UNIVERSITY OF CAMBRIDGE


University of Cambridge

Cavendish Laboratory,
Department of Physics

BA Natural Sciences


# Electrical properties of molecular arrays sandwiched between gold and graphene electrodes at room and liquid-nitrogen temperatures

**UROP Summer Student Report**


*Author:*
Victor I.COLDEA

*CRSid:*
vic21

*Cavendish Supervisor:*
Prof. Chris J.B.Ford

*PhD Student Supervisor:*
Bingxin Li


July - September 2024

# Summary

In this summer project, I measured the current-voltage (I-V) characteristics for different self-assembled molecule (SAM) devices at room and liquid nitrogen temperatures. The devices were fabricated in the cleanroom, and contain an array of 2,2'-bipyridyl-terminated undecanethiol ($HSC_{11}$BIPY) molecules sandwiched between gold and graphene layers. During the project I measured two different chips, and developed a series of Python programs to analyse the obtained data. I fitted the data to the Simmons model, and collected the parameters from different voltage sweeps to assess the overall behaviour of these molecules and usefulness of the model.

# Contents

# 1 Introduction

The field of molecular electronics allows for novel electronic characteristics that cannot be obtained from ordered metallic or crystalline lattices. This makes them ideal for use in areas such as sensors, diodes, or more complex logic gates in circuit boards. Figure 1 illustrates the energy levels inside a device, and how they can be manipulated to investigate different tunnelling effects.

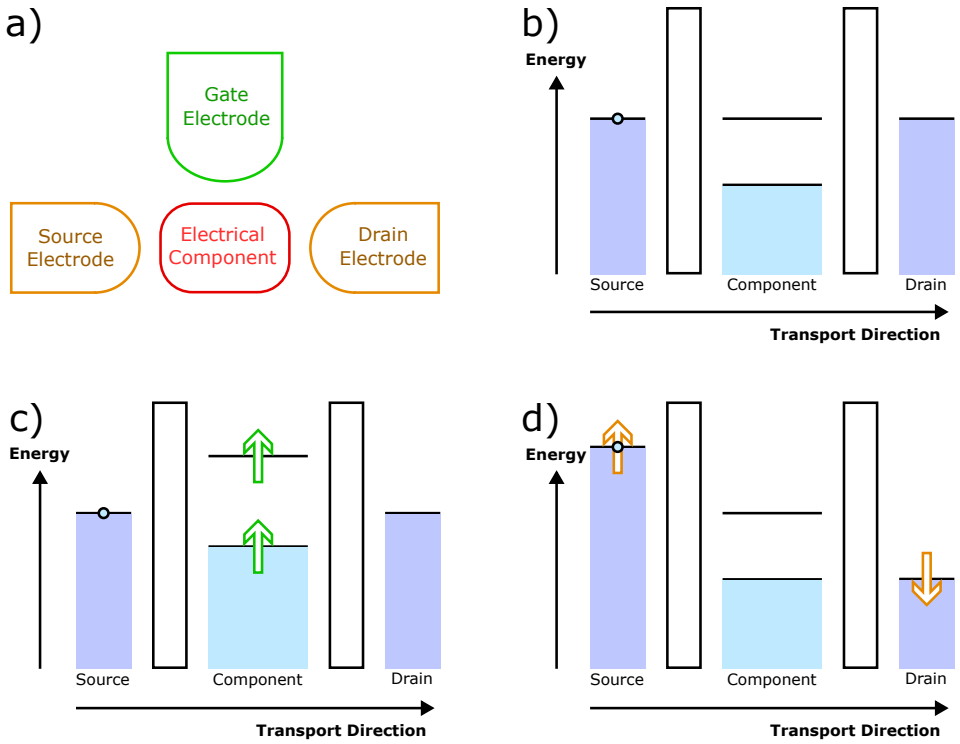


Figure 1: a) Schematic of electrical setup. The source and drain electrodes in this experiment are gold and graphene. b) Energy level diagram (ELD) for electrons in the system, with no bias or gate voltage. The coloured-in regions represent filled energy states, and the energy barriers are represented by the white boxes. c) ELD for an applied gate voltage. The energy levels of the component change in the same direction, signified by the arrows. d) ELD for an applied bias voltage. The source and drain Fermi energies change in opposite ways. This allows complete flexibility in tuning the energy levels, so that the tunnelling effects can be seen over a complete range.

# 2 Project Description

The summer project involved learning about the fabrication process of the chips in the clean room as described in Section 2.1. Next, I perform electrical measurements which required measuring the IV characteristics across different junctions of two chips using the setup shown in Figure 6. Finally I analysed these collected data with respect to the Simmons model which can describe the mechanism of charge transport across tunnelling barriers, as detailed in Section 2.3.1.

## 2.1 Device Fabrication

The research group performs experiments using a variety of different SAMs such as $C_{10}S_2$ and $C_{16}S$. They investigate how tunnelling effects are influenced by various factors such as, molecule type and size, gate voltage, and bias voltage. All the SAMs are either thiols or dithiols, since the sulfur atom can bond well with the gold.
During my project, I mainly focused on 2,2'-bipyridyl-terminated undecanethiol (see Figure 2), and I only varied the bias (sweeping) voltage across the junction.

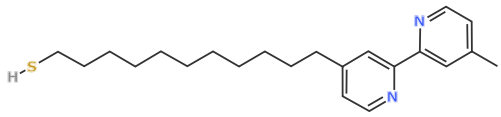


Figure 2: Skeletal formula of 2,2'-bipyridyl-terminated undecanethiol ($HSC_{11}BIPY$). The molecule is $\sim$ 2nm long, and conducts electrons along its length in the chip configuration.

The fabrication process is fully carried out in the cleanroom and takes roughly two weeks. The flowchart in Figure 3 describes the entire process from silicon wafer to fully-assembled device. Making a device requires many steps and each layer must be deposited separately, with the SAM growth occuring on the gold layer [1].

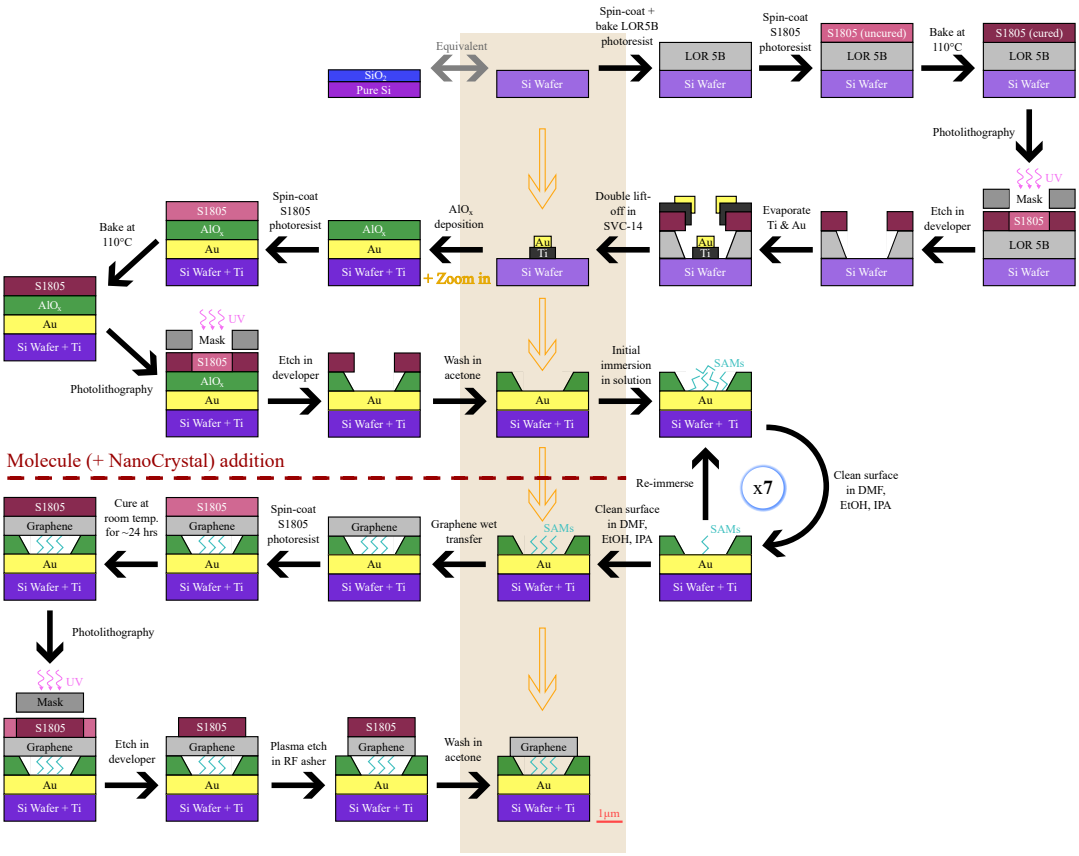


Figure 3: Flowchart of the fabrication steps. The main steps are in the central orange part and intermediate parts are branching off to the sides. The main steps were: Au deposition, $AlO_x$ deposition and etching, SAM growth, and graphene etching. The Au was deposited using a rotary evaporator, after photolithography according to the mask in Figure 4a. $AlO_x$ was deposited in the same way. The chip is submerged in a solution with the SAMs, and then repeatedly washed and resubmerged to maximise consistency among the attached SAMs and to remove any physisorbed molecules. The circuit becomes closed with the current flowing in the direction gold $\rightarrow$ SAM $\rightarrow$ graphene and vice versa. Thicknesses of layers are not to scale.

