

Graphene Junction Device

Tutorial

Version 12.2.0

Graphene Junction Device: Tutorial

Version 12.2.0

Copyright © 2008–2012 QuantumWise A/S

Atomistix ToolKit Copyright Notice

All rights reserved.

This publication may be freely redistributed in its full, unmodified form. No part of this publication may be incorporated or used in other publications without prior written permission from the publisher.

TABLE OF CONTENTS

1. Introduction	1
2. Calculating the Transmission Spectrum	3
Building a graphene junction device using the Custom Builder	3
Starting the Script Generator	4
Choosing the calculator parameters in the Script Generator	4
Adding analysis objects from the Script Generator	6
Saving a calculation	7
Running the calculation	7
Examining the results — the TransmissionSpectrum object	8
Plotting the 3-D electrostatic potential	9
3. Extracting Information from the Transmission Spectrum	11
Calculating the temperature dependent conductance	11
Investigating a longer graphene device junction	12
Electron thermal transport	14
4. Effect of the Gate Potential	16
Making a gate potential loop	16
Calculating the conductance for different gate potentials	17
A linear response calculation of the gate dependence on the conductance	18
5. Calculating the I-V Characteristics	21
Making an electrode bias loop	21
Self-consistent I-V characteristics	22
Linear response I-V characteristics	23
6. Further Analysis	25
Varying both the gate and the bias potential	25
Further analysis with ATK-SE	27
Bibliography	28
Index	29

CHAPTER 1. INTRODUCTION

The purpose of this tutorial is to present how the ATK-SE package in combination with Virtual NanoLab (VNL) can be used to investigate a nano-scale transistor. For the transistor structure you will use a graphene junction device inspired by the paper [2], where ATK was used to investigate the properties of a similar system.

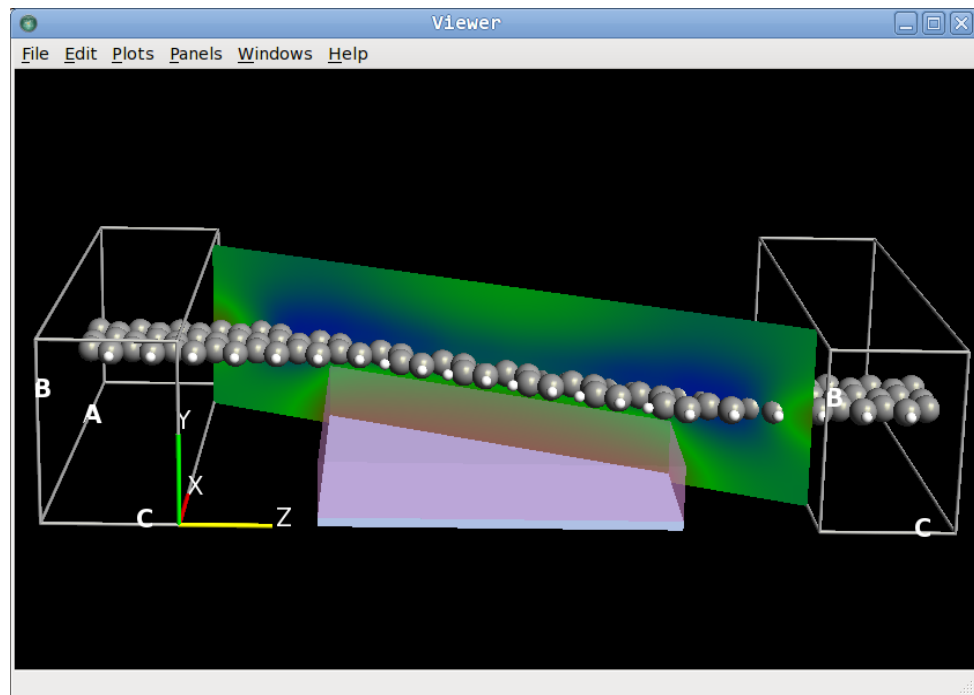


Figure 1.1: The device system considered in this tutorial consists of a z-shaped graphene nano-ribbon, on top of a dielectric and controlled by a metallic gate. The contour plot illustrates the electrostatic potential through the system.

The system is illustrated in [Figure 1.1](#). It consists of 3 regions and forms a metal-semiconductor-metal junction. By applying a gate potential to the central region, the system can function as a field effect transistor. You will calculate the following properties of the system.

- **Transmission spectrum**
- **Temperature dependent conductance**
- **Temperature induced current**
- **Conductance as function of gate potential and temperature**

-
- **Current as function of bias, gate potential and temperature**




Note

Whenever possible you will use VNL for setting up and analyzing the results. To familiarize yourself with VNL it is recommended running the [VNL Quick Tour](#).

The **Atomistix ToolKit Semi Empirical** (ATK-SE) is the underlying calculation engine for this tutorial. A complete description of all the parameters, and in many cases a longer discussion about their physical relevance, can be found in the [ATK Reference Manual](#).

CHAPTER 2. CALCULATING THE TRANSMISSION SPECTRUM

BUILDING A GRAPHENE JUNCTION DEVICE USING THE CUSTOM BUILDER

To setup the atomic coordinates of the device you will use the Custom Builder tool. Open up VNL and start up the Custom Builder tool with a **left**-click on the icon  on the Toolbar.

Within the Custom Builder select the **Graphene Junction** builder from the **Builders** menu.

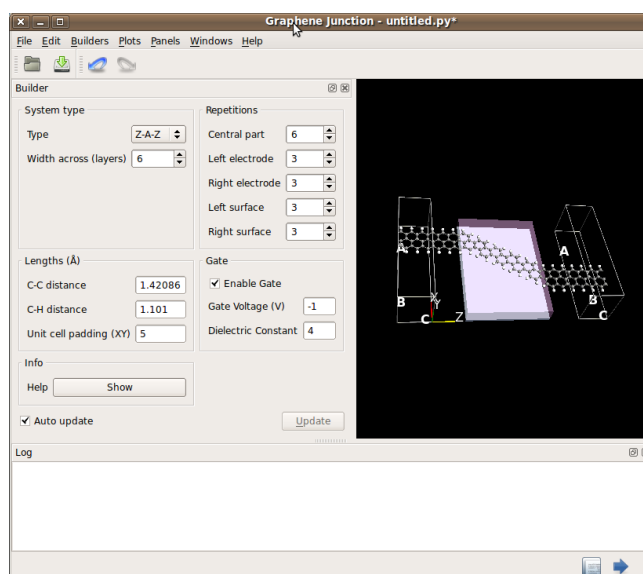
After selecting the **Graphene Junction** builder, the left side of the tool will have a number of parameters that may be edited and in the right side there is a 3-D view of the structure which corresponds to the current settings of the builder.

The structure is of the **DeviceConfiguration** type which consists of two electrodes and a central region. The electrodes are periodic structures, and their unit cells are visualized. The central region seamlessly connects the left electrode with the right electrode.

Note

You may **left**-click and drag within the 3-D view in order to rotate the view. If you **right**-click in the 3-D window and select **Properties...** from the pop-up menu, you can modify how the device is visualized in the view.

Change the settings of the builder parameters according to the following figure.





The new settings of the parameters have elongated the electrodes and the central region, as well as enabled a dielectric region and a gate potential below the structure. The dielectric region is visualized as a transparent purple region and the gate potential is a metallic region below this.

The required length of the electrode cell depends on the computational model employed. Strictly speaking, the length of the cell in the the C-direction should be longer than the inter-action distance of the model. For most models a length of about 7 Å suffices.

Tip

You can get a description of the different input parameters by **left**-clicking the Help button in the info box.

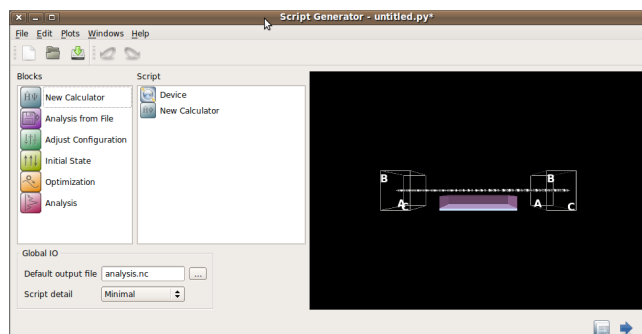
STARTING THE SCRIPT GENERATOR

The next step is to set up the analysis of e.g. the transmission spectrum for the graphene junction, as well as other properties. First, locate the **drag** icon  positioned in the bottom right-hand corner of the Database window. **Left**-click and drag-and-drop the icon on to the **Script Generator** tool  located on the VNL Main Window toolbar. The Script Generator tool then opens with the graphene junction displayed in the Script Generator's 3-D viewer indicating that this is the active atomic configuration of the tool.

