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. TIGHT-BINDING, SELF-ENERGY, GREEN'S FUNCTIONS AND TRANSMISSION

A. Quantum transport

1. 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. Therefore it is common to use what is called a ballistic model when working with graphene. Because of the strong binding between atoms in graphene hardly any phonons are present in the material, even at room temperature. Using the ballistic model, this means that electrons are not affected by phonons, i.e. no electrons will be excited by phonons, and that the electrons 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.

2. 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 approximation and Green's functions. The two concepts of Tight Binding approximation and Green's functions will elaborated further in the coming sections. 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.

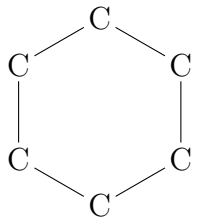


Figure 1: *Graphene lattices consists of hexagonal arrangements of carbon atoms.*

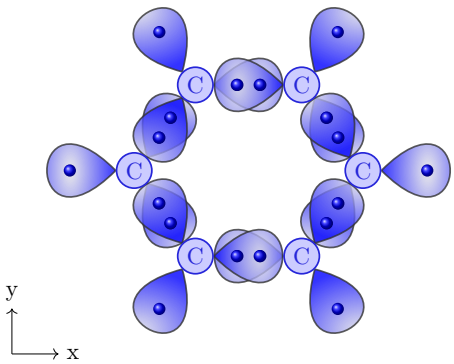


Figure 2: *Carbon atoms in a hexagonal lattice are sp^2 hybridised in the (x, y)-plane.*

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.

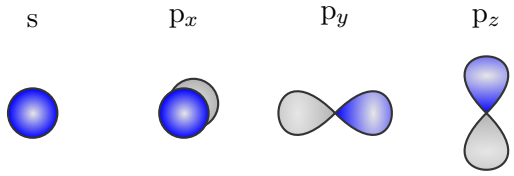


Figure 3: *The valence orbitals of carbon.*

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. For a benzene ring the π-electrons at the highest occupied molecular state will travel through the p_π-orbitals switching sign as they travel as shown in Fig. 4.

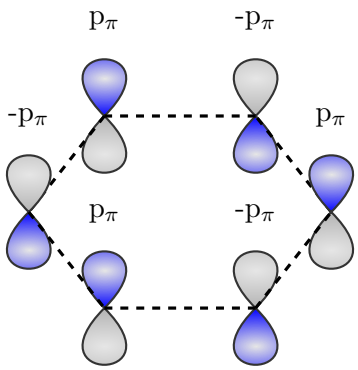


Figure 4: *When jumping from one carbon atom to another, the π-electron goes between p_π-orbitals. Such a jump is described by two matrix elements in the system's Hamiltonian.*

3. 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:

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

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:

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

Physically this means that there is a potential between the π orbitals of neighbouring atoms 1 and 2. In our tight-binding approximation we consider only hop between nearest neighbours. The element

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

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

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

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

4. 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. (A.2), the

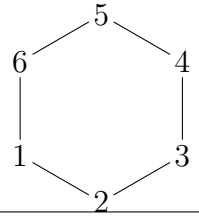


Figure 5: *Indices of a benzene molecule*

Hamiltonian reads:

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

As a helping aid, Eq. (A.5) 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:

$$\phi = \begin{pmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{pmatrix} \tag{A.6}$$

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

$$C_6\phi = \begin{pmatrix} c_2 & c_3 & c_4 & c_5 & c_6 & c_1 \end{pmatrix} \tag{A.7}$$

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 $\mathbf{H}$.

$$C_6\phi = \begin{pmatrix} c_2 & c_3 & c_4 & c_5 & c_6 & c_1 \end{pmatrix} = \lambda \begin{pmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{pmatrix} \tag{A.8}$$

This operator C_6 is represented with the matrix:

$$\mathbf{C}_6 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{A.9}$$

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

$$\phi_n = \frac{1}{\sqrt{6}} \begin{pmatrix} \lambda_n^0 & \lambda_n^1 & \lambda_n^2 & \lambda_n^3 & \lambda_n^4 & \lambda_n^5 \end{pmatrix}, \quad \lambda_n = \exp\{-i2\pi n/6\}, \quad n = 0, 1, 2, 3, 4, 5 \tag{A.10}$$

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

$$\varepsilon_n = \lambda_n + \lambda_{n-1} = 2\cos n\pi/3 \tag{A.11}$$

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

B. Hamiltonian for periodic systems

1. *Creating on-site Hamiltonian and hopping matrices for graphene systems*

In the previous section the Hamiltonian of an isolated benzene ring was defined. Such a Hamiltonian is fairly simple as the electrons could only hop within the system. However when modelling repeated periodic structures, one has to consider not only the hops within the unit cell, but also the hops in and out of said cell. 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 to the 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 Approximation, 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 matrix $\mathbf{V}$ and its conjugate $\mathbf{V}^\dagger$. This is because the Tight Binding Approximation constraints the transport for hopping of electrons between nearest neighbours which in this case is defined by the periodic boundary conditions set for the system.

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. A function called *Onsite* have been developed to make such an outer product and thus the on-site Hamiltonian. In Listing 1 the function can be seen.

```
31
32 def Onsite(xyz, Vppi):
33     h = np.zeros((xyz.shape[0], xyz.shape[0]))
34     for i in range(xyz.shape[0]):
```

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

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 a given unit cell. 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 $V_{pp\pi}$ 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:

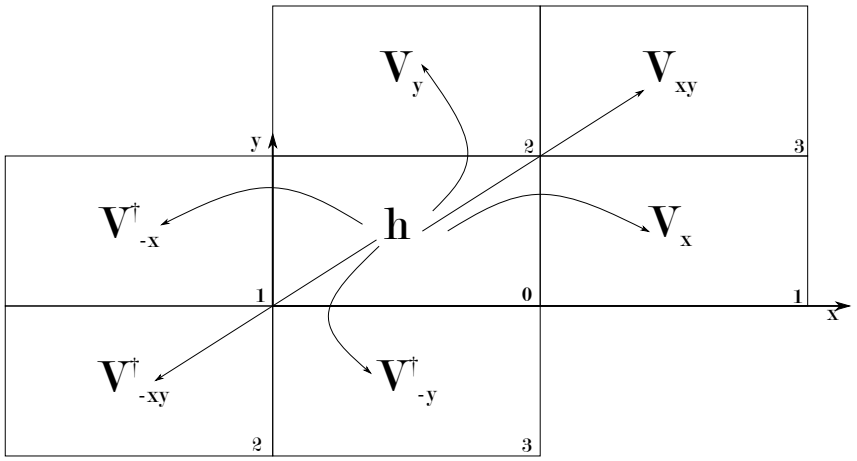


Figure 6: *Representative figure of how the on-site Hamiltonian along with its hopping matrices are structured*

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: $\mathbf{V}_{1x}^{\dagger}$, $\mathbf{V}_{2y}^{\dagger}$ and $\mathbf{V}_{3xy}^{\dagger}$. Fig. 7 is a figure of the resulting matrix-maps from calculation on a small simple system, stitched together like in Fig. 6. To see how the function running this calculation has been programmed look in Appendix A, Listing 7.

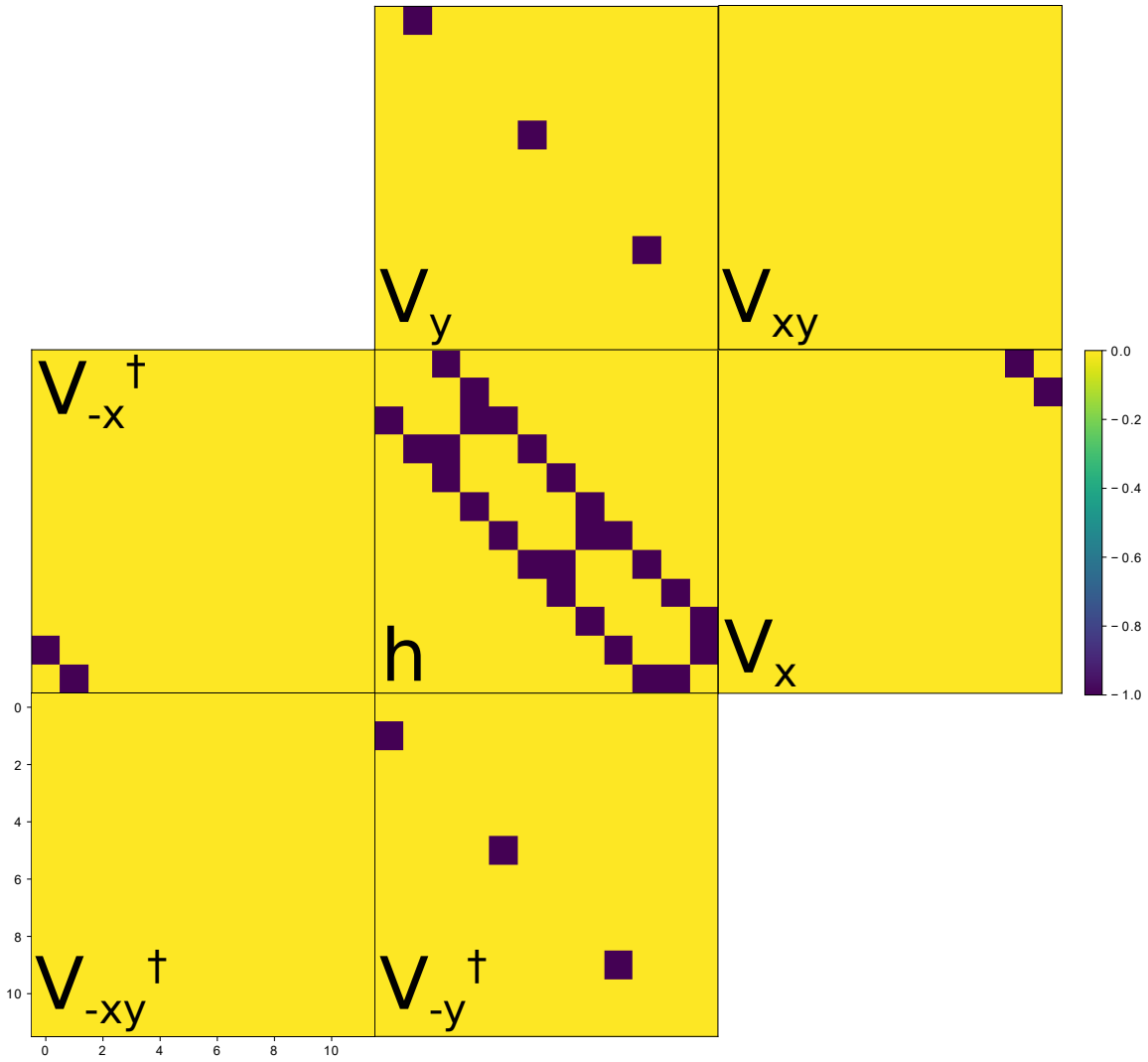


Figure 7: *Matrix maps from calculation on a small, simple unit cell. The on-site Hamiltonian along with all its hopping matrices are stitched together like the representative figure Fig. 6. All the dark spots represents a hopping of an electron to its nearest neighbour.*

