Université d'Ottawa
Faculté des sciences

University of Ottawa
Faculty of Science

PHY 3902: Physics and Applied Physics Laboratory I

LAB 2: Characteristics of a light-emitting diode

Objectives:

- Understand the operation of a diode
- Control and test a digital-to-analog converter (analog voltage source) with Arduino
- Determine the current-voltage characteristics of an electrical component
- Analyze data collected with Arduino using Python
- Determine Planck's constant

Caution: electronic devices not connected to ground are very sensitive to static electricity and can be rendered completely and irreversibly inoperative following a static discharge. Avoid working in dry carpeted rooms and discharge yourself regularly by touching a massive metallic object, copper water pipe etc.

Caution: never connect an LED directly to any board pin! Available voltages (5 V or 3.3 V) are sufficient to completely destroy the LED in a fraction of a second. One must always place a resistor in series with the LED to limit the current. See text for details.

Introduction:

In this lab, we will characterise the operation of a light-emitting diode (LED) by plotting the **current versus voltage curve**. To obtain correct and accurate data, we will generate a precise voltage using an external digital-to analog converter (DAC) since you saw that using the built-in PWM voltage output on the Arduino board might not be the ideal solution for this purpose.

Diodes are electronic devices that prevent, for instance, current from going in a direction that could be damaging to a circuit. They are considered (ideally) as one-way switches that only allow current in the so-called forward direction (*forward bias*) once a certain minimum voltage is reached, and do not let current through in the opposite direction (*reverse bias*). The word diode contains the root 'di-', meaning 'two' as

they are built by joining two semiconductor materials with different properties. There are many types of diodes that were designed according to the desired use. The most often encountered ones are as follows.

The *PN* junction diode: these are the most used ones. They consist of a junction between two parts, the anode made of material doped *P* to have a deficit of electrons, thus containing so-called `holes` which are mobile (these will become *minority carriers*), and the cathode made of material doped *N* to have a surplus of mobile electrons (these will become *majority carriers*). The whole device is electrically neutral. On either side of the *PN* junction, a *depletion region* appears at room temperature, caused by thermal diffusion of a certain quantity of majority carriers (electrons) from the *N* region to the *P* region, with the consequence that an equal number of holes seem to have diffused from the *P* region to the *N* region. This effect is temperature dependent. In this depletion region, an electric field *E* is established, which opposes circulation of current in the forward direction. This manifests itself by a potential barrier ΔV which needs to be overcome for current to flow in the forward direction (conventional current from *P* region to *N* region). A diode will thus require a voltage difference ΔV between its leads to conduct a forward current, the *activation voltage*. For a silicon diode, this voltage is about 0.7 V; for a germanium diode, it is about 0.3 V. A typical diode application is as a one-way switch: open when voltage is below activation voltage (including negative voltages) and closed when voltage is above this activation voltage.