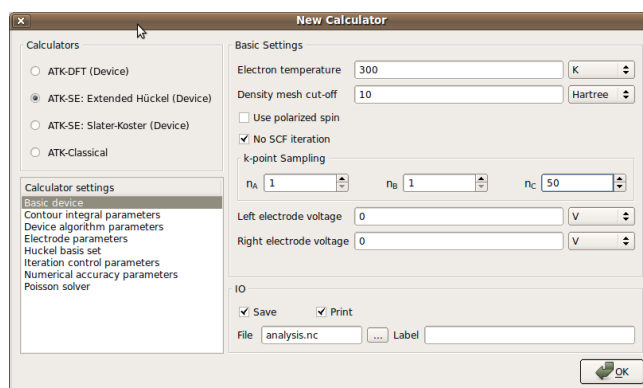
The Custom Builder tool is no longer needed, so to save space on your desktop, close the tool by pressing **Ctrl+W**.

CHOOSING THE CALCULATOR PARAMETERS IN THE SCRIPT GENERATOR

Your next task is to set up a calculation for the graphene junction and specify the physical properties that should be calculated. The first step is to create the **calculator**. To do this, **double**-click the **New Calculator** item in the left column. This creates a new calculator block, which is the first block in the input script that is build up in the right column. The Script Generator tool should look like the following



In order to modify the calculator parameters, **double**-click the newly created item in the right column. This brings up a window where you can change and fine-tune various properties of the calculator. Select the **Device Extended Hückel** calculator within this new window, and lower the number of k-points in the C direction to 50, as this should be sufficient. The window should look like this



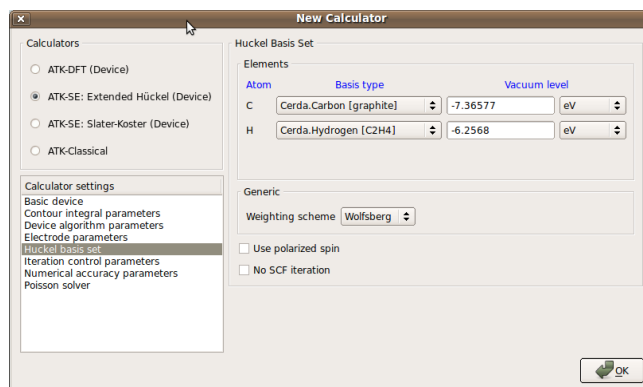
The next step is to change the basis set for the calculation to use parameters specifically fitted to give a good description for Graphite [4]. To this end **left-click** on the **Hückel basis set** line in the **Calculator settings** list and select the **Cerda.Carbon [Graphite]** and **Cerda.Hydrogen [C2H4]** basis set.

 **Note**

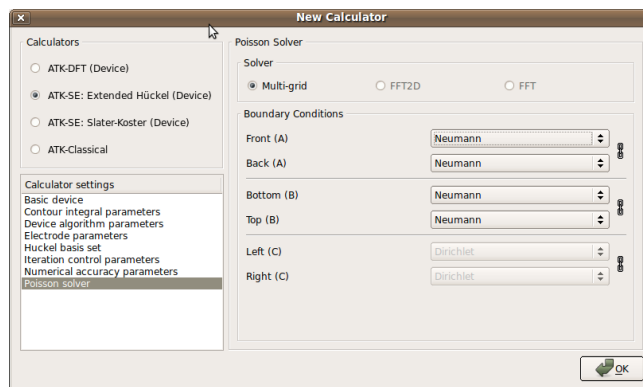
The default value of the vacuum level is adjusted such that even though the Carbon and Hydrogen parameters are fitted to different reference systems, they can be used together.

Uncheck the **No SCF iteration** box, to perform a self consistent calculation. Electro-static interactions are only included in the self consistent model.

The window should now look like this



The next part is to set up the boundary conditions for the Poisson solver. **Left-click** on the **Poisson solver** item of the calculator settings list. In the C-direction the boundary condition for the electrostatic potential is determined by the electrostatic potential in the electrodes, corresponding to a **Dirichlet** boundary condition. For the two other directions choose a **Neumann** boundary condition. The Neumann boundary condition corresponds to a zero electric field at the boundary of the computational box, and this gives the best model for studying the effect of the gate potential. This is the last calculator parameter you will need to set and the window should now look like this



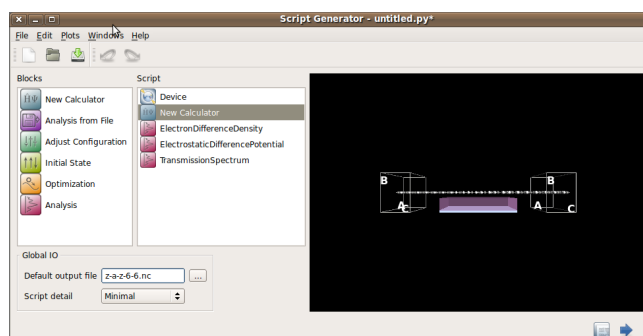
ADDING ANALYSIS OBJECTS FROM THE SCRIPT GENERATOR

Adding analysis objects for calculating physical properties of the system under study is as easy as **double-clicking** the red **Analysis** item in the left column of the Script Generator. This brings up a pop-up menu from which you can select different analyses to carry out. Select the **ElectronDifferenceDensity**, **ElectrostaticDifferencePotential**, and **TransmissionSpectrum** analysis objects. As you will learn later in this tutorial, the conductance and current of the device are obtained via the **TransmissionSpectrum** analysis object.

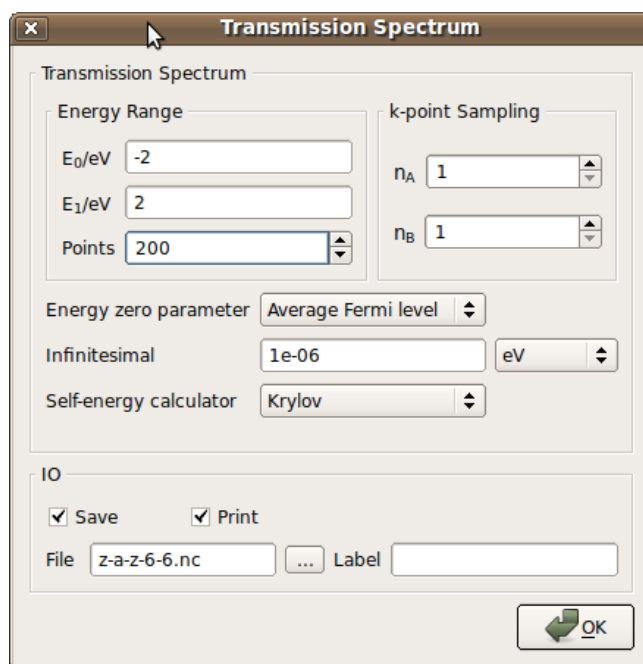
In order to save results into a more telling filename other than "analysis.nc", rename the default output file to "z-a-z-6-6.nc". This will save all calculations to this single file. Ultimately, the three analysis objects will be listed in the right column of the Script Generator and it should look as follows

Tip

Each analysis object may optionally save their results to their own custom NetCDF file via their own settings window. This window is accessible by **double-clicking** on the relevant analysis object in the right column.