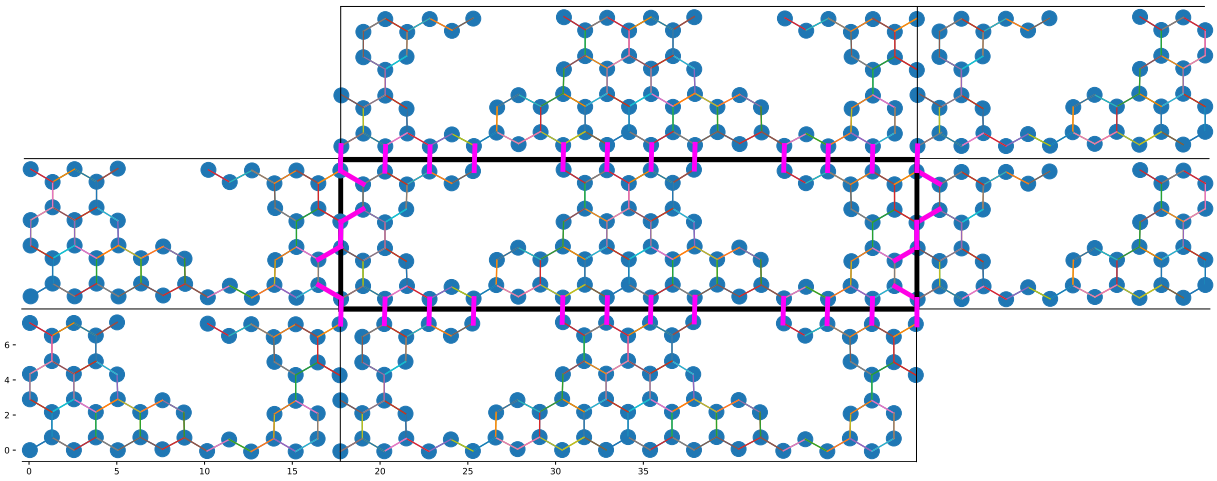


Figure 8: *Visual representation of the periodic NPG-structure. The atoms surrounded by the black box in the centre represents the unit cell. The neighbouring boxes are unit cells repeated periodically. Note that the two cells left and right with respect to the centre cell has been cut in half for figure space. The pink lines crossing the black box represents the link between the nearest neighbours in the adjacent cell.*

2. Defining the full Hamiltonian and solving the Schrödinger 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 Schrödinger 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:

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

Here k represents a continuous variable between 0 and π along the $\frac{1}{x}$- and $\frac{1}{y}$-axis. 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 here called *Hkay* 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.

```
50
51 def Hkay(Ham, V1, V2, V3, x, y):
52     Ham = Ham + (V1 * np.exp(-1.0j * x)
53                  + np.transpose(V1) * np.exp(1.0j * x)
54                  + V2 * np.exp(-1.0j * y)
55                  + np.transpose(V2) * np.exp(1.0j * y)
56                  + V3 * np.exp(-1.0j * x) * np.exp(-1.0j * y)
57                  + np.transpose(V3) * np.exp(1.0j * x) * np.exp(1.0j * y))
58     e = LA.eigh(Ham)[0]
59     v = LA.eigh(Ham)[1]
```

Listing 2: *Function producing the full hamiltonian, corresponding to Eq.* (B.1).

3. Plotting the band structures

An important intrinsic information in any solid state matter is of course the band structure as it gives information about allowed electron energies at different points in the material. Specifically for NPG, the band structure will be used to track the changes of electronic transport as the bridges in the NPG are changed. 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 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 *Hkay* function described in Section II B 2. 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:

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

$$Z: \ \mathbf{H}_Z = \mathbf{h}_0 + (\mathbf{V}_{1x} + \mathbf{V}_{1x}^{\dagger} + \mathbf{V}_{2x}e^{-ik_y} + \mathbf{V}_{2x}^{\dagger}e^{ik_y} + \mathbf{V}_{3xy}e^{-ik_y} + \mathbf{V}_{3xy}^{\dagger}e^{ik_y}) \tag{B.4}$$

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.

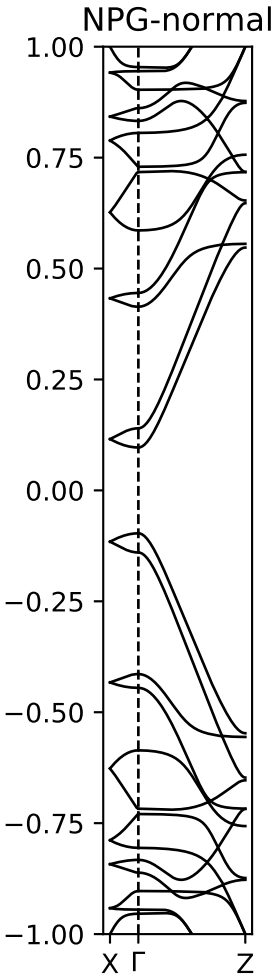


Figure 9: *Plot showing the band structure in the energy range -1 to 1 for NPG with normal bridges between symmetry points X and Z with respect to* Γ

C. Self Energy, Greens Functions and the Recursion Routine

In order to calculate the transport of electrons in NPG as well as the local density of states, LDOS, one must obtain the Green's functions and self-energies related to the unit cells of the system. *An attemp to briefly explain greens functions and self-energy. Might need*

rework as well In very basic terms the Green's function is used as a solution to the Time Dependent Schrödinger Equation. However if the Hamiltonian is time *independent* one can use the time-inde To get the self-energies and Green's functions, practically, 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 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. By using a semi-infinite tight-binding chain of unit cells, the recursion works by utilising that one can remove every second cell in the chain. 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.

