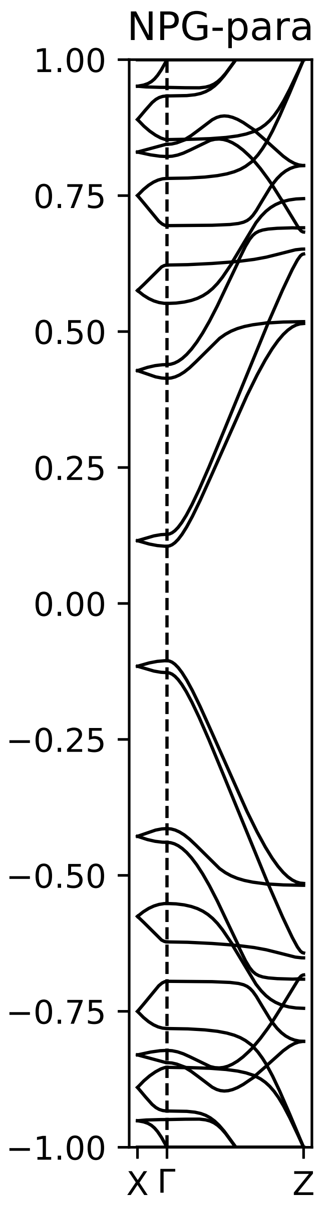
Figure 9: *Plot showing the band structure in the energy range -1 to 1 for NPG with normal bridges between symmetry points X and Z with respect to* Γ

IV. SELF ENERGY, GREENS FUNCTIONS AND THE RECURSION ROUTINE

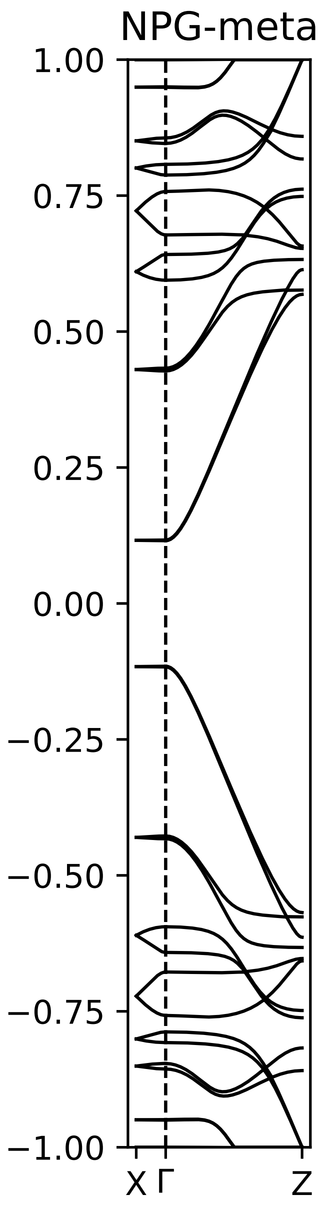
In order to calculate the transport of electrons in NPG one must obtain the Green's functions and self energies related to the unit cells of the system. In order to do so, a recursion algorithm must be implemented as calculations become too costly for the system, even if it is small. Especially the inversion of matrices required to obtain the Green's function can be very demanding computationally, when the system contains a lot of atoms. The recursion algorithm reduces the size of the system and thereby the amount



(a) *Plot showing the band structure in the energy range -1 to 1 for NPG with normal bridges between symmetry points X and Z with respect to* Γ



(b) *Plot showing the band structure in the energy range -1 to 1 for NPG with para bridges between symmetry points X and Z with respect to* Γ



(c) *Plot showing the band structure in the energy range -1 to 1 for NPG with meta bridges between symmetry points X and Z with respect to* Γ

Figure 10: *Figure showing para, meta and normal NPG band structures*

of computational time required to obtain both the first cell Green's function, the Green's function within the chain (of repeated unit cells) sometimes called $\mathbf{G}_{bulk}$ as well as the self energies related to those Green's functions. The recursion works by utilising that one can remove every second cell in an infinite chain of cells. As the chain originally was infinite, removing every second cell will just yield a new infinite chain. Every cell has with it, its hopping matrices and hamiltonian. The removal of every second cell is iterated, changing the effective interaction between the cells and thus the hopping matrices as well as the hamiltonians of the system. In the end the recursion algorithm produces re-normalised hamiltonians and hopping matrices, which can be used to obtain the Green's functions and self energies.