Finally, change the default settings for the **TransmissionSpectrum** analysis object to generate a spectrum with 200 points in the energy range $[-2,2]$ eV. **Double-click** on the **TransmissionSpectrum** object in the right column and specify parameters accordingly in the window that will open, as illustrated below



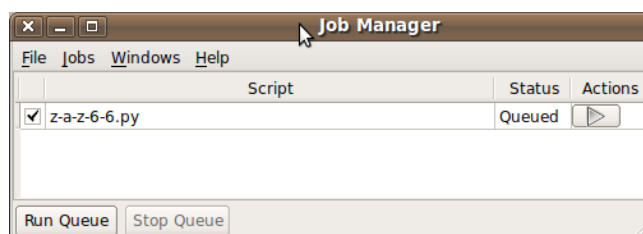
SAVING A CALCULATION

You are now done setting up the calculation. The next step is to **save** the calculation script itself. You do this by pressing the **Save As** button in the Script Generators **File** menu, which saves the defined calculation setting as a Python script on your disk. Select the name “z-a-z-6-6.py” and **left-click Save**. Now, change to the VNL Main Window. Notice that the file z-a-z-6-6.py you saved is displayed in the browser view of the Main VNL Window.

RUNNING THE CALCULATION

Once your calculation has been stored as a script on disk, the script can be executed with the **Job Manager** tool. To do this, locate the file in the browser view of the Main Window, and then drag-and-drop this file onto the Job Manager icon  located in the toolbar.

The Job Manager window then appears



To start the execution of the calculation, press the **Run Queue** button. As a result, a log window appears displaying and updating information regarding the job execution state of each script in the job queue. The job will take 5 to 15 minutes, depending on your computer.

The job will first perform a self-consistent calculation for each of the electrodes. The self-consistent Hartree potential is then used as a boundary condition for a self-consistent calculation

for the central region of the device. Finally, the transmission spectrum and other analysis objects of the structure will be calculated. Once the job has finished, the log window will contain the following lines at the end:

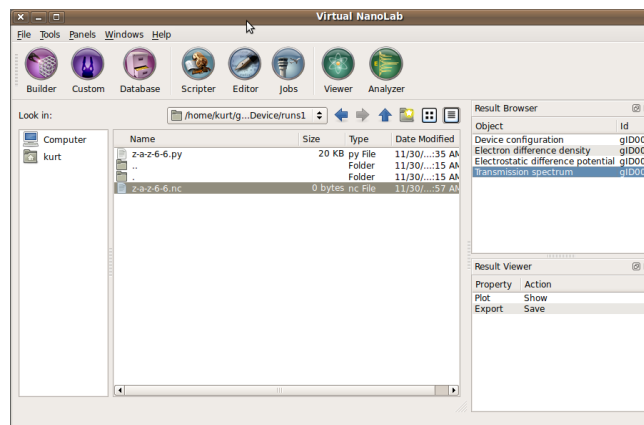
```
NanoLanguageScript finished successfully
+-----+
| NanoLanguageScript execution finished |
+-----+
```

EXAMINING THE RESULTS — THE TRANSMISSION SPECTRUM OBJECT

You will now examine the outcome of the calculation. In VNL, the Main Window is used for inspecting, analysing, extracting, and visualizing the results of your calculation.

Switch to the VNL Main Window, inspect the browser view, and notice that a NetCDF file `z-a-z-6-6.nc` has been generated. Then **left-click** this particular file in the browser view, and observe how the contents of `z-a-z-6-6.nc` is summarized in the top-right part of the Main Window, where entries for **Device Configuration**, **Electron difference density**, **Electrostatic difference potential**, and **Transmission spectrum** are displayed.

Then **left-click** to select the **Transmission spectrum** item. The bottom-right part of the Main Window now contains two entries, namely **plot** (for visualizing the transmission spectrum) and **export** (for generating a text file with the transmission spectrum data). Your VNL Main Window should look like this:



To view the transmission spectrum, press the **Show** button. This will reveal the following plot of the transmission spectrum.

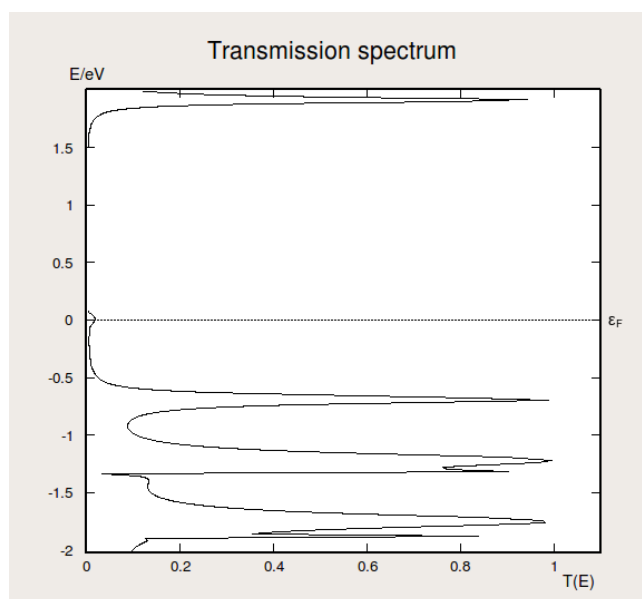



Figure 2.1: Transmission spectrum of the z-a-z-6 graphene device structure with a gate potential of -1 Volt.

Note that the transmission spectrum has a low value in the energy range [-0.5, 1.5] eV, corresponding to the energy window within the band gap of the central semi-conducting armchair-edge ribbon. The asymmetric position of the electrode Fermi levels relative to the band edges, i.e. the shift of the valence band edge of the central region towards the electrode Fermi levels, is due to the applied gate potential of 1 V.

You may zoom into details of the plot by moving the mouse while pressing the **left**-mouse button. To reset the zoom level, use the shortcut **Ctrl+R** or **right**-click the plot to access the command via a pop-up menu.

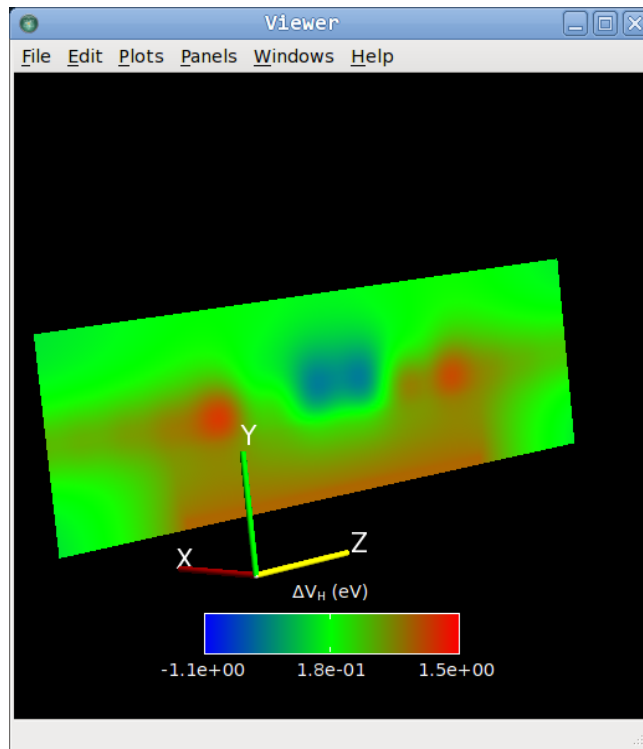
PLOTTING THE 3-D ELECTROSTATIC POTENTIAL

To visualize the effect of the gate potential, select the **Electrostatic difference potential** item and press its associated Contour **Show** button. This action launches the 3-D Viewer tool (corresponding to the  icon on the VNL Toolbar).

You may manipulate the Viewer camera with the following mouse operations:

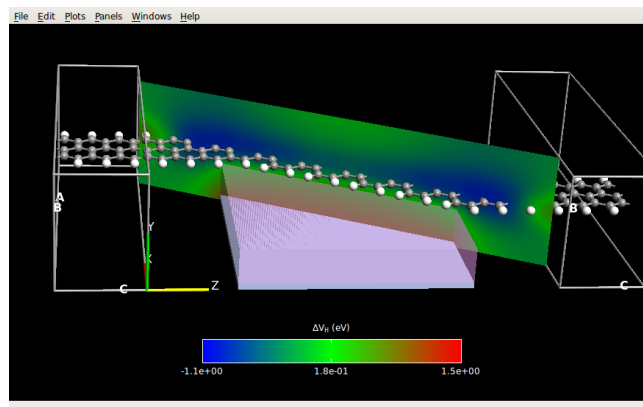
- To **rotate** the camera angle, hold down the **left** mouse button and move the mouse around to obtain the desired view angle.
- To **zoom** in and out, **scroll the mouse wheel**. Alternatively, hold down **Ctrl**, press the **left** mouse button and move the mouse backwards and forwards.
- To **pan** the camera across the view, press the **middle** mouse button (or hold down **Shift** and press the **left** mouse button) and move the mouse.