1. *Obtaining first cell self-energy and Green's function for a simple four-atom system*

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 (See Appendix A, Fig. 16). 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 in turn outputs the Green's function 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 (in the case of the four-atom system a 4x4). Here E is the energy which the function takes as an argument as well as η which is a small number (in this specific case 1×10^{-6}). Note that η should not be made too small as it will yield incorrect results. Generally one should adjust η until a satisfying result have been obtained. The rest of the variables are the Hamiltonian, hopping matrices and Green's function. They are defined as: $a0 = V^{\dagger},\ b0 = V,\ e0_s = h_s,\ e_0 = h,\ g0 = (z - e0)^{-1}$. Note that the Hamiltonian h, in this

specific case, is the same for both $e0_s$ and $e0$ as the first cell is identical to those of the rest of the molecule. Now that the variables are defined, the actual recursion can begin. The function *RecursionRoutine* has been developed. 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 Listing 3 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 Eq. (C.1).

```
62
63  def RecursionRoutine(En, h, V, eta):
64      z = np.identity(h.shape[0]) * (En - eta)
65      a0 = V.conj().transpose()
66      b0 = V
67      es0 = h
68      e0 = h
69      g0 = LA.inv(z - e0)
70      q = 1
71      while np.max(np.abs(a0)) > 1e-6:
72          ag = a0 @ g0
73          a1 = ag @ a0
74          bg = b0 @ g0
75          b1 = bg @ b0
76          e1 = e0 + ag @ b0 + bg @ a0
77          es1 = es0 + ag @ b0
78          g1 = LA.inv(z - e1)
79          a0 = a1
80          b0 = b1
81          e0 = e1
82          es0 = es1
83          g0 = g1
84          q = q + 1
85      e, es = e0, es0
86      SelfER = es - h
87      SelfEL = e - h - SelfER
88      G00 = LA.inv(z - es)
89      # print(q)
```

Listing 3: *The while loop in the recursion routine. The matrix elements are overwritten with the new variables until the resulting matrix is small enough to diagonalise*

The threshold is defined as the absolute maximum value element in the hopping matrix $a0$ approaches zero. In this specific case it was set to 1×10^{-8}. However, the value should be adjusted so that it is optimised for the specific system on which the calculation are made. 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. Notice, however, that the code (Listing 3) contains intermediate products of $a0, b0$ and $g0$. This is for run time optimisation. Following is a list of these matrix products:

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

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. $a0 = a1$) (see Listing 3 line 78-82) 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:

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

Where e_s and e are the resulting matrices from the while loop $(e0_s, e0)$. The first cell Green's function as well as the self-energies has now been obtained by recursion.

2. 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. In addition these plots gives valuable information about the local density of states at specific sites in the molecule. With a relatively simple approach, the Green's function can be obtained as a

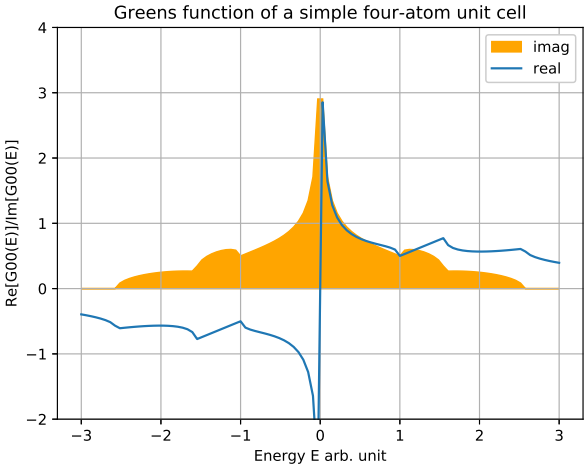


Figure 10: *A plot showing the real and imaginary part of the first cell Green's function resulting from the recursion routine on the simple system. Note that the yellow imaginary part is the representation of the local density of states.*

function of energy, using a *for loop*, looping over a range of energies which is then used as input in the *RecursionRoutine* function (Listing 3), see Listing 4:

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

Listing 4: *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 complex 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, it represents the local density of states LDOS. Note that the plot only represents the LDOS for a specific site on the molecule and that they change radically from site to site (see Appendix A, Fig. 14 for an example using the same structure as Fig. 12). The site can be changed by choosing another index in Listing 4 line 68, which corresponds to the atom indices in Fig. 16. A plot of the real and imaginary part of the first cell Green's function (index 0) obtained by the recursion routine for the simple four-atom unit cell described in Section II C 1 is shown in Fig. 10.

D. Transmission Routine

The conclusion of the preliminary work revolves around the Transmission Routine. Here all the functions producing on-site Hamiltonians, full Hamiltonians, hopping matrices, band structures, self energy and Green's functions by recursion play their part in getting the transmission through the material. *This part is an attempt to describe the basic theory behind transmission/transport and it is very likely to need rework, additions and editing* The transmission tells us the probability whether an electron will be transported trough all the possible π-orbitals in a "device region" at different energies and thus how the "device region" affects the overall current through larger systems. This means that the system will have to be well defined before one can use the functions developed for calculation of the different parts needed for production of transmission. The "device region" contains at least one central unit cell as well as a "left" and "right" unit cell. The left and right unit cell represents the contact region of the device i.e. the two parts that connects to the "rest" of the system/molecule. It is assumed that the "rest of the molecule" represent a system that is made up of unit cells which can be reduced by recursion, though not necessarily identical on each side (left and right). Knowing which "blocks" the system contains and how to obtain them with the already developed functions, three main ingredients are needed to obtain the transmission through a device. The first one is the left and right selfenergies for the device region $\mathbf{\Sigma}_{L/R}$. The device region of course includes the device Hamiltonian $\mathbf{H}_D$ and in addition, $\mathbf{H}_D$ contains a left and right Hamiltonian $\mathbf{H}_L$, $\mathbf{H}_R$ which has left and right hopping matrices $\mathbf{V}_L$, $\mathbf{V}_R$ (see Fig. 11 for a representation of all the before mentioned matrix parts). From the left/right Hamiltonian and hopping matrices one can obtain the let and right selfenergies $\mathbf{\Sigma}_L$, $\mathbf{\Sigma}_R$, on-site Green's functions $\mathbf{g}_L$, $\mathbf{g}_R$ and then again by recursion and the equations from Eq. (C.1), though slightly modified, one can obtain the second important ingredient which is the device Green's function G_D. The modified equations of those stated in Eq. (C.1) are shown below

$$\mathbf{\Sigma}_L = \mathbf{V}_L \mathbf{g}_L \mathbf{V}_L^\dagger, \;\; \mathbf{\Sigma}_R = \mathbf{V}_R \mathbf{g}_R \mathbf{V}_R^\dagger \tag{D.1}$$
$$\mathbf{G}_D = [\mathbf{1}(E + i\eta) - \mathbf{H}_D - \mathbf{\Sigma}_L(E) - \mathbf{\Sigma}_R(E)]^{-1}$$

