The LED Method of Measuring Planck's Constant

Reading the Teensy (or Arduino) Using Python

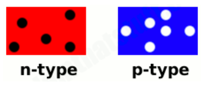
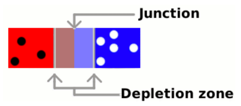
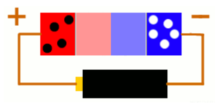
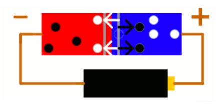
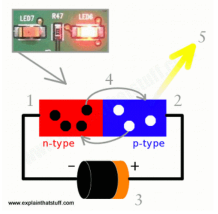
Read the handout on Reading the Teensy before starting this handout. You will need python installed on your laptop to take your data.

Background - Using LED's to Determine Planck's Constant

You saw in the Photoelectric Experiment that light of a definite frequency or higher was needed to knock electrons free from a metal. The slope of the stopping voltage versus the frequency of the light was Planck's constant, *h*. The convenient units were eV-s, that is electron volts times seconds.

LED's (or Light Emitting Diodes) also give a direct method of measuring Planck's constant. LED's are devices where two different *semiconductor* layers are manufactured one on top of the other. Semiconductors do conduct electrons, but not nearly as well as a metal. Silicon is was the first semiconductor that was used in manufacturing electronic devices. It was discovered that by intentionally adding impurities to silicon (called *doping*), extra charge carriers either positive or negative can be created in the material, and with the extra charge carriers, all kinds of interesting devices can be made. The excess positive charge carriers are called *holes* because they are like gaps in a space filled with negative electrons!

How a diode works

Here are two doped semiconductors, one with excess *holes* and one with excess electrons as charge carriers.	
When joined together some electrons move to the right to fill the holes, and some holes move to the left to suck up electrons, but when that happens an electric field is created hindering this movement until the process finds an equilibrium. The center region has the electric field and is called the *depletion region*.	
When a negative voltage is applied to the *n* end, the voltage causes an electric field that strengthens the field in the depletion zone, so no current flows.	
When a positive voltage is applied to the *p* end, it creates an electric field in the opposite direction. When the voltage is high enough, the electrons can holes do not have a barrier, so they flow to the middle and combine resulting in a current. This electron-hole recombination gives off energy.	
In an LED the diode is specially designed so that when the electrons and holes recombine in the depletion region they give off light. One photon is created each time one hole and one electron combine. The energy of the photon approximately equals the energy released when one hole and one electron recombine.	 1 2 3 4 5 n-type p-type − + www.explainthatstuff.com

The important concept for a Planck's constant experiment is that frequency of the light is proportional to the voltage it takes to make an LED start shining. There are many complications that can make the interpretation of the data harder. For example the LED's do not put out just one frequency of light, but a range.

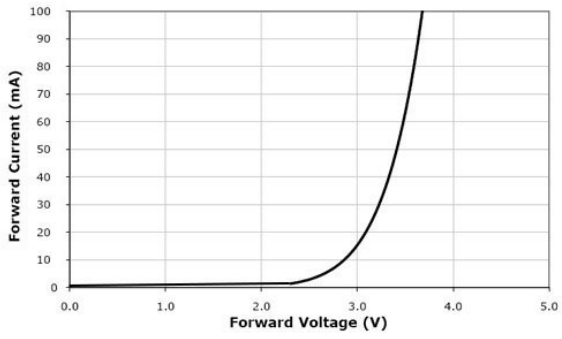
Should you use the brightest wavelength, the highest frequency, or a weighted average? You can address these issues later.

How to Measure an LED

The key concept again is *the voltage required to light and LED is proportional to the energy of the photon emitted.* Now manufacturers can make LED's with different colors and each color LED has a different *turn on* voltage. A typical I-V curve is shown in the figure to the right. There are theoretical reasons that the curve should be an exponential. You can model the curve with the function

$$I(V)=e^{(V-V_f)/V_0}$$

where V_f is the forward turn on voltage you need and V_0 is a shape parameter.