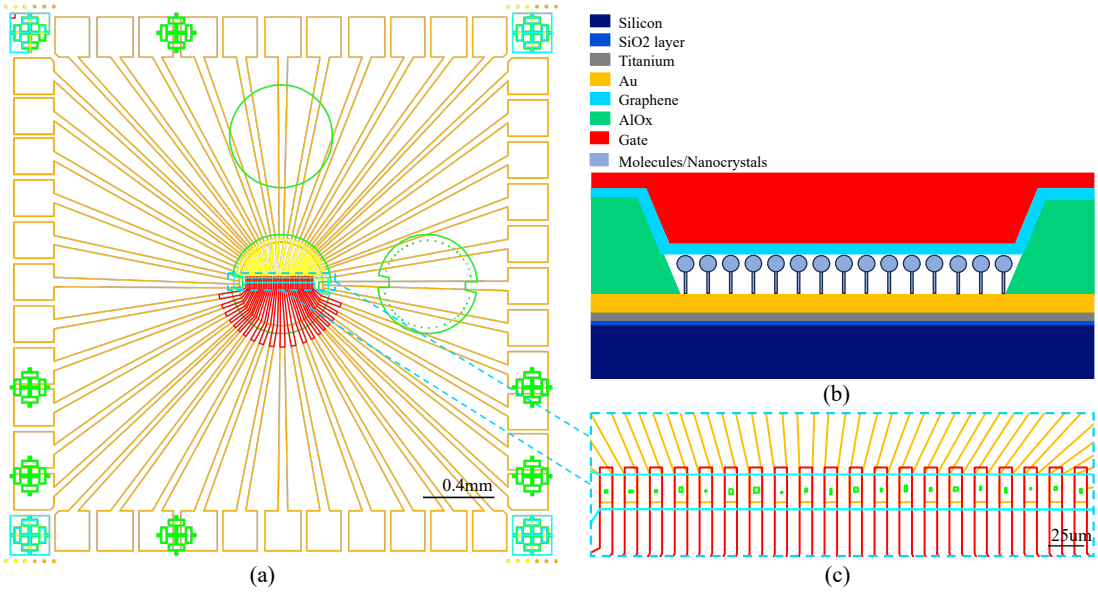


Figure 4: a) Photolithography mask for the microwell chip design, with the gold pattern as the golden shapes and $AlO_x$ windows as the green shapes. b) Schematic of final device layers at a microwell, as at the end of Figure 3. The gate was not used in this project. c) Zoomed-in part of the microwell design. The small green squares represent the $AlO_x$ microwells where the SAMs are.


Figure 5: Pictures from the cleanroom. a) The plasma etcher used to etch graphene. b) Microscope and chemical fume hood in the UV-sensitive room. All steps with the photoresist in Figure 3 were carried out in this room. c) Chemical fume hood outside the UV-sensitive room.

## 2.2 Experimental Data Collection

To measure the I-V curves a mix of DC and AC voltage was used, as shown in Figure 6a. This configuration allows for accurate current and conductance data to be gathered respectively, with DC amplitude being much larger than AC amplitude. Measurements were taken both at room temperature ($\sim$ 300K) and liquid nitrogen temperature (77K) and these studies were repeated many times to check the reproducibility.

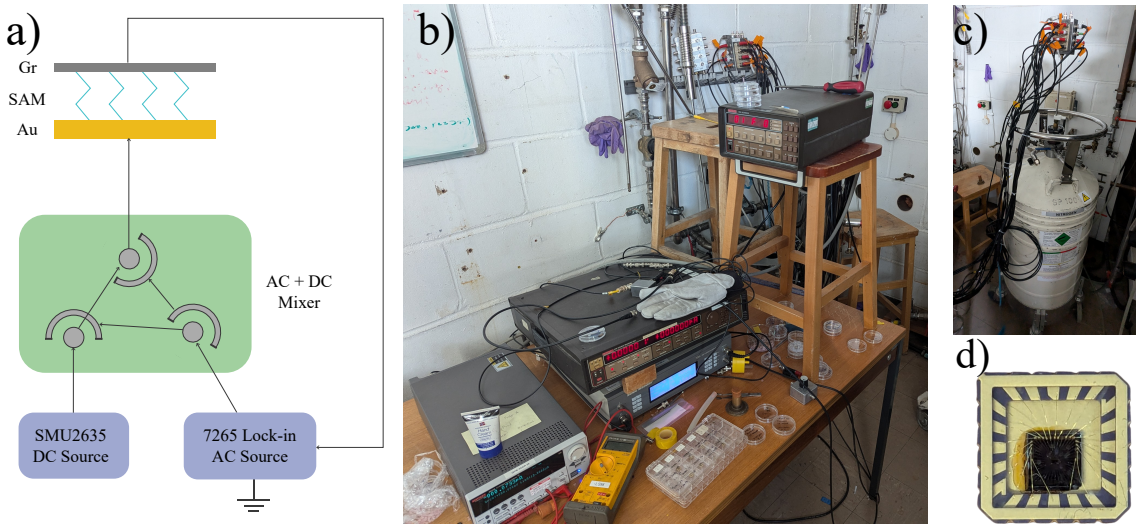


Figure 6: a) Circuit schematic of setup, showing the AC and DC sources. Signals from the SMU and Lock-in Amplifier are mixed to create an AC+DC signal. b) Photo of experimental setup in the lab. SMU (bottom-left), Lock-in Amplifier (bottom-center). c) LED Probe inside Nitrogen Dewar. Probe has a BNC box from which BNC cables are attached so that connection from device to rest of circuit can be made. d) Chip inside a leadless chip carrier (LCC) package. We need one junction which is bonded to the package to act as the ground contact in the circuit.

The data was collected using an existing program written in LabVIEW which can communicate with the instruments, and collects electrical data. To collect the data across an entire voltage range, the bias voltage (sweep voltage) across the junction is swept up to a maximum value, down to a minimum value, and then back to zero. The measurable junctions were bonded as in Figure 6d, with at least one junction to act as the ground. Each dataset involved measuring each junction up to around 1-1.5V. Examples of some of these I-V curves can be seen in the blue lines in Figure

9. This data is saved and then I analysed it with Python (more on this in Section 2.3.1).

## 2.3 Data Analysis

### 2.3.1 Simmons Model for Quantum Tunnelling

The Simmons model describes the mechanism of charge transport across a tunnelling barrier and is given by the following equation (Equation 1 in S18 [2]):

$$J = \frac{q}{4\pi^2\hbar d^2}\left(\left(\Phi_B - \frac{\alpha}{2}V\right)e^{-\frac{2d}{\hbar}\sqrt{2m^*e(\Phi_B-\frac{\alpha}{2}V)}} - \left(\Phi_B + \frac{1-\alpha}{2}V\right)e^{-\frac{2d}{\hbar}\sqrt{2m^*e(\Phi_B+\frac{1-\alpha}{2}V)}}\right) \quad (1)$$

where $q$ is the electronic charge tunnelling through, $\Phi_B$ is the barrier height, $d$ is the barrier length, $V$ is the junction bias voltage, $m^*$ is the effective electron mass, and the $\alpha$ parameter accounts for asymmetry in the relationship. In the molecular junctions, the energy levels in the molecule (i.e. HOMO and LUMO) determine the barrier height, and the size of the molecules determine the barrier length.

### 2.3.2 Quantum Dots and Single Electron Tunneling

The obtained results are in many cases inherent to that of a quantum dot. Quantum dots (QDs) have evenly spaced energy levels [3], which leads to such behaviour as Coulomb staircases (see Figure 7). In quantum dots, only one electron can tunnel through at a time before the next can enter the junction. This leads to staircase behaviour: with periods where current does not change much with bias voltage increase, and periods with a sharp increase when a new energy level is activated (such as the energy levels in Figure 1d). The staircase behaviour is usually only seen at lower temperatures, as the thermal energy is then less than the energy separation between energy levels in the QD.

The molecules themselves however should not have evenly spaced energy levels, but despite this molecular junctions with no added nanoparticles can display this behaviour, as shown in Figure 7. The current theory behind this is that Au nanoparticles are ripped from the Au layer by the molecules [4], to then act as QDs and give these staircase behaviours.

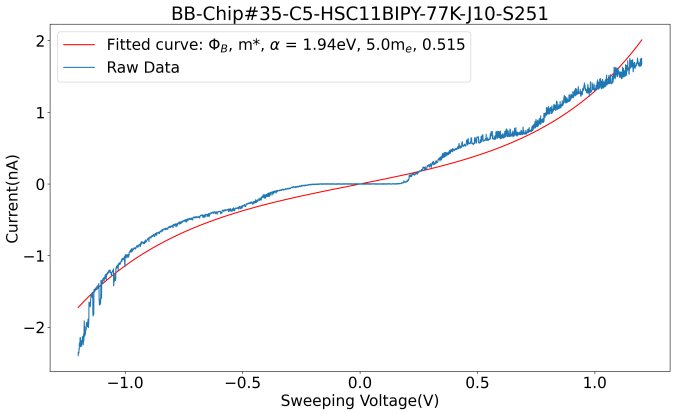


Figure 7: An example of the electric behaviour of a particular junction (in this case it is Junction 10 on Chip#35 C5), showing a Coulomb staircase with clear steps. The Simmons model is not able to account for this behaviour, as shown with the disparity between the fit and the data.

