# Quantum Transport in Nanoporous Graphene

## Bachelor defense

Christoffer Vendelbo Sørensen (s163965)

June 25, 2019

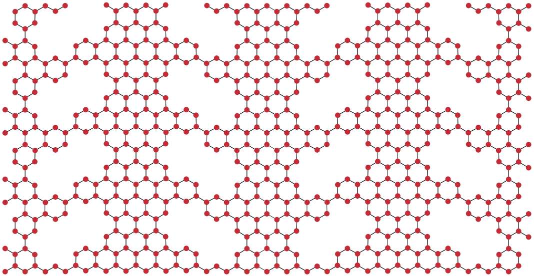
# Outline

# Project aim

"Development of tight-binding routines in Python in order to understand electron transport in novel nanoporous graphene devices (NPGs)"

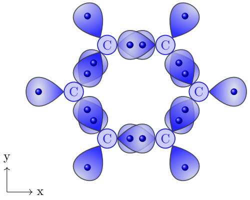
# Nanoporous graphene

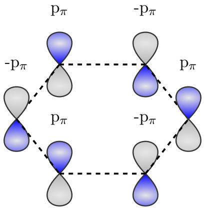
- Planar graphene sheets
- Periodically removed atoms
- Ribbons and bridges
- Ballistic electron movement
- Potential for controlling currents on nanoscale



# $\pi$-orbitals and $\pi$-electrons

- In plane electrons are bound
- 1 $p_z$-electron per site
- “Tightly bound” hops between sites



# TB approximation

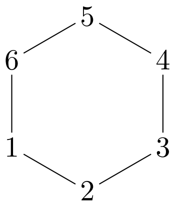
- Electrons tightly bound to sites
- Hops with potential
- Average electron energy on site
- The Hamiltonian is a hop matrix

$$V_{pp\pi} = \langle \phi_\pi(m) | \hat{\mathbf{H}} | \phi_\pi(n) \rangle$$

$$\epsilon_0 = \langle \phi_\pi(i) | \hat{\mathbf{H}} | \phi_\pi(i) \rangle$$

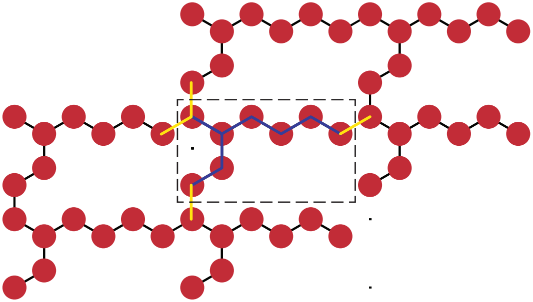
# Hamiltonian for benzene

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



# Creating the first Hamiltonian

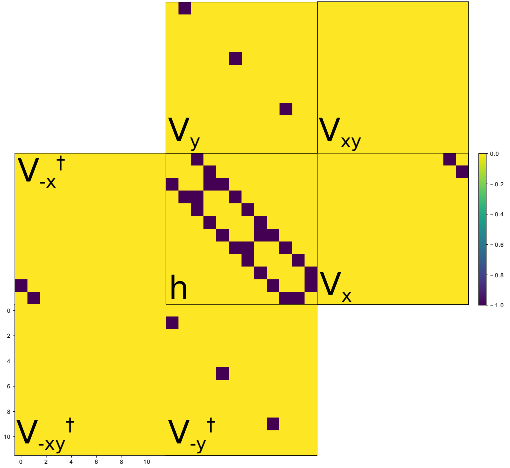
- Atom coordinates
- Interatomic distances
- Applying potential



```
h = np.zeros((xyz.shape[0], xyz.shape[0]))  # Empty matrix
for i in range(xyz.shape[0]):  # Take an atomic coordinate
    for j in range(xyz.shape[0]):  # Take another atomic coordinate
        h[i, j] = LA.norm(np.subtract(xyz[i], xyz[j]))  # Measure distances
h = np.where(h < 1.6, Vppi, 0)  # Replace distances under 1.6 angstrom with Vppi
h = np.subtract(h, Vppi * np.identity(xyz.shape[0]))  # Remove the diagonal
```