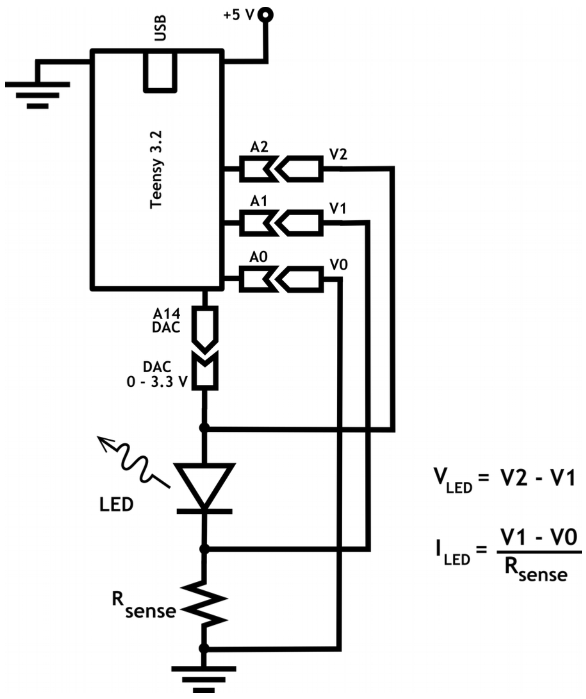


Picture 1. Forward Voltage vs. Forward Current

Measuring the I-V Curve

You can use a modification of the first circuit you made with the Teensy shown in the figure to thte right. You want to directly measure the voltage across the LED, and the current through the LED. The DAC, pin 14 on the top of the Teensy, powers the circuit. The voltage across the LED is *V2 – V1* and these voltages are read by analog input ports **`A2`**, and **`A1`**. The current can be measured using Ohm's Law and reading the voltages **`A1`**, and **`A0`**. The Teensy program will step the DAC output read the three voltages, compute V_{LED} and I_{LED}, and print the data to the USB serial port.

I use a 1 kΩ sensing resistor. You have to make sure the ReadSCPI program is using the right value for **`RSENS`**.



$$V_{LED} = V2 - V1$$

$$I_{LED} = \frac{V1 - V0}{R_{sense}}$$

Taking Data

I suggest renaming your Teensy sketch by clicking **`File → Save As`**. I use the name **`PlanckLED`**. You need to modify your Teensy sketch to set the DAC, read voltages and output the voltages and the currents. Program the sketch, debug it and start it running on the Teensy. **Note**: unlike a laptop, the Teensy will run the last program uploaded to it every time it starts. So you can unplug the Teensy and next time you plug it in, it will restart the last program.

You also need to have python on your laptop. Once you plug the Teesny in, run your python program **`ReadSCPI.py`**. I modified some of the Teensy parameters as follows.

- **`RSENS 1000`** – Make sure you have the right value for the sensing resistor.

- **`NSTEPS 1024`** – You want enough data points to capture the curve well. The code defaults to 16 steps, but that is not enough. Set **`NSTEPS`** to **`1024`**. The reason I used 1024 is because the DAC has 12 bit resolution or 4096 different steps in its output. By setting the number of steps to 1024, it takes data every fourth possible output value.
- **`NAVG 100`** – Now that you are taking real data, you should increase the number of readings that are taken and averaged at each step.

Running the System

Here is a capture of a typical session for when I took data.

First, run the program.

```
$ python ReadSCPI_18c.py
```

Next, the program prints the information the Teensy program outputs. Note that since I coded the Teensy to print the name of the sketch, it nicely tells me what program is running on the Teensy.

```
ReadSCPI_18c.py
ProfHuster
2018-02-05

This program collects data from an Arduino (or Arduino-clone) that uses
a SCPI-style command structure and saves it to a file.

Versions:
18b - Flushes stdout
```

The python program tells me the name of the data file it is using. The python program also give me a chance to enter information about this particular data set. I typed in that it was an LED with a 700 nm wavelength.

```
Opening file "SCPI-2018-02-07-1738.csv"
Enter a comment (return to end): LED 700 nm
Enter a comment (return to end):
```

Next the program needs to find and talk to the Teensy. It finds two devices connected to a USB. (This example is from a Mac.) I selected the Teensy.