Use the above mouse operations to position the contour plot of the electrostatic difference potential similar to the plot shown below



To make the plot more interesting, add a 3-D rendering of the graphene junction by first selecting the **Device configuration** item in the top-right view of the Main Window. Then click on **Device configuration** in the bottom-right view, and drag-and-drop the item onto the open 3-D Viewer window.

Next rotate the contour plot. This is done by **right-clicking** the 3-D Viewer window and choosing the **Properties...** item. Then select the contour cut plane from plots tree shown in the appearing dialog, and change the angle θ_y to 151° . Finally, apply mouse operations to fine tune the camera of the 3-D Viewer and obtain a plot similar to the plot shown below



If you wish to **export** a copy of the contour plot, **right-click** in the 3-D Viewer window, and choose **Export...** (**Ctrl+E**). Type in the name of your graphics file in the appearing file dialog browser, for example "contour.png" or "contour.jpg" (the graphics file format is chosen from the file extension you specify). Finally, press the **Save** button. The plot is now stored on disk and can be used for further processing in any other third party application of your choice.

In the next chapter you will learn how to calculate the temperature dependent conductance from the **TransmissionSpectrum** analysis object.

CHAPTER 3. EXTRACTING INFORMATION FROM THE TRANSMISSION SPECTRUM

CALCULATING THE TEMPERATURE DEPENDENT CONDUCTANCE

In this section you will learn how the transmission spectrum can be used to calculate the conductance of the device as function of the left and right electrode temperatures, T_L, T_R . At $T_L = T_R = 0$ the conductance is determined by the transmission coefficient at the Fermi Level, while for finite T_L, T_R the conductance depends on the value of the transmission coefficient in an energy window around the Fermi level. For $T_L = T_R$ the zero bias conductance is given by

$$\sigma(T_L) = \frac{2e^2}{h} \int T(E) f' \left(\frac{E - E_F^L}{k_b T_R} \right) \frac{dE}{k_b T_R}$$

where f' is the derivative of the Fermi function [8].

In semiconductors the conductance is often determined by hot electrons or holes propagating within the conduction or valence band, so-called thermionic emission. The hot electrons are located in the energy range of the tails of the Fermi function. Thus, for an accurate determination of the conductance from thermionic emission, it is important that the energy range of the transmission spectrum is such that the tails of Fermi functions are properly sampled. Figure 2.1 of the previous chapter shows that both the valence and conduction band edges of the central region is within the energy range of the transmission spectrum, thus, from the calculated data you will be able to accurately determine the temperature dependent conductance.

In order to calculate the temperature dependent conductance use the following NanoLanguage script.

```
# Read first transmission spectrum from file z-a-z-6-6.nc
transmission_spectrum = nload("z-a-z-6-6.nc", TransmissionSpectrum)[0]

#make list of temperatures
temperature_list = numpy.linspace(0, 2000, 21) * Kelvin
#make list for holding conductance
conductance_list = numpy.zeros(len(temperature_list))
#calculate the conductance for each temperature in the temperature list
for i in range(len(temperature_list)):
    conductance_list[i] = transmission_spectrum.conductance(
        electrode_temperatures = (temperature_list[i], temperature_list[i]) )

#plot the conductance as function of temperature
import pylab
pylab.figure()
pylab.semilogy(temperature_list, conductance_list)
pylab.xlabel("Temperature (K)")
```

```
pylab.ylabel("Conductance (S)")
pylab.show()
```

The script reads the transmission spectrum from the NetCDF archive file calculated in the previous section. It then uses the `conductance()` method for calculating the conductance for a number of electrode temperatures. Save the script and drop the saved script onto the Job Manager or run the command

```
atkpython conductance_t.py
```

and you will obtain the figure below as a result of the calculation.

i Tip

You may also select the text of the script and drag and drop it onto the job manager.

The script will generate the following output

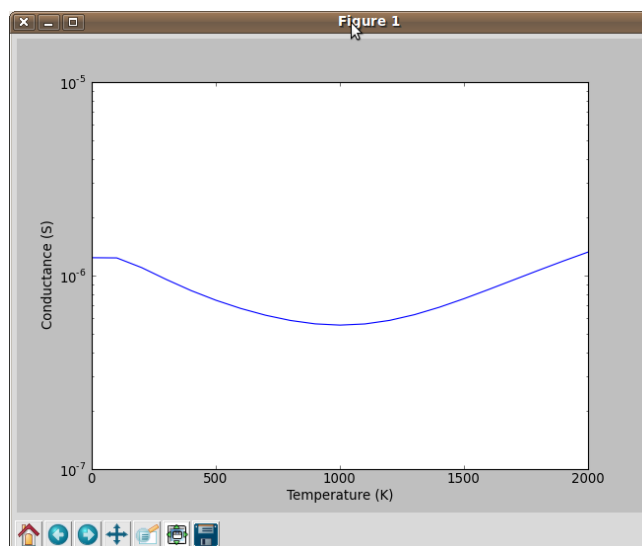


Figure 3.1: Conductance as function of the electron temperature of the electrodes.

The temperature dependent conductance for this device is rather atypical for a field effect transistor. Usually the conductance will increase as function of temperature. The decreasing conductance as function of temperature is related to a large conductance at the Fermi level caused by electron tunneling. For a longer device, the tunneling will be suppressed, and in the next section you will compare the results with the conductance of a longer device.

The figure was produced with the `pylab` package which is part of `atkpython`

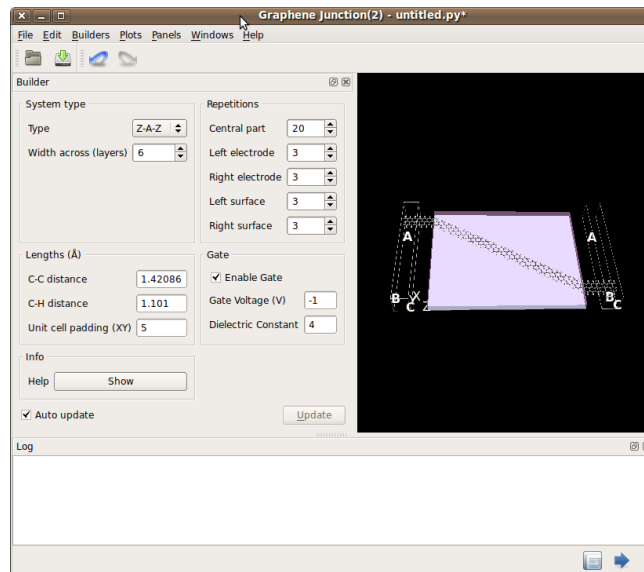
i Tip

For further information on all the options and methods available for the `TransmissionSpectrum` object, see the [ATK Reference Manual](#).

INVESTIGATING A LONGER GRAPHENE DEVICE JUNCTION

You will now setup and calculate the transmission spectrum of a 20 unit long channel, and compare its temperature dependent conductance with the shorter system investigated earlier.

To setup the calculation follow the same steps as in the previous section. The first step is to build the structure with the Custom Builder. The length of the central unit is now specified as 20 repetitions. The new settings of the custom builder is illustrated below



The next step is to setup the calculation, and this part follows exactly the same steps as in the previous chapter for generating the calculation “z-a-z-6-6.py”. As an alternative to setting up the calculation with the VNL Script generation tool, you may save the new geometry into the file “z-a-z-20-6.py” and copy the lines specifying the calculation from “z-a-z-6-6.py” (remember to change the NetCDF filename from “z-a-z-6-6.nc” to “z-a-z-20-6.nc”). The part of the script specifying the calculation is found below the lines

```
#####
# Calculator
#####
```

You may now run the program with the command

```
atkpython z-a-z-20-6.py > z-a-z-20-6.out
```

where all output is redirected to the file “z-a-z-20-6.out”. The calculation will take approximately twice as much time as the previous calculation for the shorter junction.