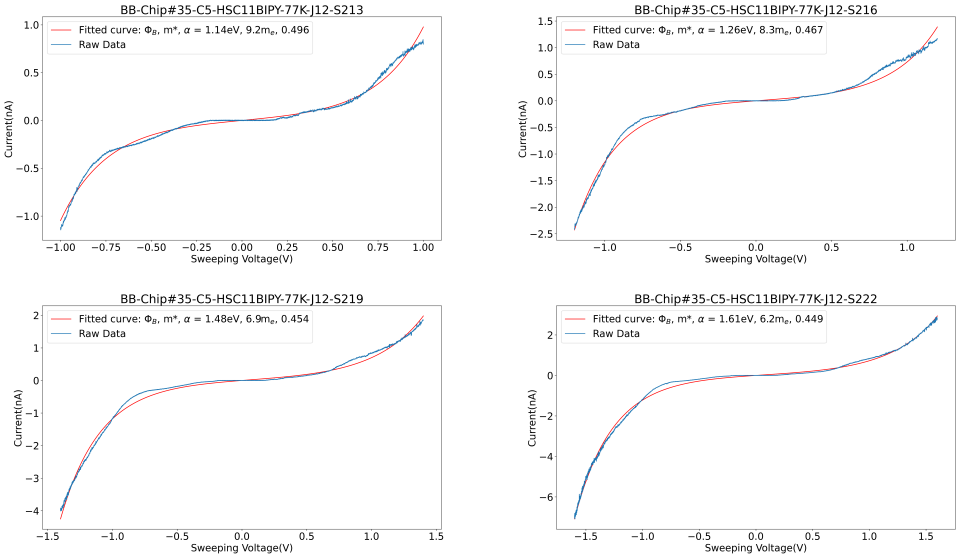


Figure 8: Consecutive sweeps on the same junction in the same chip at the same temperature, in chronological order. The maximum voltage increases by 0.2V with each consecutive sweep. As more sweeps are taken, the staircase behaviour can be seen to slowly disappear.

In some junctions the staircase behaviour was seen to change as more sweeps were taken (Figure 8), likely due to high electric fields across the junction meaning the Au nanoparticles moved within it. If the nanoparticles reach either the Au layer again or the graphene layer, they will no longer exhibit QD behaviour since they will become part of the conductive layer. Each junction contained approximately $10^6$ molecules, meaning its overall behavior was governed by the collective average of all these molecules and nanoparticles. The motion of just a few molecules would only slightly alter the junction's properties, allowing for a wide range of electrical behaviors to emerge. The staircase behaviour was seen in roughly 15% of the junctions I measured and the Simmons model was inadequate to describe these curves, so they are excluded from the final parameter extraction.

### 2.3.3 Tunnelling Results described by the Simmons model

I developed three Python programs to extract parameters from sweep data, and use the status data (data about the status of each sweep) to sort this for each junction in order to better understand the resultant distribution of parameters. To achieve this, I employed the use of many Python libraries, most notably `pandas`, `numpy`, `scipy.optimize`, `matplotlib.pyplot`, `csv` and `os`. More information about the format of the created code can be found in Appendices A.1-A.3).

Figure 9 is a collection of various raw data together with fits which show a good agreement with the Simmons model. These are just a few examples from over 100 different sweeps that were fitted, with the extracted parameters used for the parameter plotting in Figures 10 and 11.

Figure 10 emphasizes that there is a correlation between the $\Phi_B$ and $m^*$ parameters. The proportionality constant between $\Phi_B$ and $1/m^*$ is approximately $\Phi_B m^* \approx 11 \pm 3$. This suggests that the product $\Phi_B m^*$ can be used to avoid the issue of over-fitting the data. This is clear from Equation 1, since the exponentials dominate in determining the shape of the fit, and so $\Phi_B m^*$ is the appropriate variable to consider.

Figure 11 outlines the spread of each of the three parameters. From the datasets: $\Phi_B$ was found to be between 0.6 and 2 eV, $m^*$ was between 4 and 15, and $\alpha$ was approximately between 0.45 and 0.55. The aforementioned relation between $\Phi_B$ and $m^*$ can be clearly seen in the first two columns, with smaller $\Phi_B$ corresponding to larger $m^*$ and vice versa. This means that exact values for these two parameters are unclear. Furthermore, the variation in distributions for different datasets means the bounds mentioned above are only approximate.

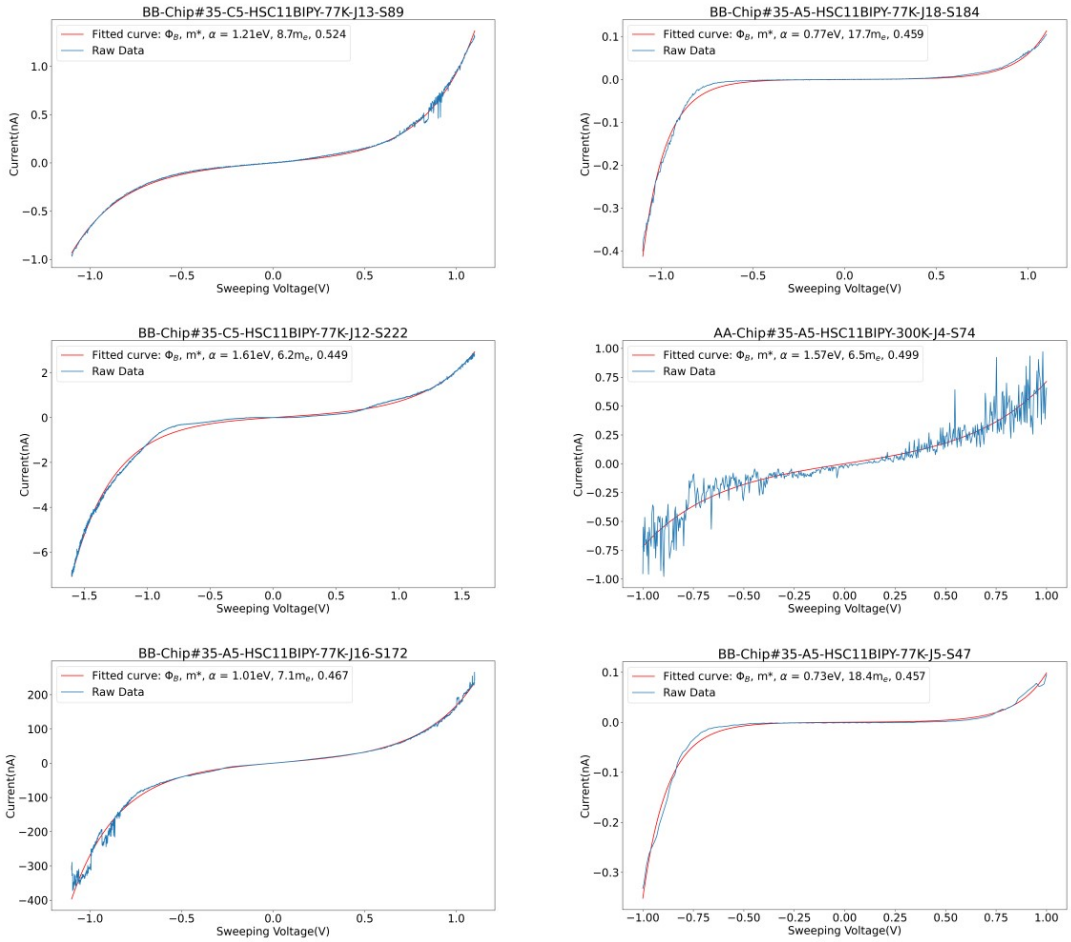


Figure 9: Examples of I−V curves (blue solid curves) and fits to the Simmons model (red solid line) for different junctions at different temperatures. These display behaviour described well by the Simmons model (Equation 1). The extracted parameters are provided in the legend. The title of each subplot contains the dataset prefix, chip number, SAM name, temperature, junction number, and sweep number, in that order. I analysed two different chips (Chip#35 C5 and Chip#35 A5) at both 77K (dataset BB) and room temperature (dataset AA). This title naming scheme is also present in Figures 7 and 8.