Troubleshooting

If the Teesny doesn't show up, or you are not sure which device is the Teensy try these steps.

- Check the Arduino program. It can be running, *but the Serial Monitor window must be closed!*
- Unplug the Teensy run the python program and see which devices are listed. Kill the program, plug in the Teensy and try again. A new device should show up.
- Failing these steps, google **`Teensy does not show up`**. You know the routine.

```
The available ports are:
0, /dev/tty.Bluetooth-Incoming-Port
1, /dev/tty.usbmodem3634801
Enter serial port number: 1
```

After the python program sends a prompt, I use **`conf?`** To get the configuration to see if I have to change anything.

```
Enter command line (help? for help)
Enter 'q' to quit
: conf?
>rSens = 1000.00<
>nSteps = 1024<
```

```
>nAvg = 100<
>nMsSleep = 1<
```

Finally take the data by typing **data?**. When the data is done, the python program prints **Done**.

```
: data?
Done
```

And I quit the program by typing **q**.

```
: q
$
```

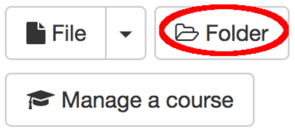
Now check the folder you were working in. There should be a **.csv** file with a time and data that is close to the current time. You can open it with a text editing program like TextEdit or NotePad. I avoid opening it with Excel because I might accidentally save it in a different format. The first several lines of the data file are shown below.

```
# LED 700 nm
# CONF?

# rSens = 1000.00

# nSteps = 1024

# nAvg = 10

# nMsSleep = 1

# V0  ,   V1  ,   V2  ,   Vs  , Is (mA)
0.0009, 0.0008, 0.0054, 0.0046, -0.0001
0.0008, 0.0008, 0.0054, 0.0046, 0.0000
0.0008, 0.0008, 0.0077, 0.0068, 0.0000
```

Note: The LED will turn on briefly near the end of taking data.

Analyzing the LED Data

Analyzing the LED data has two parts, first, fitting each I-V curve to get a turn-on voltage, and second, fitting the turn-on voltage versus frequency to get Planck's constant.



You need to upload the data files to your CoCalc account.

- Go to your **ModPhysLab-Sp19** project.
- You should have a folder named **LED_Experiment** in folder named **EXPERIMENTS** in your home project folder. Click to enter that folder.
- On your laptop, select the files, drag and drop them where the page says ➔ **Drop Files to upload**.
- Note that I added the LED wavelength to the file name, so my files are named like **SCPI-2018-02-07-1356-700nm.csv**.
- Now you need to fit the I-V curves.

Fitting LED I-V Curves

I have given you a jupyter notebook to help analyze the I-V data. It has missing code you have to supply. The next section takes you through the code and points our what you have to add.

These lines import the modules. The should mostly look familiar to you.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.odr as odr
```

```
from glob import glob
import re
%matplotlib inline
```

The next cell lists the .csv files in your working folder and prompts you which one to use. Once it opens the file and reads the data, it makes variables with working names. The most important columns for your application are the **`current`** and the **`voltage`**.

It also is going to save your figure into a **`.png`** image file. The line with **`re.sub(r'csv$', "png", dataFile)`** uses the exact name as the data file but substitutes the **`.png`** suffix for the **`.csv`** suffix.

The last part of the code reopens the file, reads the first line and saves it for the figure title.