The following script calculates the temperature dependent conductance of the two junctions

```
# Read first transmission spectrum from file z-a-z-6-6.nc and z-a-z-20-6.nc
transmission_spectrum6 = nload("z-a-z-6-6.nc",TransmissionSpectrum)[0]
transmission_spectrum20 = nload("z-a-z-20-6.nc",TransmissionSpectrum)[0]

#make list of temperatures
temperature_list=np.linspace(0,2000,21)*Kelvin
#make list for holding conductance
conductance_list6=np.zeros(len(temperature_list))
conductance_list20=np.zeros(len(temperature_list))
for i in range(len(temperature_list)):
    conductance_list6[i]=transmission_spectrum6.conductance(
        electrode_temperatures=(temperature_list[i],temperature_list[i]) )
    conductance_list20[i]=transmission_spectrum20.conductance(
        electrode_temperatures=(temperature_list[i],temperature_list[i]) )
```

```

#plot the conductance as function of temperature
import pylab
pylab.figure()
pylab.semilogy(temperature_list,conductance_list6,label="short junction")
pylab.semilogy(temperature_list,conductance_list20,label="long junction")
pylab.xlabel("Temperature (K)")
pylab.ylabel("Conductance (S)")
pylab.legend(loc="lower left")
pylab.grid(True)
pylab.show()

```

Below is shown the result of the calculation. The green curve shows the result for the longer junction. Note that in this case the conductance increases strongly with temperature, indicating that the dominating transport mechanism is thermionic emission.

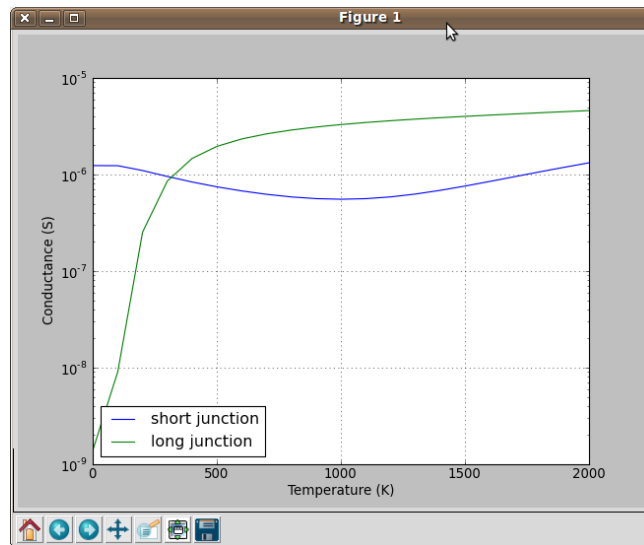


Figure 3.2: Conductance as function of the electron temperature for the short (blue) and 20 unit long (green) graphene junction.

ELECTRON THERMAL TRANSPORT

In the previous section you calculated the conductance of the junction. To obtain an electrical current it is usually necessary to apply an external bias. This section shows how it is also possible to drive a current through the junction by having the two electrodes at different temperatures. The NanoLanguage script for the calculation is displayed below

```

# Read first transmission spectrum from file z-a-z-6-6.nc and z-a-z-20-6.nc
transmission_spectrum6 = nload("z-a-z-6-6.nc",TransmissionSpectrum)[0]
transmission_spectrum20 = nload("z-a-z-20-6.nc",TransmissionSpectrum)[0]
#make list of temperatures
temperature_list=numpy.linspace(0,2000,21)*Kelvin
#make list for holding current
current_list6=numpy.zeros(len(temperature_list))
current_list20=numpy.zeros(len(temperature_list))
#fix the temperature in the left electrode
temperature_left=300*Kelvin
for i in range(len(temperature_list)):
    current_list6[i]=transmission_spectrum6.current(
        electrode_temperatures=(temperature_left,temperature_list[i]) )
    current_list20[i]=transmission_spectrum20.current(
        electrode_temperatures=(temperature_left,temperature_list[i]) )

```

```

#plot the current as function of temperature in right electrode
import pylab
pylab.figure()
pylab.plot(temperature_list,current_list6,label="short junction")
pylab.plot(temperature_list,current_list20,label="long junction")
pylab.xlabel("Right Electrode Temperature (K)")
pylab.ylabel("Current (A)")
pylab.legend(loc="lower left")
pylab.grid(True)
pylab.show()

```

The electron temperature of the left electrode is now fixed at 300 Kelvin, while the temperature of the right electrode is varied from 0 to 2000 Kelvin. The figure below shows the output of the calculation. The change in temperature has the strongest effect on the longer junction (the green curve). The positive direction of the current is from left to right, thus, the electrical current goes from the high temperature to the low temperature. Figure 2.1 shows that the valence band of the central region is closest to the Fermi Level, thus, the transport is dominated by hole transport. The electrode with the highest temperature will have most holes, which will propagate to the other electrode giving rise to a current in this direction.

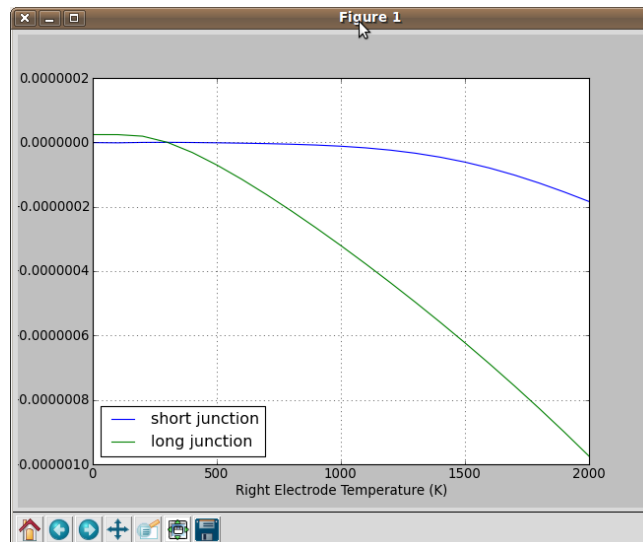


Figure 3.3: Current as function of the electron temperature of the right electrode for the short (blue) and 20 unit long (green) graphene junction. The electron temperature of the left electrode is fixed at 300 Kelvin.

CHAPTER 4. EFFECT OF THE GATE POTENTIAL

MAKING A GATE POTENTIAL LOOP

You will now investigate the effect of changing the gate potential. In the previous section the gate potential was fixed at -1 V. You will now change the value in the range [-2,2] V, and for each value make a new self-consistent calculation and obtain the corresponding transmission spectrum. Such a loop is conveniently done by the following NanoLanguage script.

```
#read in the old configuration
device_configuration = nload("z-a-z-6-6.nc",DeviceConfiguration)[0]
calculator = device_configuration.calculator()
metallic_region0 = device_configuration.metallicRegions()[0]

# Define gate voltages
gate_voltage_list=numpy.linspace(-2.0,2.0,17)*Volt

for gate_voltage in gate_voltage_list:
    device_configuration.setMetallicRegions(
        [metallic_region0(value = gate_voltage)] )

    # make a copy of the calculator and attach it to the configuration
    # restart from the previous scf state
    device_configuration.setCalculator(calculator(),
        initial_state=device_configuration)

    #Analysis
    filename= 'gatescan-6-6.nc'
    electron_density = ElectronDifferenceDensity(device_configuration)
    nlsave(filename, electron_density,object_id='dens'+str(gate_voltage))

    electrostatic_potential = ElectrostaticDifferencePotential(device_configuration)
    nlsave(filename, electrostatic_potential, object_id='pot'+str(gate_voltage))

    transmission_spectrum = TransmissionSpectrum(
        configuration=device_configuration,
        energies=numpy.linspace(-2,2,200)*eV,
        )

    nlsave(filename, transmission_spectrum,object_id='trans'+str(gate_voltage))
    nlprint(transmission_spectrum)
```

The script has the following flow:

1. Read in the self-consistent calculation obtained in the previous chapter and use this as the starting point for the calculations

```
#read in the old configuration
device_configuration = nload("z-a-z-6-6.nc",DeviceConfiguration)[0]
```

```
calculator = device_configuration.calculator()
metallic_region0 = device_configuration.metallicRegions()[0]
```

2. Set up a list of gate voltage values [-2.0, -1.75, ..., 2.0]*Volt.

```
# Define gate_voltages
gate_voltage_list=numpy.linspace(-2.0,2.0,17)*Volt
```

3. Loop through the gate voltage list, and for each value modify the value of the gate voltage in the metallic region.

```
for gate_voltage in gate_voltage_list:
    device_configuration.setMetallicRegions(
        [metallic_region0(value = gate_voltage)] )
```

4. Initialize the self-consistent state of the **DeviceConfiguration**, by calling the **calculator()** method. The **calculator()** method will make a copy of the previous **DeviceHuckelCalculator** using the same settings. Since a new calculator is attached, the device_configuration will now perform a self-consistent calculation automatically the next time a property of the system is required.

```
# make a copy of the calculator and attach it to the configuration
# restart from the previous scf state
device_configuration.setCalculator(calculator(),
    initial_state=device_configuration)
```

5. Finally, calculate the same analysis objects as in the previous chapter and save them into a NetCDF file. Note that for each analysis object a specific **object_id** is specified. This will make it easier to retrieve the correct entries from the NetCDF file.