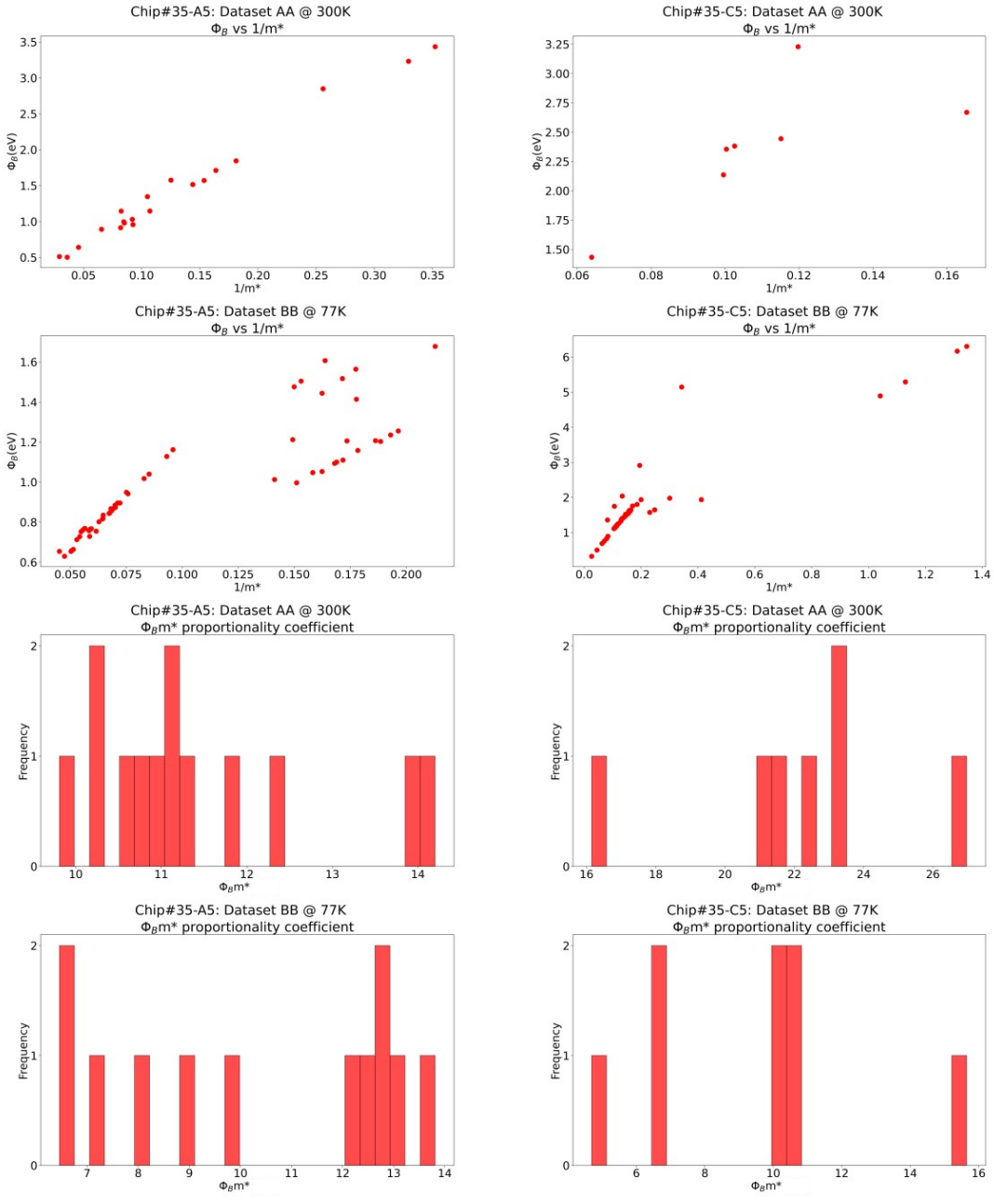


Figure 10: Scatter plots of $\Phi_B$ vs $1/m^*$ (top) and histograms of $\Phi_B m^*$ per junction (bottom). There is an unmistakeable positive correlation between these two parameters, suggesting that the curve fit is over-fitting the data. The histograms display the proportionality constant between these parameters and, apart from that for Chip#35 C5 Dataset AA, show an approximate value of $\Phi_B m^* = 11 \pm 3$.

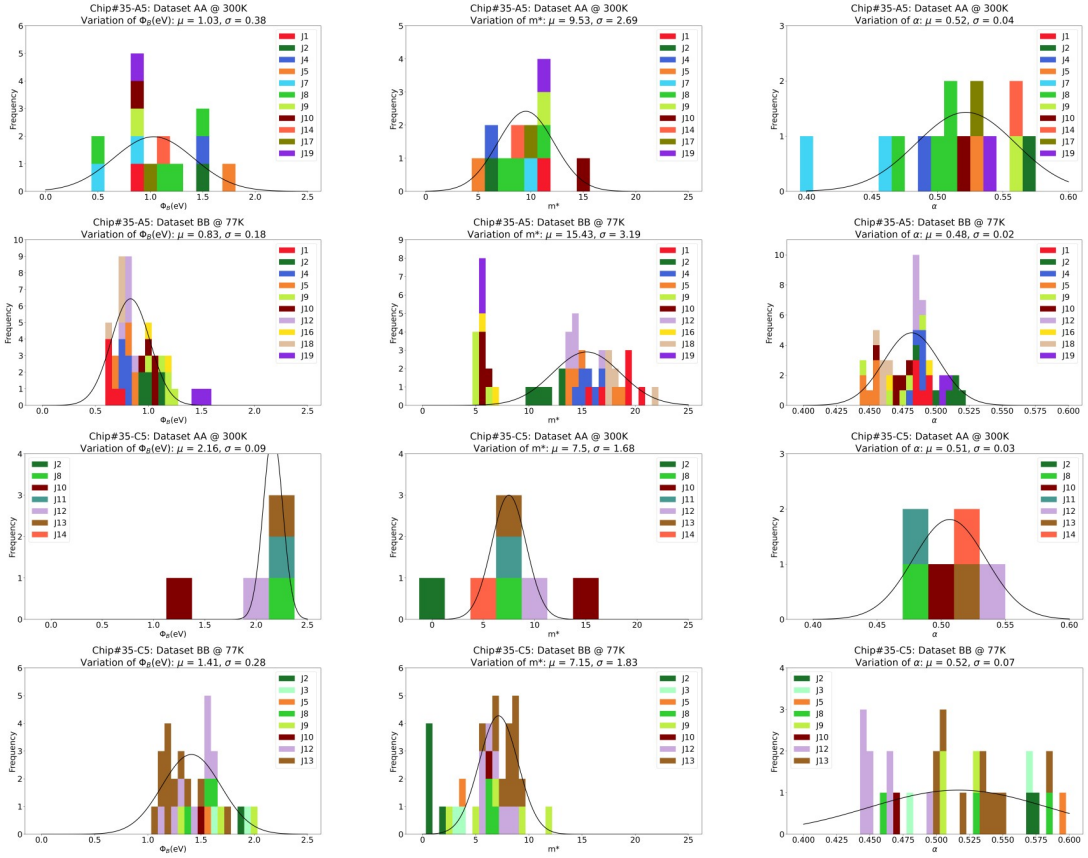


Figure 11: Histograms of each of the three extracted parameters, for four datasets, and split per junction. The mean and standard deviation of the data are given in the plot title, and the junction numbers are given in the legend. The room temperature measurements had less data, so the bins are wider. These plots show significant variation in $\Phi_B$ and $m^*$ across datasets, but each dataset can be approximated to a gaussian. Literature had no value for $m^*$, but $\Phi_B$ was given as 1.4eV [5], which agrees very well with the dataset on the last row. The values for $\alpha$ also varied a lot, with points further from 0.5 describing more asymmetric data. The data for Chip#35 A5 at 77K was skewed to have larger currents at negative bias so $\alpha$ is less than 0.5, whereas the opposite is true for the 300K data.

# 3 Conclusion and Future work

In this project I measured and analyzed the I-V characteristics of two different chips with multi-terminal configurations. Around 15% of the junctions measured showed Coulomb staircases and could not be fitted by the Simmons model. This behaviour was attributed to the HSC11BIPY molecules ripping gold nanoparticles out from the gold layer of the device, which then acted as quantum dots.

Among the junctions which showed good agreement with the Simmons model, there was a clear correlation between the $\Phi_B$ and $m^*$ extracted parameters. The correlation coefficient has an approximate value of $\Phi_B m^* = 11 \pm 3$, which suggests a more suitable fitting parameter might be $\Phi_B m^*$. Considering all the parameters independently resulted in roughly Gaussian distributions for all of the junctions on each chip. This gave a value for $\Phi_B$ between 0.6 and 2eV, which is in the range of the literature value of 1.4eV. The values for the effective mass, $m^*$, were between 4 and 15 $m_e$, while $\alpha$ was between 0.45 and 0.55.

Despite the fact that the junctions without a Coulomb staircase present generally had a good fit with the Simmons model, some of the extracted parameters varied with more than two standard deviations between datasets. Moreover, this data was at low voltage, whereas at higher voltages a different tunnelling mechanism takes over, so the Simmons model is mainly useful for finding where the tunnelling mechanism changes.

A significant challenge with this work was the difficulty to produce reproducible results with different chips. Each chip exhibited a distinct distribution compared to other chips, even when at the same temperature. Additionally, each chip's behavior varied further when the temperature changed, and individual junctions within each chip also showed differences. This disparity arises from the SAM growth in the fabrication, and is unavoidable with the current technique.

Further steps could be taken to analyse the correct mechanism of tunnelling through these molecules, and characterise them accordingly. New fitting models can be developed to take into account the contribution from the Coulomb staircases. Additionally, future studies must consider much larger datasets, to get more representative parameterization of these systems.