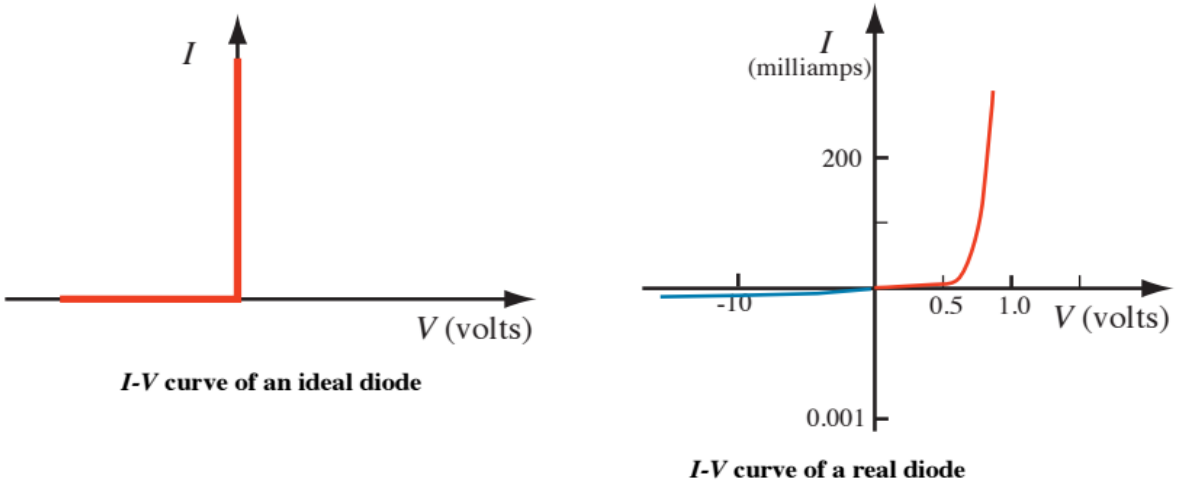


Figure 1: Current vs. Voltage curves for ideal and real diodes

The ideal diode is a theoretical description of the operation of a diode and has the characteristics of a switch. In the above picture (left of Figure 1), the activation voltage has been neglected. One could have translated the curve to the right by a value ΔV of about 0.7 V. The real diode has an exponential response above the activation voltage as opposed to an immediate response. There is a simplified relationship giving the current, ***I***, through a *PN* junction:

$$I = I_s \left(e^{\left(\frac{qV}{nkT}\right)} - 1 \right),$$

where $\boldsymbol{I_s}$ is the saturation current, typically somewhere around 10 x 10^{-12} A, ***q*** the elementary charge, ***V*** the voltage applied to the diode, ***k*** is Boltzmann's constant and ***T*** the temperature of the junction. The

ideality factor is represented by ***n***. The ideality factor is included to account for second-order differences between theory and real diode behaviour. This relationship models the positive part of the diode curve.

By the way, since the voltage across a diode when it conducts is about 0.7 V, and since a typical diode like the 1N4148 can dissipate up to about 1/4 W of power as heat, the current must be limited: $I_{max} < \frac{0.25\ W}{0.7\ V} < 350\ mA$, in fact the datasheet recommends that continuous currents remain below 300 mA.

The light-emitting diode (LED): an LED is essentially a *PN* junction diode that emits light due to the flow of electrons inside the depletion region of a suitable semiconductor material. LEDs emit light but consume significantly less energy than incandescent lights because they are much more efficient, wasting much less power in the form of heat. The activation voltage of a light-emitting diode varies by the colour of the LED, but is typically a little higher than that of standard diodes. This higher activation voltage can have important consequences. Such LEDs are embedded in plastic and can only dissipate 100 mW or so. Therefore, for an LED with an activation voltage of 1.6 V, the maximum current which can flow through the diode safely would be $\frac{0.1\ W}{1.6\ V} = 63\ mA$. In fact, the datasheet for the LEDs we are using recommends that continuous currents remain below 20 mA. ***Therefore, a suitable resistor needs to be placed in series with the LED in order to limit the current.***

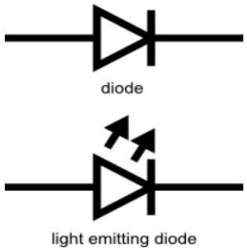


Figure 2: Difference between diode and LED symbols. Current flows from left to right.

The photodiode: This one is a modified *PN* diode. It is light sensitive and converts light into a current. Solar panels are large photodiodes with a large collecting surface to convert solar light into electricity (all *PN* diodes are photosensitive, therefore regular ones are in opaque packages).

Figure 3: Photodiode symbol.

The Zener diode: The Zener diode is used as a voltage reference. It allows for instance the protection of a circuit from abrupt supply voltage variations. The reverse (negative) Zener voltage is relatively constant,

and such a diode can thus be used as a reference voltage. These diodes come in a variety of Zener voltages, for instance 12 V. Here again, one must limit the current!

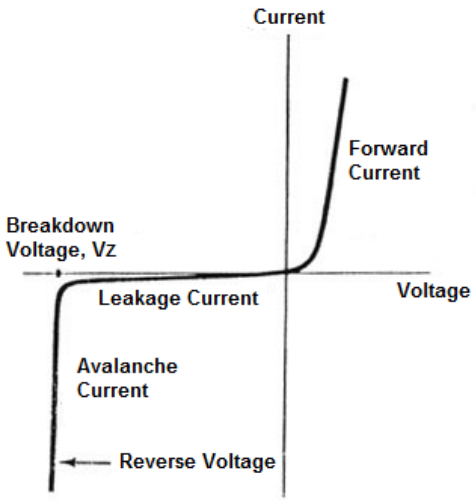


Figure 4: I-V characteristic of a Zener diode

Required components:

- Arduino Uno R3
- ~220 Ω resistor
- USB 2.0 cable to connect Arduino to your computer
- Approximately 9 male-male wires
- 3 LEDs (red, green, yellow)
- Prototyping board
- Arduino IDE
- MCP4725 DAC
- Python installed
- Text editor (Idle, VS Code …) or IDE (Pycharm, Spyder …)

Part A: Getting started with the MCP4725 DAC

A DAC or "digital-to-analog converter" is an electronic device which can convert a binary digital command into an analog voltage output, in this case between 0 V and 5 V. In other words, the DAC shield (circuit board) will allow us to output a true analog voltage. We will use the common 12-bit DAC shield MCP4725 as can be seen below:

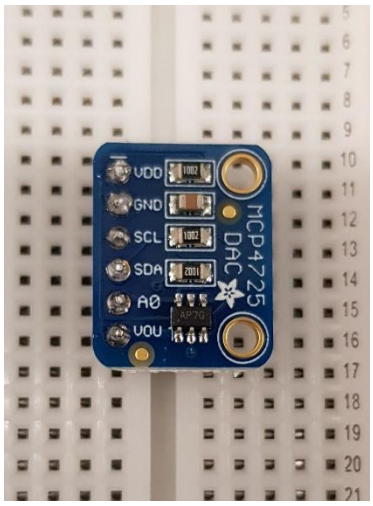


Figure 5: DAC shield

The difference between a DAC and the PWM signal available from the Arduino is:

- A PWM signal is characterized by its duty cycle and tries to mimic a continuous voltage on average by switching rapidly between 0 V and 5 V. This technique causes a large noise at the switching frequency (1 kHz for pins 5 and 6; 500 Hz for pins 3, 9, 10 and 11 if one uses default parameters), and forces the user to insert a low-pass filter to eliminate the noise and obtain good precision.
- The signal produced by the DAC, on the other hand, is a real continuous voltage with good precision. In our case, the MCP4725 has **12-bit resolution**, corresponding to a digital range from 0 to 4095 (as opposed to the range 0 to 255 for the 8-bit Arduino PWM), for voltages from 0 V to 5 V, respectively. This is a significant gain in resolution, perfect for our experiment!

For this lab, Arduino IDE is already installed on your computer, but we will need to add two additional libraries **Wire.h** and **Adafruit_MCP4725.h** at the beginning of your sketches. The first library, **Wire.h**, is for I2C serial communication and is pre-installed with Arduino IDE so you only need to install the second one. The best way to go about this is to go to "**Tools > Manage libraries**" or Ctrl+Shift+I for shortcut. In the search bar, type: MCP4725 and you should see something like this:

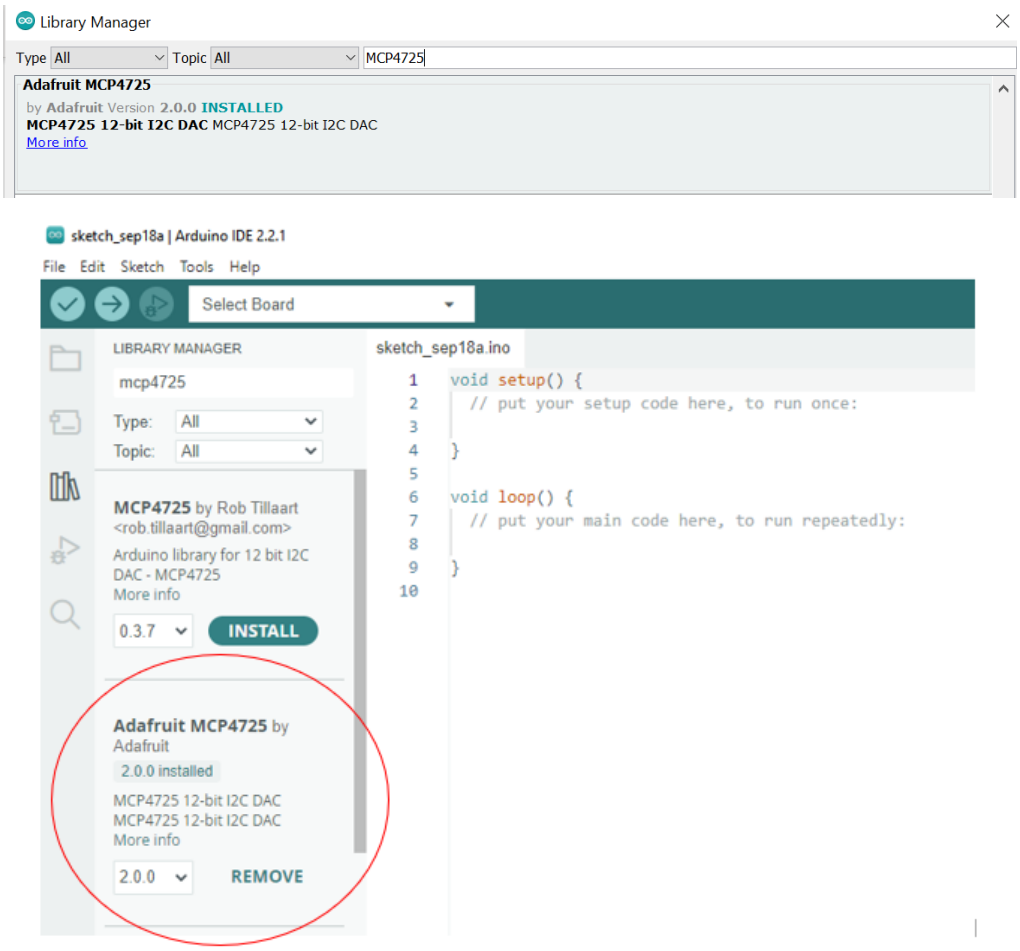


Figure 6: Arduino Library Manager. It will look a little different depending on whether you are using Arduino IDE version 1 or version 2.

In my case, you can see that the library is already installed but, in your case, you might have the option to click on install, which when you do, you will now be able to include this library in your Arduino code as it will be listed in your list of choices.

Note: *You might have to close the Arduino IDE and start it again for the changes to take effect.*

Plug your MCP4725 on the prototyping board such that each pin is connected to a separate column.

We will need to connect pins **VDD**, **GND**, **SCL**, **SDA** and **VOU** to the Arduino. Connect pin **VDD** to pin **5V** on the Arduino board, pin **GND** to pin **GND** of the board, pin **SCL** to pin **A5** on the analog part of the board (this will act as a clock for communications between board and MCP4275), finally pin **SDA** to pin **A4** (this will be for actual serial data transfer).

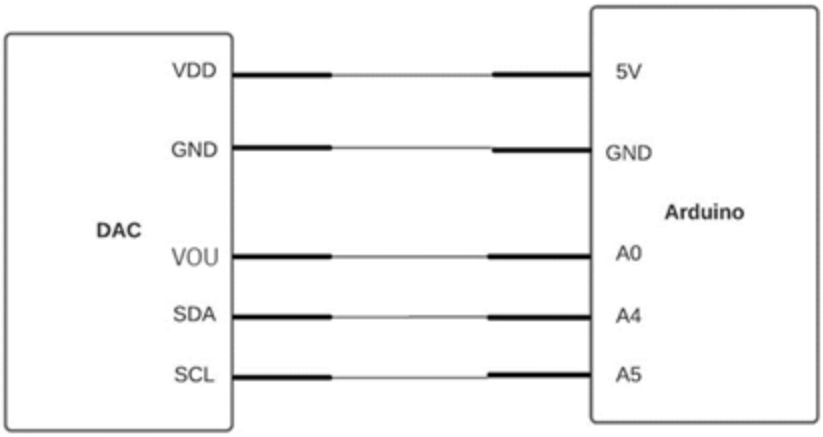


Figure 7: Schematic diagram for interfacing the DAC with Arduino

Pin **VOU** is the actual voltage output of the DAC. We are connecting it to the A0 input pin since we will be measuring the output voltage as a verification in Part B. But otherwise, you would connect it to your circuit.

PART B: TESTING THAT the DAC WORKS:

In this part, you will connect the **VOU** output to one of the input pins (e.g. **A0**) of your Arduino. This will allow you to output specific voltages and measure them with the ADC (analog-to-digital converter). Of course, if everything works, you should measure exactly what you outputted! Now, this might seem trivial, but it is very important to test each part of an experiment that you design. This is particularly true for complex experiments. This extra step can save you a lot of time trying to debug an experiment that might not be working as expected.