A. Obtaining first cell self energy and Green's function

For simplicity and in order to check whether the routine would yield the results expected, the starting point is a simple system containing only 4 atoms in the unit cell. The idea is to make a function that takes in an energy, a hamiltonian (alternatively two in case there is a special site in the molecule) and a hopping matrix as arguments and outputs the Green's function at the zeroth site as well as the self energies going left and right. Firstly all the necessary variables are defined. The first of these consists of a complex number matrix which has the value $E + i\eta$ in the diagonal and dimension identical to that of the hamiltonian given as argument. Here E is the energy which the function takes as an argument and η is a small number (in specific case 1×10^{-6}). Note that η should not be made too small as it will yield incorrect results. The rest of the variables are the hamiltonian, hopping matrices and Green's functions. They are defined as: $a0 = V^{\dagger},\ b0 = V,\ e0_s = h,\ e_0 = h,\ g0 = (z - e0)^{-1}$. Note that the hamiltonian h here is the same for both $e0_s$ and $e0$ as the zeroth cell is identical to those of the rest of the molecule. Now that the variables is defined, the actual recursion can begin. As the recursion is an algorithm that runs for an arbitrary amount of iterations, a while loop is chosen to run the iterations until a threshold has been reached. In **??** the code for the routine is shown. Line 64-70 is the listing of variables, 71-78 is the while loop with the matrix multiplication, 80-84 is the redefinition of new variables in terms of the old and 86-89 is the definition of the outputs as per **??**.

The threshold is defined as the absolute maximum value in the hopping matrix $a0$

```python
def RecursionRoutine(En, h, V):
    z = np.identity(h.shape[0]) * (En - 1e-6j)
    a0 = np.transpose(V)
    b0 = V
    es0 = h
    e0 = h
    g0 = LA.inv(z - e0)
    q = 1
    while np.max(np.abs(a0)) > 0.000000001:
        ag = a0 @ g0
        a1 = ag @ a0
        bg = b0 @ g0
        b1 = bg @ b0
        e1 = e0 + ag @ b0 + bg @ a0
        es1 = es0 + ag @ b0
        g1 = LA.inv(z - e1)
```

approaches zero. In this specific case it was set to 1×10^{-8}. Within this while loop a range matrix multiplications are computed. These constitutes the actual recursion and their order are of big importance for a correct result. Following is a list of these matrix

```python
        a0 = a1
        b0 = b1
        e0 = e1
        es0 = es1
        g0 = g1
        q = q + 1
    e, es = e0, es0
    SelfER = es - h
    SelfEL = e - h - SelfER
    G00 = LA.inv(z - es)
    # print(q)
    return G00, SelfER, SelfEL
```

multiplications:

$$
\begin{aligned}
a1 &= a0\ g0\ a0 \\
b1 &= b0\ g0\ b0 \\
e1 &= e0 + a0\ g0\ b0 + b0\ g0\ a0 \\
e1_s &= e0_s + a0\ g0\ b0 \\
g1 &= (z - e1)^{-1}
\end{aligned}
$$

As stated above in the equations, the matrix product are defined as new variables with a +1 variable name. This is all very well but the while loop would stop here because of the definition of new variables. It is therefore necessary to redefine the new variables in terms of the old ones (f.ex. $a0 = a1$) an as such the while loop will continue until the threshold has been reached. For the rest of the function the only thing left to do is to define the self energies and the first cell Green's function using the variables obtained from the while loop. These are simply given as:

$$
\begin{aligned}
\mathbf{\Sigma}_R &= e_s - h \\
\mathbf{\Sigma}_L &= e - h - \mathbf{\Sigma}_R \\
\mathbf{G00} &= (z - e_s)^{-1}
\end{aligned}
$$

Where e_s and e are the resulting matrices from the while loop. The first cell Green's function as well as the self energies has now been obtained by recursion.

B. Plotting the real and imaginary part of the first cell Green's function

As a good way to check whether the recursion routine has been implemented successfully, the real and imaginary part of the Green's function can be plotted. With a relatively simple approach, the Green's function can be obtained as a function of energy, using a 'for loop', looping over a range of energies which is then used as input in the function **??** (see Listing 3):

```
G00 = np.zeros((En.shape[0]), dtype=complex)
for i in range(En.shape[0]):
    G, SelfER, SelfEL = RecursionRoutine(En[i], h, V)
    G = np.diag(G)
    G00[i] = G[0]
```

Listing 3: *Code showing the loop which produces the Green's function (or y) values for a range of energies used in the plot.*

The output from this loop will be the real and imaginary y-values for the plot. These will have to be sorted to imaginary and real parts before plotting. The x-values will just be the energy range used for the loop.The imaginary part of the Green's function is especially important as it represents the local density of states LDOS. Following is the plot of the real and imaginary part of the first cell Green's function obtained by the recursion routine for the simple four atom unit cell described in Section IV A. In Fig. 11 a plot of the imaginary and real part of the resulting Green's function can be seen.

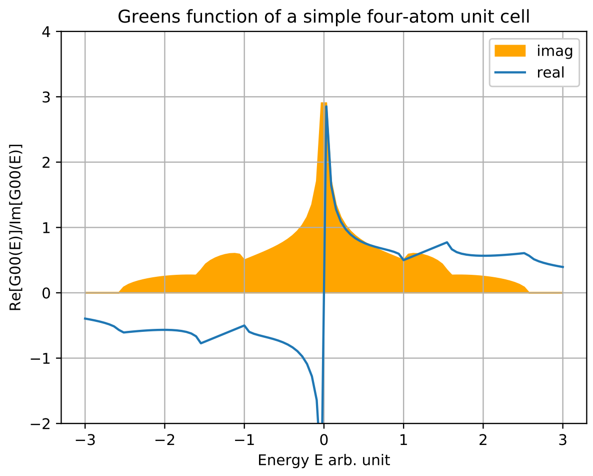


Figure 11: *A plot showing the real and imaginary part of the first cell Green's function resulting from the recursion routine on the simple system. Note that the yellow imaginary part is the representation of the density of states.*

ACKNOWLEDGMENTS

The authors would like to thank...