The last main ingredient are $\mathbf{\Gamma}_L$, $\mathbf{\Gamma}_R$ which are matrix operators describing the coupling between the left and right parts of Hilbert space. They are called rate equations because..... The rate equation are obtained via the self energies in the following fashion:

$$\mathbf{\Gamma}_{L/R} = i(\mathbf{\Sigma}_{L/R} - \mathbf{\Sigma}_{L/R}^\dagger) \tag{D.2}$$

Now the rate matrices have another important function as they can be used to rewrite the equation for the *spectral function*:

$$\mathbf{A}(E) = -2\mathrm{Im}[\mathbf{G}(E)] \tag{D.3}$$

$$\downarrow \tag{D.4}$$

$$\mathbf{A}(E) = \mathbf{G}(E)\mathbf{\Gamma}(E)\mathbf{G}^{\dagger}(E) \tag{D.5}$$

To explain what the spectral function is, it is convenient to know that it too kan be divided into two parts, a left and a right one.

$$\mathbf{A}(E) = \mathbf{A}(E)_L + \mathbf{A}(E)_R \tag{D.6}$$

$$\mathbf{A}(E)_L = \mathbf{G}(E)\mathbf{\Gamma}(E)_L\mathbf{G}^{\dagger}(E) \tag{D.7}$$

$$\mathbf{A}(E)_R = \mathbf{G}(E)\mathbf{\Gamma}(E)_R\mathbf{G}^{\dagger}(E) \tag{D.8}$$

Taking the left spectral function as an example, it represents the density of states (note Eq. (D.3)) of a wave coming from the left entering the device. Now one can also write the equation

$$\mathbf{A}_L(E)\mathbf{\Gamma}_R(E) = \mathbf{G}(E)\mathbf{\Gamma}(E)_L\mathbf{G}^{\dagger}(E)\mathbf{\Gamma}_R(E) \tag{D.9}$$

This corresponds to the density of states coming from the left, which then passes on through the right electrode by the rate $\mathbf{\Gamma}_R(E)$ and because of time-reversal symmetry one can also get the density of states coming from the other direction.

$$\mathbf{A}(E)_L\mathbf{\Gamma}(E)_R = \mathbf{A}(E)_R\mathbf{\Gamma}(E)_L \tag{D.10}$$

This ultimately leads to transmission because transmission is in essence an expression of how much of the density of states passes trough the device and as explained Eq. (D.9) is exactly the density of states, coming from the left (or right) and then passes through the right (left) by rate $\mathbf{\Gamma}_{L/R}(E)$. So using the Green's function, left/right self energies as well as the left/right rate matrices, the transmission, as a function of energy, can be obtained via the following equation:

$$T(E) = \mathrm{Tr}[\mathbf{\Gamma}_R\mathbf{G}_D\mathbf{\Gamma}_L\mathbf{G}_D^{\dagger}](E) \tag{D.11}$$

Where Tr is the trace of the matrix product.

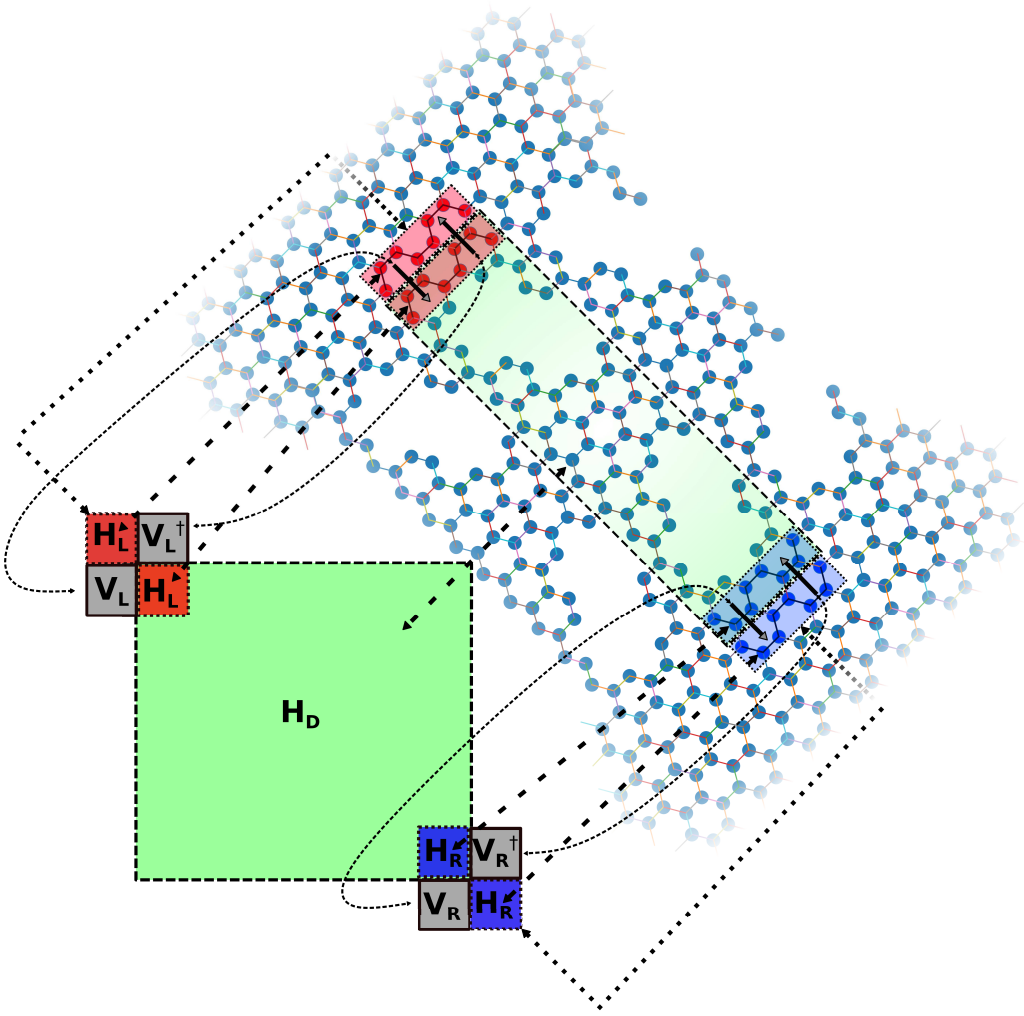
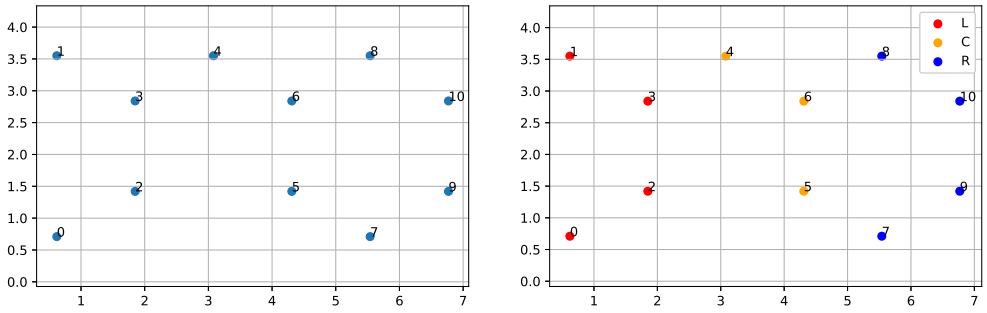