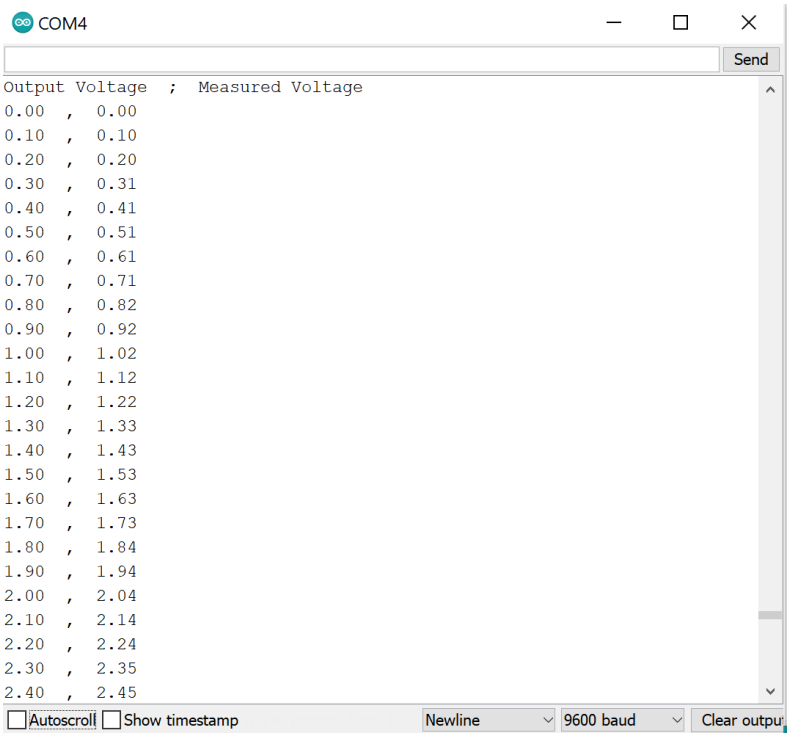
- **VOU** is connected to pin **A0** on Arduino to read the voltage out of the DAC using the Arduino ADC.
- Program a sketch that will output a series of voltages, from 0 V to 5 V in steps of 0.1 V. If you are curious how to do this, you might want to inspire yourself by a built-in example accessible through the menu: "**File>Examples>Adafruit MCP4725>Triangularwave**". This example sketch outputs a triangular wave. In this code, you will notice the two libraries that are added, the setup command (***dac.begin(0x62)***) and the actual command for outputting a voltage (***dac.setVoltage(value,false)***). Here is a screen capture of the code:

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;

void setup(void) {
  Serial.begin(9600);
  Serial.println("Hello!");

  // For Adafruit MCP4725A1 the address is 0x62 (default) or 0x63 (ADDR pin tied to VCC)
  // For MCP4725A0 the address is 0x60 or 0x61
  // For MCP4725A2 the address is 0x64 or 0x65
  dac.begin(0x62);

  Serial.println("Generating a triangle wave");
}

void loop(void) {
    uint32_t counter;
    // Run through the full 12-bit scale for a triangle wave
    for (counter = 0; counter < 4095; counter++)
    {
      dac.setVoltage(counter, false);
    }
    for (counter = 4095; counter > 0; counter--)
    {
      dac.setVoltage(counter, false);
    }
}
```

- In your case, I suggest you use a similar FOR LOOP to ramp the voltage up from 0V to 5 V. I also suggest you figure out a way to run this FOR LOOP only once, so that it does not repeat forever. It is common to use a variable that you can call "**hasRun**" (or something like that!) to accomplish this. It can be set to 0 initially and then changed to 1 after running the first iteration of the LOOP. Then, you can ensure that the LOOP is run a single time by adding an **IF** statement.