# Hopping matrices

- Shift by lattice vector
- Resulting matrices: $\mathbf{h_0}, \mathbf{V}, \mathbf{V}^\dagger$



# Full Hamiltonian and first band plots

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

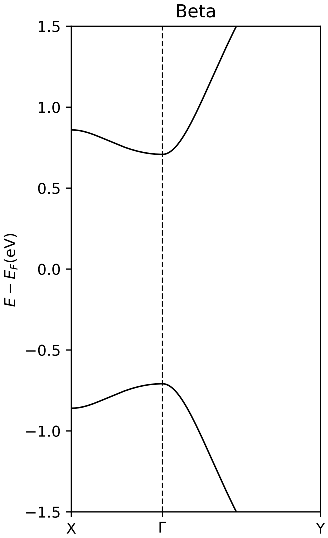
$$\mathbf{H}(k_x, k_y) = \mathbf{h}_0 + (\mathbf{V}_x e^{-ik_x} + \mathbf{V}_x^{\dagger} e^{ik_x} + \mathbf{V}_y e^{-ik_y} + \mathbf{V}_y^{\dagger} e^{ik_y} + \mathbf{V}_{xy} e^{-ik_x} e^{-ik_y} + \mathbf{V}_{xy}^{\dagger} e^{ik_x} e^{ik_y})$$

```python
    Ham = Ham + (V1 * np.exp(-1.0j * x)  # Onsite hops and hops in x
                + np.transpose(V1) * np.exp(1.0j * x)  # Hops in -x
                + V2 * np.exp(-1.0j * y)  # Hops in y
                + np.transpose(V2) * np.exp(1.0j * y)  # Hops in -y
                + V3 * np.exp(-1.0j * x) * np.exp(-1.0j * y)  # Hops in both x and y
                + np.transpose(V3) * np.exp(1.0j * x) * np.exp(1.0j * y))  # Hops in b
    e = LA.eigh(Ham)[0]  # Eigen energies from the Hamiltonian
    v = LA.eigh(Ham)[1]  # Eigen vectors from the Hamiltonian
```

# Full Hamiltonian and first band plots

$$\text{X:}\quad \mathbf{H}_X = \mathbf{h}_0 + (\mathbf{V}_x e^{ik_x} + \mathbf{V}_x^\dagger e^{-ik_x} + \mathbf{V}_y + \mathbf{V}_y^\dagger + \mathbf{V}_{xy} e^{ik_x} + \mathbf{V}_{xy}^\dagger e^{-ik_x})$$

$$\text{Y:}\quad \mathbf{H}_Y = \mathbf{h}_0 + (\mathbf{V}_x + \mathbf{V}_x^\dagger + \mathbf{V}_y e^{-ik_y} + \mathbf{V}_y^\dagger e^{ik_y} + \mathbf{V}_{xy} e^{-ik_y} + \mathbf{V}_{xy}^\dagger e^{ik_y})$$



# Green's matrix

- Solution to the Scödinger Equation
- Propagator

$$[(E + i\eta)\mathbf{1} - \mathbf{H}]\mathbf{G}(E) = \mathbf{1}$$

$$\downarrow$$

$$\mathbf{G}(E) = \mathbf{1}([(E + i\eta)\mathbf{1} - \mathbf{H}])^{-1}$$

# Recursion

- Semi-infinite chain

$$\begin{pmatrix} z\mathbf{1} - \mathbf{H}_c & -\mathbf{V}^\dagger \\ -\mathbf{V} & (z-\varepsilon')\mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{X} & \mathbf{G}_{0c} \\ \mathbf{G}_{c0} & \mathbf{G}_{00} \end{pmatrix} = \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$$