# 4 Acknowledgements

Firstly, I would like to thank Professor Chris Ford for setting up the project and his guidance, and the Cavendish Laboratory for providing the UROP funding. Special thanks to Bingxin Li for his assistance and supervision throughout the 8 weeks. I acknowledge that Figure 4 was made by Bingxin. Thank you also to Andy Li for collaborating with me during the data collection.

# 5 Personal Reflection

**Research Experience:** During the first few weeks I dedicated time to the understanding the theory behind the project by reading a thesis on the topic by Dr. Shanglong Ning. Throughout the project, I proactively engaged with other group members to aid my understanding. I also gained valuable experience as a research student which consisted of lots of patience and resilience to persevere despite unprecedented challenges.

**Performing Experiments:** Carrying out the experiments and analysing the data helped develop intuition for the electrical behaviour of these molecules. This allowed me to participate and actively engage in discussion about the origin of this behaviour, which was especially rewarding.

**Data Analysis and Management:** I learnt how to use many more Python libraries, and gained a lot of confidence and experience with data analysis in Python. In my Python code, I used a variety of data structures to achieve my aims, as well as experiment with creating and accessing many external files. Another key concept I learned was the importance of highly organised files. My Python programs created folders and files using the Python **os** library. Everything I created was in the same format so that it was straightforward to access the csv files from different programs and find everything efficiently.

# References


[1] Shanglong Ning and Christopher Ford. PhD thesis, University of Cambridge, 2022.

[2] Junwoo Park, Lee Belding, Li Yuan, Maral P. Mousavi, Samuel E. Root, Hyo Jae Yoon, and George M. Whitesides. Rectification in molecular tunneling junctions based on alkanethiolates with bipyridine–metal complexes. *Journal of the American Chemical Society*, 143(4):2156–2163, Jan 2021.

[3] Joel M. Fruhman, Hippolyte P.A.G. Astier, Bruno Ehrler, Marcus L. Böhm, Lissa F. Eyre, Piran R. Kidambi, Ugo Sassi, Domenico De Fazio, Jonathan P. Griffiths, Alexander J. Robson, and et al. High-yield parallel fabrication of quantum-dot monolayer single-electron devices displaying coulomb staircase, contacted by graphene. *Nature Communications*, 12(1), Jul 2021.

[4] Shu Hu, Junyang Huang, Rakesh Arul, Ana Sánchez-Iglesias, Yuling Xiong, Luis M. Liz-Marzán, and Jeremy J. Baumberg. Robust consistent single quantum dot strong coupling in plasmonic nanocavities. *Nature Communications*, 15(1), Aug 2024.

[5] Jiung Jang, Gyu Don Kong, and Hyo Jae Yoon. Electrically stable self-assembled monolayers achieved through repeated surface exchange of molecules. *Accounts of Materials Research*, Aug 2024.

# Appendices

## A Python Code

### A.1 Junction Sweep Association

This program is used to determine which junction the sweeps correspond to. It takes in the status data from a csv and creates a dictionary, which it then writes to a new junction-sweep csv for the next program to use.

```
import pandas as pd
from collections import defaultdict
import os

def extract_junction_sweep():
    '''Function to extract data describing which sweeps are associated with which junction, from .csv.'''
    # Initialise header
    header_names = ['Sweep number', 'Row offset', 'sweep length', 'x column', 'SD type', '<VarA>',
                    'V (V)', 'Current(nA)', 'Common contact no.', 'Contact no.', 'Sweep rate', 'date-2000',
                    'time', 'Compliance (nA)', 'x Scaling F', 'lock-in compliance (nA)', 'SEN1 (nA)',
                    'FRQ (Hz)', 'TC1 (s) ', 'Osc Amp (V)', 'Leakage Current (nA)', 'SEN2 (nA)', 'TC2 (s) ',
                    'Probe Height (cm)','Dipping Direction (in+)', 'Ref. theta1', 'Ref. theta2',
                    'Resistance (Ohm)', 'Conductance (S)', 'G/G0', 'temperature (K)', 'diode voltage (V)',
                    'gate contact no.', 'Gate Voltage (V)', 'gate increment direction', 'Relay #']
    # NOTE: Edit filename and filepath as necessary
    filename = f"Data_vic\\Chip#{chip_name}\\{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K
    ↪  -status_data.csv"
    try:
        df = pd.read_csv(filename, header=None, names=header_names)
    except FileNotFoundError as e:
        print(f"Oops, error found: {e}\nCheck that you put the right chip name ({chip_name}), and that your files
        ↪  are arranged appropriately!\n")
        print(f"Put the status_data.csv file in the folder: Chip#{chip_name} with name format of
        ↪  {data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K-status_data.csv")
        quit() # Restart if file not found
    # Take out columns of interest
    return df['Contact no.'], df['Sweep number']

def association_junc_sweep(contact_nums, sweep_nums):
    '''Creates dictionary of contact numbers and corresponding sweep numbers'''
    # Create a dictionary to group values
    grouped_dict = defaultdict(list)
    # Add key,value pair to dictionary
    for key, value in zip(contact_nums, sweep_nums):
        grouped_dict[key].append(value)
    return grouped_dict

def write_dict_to_csv(dict):
    '''Writes the junction-sweep correspondence to a dedicated .csv file.'''
    global current_dir
    # Prepare the data to be written
    rows = []
    for key, values in dict.items():
    # Format the row as 'key:values' where values are comma-separated
        row = f"J{key},S{';'.join(map(str, values))}\n"
        rows.append(row)
    # Make the folder for it; tidy things up a bit
    folder_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}")
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
```

```
    # Write to CSV in write mode
    with open(f'{folder_path}\\{data_prefix}-Chip#{chip_name}-junction_sweep_correspondence.csv', mode='w',
    ↪  newline='') as file:
        file.writelines(rows) #writes it all in one go (hence why row has \n in it)

# Get the directory where the current script is located
current_dir = os.path.dirname(os.path.abspath(__file__))
# Get chip name and data prefix
chip_name = input("What's the chip name? (e.g. 35-C5) ")
data_prefix = input("What's the data prefix (e.g. AA or BB etc) ")
temperature = int(input("What's the temperature (e.g. 77 or 300K)? "))
# Extract contact and sweep numbers
contact_nums, sweep_nums = extract_junction_sweep()
dictionary_junc_sweeps = association_junc_sweep(contact_nums, sweep_nums)
# Print the result in the desired format
sorted_dict = {key: dictionary_junc_sweeps[key] for key in sorted(dictionary_junc_sweeps)} # to sort the keys
write_dict_to_csv(sorted_dict)
```