```
if (hasRun == 0) {
  run your code
}
}
else {
  Do nothing
}
```

- Use the **Serial.print()** and **Serial.println()** command to output your results to the serial monitor. By formatting your data into two columns (output voltage, measured input voltage), it will be easy for you to plot your data to verify that the slope is unity. You can copy/paste your data into a .csv file that you can import into Excel, but you should really plot your data using Python!



PART C: MEASURING THE CURRENT-TO-VOLTAGE (IV) CHARACTERISTICS OF AN LED

You will use the DAC voltage output to light up an LED. By measuring various voltages in your circuit, you will be able to calculate the current passing through the LED. You will then plot these data to confirm that it behaves as expected!

- Wire up your electronics as follows, remembering to add a series resistor (~200-330 ohms). This will create a simple voltage divider.

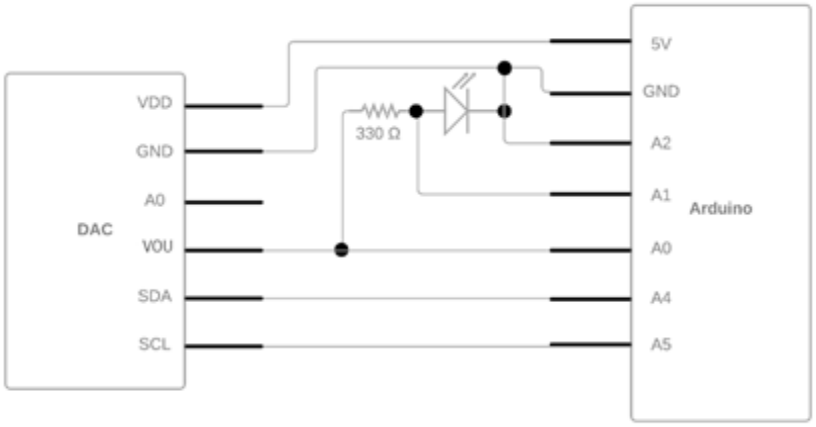


Figure 8: Schematic diagram for interfacing the DAC with Arduino (adding a resistor and an LED)

- Write a sketch that will output a voltage from the DAC. If you output 5V, the LED should light up brightly.
- Include a FOR LOOP to ramp the output voltage from 0 V to 5 V in small increments. Include a delay at each iteration to slow down this ramp! Measure the voltages at pins A0, A1 and A2 using the command **analogRead()**.
- Convince yourself that the current passing through the resistor will be the same as the current passing through the LED. If this is true, then you can calculate the current passing through the LED by measuring the voltage drop across the series resistor (**A0 - A1**) and dividing by the resistance value. This is Ohm's law.
- Print the voltage across the LED (**A1 – A2**) and the calculated current to the serial monitor. If you format these data into columns, it will be easier to copy/paste them into a .csv file for import into your graphing software (Excel or Python). CSV files have "Comma Separated Values", so each row of data can have many values all separated by commas. ***Think about how you plan on including error bars (standard deviation? standard error?) in your graphs… (all measurements have uncertainties, as you know!)***

PART D: PROCESSING YOUR DATA USING PYTHON

Some of you might already be familiar with the capabilities of the Python programming language. While I mentioned that you can use Excel to plot your graphs, you can also use Python. It is also possible to use advanced pre-written functions (added as "**libraries**" to your code) to manipulate your data prior to plotting. While the capabilities of Python are vast, we will focus on using some curve fitting and plotting functions. The objective here is for you to get a glimpse into Python's capabilities, in the hopes of inspiring you to delve into using Python in the future. (Spoiler alert!) These fitting functions are likely to be useful in future experiments!

Python is widely used for data analysis and data visualization using popular libraries such as **numpy**, **pandas** and **matplotlib**.

TASKS:

- Make sure Python is installed on your computer. It should be installed on the lab computers already. If you want information about how to install Python on your own computer, and how to make sure added libraries are installed and ready to use in your own code, please refer to the **"Python Installation Instructions"** available on Brightspace.
- You can write your Python code in any text file editor, but the one that comes with Python is called IDLE. Open an IDLE window.
- Open the file **resistor_properties.py** that has been prepared for you (available on Brightspace). Python syntax is different than it is for Arduino language, so play close attention to this in the sample Python code provided. Comments are added to help explain what each line of code does.
- This Python code:
    1. Opens a data file called "**ohm_law_verification.csv**" that should be located in the same folder as that of your python file. It then stores the data in variables. This data file has two

columns for voltage across and current passing through a plain resistor. You can certainly generate your own data file using your Arduino, **and I encourage you to do so!**

2. Defines a fitting function that will be used to fit your experimental data.
3. Defines the variables within this fitting function.
4. Fits your data with the fitting function to perform a best fit of your data. This will return the best fitting parameters by minimizing error between the data and the fitting curve.
5. Plots your data and the calculated best fit to a graph and displays it to the screen.

Notes:

- Notice that the actual fitting is done via a single line of code. This is extremely impressive if you think about it! This is possible since we included (imported) a few powerful libraries (pandas, numpy, matplotlib, scipy, etc.).