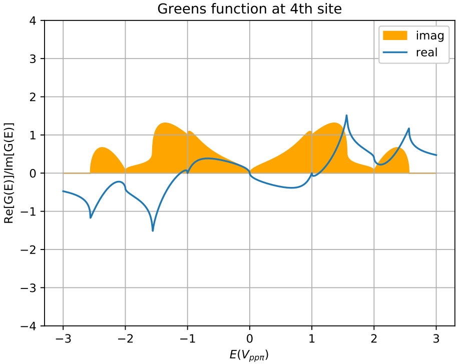
$$\begin{aligned} \mathbf{G}_{00}(z) &= \left[(z-\varepsilon') - \mathbf{V}(z\mathbf{1} - \mathbf{H}_c)\mathbf{V}^\dagger\right]^{-1} \\ &= (z - \varepsilon' - \Sigma(z))^{-1} \\ &\text{where} \\ z &= E + i\eta \end{aligned}$$

$$\begin{aligned} a_0 &= \mathbf{V}^\dagger, \quad b_0 = \mathbf{V} \\ e_{s0} &= \mathbf{h}_s, \quad e_0 \ = \mathbf{h} \\ &\text{in loop:} \\ a_1 &= a_0 \times g_0 \times a_0 \\ b_1 &= b_0 \times g_0 \times b_0 \\ e_1 &= e_0 + a_0 \times g_0 \times b_0 + b_0 \times g_0 \times a_0 \\ e_{1s} &= e_{0s} + a_0 \times g_0 \times b_0 \\ g_1 &= (z - e_1)^{-1} \\ \mathbf{\Sigma}_R &= e_s - h \\ \mathbf{\Sigma}_L &= e - h - \mathbf{\Sigma}_R \\ \mathbf{G00} &= (z - e_s)^{-1} \end{aligned}$$

# Recursion

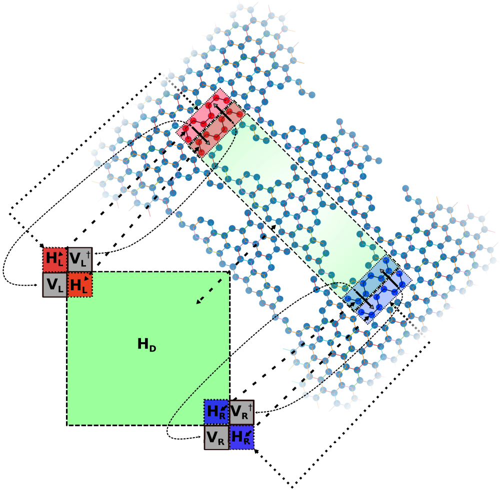
```python
    while np.max(np.abs(a0)) > 1e-6:  # Loop with hop matrix as threshold
        ag = a0 @ g0  # Product defined here and used multiple times
        a1 = ag @ a0  # New hop matrix (transposed)
        bg = b0 @ g0  # Product defined here and used multiple times
        b1 = bg @ b0  # New hop matrix
        e1 = e0 + ag @ b0 + bg @ a0  # New onsite for "other cells"
        es1 = es0 + ag @ b0  # New onsite
        g1 = LA.inv(z - e1)  # New Green's function
        a0 = a1  # Overwrite old variable
        b0 = b1  # Overwrite old variable
        e0 = e1  # Overwrite old variable
        es0 = es1  # Overwrite old variable
        g0 = g1  # Overwrite old variable
        q = q + 1  # Counter (for diagnostic purposes)
    e, es = e0, es0  # Define the onsite Hamiltonians
    SelfER = es - h  # Self-energy from the right
    SelfEL = e - h - SelfER  # Self-energy from the left
    G00 = LA.inv(z - es)  # Green's functions
```

# LDOS

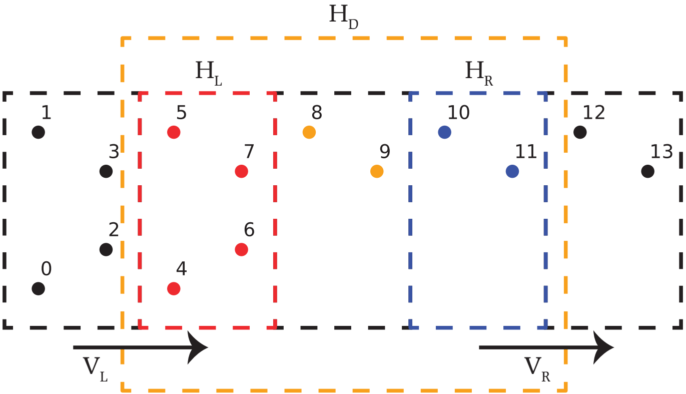



```python
G00 = np.zeros((En.shape[0]), dtype=complex)  # Empty data matrix for Green's functions
for i in range(En.shape[0]):  # Loop iterating over energies
    G, SelfER, SelfEL = RecursionRoutine(En[i], h, V, eta)  # Invoking the RecursionRou
    G = np.diag(G)  # The Green's functions for each site is in the diagonal of the G ma
    G00[i] = G[4]  # Chosen Green's function (here the 4th site)
```

Transmission is the probability of an electron being transported through a specific region for a specific range of energies.