The script is executed with the command

```
atkpython gatescan-6-6.py > gatescan-6-6.out &
```

The script will take a while to complete, since 17 self-consistent calculations are needed. Thus, you might need to wait 2-8 hours for job completion, depending on the hardware you are running on.

CALCULATING THE CONDUCTANCE FOR DIFFERENT GATE POTENTIALS

The script in the previous section produced the file "gatescan-6-6.nc". Try to inspect the file with VNL to see how the transmission spectrum changes with the gate potential. You may find that all the transmission spectra look rather similar and the main difference between the different spectra is the shift of valence and conduction band edges relative to the Fermi level.

You will now use the data to calculate the conductance of the junction as a function of the gate potential for different electrode temperatures. For this purpose use the following script, which is a slight modification of the script used in the previous section.

```
#make list of relevant temperatures
temperature_list=numpy.linspace(0,2000,21)*Kelvin
#make list of relevant gate voltages
gate_voltage_list=numpy.linspace(-2.0,2.0,17)*Volt
```

```

#make list to hold the conductance calculations
conductance_list=numpy.zeros(len(gate_voltage_list)*len(temperature_list))
conductance_list=conductance_list.reshape(len(gate_voltage_list),
                                          len(temperature_list))

#specify the filename for the netcdf data file
filename="gatescan-6-6.nc"
#loop through the gate voltages
for n in range(len(gate_voltage_list)):
    transmission_spectrum=nload(filename,
                                object_id="trans"+str(gate_voltage_list[n]))[0]
    #loop through the temperature list
    for i in range(len(temperature_list)):
        conductance_list[n,i]=transmission_spectrum.conductance(
            electrode_temperatures=(temperature_list[i],temperature_list[i]))

#plot the conductance as function of gatevoltage
import pylab
pylab.figure()
# make curve for each temperature
for i in range(len(temperature_list)):
    pylab.semilogy(gate_voltage_list,conductance_list[:,i])
pylab.xlabel("Gate Voltage (V)")
pylab.ylabel("Conductance (S)")
pylab.show()

```

The script produces the following figure

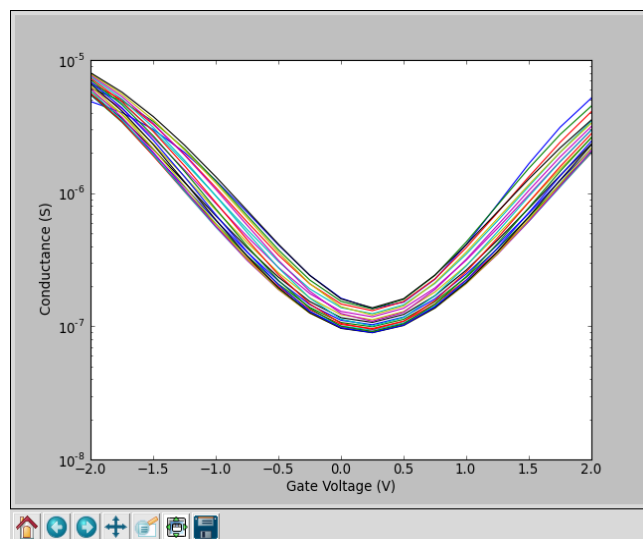


Figure 4.1: Conductance as function of the gate potential. Different curves are for different electron temperatures, namely 0, 100, 200, ..., 2000 Kelvin.

The conductance has a minimum for a gate potential of about 0.25 Volt. For this gate potential, the Fermi level is positioned in the mid gap between the valence and conduction band of the central ribbon. Note that there is only a weak dependence of the conductance on the electron temperature, and this illustrates that the transmission is dominated by tunnelling.

A LINEAR RESPONSE CALCULATION OF THE GATE DEPENDENCE ON THE CONDUCTANCE

Previously, you saw that the transmission spectra are rather similar and the main difference is the shift of the band edges relative to the Fermi level. This suggests that you may avoid the self-consistent loop and simply shift the Fermi level rigidly.

The following script implements a linear response calculation of the conductance as function of the gate potential, assuming that the effect of the gate potential is a simple shift of the relative positions of the electrode Fermi levels within the transmission spectrum.

```
#make list of relevant temperatures
temperature_list=numpy.linspace(0,2000,21)*Kelvin
#make list of relevant gate voltages
gate_voltage_list=numpy.linspace(-2.0,2.0,17)*Volt
#make list to hold the conductance calculations
conductance_list=numpy.zeros(len(gate_voltage_list)*len(temperature_list))
conductance_list=conductance_list.reshape(len(gate_voltage_list),
                                          len(temperature_list))

#read the reference transmission spectrum from the netcdf data file
gate_potential_ref = 0.*Volt
transmission_spectrum = nload("gatescan-6-6.nc",
                              object_id="trans"+str(gate_potential_ref))[0]

#loop through the gate voltages
for n in range(len(gate_voltage_list)):
    #loop through the temperature list
    for i in range(len(temperature_list)):
        conductance_list[n,i]=transmission_spectrum.conductance(
            electrode_temperatures=(temperature_list[i],temperature_list[i]),
            electrode_voltages=(gate_voltage_list[n]-gate_potential_ref,
                               gate_voltage_list[n]-gate_potential_ref ) )

#plot the conductance as function of gatevoltage
import pylab
pylab.figure()
# make curve for each temperature
for i in range(len(temperature_list)):
    pylab.semilogy(gate_voltage_list,conductance_list[:,i])
pylab.xlabel("Gate Voltage (V)")
pylab.ylabel("Conductance (S)")
pylab.show()
```

1. The script reads in the transmission spectrum from a calculation with gate potential 0 Volt.

```
gate_potential_ref = 0.*Volt
transmission_spectrum = nload("gatescan-6-6.nc",
                              object_id="trans"+str(gate_potential_ref))[0]

#loop through the gate voltages
for n in range(len(gate_voltage_list)):
```

2. By setting the **electrode_voltages** of the transmission spectrum, the Fermi levels of the electrodes are moved relatively.

```
        electrode_voltages=(gate_voltage_list[n]-gate_potential_ref,
                            gate_voltage_list[n]-gate_potential_ref ) )

#plot the conductance as function of gatevoltage
import pylab
```

The script produces the figure below. The figure shows that the main trends of [Figure 4.1](#) are reproduced by the linear response calculation.

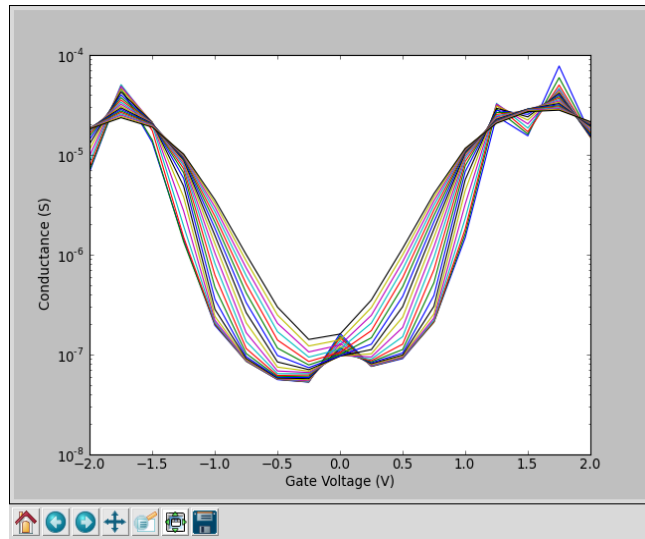


Figure 4.2: Conductance as function of the gate potential calculated using linear response. Different curves are for electron temperatures 0, 100, 200, ..., 2000 Kelvin.