## A.2 Curve Fitting Simmons

This program is used to extract the Simmons model parameters, and send this information to an external csv as well as plot fits for the user to distinguish which fits are useful (e.g. Figures 7-9). It uses csv files containing sweep data to fit the model for each sweep, and then the junction-sweep csv to sort these parameters according to the junction they correspond to.

```
import pandas as pd # For creating dataframe
import matplotlib.pyplot as plt # For data representation/graph plotting
import numpy as np
from scipy.optimize import curve_fit # For curve fitting
import scipy.constants as cnst # For scientific constants
import csv # For writing extracted parameters to csv
import os

# Plot aesthetics :)
import matplotlib as mpl # For font size
mpl.rcParams.update({'font.size': 22})  # Adjust the font size as needed

###############~~~~~ C O D E ~~~~~###############
def string_nums_convert_to_range(input_range):
    '''Returns range (e.g "1-6,8") as a list of every number (as integers) implicated in that range.'''
    list_full = []
    # Split input first by commas
    range_segments = input_range.split(',')
    for segment in range_segments:
        # Check if the segment is a range (i.e. 1-5)
        if '-' in segment:
            start_range, end_range = segment.split('-')
            list_full.extend(range(int(start_range), int(end_range)+1))
        else:
            list_full.append(int(segment))
    return list_full

def extract_sweep_data(sweep_num):
    '''Function to extract sweep voltage and current data from .csv. Takes in sweep_num to know which datafile to
    ↪  read.'''
    # Initialise header
    header_names = ['<ChA>', 'Current (nA)', 'Time (s)', 'Voltage (V)', 'MAG1 (nA)', 'MAG2 (nA)', 'X1 (nA)', 'X2
    ↪  (nA)', 'Y1 (nA)', 'Y2 (nA)', 'Theta1', 'Theta2', 'Leakage Current (nA)', 'Gate Voltage (V)']
    # NOTE: Edit filename as necessary
```

```
    datafile_name = f'Data_vic/Chip#{chip_name}/CSV-data_files/{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K-S{sweep_num}.csv'
    try:
        # Try to create the dataframe from the csv file which needs to be created beforehand.
        df = pd.read_csv(datafile_name, header=None, names=header_names)
    except FileNotFoundError:
        print(f'File not found at\n Chip#{chip_name}/CSV-data_files/{data_prefix}-Chip#{chip_name}-HSC11BIPY {temperature}K-S{sweep_num}.csv')
        print("Make sure this file path exists!")
    # Take out columns of interest
    df = df.replace([np.inf, -np.inf], np.nan)
    df = df.dropna()
    return df['<ChA>'], df['Current (nA)']

# Curve fitting equation
def Simmons_equation_alpha(V, Phi_B=5, m=1, alph=0.5):
    '''Function for the Simmons equation of tunnelling, accouting for asymmetry with the alpha term.'''
    # Defining equation constants
    q, e, h_bar, d, m_e = cnst.e, cnst.e, cnst.hbar, 1e-9, cnst.m_e
    return q*e/(4*np.pi**2*h_bar*d**2) * (((Phi_B-1/2*V*alph)*(np.exp(-2*d/h_bar*np.sqrt(2*m*m_e*e * Phi_B-1/2*V*alph)))) - ((Phi_B+1/2*V*(1-alph))*(np.exp(-2*d/h_bar*np.sqrt(2*m*m_e*e * Phi_B+1/2*V*(1-alph))))))
    # junction_area*1e7

def extract_parameters(sweep_voltage, current):
    '''Function to extract parameters from Simmons equation.'''
    return curve_fit(Simmons_equation_alpha, np.array(sweep_voltage), current, bounds = ([0, 0, 0], [np.inf, np.inf, 1]), maxfev=5000)

def make_figures_save(sweep_voltage, current, params, sweep_num):
    '''Function to save figures in a new folder for sanity check.'''
    # Define a folder relative to the current script directory
    folder_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}/Figure_Fits/J{junction_num}")
    # If folder doesn't exist yet, create it
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
    ## Create a figure
    plt.figure(figsize=(16,9)) # create a new figure that is full-screen size
    plt.plot(sweep_voltage, Simmons_equation_alpha(sweep_voltage, *params), label=f'Fitted curve: $\\Phi_B$, m*, $\\alpha$ = {round(params[0],2)}eV, {round(params[1],1)}m$_e$, {round(params[2],3)}', color='red') # plot of fit
    plt.plot(sweep_voltage, current, label=f'Raw Data') # plot of raw data
    plt.title(f"{data_prefix}-Chip#{chip_name}-HSC11BIPY {temperature}K-J{junction_num}-S{sweep_num}")
    plt.ylabel("Current(nA)")
    plt.xlabel("Sweeping Voltage(V)")
    plt.legend()
    # Define the full file path, including the folder and file name
    file_path = os.path.join(folder_path, f"{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K-J{junction_num}-S{sweep_num}.png")
    # Save the figure to the specified folder
    plt.savefig(file_path, dpi=300)
    # Close the figure so a new one can be created next time
    plt.close()

def sweep_size_determiner(sweep_voltage):
    '''Function to determine whether the sweep is purely positive, purely negative, or both.'''
    # Find the minimum and maximum voltage of the sweep
    voltage_max, voltage_min = max(sweep_voltage), min(sweep_voltage)
    # Determine the type of sweep
    if voltage_min == 0: # (this still includes sweeps that DNF)
        sweep_type = 'positive'
    elif voltage_max == 0: # if voltage max = 0
        sweep_type = 'negative'
    elif voltage_max == -voltage_min:
        if voltage_max < 0.4: # if the sweep is small
            sweep_type = 'small' # small sweep
```

```
            else:
                sweep_type = 'both' # both/long/middle sweeps. Refers to the ones that sweep the entire range
        else:
            sweep_type = 'DNF' # DNF = Did Not Finish
        return sweep_type

    def write_parameters_to_csv(params, sweep_type):
        '''Writes the extracted parameters to a dedicated .csv file.'''
        # Make the folder for it; tidy things up a bit
        folder_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}/Extracted_Parameters")
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

        # Send to csv
        row_for_csv = [sweep_num, *np.round(params, 3), sweep_type]
        # Open file in append mode ('a') and write new rows in each iteration
        # Write to junction-specific CSV, but also to chip-specific csv
        with open(f'{folder_path}/J{junction_num}-extracted_parameters.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(row_for_csv)

        with open(f'{folder_path}/extracted_parameters.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(row_for_csv)

    def write_excluded_sweeps_to_csv(sweeps_to_exclude_range):
        '''Writes the excluded sweeps for a particular junction to a dedicated .csv file.'''
        # Make the folder for it; tidy things up a bit
        folder_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}/Excluded_Sweeps")
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)
        # Row entry for csv
        entry_for_csv = [f"J{junction_num}",f"S{sweeps_to_exclude_range}"]
        # Open file in append mode ('a') and write new rows in each iteration
        with open(f'{folder_path}/J{junction_num}-excluded_sweeps.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(entry_for_csv)
        # Write to non-junction-specific CSV as well
        with open(f'{folder_path}/excluded_sweeps.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(entry_for_csv)

    def main(sweep_num):
        '''Main function to run all other functions in this file.'''
        sweep_voltage, current = extract_sweep_data(sweep_num)
        # Extract parameters:
        params, _ = extract_parameters(sweep_voltage, current)
        print(f'S{sweep_num}', params) #sanity check
        # Save figures in other folder for easy access and separation
        sweep_type = sweep_size_determiner(sweep_voltage)
        # Select only long sweeps to plot
        if sweep_type == "both":
            make_figures_save(sweep_voltage, current, params, sweep_num)
        # Write the extracted parameters to a csv file
        write_parameters_to_csv(params, sweep_type)

    def print_all_junction_options():
        '''Print all junction options to choose from.'''
        file_path = f"{current_dir}/Chip#{chip_name}-{data_prefix}/
        ↪  {data_prefix}-Chip#{chip_name}-junction_sweep_correspondence.csv"
        df = pd.read_csv(file_path, header=None)
        junctions_df = df[0]
        junctions_list = junctions_df.tolist()
        print("Junctions available to choose from are: ")
        for junction in junctions_list:
            print(f"{junction}", end="\t")
```

```
    print(end="\n")

def process_junction_data_from_csv(junction_num):
    '''Read the CSV file into a DataFrame (without predefined column names)'''
    global junc_sweep_dict
    # Filepath of the sweep-junction data
    file_path = f"{current_dir}/Chip#{chip_name}-{data_prefix}/{data_prefix}-Chip#{chip_name}-junction_sweep_correspondence.csv"
    # Convert file to a dataframe
    df = pd.read_csv(file_path, header=None)
    # Convert to dictionary
    junc_sweep_dict = pd.Series(df[1].values, index=df[0]).to_dict()
    # Extract sweeps corresponding to correct junction
    sweeps_to_analyze_string = junc_sweep_dict[f"J{junction_num}"]
    # Converts sweep numbers from string to integer, and removes 'S' from sweeps_to_analyze_string
    sweeps_to_analyze_list = [int(sweep_num) for sweep_num in sweeps_to_analyze_string[1:].split(";")]
    print(f"Sweeps corresponding to junction {junction_num} are: {sweeps_to_analyze_list}")
    return sweeps_to_analyze_list

def initialise_constants():
    '''Initialise the variables which are constant for the whole duration of the program.'''
    #NOTE: Here is where to change values for different datasets
    global current_dir, chip_name, data_prefix, junction_areas, temperature
    # Get the directory where the current script is located
    current_dir = os.path.dirname(os.path.abspath(__file__))
    # Get chip name and data prefix
    chip_name = input("What's the chip name? (e.g. 35-C5 or 102-D4 etc) ")
    data_prefix = input("What's the data prefix (e.g. AA or BB etc) ")
    temperature = input("What's the temperature (e.g. 77 or 300K)? ")

########################################
## CODE to run through all relevant sweeps (specified by the user)
initialise_constants()

def how_many_junctions():
    '''Function to determine how many junctions should be analysed.'''
    global junc_sweep_dict, junction_nums
    # Filepath of the sweep-junction data
    file_path = f"{current_dir}/Chip#{chip_name}-{data_prefix}/{data_prefix}-Chip#{chip_name}-junction_sweep_correspondence.csv"
    # Convert file to a dataframe
    df = pd.read_csv(file_path, header=None)
    # Convert to dictionary
    junc_sweep_dict = pd.Series(df[1].values, index=df[0]).to_dict()
    junctions = list(junc_sweep_dict.keys())
    numeric_parts = [int(junc[1:]) for junc in junctions]
    print(numeric_parts)
    how_many_junc = int(input("Would you like to only look at one junction [1], some junctions[2], or all junctions[3]? "))
    if how_many_junc == 1:
        print("You've chosen just one junction!")
        junction_num = int(input("Which junction would you like to include? "))
        junction_nums = [junction_num]
    elif how_many_junc == 2:
        junctions_range_string = input("Which junctions would you like to include? Put them in the form 1,5-8,9 .etc\n")#"2,3,5,8,9,10,12,13"#"1,2,3,4,5,6,7,8,9,10,12,14,16,17,19"#"3,5,8,9,10,11,12,13,14"
        junction_nums = string_nums_convert_to_range(junctions_range_string)
    elif how_many_junc == 3:
        junction_nums = numeric_parts # All junctions
    else:
        print("Pick 1, 2 or 3 please. Restart the program.")
        quit()
    junction_nums.sort()

# Decide which junctions
how_many_junctions()
```

```
# Analyse each junction
for junction_num in junction_nums:
    # Sweep range for each junction
    sweep_range = process_junction_data_from_csv(junction_num)
    # Analyzes dataset for each sweep
    for sweep_num in sweep_range:
        try:
            main(sweep_num)
        except (ValueError, RuntimeError) as e:
            print(f"Skipping sweep_num {sweep_num} due to: {e}")
            continue
    # Write the excluded sweeps to a csv to automatically be taken into account by the next program (parameter_plotting-XXXXX.py)
    figures_folder_path, satisfied_range = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}\\Figure_Fits\\J{junction_num}"), False
    print(f"\nCheck the images I've just put in {figures_folder_path}, and let me know which sweeps to exclude in further analysis!")
    sweeps_to_exclude_range_commas = input("Would you like to exclude any sweeps from this junction? If so, put them in the form 1-5,8,9-11 .etc \tDefault is none.\n")
    sweeps_to_exclude_range = sweeps_to_exclude_range_commas.replace(",",";")
    # Write the excluded sweeps to a csv. If none, write 0
    if sweeps_to_exclude_range != "":
        write_excluded_sweeps_to_csv(sweeps_to_exclude_range)
    else:
        write_excluded_sweeps_to_csv("0")
```