# The left and right self-energy




```
        for i in En:   # Iteration over each energy
            gl, scrap, SEL = RecursionRoutine(i, HL, VL, eta=eta)   # From the left
            gr, SER, scrap = RecursionRoutine(i, HR, VR, eta=eta)   # From the right
```

# Getting Transmission

$$\mathbf{G}_D = \left[\mathbf{1}(E + i\eta) - \mathbf{H}_D - \mathbf{\Sigma}_L(E) - \mathbf{\Sigma}_R(E)\right]^{-1}$$

$$\mathbf{\Gamma}_{L,R} = i\left(\mathbf{Sigma}_{L,R} - \mathbf{\Sigma}^{\dagger}_{L,R}\right)$$

```
GD["GD{:d}".format(q)] = scp.linalg.inv(  # Device Green's functions are saved
    scp.identity(HD.shape[0]) * (i + eta) - HD - SEL - SER)  # in a dictionary.
GammaL["GammaL{:d}".format(q)] = 1j * (SEL - SEL.conj().transpose())  # Rate ma
GammaR["GammaR{:d}".format(q)] = 1j * (SER - SER.conj().transpose())  # likewis
```

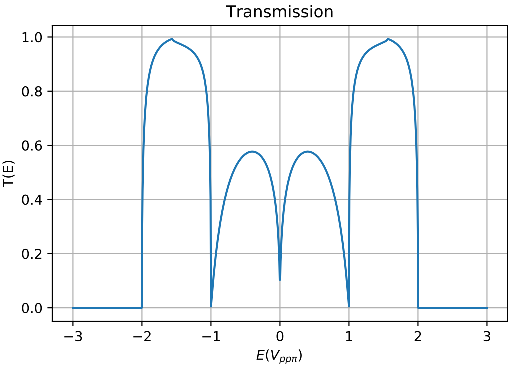
# Getting Transmission

$$T(E) = \mathrm{Tr}\left[\mathbf{\Gamma}_R \mathbf{G}_D \mathbf{\Gamma}_L \mathbf{G}^{\dagger}\right](E)$$

```
    for i in range(En.shape[0]):  # Iteration for every energy point.
        T[i] = np.trace((GammaR["GammaR{:d}".format(i)] @ GD["GD{:d}".format(
            i)] @ GammaL["GammaL{:d}".format(i)] @ GD["GD{:d}".format(i)].conj(
        ).transpose()).todense())  # Calculate transmission (equation V.9).
```

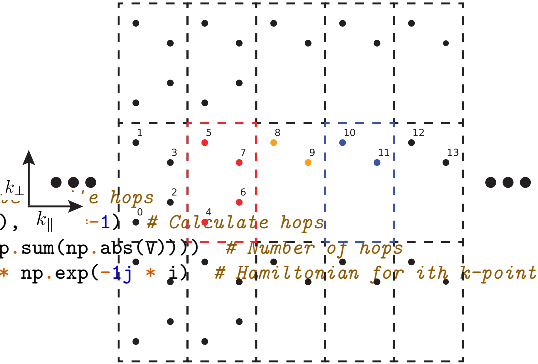
# Getting Transmission

- Text1
- Text2
- Text3



# Transmission in 2D

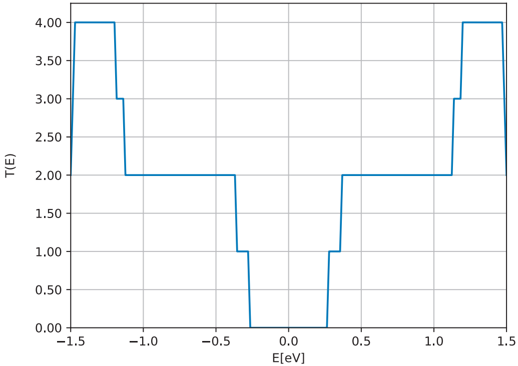
$$\mathbf{H} = \mathbf{h} + \mathbf{V}\mathrm{e}^{ik_{\perp}} + \mathbf{V}^{\dagger}\mathrm{e}^{i(-k_{\perp})}$$