[1] G. Calogero, N. R. Papior, B. Kretz, A. Garcia-Lekue, T. Frederiksen, and M. Brandbyge, Electron Transport in Nanoporous Graphene: Probing the Talbot Effect, Nano Letters **19**, 576 (2019).

LIST OF FIGURES

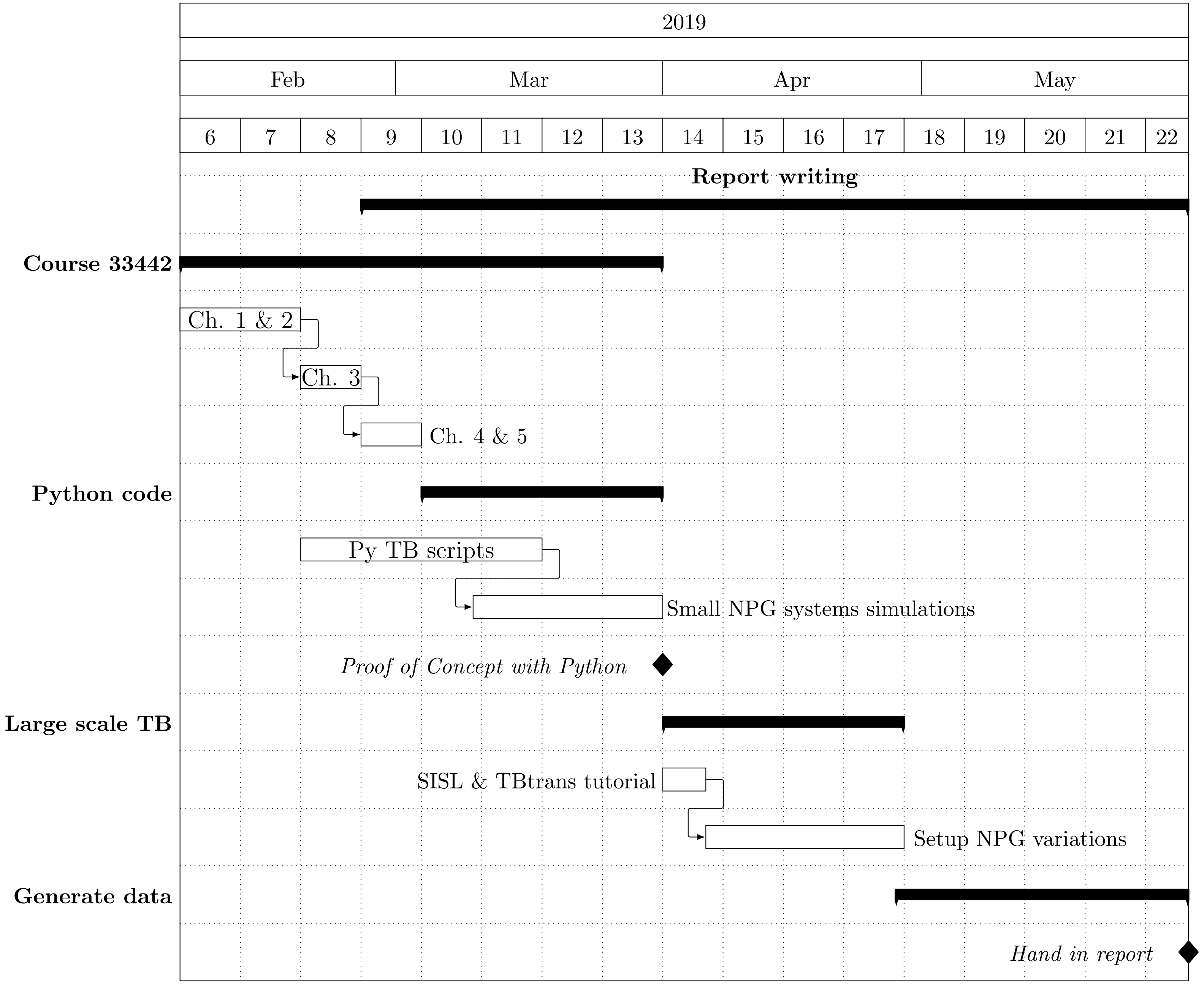
LIST OF TABLES

LISTINGS

Appendices

Appendix A: Project overview

A Gantt chart is provided on the next page. **Not Updated.**


2019
Feb
Mar
Apr
May
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
Report writing
Course 33442
Ch. 1 & 2
Ch. 3
Ch. 4 & 5
Python code
Py TB scripts
Small NPG systems simulations
Proof of Concept with Python
Large scale TB
SISL & TBtrans tutorial
Setup NPG variations
Generate data
Hand in report