- You can and should edit this code to improve upon its capabilities.
- Using the current-voltage data you acquired for the LED, perform a similar fit and determine the saturation current I_s and the ideality factor n for the LEDs of different colour. Comment on your findings. Are these results what you expected?

Hints:

- The I-V characteristics for a diode is given by the following equation:

$$I = I_s * (e^{\frac{V}{n*V_t}} - 1),$$

where:
I = is the current flowing through the diode.
I_s = the saturation current or the diode leakage current density in the absence of light.
V = the applied voltage across the terminals of the diode.
n = the ideality factor.
V_t = (k * T) / q
k = Boltzmann's constant => $1.38064852*10^{-23}$ J/K
T = the absolute temperature in Kelvin (K) => ~298.15 K (room temperature)
q = the elementary charge in coulombs => $1.60217662*10^{-19}$ C

- To make your code easier to use, it is recommended that you define a variable B = 1 / ($n*V_t$). Therefore, the equation you will be working with in python is

$$I = I_s * (e^{B*V} - 1),$$

where ***V***, is the applied voltage across the diode's terminals. You will therefore need to define your constants such as ***k***, ***T***, and ***q*** at the beginning of your program so that once you fit your data and determine the fitting parameter ***B***, you can compute the value of the ideality factor ***n***. Of

course, the value of I_s will also be determined from your fit as well. The rest of your program will follow the same logic as what was done for the "resistor_properties.py".

- Change the color of your diode and collect the data from your serial monitor and export it to python. Ideally, you want to do this for at least 3 different colours (red, green, yellow for example). Again, make sure to include your csv files in the same base directory as your python files. Execute your "diode_properties.py" file again.
- How does your fit compare with the data that you collected? Your report should include and discuss these I-V curves.
- Are the values of I_s and n the same? Can you explain why or why not?

PART E

It is possible to determine Planck's Constant from the measurements you made above! Planck's constant shows up in several equations related to quantum phenomena. Without getting into too much detail, the Planck-Einstein relation relates energy and wavelength of a photon. This relation is written as $\boldsymbol{E = \frac{hc}{\lambda}}$ where **h** is Planck's constant, **c** is the speed of light and $\boldsymbol{\lambda}$ is the wavelength of the photon.

When you apply a voltage to an LED, you are essentially providing energy to electrons. If an electron has enough energy, it can release this energy by jumping between energy levels as a photon. The photon's energy is quantized, which explains why this photon has a specific colour (wavelength).

The voltage at which an LED starts to emit photons (i.e. the voltage at which the current starts to increase in your I-V curves) is called the "forward voltage, the "knee voltage", the "threshold voltage" or the "activation voltage". At this forward voltage (V_f), the electron will have energy $\boldsymbol{E = eV_f}$ where the electron charge $e = 1.60217662 \times 10^{-19} C$.

Combining this information, we can obtain a relation between $\boldsymbol{V_f}$ and $\boldsymbol{\lambda}$. Work out what this relationship should be. In fact, if you plot $\boldsymbol{V_f}$ as a function of $\frac{\boldsymbol{c}}{\boldsymbol{\lambda e}}$ for a few different LEDs, you should obtain a straight line and the slope of the line should be related to Planck's constant. ***Determine Planck's constant and compare your result with the accepted value of*** $\boldsymbol{h = 6.62607015 \times 10^{-34}\ m^2\ kg\ /\ s}$.

There is lots of information about using an Arduino to compute Planck's constant online, so you can gather more information from various sources. But I will upload a report published by students from the University of Havana (*Rev. Cubana Fis.36, 125 (2019)*). This report is not perfect, but I like that it is structured like a real scientific journal article.

Notes:

- Acquire current-voltage data for several LEDs of different colours, as you did above. You can determine the forward voltages of each LED by extending the linear part of the IV curve towards the x-axis. You can do this by hand or you can certainly fit your data using Python if you want to be a little more precise!

- Some LEDs have a coloured plastic casing to enhance the colours. I suggest you use LEDs with clear plastic casings for more accurate results. Why is this?
- Question: What is the wavelength of the white LED??!