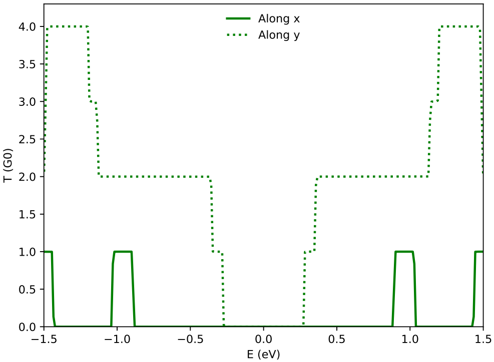



```python
h, p = Onsite(xyz=xyz, Vppi=-1, f=1)  # Calculate onsite hops
V = Hop(xyz=xyz, xyz1=xyz + np.array([0, UY, 0]), Vppi=-1)  # Calculate hops
print('Number of hopping elements: {}'.format(np.sum(np.abs(V))))  # Number of hops
Ham = h + V * np.exp(1j * i) + np.transpose(V) * np.exp(-1j * i)  # Hamiltonian for ith k-point
```

# Code validity

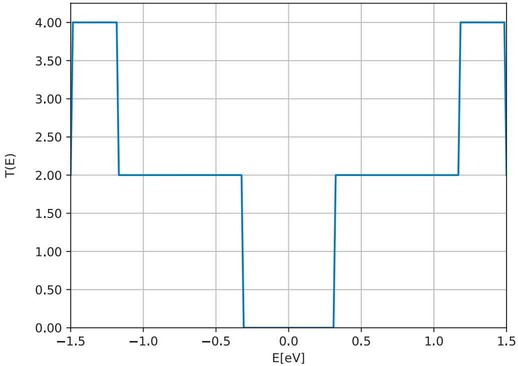
$$k = \frac{\pi}{2}$$

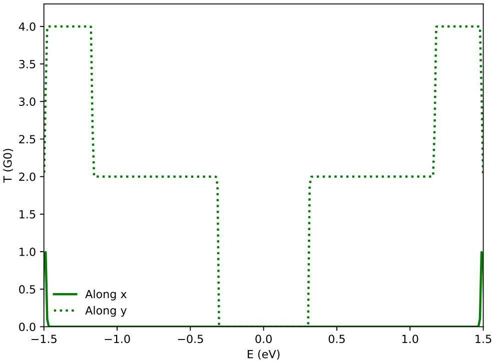


# Code validity

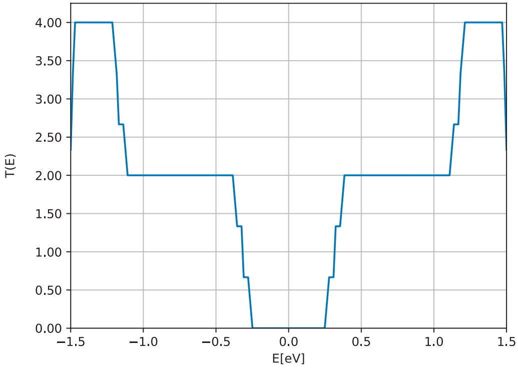