Figure 11: *Illustration showing how the different parts of the system are translated into matrix blocks. On the graphene the green box is the unit cell of the device. It includes one red and blue box which themselves are unit cells of the left and right contacts. These unit cells can be translated into Hamiltonian matrices* $\boldsymbol{H}_D$, $\boldsymbol{H}_L$, $\boldsymbol{H}_R$ *as illustrated. Note that because the first of the left and right unit cells (red and blue) are inside the device (green region), they can be picked out of* $\boldsymbol{H}_D$ *directly and so it is not necessary to make them from scratch. The two other unit cells lying outside the device region represents what could be an infinite contact region. The contact region is therefore potentially of an infinitely bigger dimension compared to the device region, however, by recursion, this region can be collapsed into a single Hamiltonian of same dimension as the one inside the device region. Finally the two fat black arrows (not dotted) on each side of the device represents the hopping between the device and contact region. Note that the direction of hopping corresponds to a specific hopping matrix. F.ex. left-to-right is the ordinary hopping matrix while right-to-left is its conjugate (for both left and right side of the device).*



(a) *The basic structure plotted with indices*

(b) *The same structure now shown with indices marked red and blue as per users choice of contact atoms*

Figure 12: *Figure showing the interface integrated into the script.*

1. Transmission in 1D a simple example

As with the Recursion Routine, the development of this routine will be build up around a small system in order to make sure that the obtained results are as expected, thereafter generalising the routine to suit all kinds and sizes of system. First thing is to define the device in the same manner as Fig. 11 so that the device Hamiltonian $\mathbf{H}_D$ can be obtained through the already defined function *Onsite*. The left and right Hamiltonian $\mathbf{H}_{L/R}$ are thus picked out as described in Fig. 11. A smart function has been implemented in order to allow the user to choose the left and right contact cells using indices given to each atom in the system of choice (See Fig. 12b for an example). This allows the user to get the dimensions needed to define the left and right Hamiltonians so they can be picked out of the device Hamiltonian. From the left and right Hamiltonian the corresponding hopping matrices are defined using the *Hop* function. Then the *EnergyRecursion* is used to obtain the Green's function for the device. This function is a more elaborate version of the earlier mentioned *RecursionRoutine*. It uses the old recursion routine to calculate the self energies for the left and right cells ($\mathbf{\Sigma}_{L/R}$) (see line 164-165 Listing 5) and then uses those to calculate the device Green's function $\mathbf{G}_D$ as well as the left and right rate matrices $\mathbf{\Gamma}_{L/R}$, using the equations Eq. (D.2), Eq. (D.1) (see Listing 5 line 176-179).

```
156     plt.scatter(Restxyz[:, 0], Restxyz[:, 1], c='k')
157     plt.legend()
158     plt.axis('equal')
```

```
159     for i in range(xyz[:, 0].shape[0]):
160         s = i
161         xy = (xyz[i, 0], xyz[i, 1])
162         plt.annotate(s, xy)
163     plt.grid(b=True, which='both', axis='both')
164     plt.show()
165     return RestL, L, R, C, RestR
166
167
168 def EnergyRecursion(HD, HL, HR, VL, VR, En, eta):
169     HD = scp.csr_matrix(HD)
170     HL = scp.csr_matrix(HL)
171     HR = scp.csr_matrix(HR)
172     VL = scp.csr_matrix(VL)
173     VR = scp.csr_matrix(VR)
174     start = time.time()
175     GD = {}
176     GammaL = {}
177     GammaR = {}
178     bar = Bar('Running Recursion          ', max=En.shape[0])
179     q = 0
180     for i in En:
181         gl, crap, SEL = RecursionRoutine(i, HL, VL, eta=eta)
182         gr, SER, crap = RecursionRoutine(i, HR, VR, eta=eta)
183         SS = SEL.shape[0]
184         Matrix = np.zeros((HD.shape), dtype=complex)
185         Matrix[0:SS, 0:SS] = SEL
```