CHAPTER 5. CALCULATING THE I-V CHARACTERISTICS

MAKING AN ELECTRODE BIAS LOOP

You will now investigate the effect of changing the external bias in the range [0,2] Volt. The calculation follows a similar procedure as the gate potential scan in the previous section, and the corresponding script is given below.

```
#read in the old configuration
device_configuration = nload("z-a-z-6-6.nc",DeviceConfiguration)[0]
calculator = device_configuration.calculator()

# Define bias voltages
voltage_list=numpy.linspace(0.0,2.0,9)*Volt
for voltage in voltage_list:

    # Set new calculator with modified electrode voltages on the configuration
    # use the self consistent state of the old calculation as starting input.
    device_configuration.setCalculator(
        calculator(electrode_voltages=(-0.5*voltage,0.5*voltage)),
        initial_state=device_configuration)

    #Analysis
    filename = 'ivscan-6-6.nc'
    electron_density = ElectronDifferenceDensity(device_configuration)
    nlsave(filename, electron_density,object_id='dens'+str(voltage))

    electrostatic_potential = ElectrostaticDifferencePotential(device_configuration)
    nlsave(filename, electrostatic_potential, object_id='pot'+str(voltage))

    transmission_spectrum = TransmissionSpectrum(
        configuration=device_configuration,
        energies=numpy.linspace(-2,2,200)*eV,
        )

    nlsave(filename, transmission_spectrum,object_id='trans'+str(voltage))
    nlprint(transmission_spectrum)
```

The calculation takes reference in the calculation for the junction with an applied gate potential of -1 Volt. This value for the gate potential is fixed through the entire calculation. The electrode voltages are modified through the **electrode_voltages** variable of the **DeviceHuckelCalculator**. The command **calculator()**(**electrode_voltages** = (-0.5*voltage, 0.5*voltage)) returns a new **DeviceHuckelCalculator** object with exactly the same settings as the previous calculator, except for the **electrode_voltages**.

The entire calculation will take more than 10 times longer than the single zero bias calculation. During the loop the results will be saved in the file "ivscan-6-6.nc". The file may be inspected

during the calculation to follow how the calculation is progressing. The plot below shows the electro-static potential for a bias of 1 Volt, i.e. -0.5 Volt on the left electrode and 0.5 Volt on the right electrode. Notice how the electro-static potential in the left electrode is closer to the electro-static potential of the gate.

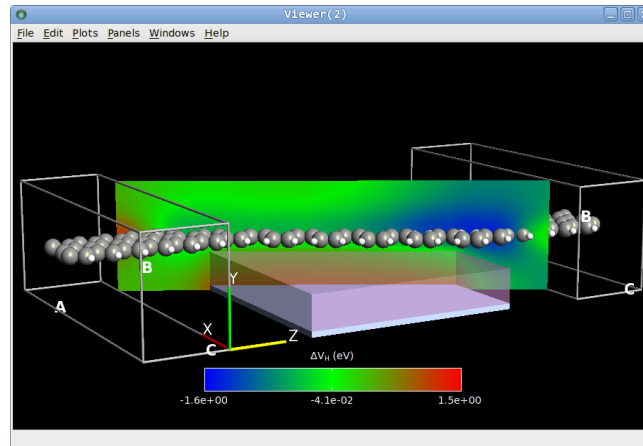


Figure 5.1: Electrostatic potential of the junction with symmetrically applied bias of 1 Volt and a gate potential of -1 Volt.

SELF-CONSISTENT I-V CHARACTERISTICS

You can now use all the **TransmissionSpectrum** objects in "ivscan-6-6.nc" to calculate an I-V curve for a gate potential of -1 Volt. For the calculation use the script below, which combines the methods presented in the previous sections.

```
#make list of relevant temperatures
temperature_list=numpy.linspace(0,2000,21)*Kelvin
#make list of relevant gate voltages
voltage_list=numpy.linspace(0.0,2.0,9)*Volt
#make list to hold the current calculations
current_list=numpy.zeros(len(voltage_list)*len(temperature_list))
current_list=current_list.reshape(len(voltage_list),len(temperature_list))

#specify the filename for the netcdf data file
filename="ivscan-6-6.nc"
#loop through the gate voltages
for n in range(len(voltage_list)):
    transmission_spectrum=nlread(filename, object_id="trans"+str(voltage_list[n]))[0]
    #loop through the temperature list
    for i in range(len(temperature_list)):
        current_list[n,i]=transmission_spectrum.current(
            electrode_temperatures=(temperature_list[i],temperature_list[i]))

#plot the current as function of voltage
import pylab
pylab.figure()
# make curve for each temperature
for i in range(len(temperature_list)):
    pylab.plot(voltage_list,current_list[:,i])
pylab.xlabel("Bias (V)")
pylab.ylabel("Current (A)")
pylab.show()
```

The result of the calculation is illustrated below

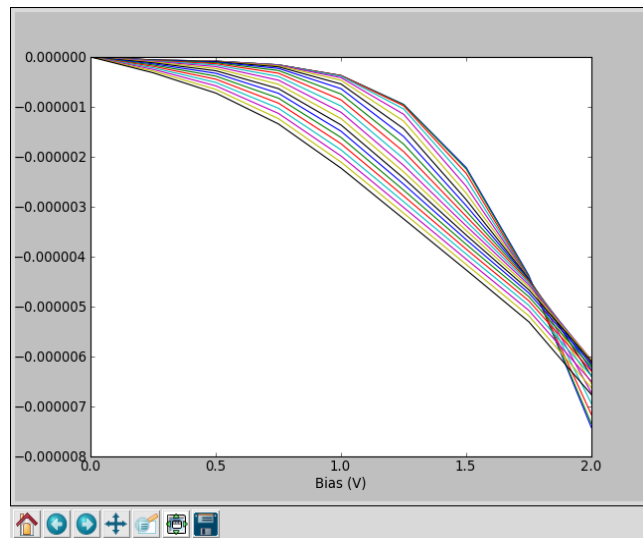


Figure 5.2: Self-consistent current-voltage characteristics of the graphene junction with -1 V gate potential. Curves are for electrode temperatures 0, 100, 200, .. 2000 Kelvin. The current shows an increase with the temperature.

LINEAR RESPONSE I-V CHARACTERISTICS

Similar to the linear response approximation for the gate potential presented in the previous section, you may also use the **TransmissionSpectrum** at zero bias to approximate the finite bias **TransmissionSpectrum**. The following script performs a calculation of the linear response current for an electrode temperature of 300 Kelvin

```
#make list of relevant temperatures
temperature=300*Kelvin
#make list of relevant gate voltages
voltage_list=numpy.linspace(0.0,2.0,9)*Volt
#make list to hold the current calculations
current_list=numpy.zeros(len(voltage_list))
current_list_lin=numpy.zeros(len(voltage_list))

#read the reference transmission spectrum from the netcdf data file
potential_ref = 0.*Volt
filename="ivscan-6-6.nc"
transmission_spectrum0 = nload(filename, object_id="trans"+str(potential_ref))[0]

for n in range(len(voltage_list)):
    transmission_spectrum=nload(filename, object_id="trans"+str(voltage_list[n]))[0]
    current_list[n]=transmission_spectrum.current(
        electrode_temperatures=(temperature,temperature))
    current_list_lin[n]=transmission_spectrum0.current(
        electrode_temperatures=(temperature,temperature),
        electrode_voltages=(-0.5*(voltage_list[n]-potential_ref),
            0.5*(voltage_list[n]-potential_ref) )
    )

#plot the current as function of bias
import pylab
pylab.figure()
pylab.plot(voltage_list,current_list,label="self consistent")
pylab.plot(voltage_list,current_list_lin,label="linear response")
pylab.xlabel("Bias (V)")
pylab.ylabel("Current (A)")
pylab.legend(loc="lower left")
```

```
pylab.grid(True)
pylab.show()
```

The figure below shows the output of the calculation which is rather similar to [Figure 5.2](#). Thus, the linear response current is a good approximation to the self-consistent current.