$$k = \pi$$

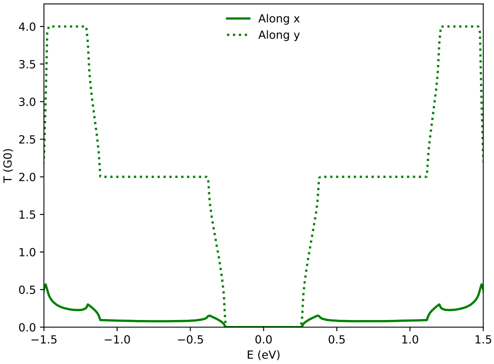


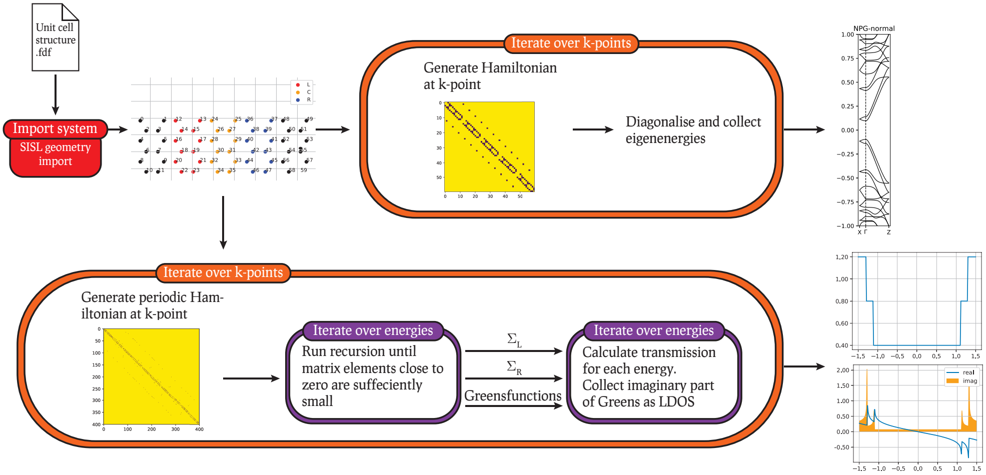
# Code validity

$$k = \text{AVG}\left(0, \frac{\pi}{2}, \pi\right)$$

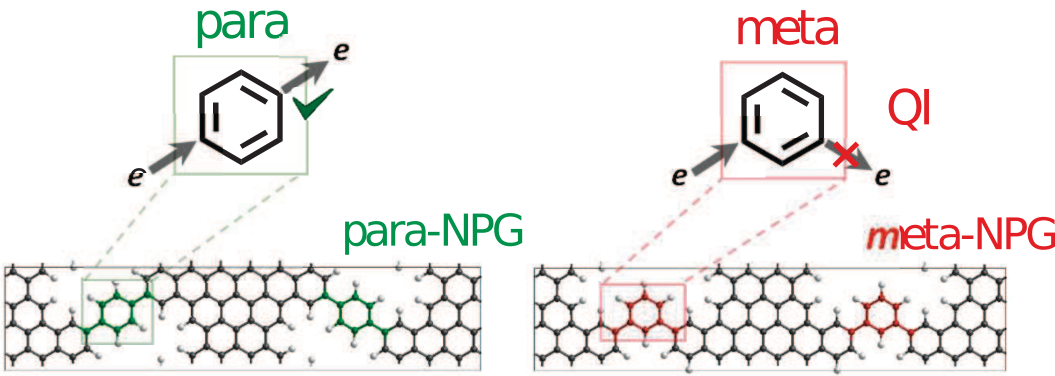


# Summary of code structure



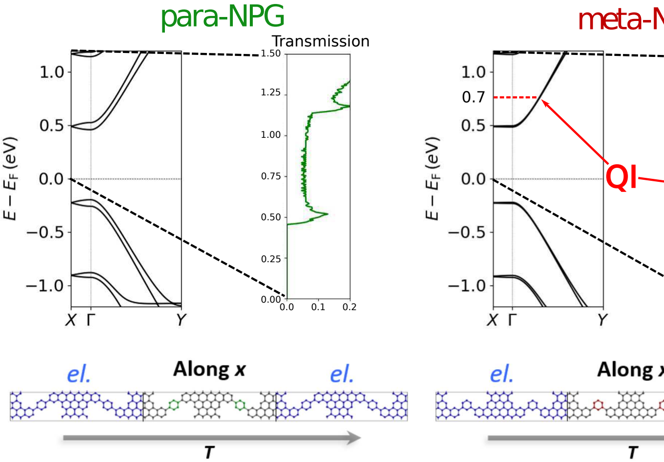
# Para and meta bridges




- Test

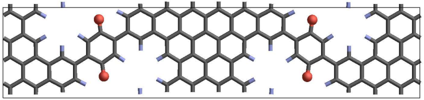
# Para and meta bridges

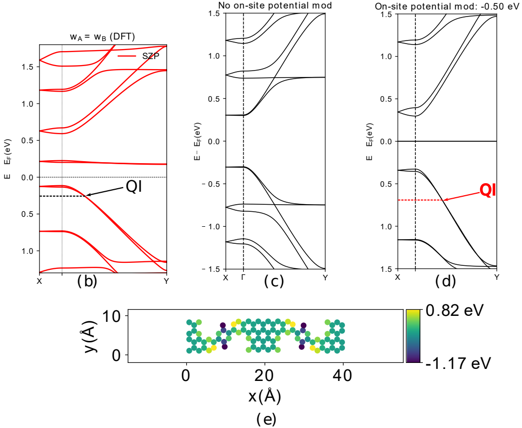
- Text1



# Para-$O_4$-NPG

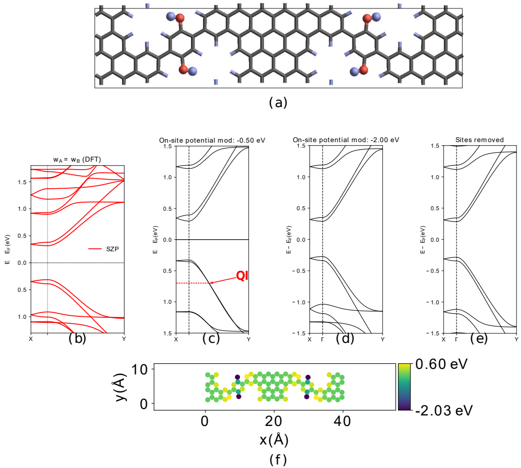
- text