Listing 5: *lalala*

The output of the energy recursion function is the two rate matrices (left and right) as well as the device Green's function and as per Eq. (D.11) the matrices needed for transmission have been obtained. As seen in Listing 6 the transmission function *Transmission* simply

carries out the matrix product and subsequent trace of the resulting matrix and outputs a range of transmission probabilities which is then plotted against an energy range. Do mind that this is still just 1D in the sense that the transmission only moves in one direction. A plot of the transmission for such a simple 1D system (the one in Fig. 12a) can be seen in Fig. 13.

```
188         SS = SER.shape[0]
189         Matrix = np.zeros((HD.shape), dtype=complex)
190         Matrix[-SS:, -SS:] = SER
191         SER = Matrix
192
193         SEL = scp.csr_matrix(SEL)
194         SER = scp.csr_matrix(SER)
195         GD["GD{:d}".format(q)] = scp.linalg.inv(
```

Listing 6: *lalala*

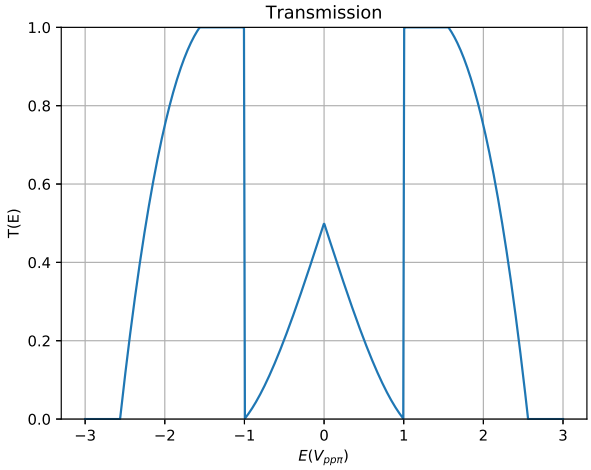


Figure 13: *Transmission plot for the system in Fig. 12*

2. Development of transmission to 2D

Lastly the transmission routine needs to be generalised so it can handle transport in two directions. The most convenient approach is to work with five real space unit cells as a starting point. One center cell, a right and a left cell representing the contacts and then

two additional cells on the left and right, representing the rest of the contact region. This would be the minimum amount of cells needed to generalise the transmission. One might have more center cells if the structure changes from one of those cells to the other. First these five unit cells will be defined using already developed tools. Then a big Hamiltonian, including all coordinates from the five cells representing real space are created, again using existing functions. One could call this big Hamiltonian $\mathbf{H}_{\text{Bigreal}}(xyz_0, xyz_0)$. It is a function of two sets of identical coordinates, namely the ones used to create the Hamiltonian itself. This has also been the case for all previous calculations, but the following steps will make it clear why it is explicitly stated for this Hamiltonian. The left/right on-site Hamiltonians and hopping matrices can thus be picked out of $\mathbf{H}_{\text{Bigreal}}(xyz_0, xyz_0)$ as before to get the Green's functions, self energies for transmission in real space, just as in the previous section. But instead, before the Green's function, self energies and transmission is calculated, the left/right onsite-Hamiltonian as well as their hopping matrices are defined as functions of a variable k and the hopping will additionally have a phase added which depends on a variable q. As an example the following is the equation for the full right side Hamiltonian using the right on-side Hamiltonian with its hopping matrices: $\mathbf{H}_R(k,q) = \mathbf{V}_R(k)e^{iq} + \mathbf{V}_R^\dagger(k)e^{-iq} + \mathbf{h}_R(k)$. The k represents .... and the q represents ..... The next part is to obtain the hop from $\mathbf{H}_{\text{Bigreal}}(xyz_0, xyz_0)$ to an equal cell in the other (transverse) direction, so that the transmission will become truly 2D. Firstly the hopping between $\mathbf{H}_{\text{Bigreal}}(xyz_0, xyz_0)$ and a transversely shifted Hamiltonian is defined as $\mathbf{W} = \mathbf{H}_{\text{Bigtrans}}(xyz_0, xyz_1)$ (Here using $\mathbf{W}$ as not to confuse it with $\mathbf{V}$ which is hopping between cells in real space). Note that the index of one of the coordinate sets have changed to 1. This corresponds to a shift by the lattice vector going in the transverse direction to that of the transmission direction defined in 1D. With the hopping matrices for the transverse direction defined a Hamiltonian dependent on k-point values can be defined as:

$$\mathbf{H}(k) = \mathbf{h} + \mathbf{W}e^{ik} + \mathbf{W}^\dagger e^{-ik} \tag{D.12}$$

$$= \mathbf{H}_{\text{Bigreal}}(xyz_0, xyz_0) + \mathbf{H}_{\text{Bigtrans}}(xyz_0, xyz_1)e^{ik} + \mathbf{H}^\dagger_{\text{Bigtrans}}(xyz_0, xyz_1)e^{-ik} \tag{D.13}$$

Now a Hamiltonian, dependent of a variable k has been defined, and thus it is now possible to get self energies, Green's functions that is k-dependent as well as a transmission (also k-dependent) which can be found for different k-points i.e. different points in the transverse direction (inverse space). This hereby concludes the all the initial effort to develop a code which can do calculations of different points of interest (Green's function plots, band structures and transmission) in a two dimensional material such as NPG. Following is a

walk through as to how this last step has been implemented through code programming.

III. PYTHON IMPLEMENTATION

IV. EXPLORING FUNCTIONALITY OF NANO-RIBBON BRIDGES

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).