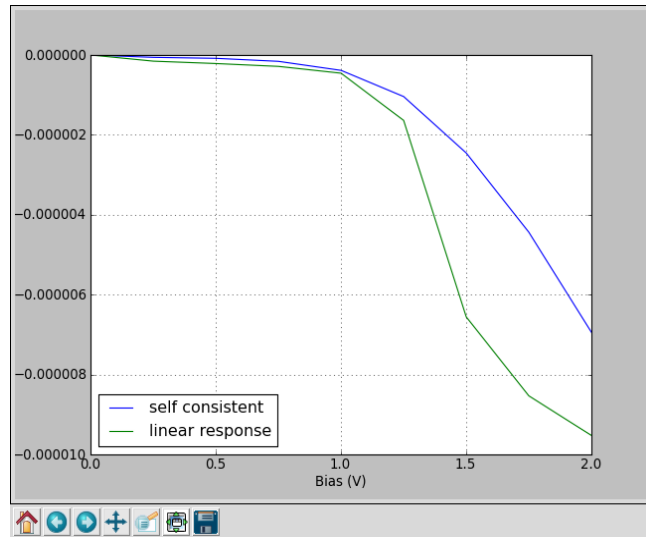


Figure 5.3: Comparison of the self-consistent (blue) and linear response (green) current-voltage characteristics of the graphene junction with -1 V gate potential and electrode temperature of 300 Kelvin.

CHAPTER 6. FURTHER ANALYSIS

VARYING BOTH THE GATE AND THE BIAS POTENTIAL

In the previous sections illustrates how to perform calculations of the current as function of the electrode bias or the gate potential. Even though the methodology is based on the extended Huckel formalism, such calculations can be quite time consuming, and an alternative is to investigate the effect non-self-consistently using linear response.

The scripts presented in the previous sections can be easily modified to vary the gate potential and the electrode bias simultaneously. Below is shown a comparison between the self-consistent current and the linear response current. For a small bias and gate potential the two are in good agreement, illustrating that the linear response current is a good first approximation.

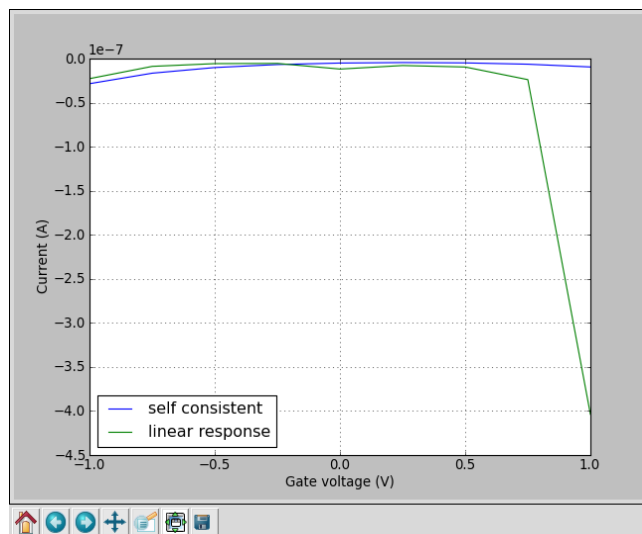


Figure 6.1: Current for an applied bias of 0.2 Volt as function of the gate potential. The blue curve was obtained from a full self-consistent calculation, while the green curve was obtained from a linear response calculation using the transmission spectrum of a system with zero gate potential and bias.

The script for generating the self-consistent data is given below

```
#read in the old configuration
device_configuration = nload("z-a-z-6-6.nc",DeviceConfiguration)[0]
calculator = device_configuration.calculator()
metallic_region0 = device_configuration.metallicRegions()[0]

# Define gate voltages
gate_voltage_list=numpy.linspace(-1.0,1.0,9)*Volt
for gate_voltage in gate_voltage_list:
    device_configuration.setMetallicRegions(
```

```

        [metallic_region0(value = gate_voltage)] )

# Set calculator with applied bias on the configuration
# use the old selfconsistent state as starting input.
device_configuration.setCalculator(
    calculator(electrode_voltages=(-0.1*Volt,0.1*Volt)),
    initial_state=device_configuration)

#Analysis
filename= 'gateivscan-6-6.nc'
electron_density = ElectronDifferenceDensity(device_configuration)
nlsave(filename, electron_density,object_id='dens'+str(gate_voltage))

electrostatic_potential = ElectrostaticDifferencePotential(device_configuration)
nlsave(filename, electrostatic_potential, object_id='pot'+str(gate_voltage))

transmission_spectrum = TransmissionSpectrum(
    configuration=device_configuration,
    energies=np.linspace(-2,2,200)*eV,
)

nlsave(filename, transmission_spectrum,object_id='trans'+str(gate_voltage))
nlprint(transmission_spectrum)

```

The script below produced [Figure 6.1](#)

```

#make list of relevant temperatures
temperature=300*Kelvin
#make list of relevant gate voltages
gate_voltage_list=np.linspace(-1.0,1.0,9)*Volt
#make list to hold the current calculations
current_list=np.zeros(len(gate_voltage_list))
current_list_lin=np.zeros(len(gate_voltage_list))

#read the reference transmission spectrum from the netcdf data file
gate_potential_ref = 0.*Volt
bias_potential_ref= 0.2*Volt
transmission_spectrum0 = nload("gatescan-6-6.nc",
                                object_id="trans"+str(gate_potential_ref))[0]

for n in range(len(gate_voltage_list)):
    transmission_spectrum=nload("gateivscan-6-6.nc",
                                object_id="trans"+str(gate_voltage_list[n]))[0]
    current_list[n]=transmission_spectrum.current(
        electrode_temperatures=(temperature,temperature))
    current_list_lin[n]=transmission_spectrum0.current(
        electrode_temperatures=(temperature,temperature),
        electrode_voltages=
        (-0.5*bias_potential_ref+gate_voltage_list[n]-gate_potential_ref,
         0.5*bias_potential_ref+gate_voltage_list[n]-gate_potential_ref)
    )

#plot the current as function of gatevoltage
import pylab
pylab.figure()
pylab.plot(gate_voltage_list,current_list,label="self consistent")
pylab.plot(gate_voltage_list,current_list_lin,label="linear response")
pylab.xlabel("Gate voltage (V)")
pylab.ylabel("Current (A)")
pylab.legend(loc="lower left")
pylab.grid(True)
pylab.show()

```

FURTHER ANALYSIS WITH ATK-SE

Some of the other analysis that can be performed with VNL are

- 3-D visualization of scattering states
- 3-D visualization of transmission pathways
- Projected density of states

The main part of the analysis in this tutorial was performed using NanoLanguage scripts. However, the analysis can in most cases also be performed using VNL. For further insight into the Analysis see the [ATK device Tutorial](#) .

BIBLIOGRAPHY

- [1] Atomistix ToolKit Manual “ATK version 12.2.0”, QuantumWise A/S (www.quantumwise.com).
- [2] Q. Yan, B. Huang, J. Yu, F. Zheng, J. Zang, J. Wu, B.-L. Gu, F. Liu, and W. Duan. *Nano Letters*, **7**, 1469, 2007. ([link](#))
- [3] S. Datta, “Electronic Transport in Mesoscopic Systems”, Cambridge University Press (Cambridge), 1997
- [4] J. Cerda, and F. Soria, *Phys. Rev. B*, **61**, 7965, 2000 ([link](#)).
- [5] M. Wolfsberg, and L. Helmholtz. *J. Chem. Phys.*, **20**, 837, 1952
- [6] M. H. Whangbo, and R. Hoffmann. *J. Chem. Phys.*, **68**, 5498, 1978
- [7] J. H. Ammeter, H. B. Burgi, J. C. Thibeault, and R. Hoffmann. *J. Am. Chem. Soc.*, **100**, 3686, 1978
- [8] H. H. Sørensen, P. C. Hansen, D. E. Petersen, S. Skelboe, and K. Stokbro. *Phys. Rev. B*, **79**, 205322, 2009. ([link](#))

INDEX

A

ATK, 1

ATK Reference Manual, 2

Atomistix ToolKit (ATK), 1

C

calculation engine, 2

L

linear response, 19

W

work flow, 1