## A.3 Parameter Plotting

This program is used to plot the extracted parameters: both in relation to one another and as individual histograms for each parameter. These plots can be seen in Figure 11.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.optimize import curve_fit

# Aesthetics :)
import matplotlib as mpl # For font size
mpl.rcParams.update({'font.size': 20})  # Adjust the font size as needed
from matplotlib.ticker import MaxNLocator # For integer y-axis ticks

###############~~~~~ C O D E ~~~~###############
def how_many_junctions():
    '''Function to determine whether only the long sweep or all sweeps should be analysed.'''
    # List all files in the folder
    files = os.listdir(folder_path)
    # Extract the numeric parts from filenames
    numeric_parts = [int(f.split('-')[0][1:]) for f in files if f.endswith('.csv') and f.startswith('J')]
    print(f"Available junctions to analyze are: {numeric_parts}")
    # User input
    how_many_junc = int(input("Would you like to only look at one junction [1], some junctions[2], or all junctions[3]? "))
    if how_many_junc == 1:
        print("You've chosen just one junction!")
        junction_num = int(input("Which junction would you like to include? "))
        junction_nums = [junction_num]
    elif how_many_junc == 2:
        junctions_range_string = input("Which junctions would you like to include? Put them in the form 1,5-8,9 .etc\n")
        junction_nums = string_nums_convert_to_range(junctions_range_string)
```

```
    elif how_many_junc == 3:
        junction_nums = numeric_parts# all junctions
    else:
        print("Pick 1, 2 or 3 please. Restart the program.")
        quit()
    junction_nums.sort()
    return junction_nums

def create_dict_of_sweeps_to_exclude_from_csv():
    '''Create dictionary of the sweeps to exclude from the appropriate csv file'''
    global junc_sweep_exclude_dict
    # Filepath of the sweep-junction data
    file_path = f"{current_dir}/Chip#{chip_name}-{data_prefix}/Excluded_Sweeps/excluded_sweeps.csv"
    # Convert file to a dataframe
    df = pd.read_csv(file_path, header=None)
    # Convert to dictionary, with junction numbers being the keys and the corresponding sweep numbers being the values
    junc_sweep_exclude_dict = pd.Series(df[1].values, index=df[0]).to_dict()
    ## This way of seeing which sweeps to exclude means the dictionary is not created each time, and so saves time.

def extract_sweeps_to_exclude_from_dict(junction_num):
    '''Selecting junction from dictionary, and extracting the sweeps to exclude from this.'''
    # Extract sweeps corresponding to correct junction
    sweeps_to_exclude_string = junc_sweep_exclude_dict[f"J{junction_num}"]
    # Convert string to range
    sweeps_to_exclude = string_nums_convert_to_range(sweeps_to_exclude_string[1:])
    # Converts sweep numbers from string to integer, and removes 'S' from sweeps_to_analyze_string
    print(f"Sweeps to exclude corresponding to this junction are: {sweeps_to_exclude}")
    return sweeps_to_exclude

def extract_param_data(csv_file, junction_num):
    '''Function to extract parameter data from .csv'''
    # Initialise header
    header_names = ['SweepNum','Phi_B','m*','alpha','SweepType']
    # Read CSV into DataFrame
    df = pd.read_csv(csv_file, header=None, names=header_names)
    df_unique = df.drop_duplicates()
    # Sweeps to exclude:
    try:
        sweeps_to_exclude = extract_sweeps_to_exclude_from_dict(junction_num)
        df_filtered = df_unique[~df_unique['SweepNum'].isin(sweeps_to_exclude)] # Without sweeps to exclude
        # df_excluded = df_unique[df_unique['SweepNum'].isin(sweeps_to_exclude)]  # With sweeps to exclude (here for debugging purposes)
    except KeyError as e:
        print(f"{e}-excluded_sweeps.csv not found.")
        df_filtered = df_unique
        pass
    # Print the parameter being analysed
    print(f'Analysing param \"{parameters[i]}\", at junction {junction_num}...')
    # Separate sweeps in accordance with SweepType
    both_df = df_filtered[df_filtered['SweepType'] == 'both']
    # Convert the DataFrames to arrays (list of lists)
    both_array = np.array(both_df.values.tolist())
    return both_array.T # Swap rows and columns

def string_nums_convert_to_range(input_range):
    '''Returns range string(e.g "1-6,8") as a list of every number implicated in that range (e.g. [1,2,3,4,5,6,8]).'''
    list_full = []
    # Split input first by commas
    input_range_commas = input_range.replace(";",",")
    range_segments = input_range_commas.split(',')
    for segment in range_segments:
        # Check if the segment is a range (i.e. 1-5)
        if '-' in segment:
            start_range, end_range = segment.split('-')
```

```
            # print(f"start_range: {start_range}, end_range: {end_range}")
            list_full.extend(range(int(start_range), int(end_range)+1))
        else:
            list_full.append(int(segment))
    return list_full

def make_figures_save_multiple_junc(junction_nums):
    '''Function to save figures in a new folder for sanity check.'''
    # Define a folder relative to the current script directory
    folder_figures_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}/Parameter_Plots")
    # If folder doesn't exist yet, create it
    if not os.path.exists(folder_figures_path):
        os.makedirs(folder_figures_path)
    ## Create a figure
    # Initialise bins
    num_bins = 21
    preset_range = [(0,2.5),(0,25),(0.4,0.6)]
    xlimits = preset_range
    bin_width = (preset_range[i][1]-preset_range[i][0])/(num_bins-1)
    print(f"Bin width = {bin_width}")

    # Histogram plots
    sum_hists = np.zeros(num_bins-1)
    plt.figure(figsize=(16,9)) # Create a new figure that is big enough to see titles
    for junction_num in junction_nums:
        datafile_name = f'{folder_path}/J{junction_num}-extracted_parameters.csv'
        try: # Try to extract the data
            both_array = extract_param_data(datafile_name, junction_num)
        except ValueError as e: # If there's an error, print it and continue to the next junction
            print(f"ValueError: {e}")
            continue
        # Define the number of bins
        bins = np.linspace(*preset_range[i], num_bins)
        try: # Try to extract the data
            int_both_array = both_array[i+1].astype(float)
        except IndexError as e: # If there's an error, print it and continue to the next junction
            print(f"IndexError: {e}")
            print(f"Size of both_array: {both_array.shape}")
            continue
        # Calculate the histogram data for each dataset
        hist_data, _ = np.histogram(int_both_array, bins=bins, weights=None)
        # Stack the histograms
        plt.bar(bins[:-1], hist_data, width=np.diff(bins), bottom=sum_hists, color=colors_dict[junction_num],
        ↪  label=f'J{junction_num}')
        sum_hists = sum_hists + hist_data
    # Print histogram data for debugging
    print(f"sum_hists after junction {junction_num} is: {sum_hists}")
    x_points = np.arange(preset_range[i][0], preset_range[i][1], bin_width)
    popt = fit_gaussian(sum_hists, x_points, xlimits[i])
    # Aesthetics
    plt.xlim(None)#list(xlimits[i]))
    plt.ylim([None,max(sum_hists)+1])
    plt.ylabel("Frequency")
    plt.xlabel(parameter)
    plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True)) # Make y-axis have integer ticks
    plt.legend()#loc='lower right')
    plt.title(f"Variation of {parameter}: $\mu$ = {round(popt[0],2)}, $\sigma$ = {round(np.sqrt(popt[1]),2)}")
    plt.suptitle(f"Chip#{chip_name}: Dataset {data_prefix} @ {temperature}K") # Over-arching title for the whole
    ↪  figure
    # Define the full file path, including the folder and file name
    plot_file_path = os.path.join(folder_figures_path, f"{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K
    ↪  -J{[j_num for j_num in junction_nums]}-parameter{i+1}.png")
    # Save the figure to the specified folder
    plt.savefig(plot_file_path, dpi=300)
    # Close the figure so a new one can be created next time
    plt.close()
```