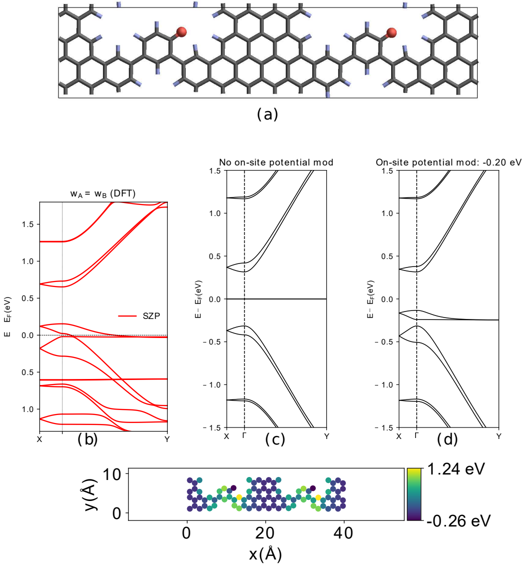
# Para-(OH)$_4$-NPG

- text

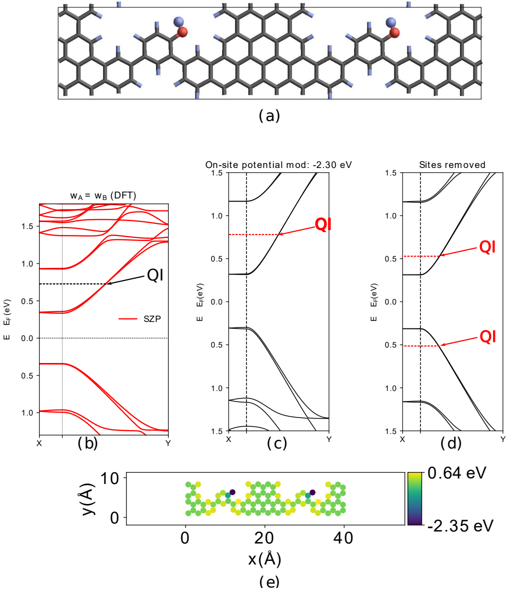


# Meta-$O_2$-NPG

- text



- text


(a)
$w_A = w_B$ (DFT)
QI
SZP
(b)
On-site potential mod: -2.30 eV
QI
(c)
Sites removed
QI
QI
(d)
0.64 eV
-2.35 eV
y(Å)
x(Å)
(e)

# Conclusion

"Development of tight-binding routines in Python in order to understand electron transport in novel nanoporous graphene devices (NPGs)"

# Questions

Christoffer Vendelbo Sørensen (s163965)

June 25, 2019

# Energy and velocity fits

- Appendix