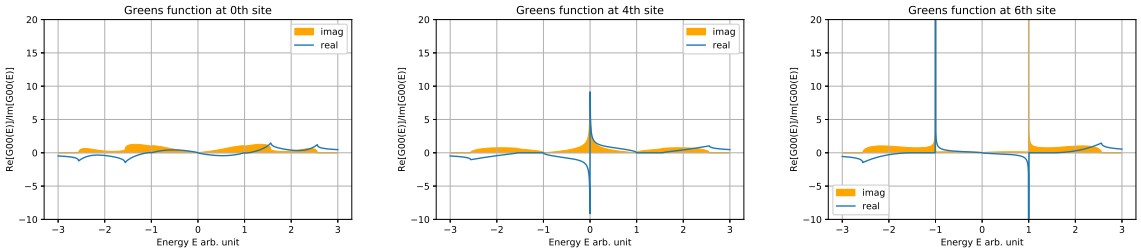
LIST OF FIGURES

LIST OF TABLES

LISTINGS

Appendices

Appendix A: Additional figures



(a) *Figure showing a plot of the Green's function at the 0th site*

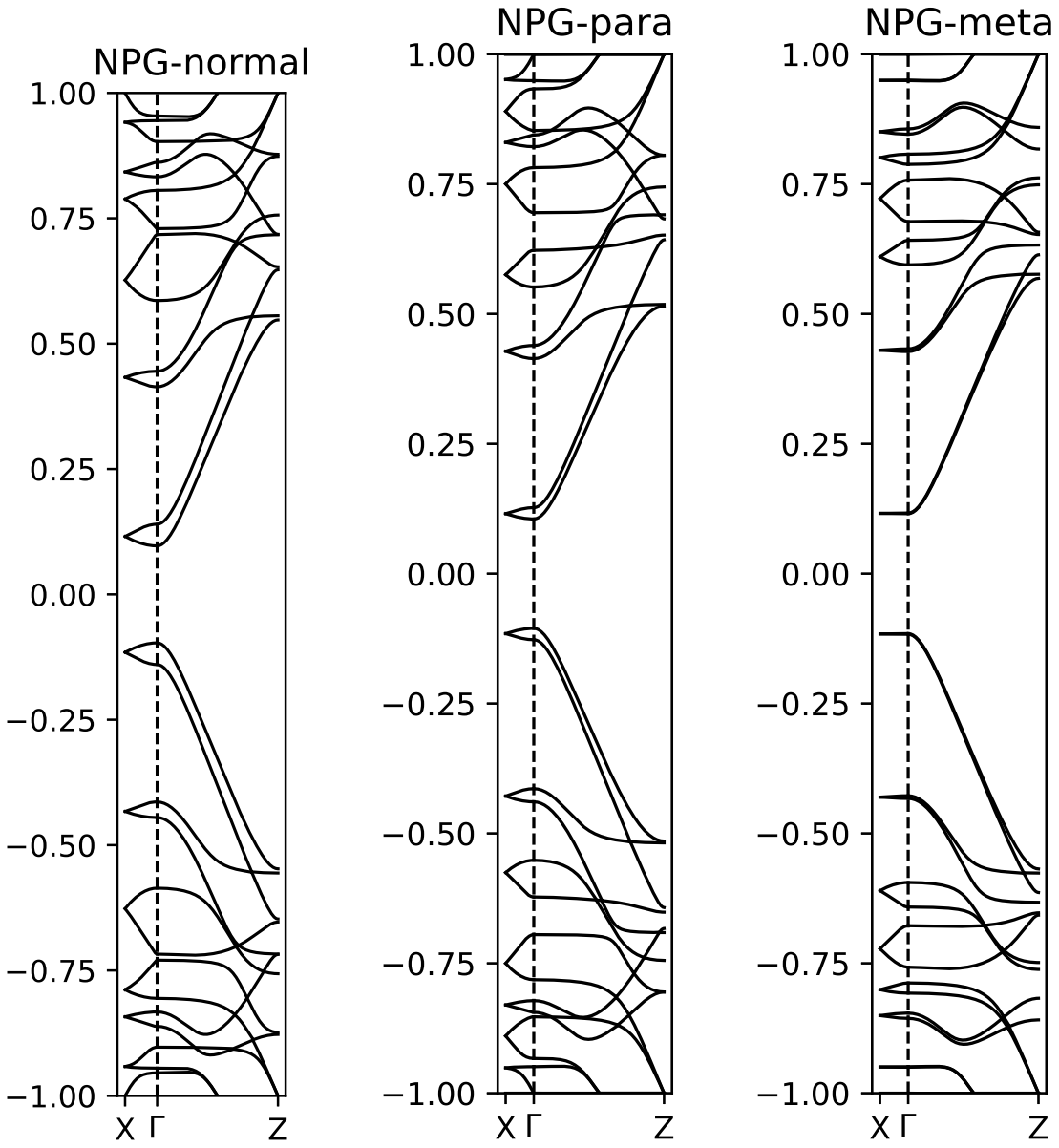
(b) *Figure showing a plot of the Green's function at the 4th site*

(c) *Figure showing a plot of the Green's function at the 6th site*

Figure 14: *Three plots showing how the Green's function changes as the site is changed. The 0th, 4th and 6th sites are corresponding to atoms of those indices (0, 4, 6) in Fig. 12. Note how the LDOS changes (imaginary part) for the different sites.*

```
41
42  def Hop(xyz, xyz1, Vppi):
43      hop = np.zeros((xyz1.shape[0], xyz.shape[0]))
44      for i in range(xyz1.shape[0]):
45          for j in range(xyz.shape[0]):
46              hop[i, j] = LA.norm(np.subtract(xyz1[i], xyz[j]))
47      hop = np.where(hop < 1.6, Vppi, 0)
```

Listing 7: *Function creating the hopping matrices between two sets of coordinates*



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

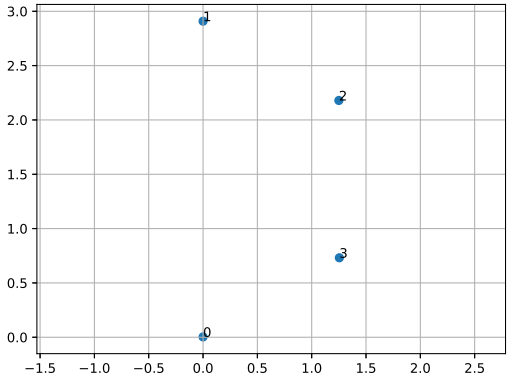


Figure 16: *Plot showing the simple four-atom system with indices, used for the first tests of the recursion function.*