```
files = glob("*.csv")
files.sort()
for (i,file) in enumerate(files):
    print "%d, %s" % (i, file)
iFile = int(raw_input("Enter file number: "))
dataFile = files[iFile]
fpData = open(dataFile)
data = np.genfromtxt(dataFile, comments='#', delimiter=',')
v0 = data[:,0]
v1 = data[:,1]
v2 = data[:,2]
voltage = data[:,3]
current = data[:,4]

# Make a file name for the figure
figFile = re.sub(r'csv$', "png", dataFile)
print "figFile = ", figFile

# Now go back and grab the first comment line
fpData = open(dataFile, 'r')
line = fpData.readline().strip()
figureTitle = line[2:]
print "fileComment >%s<" % (fileComment)
```

The next cell plots the data. I always plot data right after I read it in just to see if the data makes sense. I intentionally left out the plot command. You should plot the data with a blue line.

```
fig = plt.figure(figsize=(15,5))
ax = fig.add_subplot(111)
ax.grid(True)
ax.set_title("Comment: %s" % (figureTitle))

# You have to fill this line in!!!!

ax.set_xlabel('Voltage (V)')
ax.set_ylabel('current (mA)')
```

Next you want to fit the data to a model function. *You have to write the python function that is the right model for the data.*

Next you have to wrap the model and the data in the **`odr`** methods. Note that I als

```
def LEDModel(Params, V):
    V0 = Params[0]
    Vf = Params[1]
    return "You have to fill in this function!"

myModel = odr.Model(LEDModel)
```

You have to look at your data and decide what reasonable values are for the current and voltage uncertainties. Think!

```
myData = odr.RealData(voltage, current, sx=”Voltage Uncertainty”, sy=”Current
Uncertainty”)
```

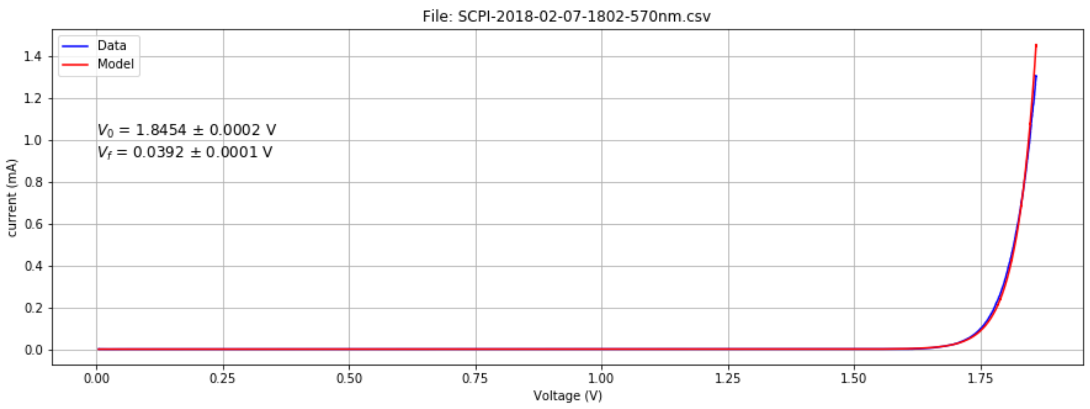
Finally the rest of the code in the cell gives reasonable values to the initial parameters, runs the fit program, and names the best fit values and uncertainties reasonable names. The results are then saved in a formatted string **`resultStr`**.

```
initParams = np.array([1.0,2.0])
myOdr = odr.ODR(myData, myModel, beta0=initParams)
myOutput = myOdr.run()
myOutput.pprint()
(V0_Fit, Vf_Fit) = myOutput.beta
(V0_SD, Vf_SD) = myOutput.sd_beta
resultStr = \
r"""$V_0$ = %.4f $\pm$ %.4f V
$V_f$ = %.4f $\pm$ %.4f V""" % (V0_Fit, V0_SD, Vf_Fit, Vf_SD)
print resultStr
```

Finally you have to plot your data and best fit model together. The cell below makes the figure and saves it to the figure name created above.