```
    print("Next parameter")

def plot_phiB_vs_m():
    '''Plot Phi_B vs m*'''
    # Extract phiB and m* columns from dataframe.
    csv_file = f'Chip#{chip_name}-{data_prefix}/Extracted_Parameters/extracted_parameters.csv'
    # Initialise header
    header_names = ['SweepNum','Phi_B','m*','alpha','SweepType']
    # Read CSV into DataFrame
    df = pd.read_csv(csv_file, header=None, names=header_names)
    df_unique = df.drop_duplicates()
    # Separate out sweeps_to_exclude
    sweeps_to_exclude = list(junc_sweep_exclude_dict.values())
    df_filtered = df_unique[~df_unique['SweepNum'].isin(sweeps_to_exclude)] # Without sweeps to exclude
    # Separate sweeps in accordance with SweepType
    both_df = df_filtered[df_filtered['SweepType'] == 'both']
    # Take out relevant columns
    sweep_nums, phiB_data, effm_data = np.array(both_df['SweepNum'].tolist()), np.array(both_df['Phi_B'].tolist()),
    ↪  np.array(both_df['m*'].tolist())
    # # Filter data to remove outliers
    # Define a condition to ignore the points with _B > 10
    condition = (phiB_data <= 10)
    # Filter the data
    effm_filtered = effm_data[condition]
    phiB_filtered = phiB_data[condition]
    sweep_nums_filtered = sweep_nums[condition]
    # Plot
    mpl.rcParams.update({'font.size': 24})
    plt.figure(figsize=(16,9)) # Create a new figure that is full-screen size
    plt.scatter(1/effm_filtered, phiB_filtered, s=100, color='red') # Plot data
    plt.title("$\\Phi_B$ vs 1/m*")
    plt.suptitle(f"Chip#{chip_name}: Dataset {data_prefix} @ {temperature}K")
    plt.ylabel("$\\Phi_B$(eV)")
    plt.xlabel("1/m*")
    ####  SAVE FIG
    # Define a folder relative to the current script directory
    folder_figures_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}/Parameter_Plots")
    # If folder doesn't exist yet, create it
    if not os.path.exists(folder_figures_path):
        os.makedirs(folder_figures_path)
    # Define the full file path, including the folder and file name
    plot_file_path = os.path.join(folder_figures_path,
    ↪  f"{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K-PhiB-vs-m_eff.png")
    # Save the figure to the specified folder
    plt.savefig(plot_file_path, dpi=300)
    plt.close()
    return sweep_nums_filtered, phiB_filtered, effm_filtered

def average_phiB_m_proportionality_coeff(sweep_nums, phiB, m_eff):
    '''Average the proportionality coefficient between Phi_B and m* per junction'''
    # Create an array of junction numbers corresponding to the sweeps
    junction_sweep_nums = np.array([sweep_junc_dict[sweep_num] for sweep_num in sweep_nums])
    # Create an array to store the average Phi_B m* for each junction
    array_phiB_m_eff = []
    # Run through each junction number
    for junction_num in junction_nums:
        # Find the indices of the sweeps corresponding to the junction number
        indices_to_average = np.where(junction_sweep_nums == junction_num)[0]
        if len(indices_to_average) == 0: # If there are no sweeps for this junction
            # print(f"Junction {junction_num} has no data to average.") # Debugging purposes
            continue
        relevant_phiB = phiB[indices_to_average] # Extract the relevant data
        relevant_m_eff = m_eff[indices_to_average]
        relevant_phiB_m_eff = relevant_phiB * relevant_m_eff
        average_phiB_m_eff = np.mean(relevant_phiB_m_eff) # Average the data (spread is ignored)
        # print(f"Average Phi_B*m* for junction {junction_num} is: {average_phiB_m_eff}")
```

```
            array_phiB_m_eff.append(average_phiB_m_eff) # Append value to the array, to be plotted
        plot_phiB_m_proportionality_coeff(array_phiB_m_eff)

    def plot_phiB_m_proportionality_coeff(average_phiB_m_eff):
        '''Plot the average (per junction) proportionality coefficient between Phi_B and m* in histograms'''
        # Plot
        mpl.rcParams.update({'font.size': 24})
        plt.figure(figsize=(16,9)) # Create a new figure that is full-screen size
        plt.hist(average_phiB_m_eff, bins=25, color='red', edgecolor='black', alpha=.7) # Plot data
        plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True))
        plt.title("$\\Phi_B$m* proportionality coefficient")
        plt.suptitle(f"Chip#{chip_name}: Dataset {data_prefix} @ {temperature}K")
        plt.ylabel("Frequency")
        plt.xlabel("$\\Phi_B$m* (eV)")
        ####  SAVE FIG
        # Define a folder relative to the current script directory
        folder_figures_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}/Parameter_Plots")
        # If folder doesn't exist yet, create it
        if not os.path.exists(folder_figures_path):
            os.makedirs(folder_figures_path)
        # Define the full file path, including the folder and file name
        plot_file_path = os.path.join(folder_figures_path, f"{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}
        ↪  K-PhiB-m_eff-proportionality.png")
        # Save the figure to the specified folder
        plt.savefig(plot_file_path, dpi=300)
        plt.close()

    def junc_sweep_dict_creation():
        '''Create dictionary of junction-sweep correspondence.'''
        # Filepath of the sweep-junction data
        file_path = f"{current_dir}/Chip#{chip_name}-{data_prefix}/
        ↪  {data_prefix}-Chip#{chip_name}-junction_sweep_correspondence.csv"
        # Convert file to a dataframe
        df = pd.read_csv(file_path, header=None)
        # Convert to dictionary
        junc_sweep_dict = pd.Series(df[1].values, index=df[0]).to_dict()
        # Convert the values from a string to a list of integers
        junc_sweep_dict_int = {int(key[1:]): string_nums_convert_to_range(value[1:]) for key, value in
        ↪  junc_sweep_dict.items()}
        # Create a new dictionary where values are now keys
        swapped_dict = {value: key for key, values in junc_sweep_dict_int.items() for value in values}
        # Sort the dictionary by key
        sorted_swapped_dict = {key: swapped_dict[key] for key in sorted(swapped_dict)}
        return sorted_swapped_dict

    def gaussian_equation(x, mean, var=0.4, A=5):
        '''Equation for a Gaussian'''
        return A * np.exp(-1/2*(x-mean)**2/(var))

    def fit_gaussian(y_data, x_data, x_limits):
        '''Plot a fitted Gaussian on the graph'''
        # Initial guess for the parameters: mean, std deviation, amplitude
        initial_guess = [[1.5, 0.08, 2], [8, 2, 5], [0.5, 0.001, 2]]
        # Fit the data to a Gaussian
        popt, _ = curve_fit(gaussian_equation, x_data, y_data, bounds = ([0, 0, 0], [50, np.inf, np.inf]),
        ↪  p0=initial_guess[i], maxfev=10000)
        print(f"Extracted Gaussian parameters. Mean, Variance, Amplitude: {popt}") # Print the parameters
        # Plot the original data and the Gaussian fit
        # plt.scatter(x_data, y_data, label='Original data')  # Plot original data
        plt.plot(np.linspace(*x_limits,500), gaussian_equation(np.linspace(*x_limits,500), *popt), 'k-', linewidth=2)
        ↪  # Plot Gaussian fit on top. label=f"£\mu£ = {round(popt[0],2)}",
        # plt.show()
        return popt

    def initialise_constants():
        '''Initialise all values which are constant for the duration of the program'''
```

```
    global current_dir, chip_name, data_prefix, folder_path, junction_nums, temperature
    # Get the directory where the current script is located
    current_dir = os.path.dirname(os.path.abspath(__file__))
    # Get chip name and data prefix
    chip_name = input("What's the chip name? (e.g. 35-C5) ")
    data_prefix = input("What's the data prefix? (e.g. AA or BB etc) ")
    temperature = input("What's the temperature(in K)? (e.g. 77 or 300) ")
    folder_path = f'Chip#{chip_name}-{data_prefix}/Extracted_Parameters'
    create_dict_of_sweeps_to_exclude_from_csv()
    junction_nums = how_many_junctions()

#### CODE
# Dictionary of colors for plotting junction numbers
colors_dict = {1: "#f6192B", 2: "#1c742b", 3: "#aaffc3", 4: "#4363d8", 5: "#f58231",  # Red, Dark Green, Light Mint, Blue, Orange
               6: "#911eb4", 7: "#42d4f4", 8: "#32CD32", 9: "#bfef45", 10: "#800000",  # Purple, Light Blue, Green, Lime, Maroon
               11: "#469990", 12: "#caaade", 13: "#9A6324", 14: "#ff6347", 15: "#daa8b9",  # Teal, Light Purple, Brown, Tomato, Pink
               16: "#ffe119", 17: "#808000", 18: "#dfbf9f", 19: "#8a2be2", 20: "#a9a9a9",}  # Yellow, Olive, Peach, Blue Violet, Dark Gray

initialise_constants()
# Plot Phi_B vs m*
sweepnums, phiB, meff = plot_phiB_vs_m()
# Plot the proportionality coefficient between Phi_B and m*
sweep_junc_dict = junc_sweep_dict_creation()
average_phiB_m_proportionality_coeff(sweepnums, phiB, meff)
# Go through each parameter and plot histograms
parameters = ['$\\Phi_B$(eV)','m*','$\\alpha$']
for i, parameter in enumerate(parameters):
    make_figures_save_multiple_junc(junction_nums)

plots_folder_path = os.path.join(current_dir, f"Chip#{chip_name}-{data_prefix}\\Parameter_Plots")
plots_file_name = f"{data_prefix}-Chip#{chip_name}-HSC11BIPY-{temperature}K -J{[j_num for j_num in junction_nums]}-parameterX.png"
print(f"\nParameter plots have been created and can be found in {plots_folder_path}, \nunder the names of {plots_file_name}")
```