```
fig = plt.figure(figsize=(15,5))
ax = fig.add_subplot(111)
ax.grid(True)
ax.set_title("Comment: %s" % (figureTitle))
ax.plot(voltage, current, 'b-', label='Data')
ax.set_xlabel('Voltage (V)')
ax.set_ylabel('current (mA)')

ax.plot(voltage, LEDModel(myOutput.beta, voltage), 'r-', label='Model')

ax.text(0, 0.7 * np.max(current), resultStr, size='large')

jnk = ax.legend()
fig.savefig(figFile)
```

Here is one of my figures:



You then have to do this for each LED! Enjoy!

Fitting the LED Data to get Planck's Constant

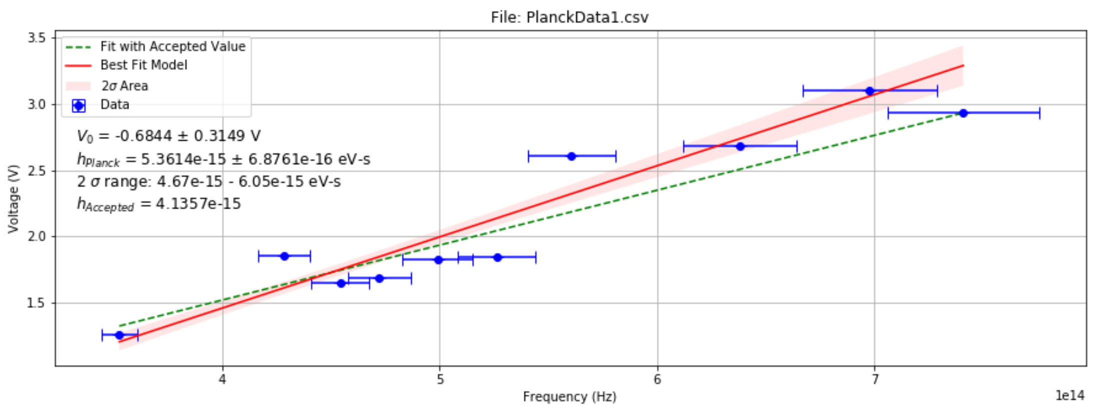
You are almost there! For each LED you have to manually copy the best fit forward voltage and the uncertainty of that voltage to a new data file. Here is the first few lines of my data file. Note that I put comments in (lines that begin with a hash tag #.) The first column is the wavelength in nm. The second column is the uncertainty of that wavelength. From looking at the specification for each of the LED's I estimate there is about a 20 nm uncertainty in each wavelength. The third column is the forward voltage I got from fitting the data of each LED. The uncertainties are in the fourth column. The first couple are small, but some of the others are bigger. I named this file **`PlanckData1.csv`**.

```
# My data
# 2018-02-07
# ProfHuster
# wavelength (nm), Uncertainty Wavelength (nm), V0 (V), Uncertainty V0 (V)
850.0, 20.0, 1.2603, 0.0001
700.0, 20.0, 1.8540, 0.0001
```

You have to create your own data file.

Fitting The Planck Data

Now you need new jupyter notebook that reads your new Planck data file. Fits it to a straight line, and plots it. I got carried away making mine fancy, but here it is.



I used error bars in plotting my data by using the method

`ax.errorbar(freq, V0, xerr=freqSig, yerr=V0Sig, fmt='bo', capsize=5)`

instead of **`ax.plot`**. The red line is my best fit and the shaded pink area are the lines that my data is consistent with 95% (or two standard deviations) of the time. Finally, the green dashed line shows the accepted value for Planck's constant.

My Conclusion

So I have to conclude that I measured $h_{Planck} = 5.4 \pm 0.9 \times 10^{-15} eV - s$ which is **not consistent** with the accepted value of $h_{Planck} = 4.14 \times 10^{-15} eV - s$.