INTRODUCTION

Python is a widely used programming language that strikes a middle ground between rigorous compiled languages like C++ or Java and easy to use scripting languages like perl, MATLAB, or Maple. It has many advanced numerical, plotting, and scientific libraries, and fast numerical libraries.

Jupyter is an interactive python shell that has extensions that make it easy to develop and run code interactively. Jupyter also allows formatting "cells" as typeset text. This combination makes it easy to create documentation, derivations, calculations, and numerical analysis all in the same document.

Finally, SageMath Cloud, or SMC, (**http://cloud.sagemath.com**) is a free website that hosts a computing environment that includes Jupyter Notebook, and much more, complete Linux machines, LaTeX document production, file storage, etc. SageMath Cloud also allows a fully collaborative environment where multiple people, like lab partners, can simultaneously edit the same file.

We purchased a course upgrade to your SageMath Cloud account for this course. You will start using SMC for your lab notes, data analysis, and document writing. You will start by using SMC to plot your data.

PLOTTING WITH PYTHON & MATPLOTLIB ON SAGEMATH CLOUD[1]

Start by logging in to your SMC account. When you login, you should have a folder called (with your name substituted for **Your Name**, of course) **Your Name – Optics Lab Fall 2016**. Click on that link. Inside that folder you should find a folder named **Lab08-Polarization**. Click on that link to enter that folder. This file is in that folder and is named something like **Optics Lab08 Jupyter-Python Plotting.pdf** (maybe with a version number.) Click on that link to open the file in SMC.

CREATING A DATA FILE

To start, you need to get your data into a file. A very general format is to save your data in a file with two columns separated by commas. You should also put some comment lines and column labels in the file. In the example below any line starting with # is ignored as a comment. Note that it is ***always*** a good idea to put some basic information in the comments in your data file. A file full of just numbers is useless. Below is the data file this guide uses as an example, but *enter your own data in the file*.

```
# Polarization Data
# Run 1
# ProfHuster
# 2016-09-16
# "Theta", "Intensity"
0, 2.4
45, 1.2
90, 0.2
135, 1.4
180, 2.6
```

To create a data file

1) Click on the **Files** button (2 in the figure below.)

2) Click in the box **Filename**, and type the name of the file you are going to create, **run1.csv**. Press **Enter** and the file should be created and a text editor will open with this file.

3) Enter your own data in the file using this format.

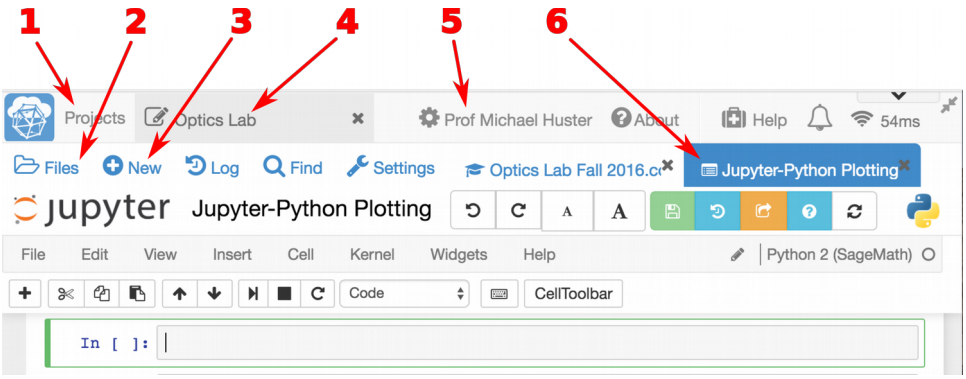
4) Click on the green **Save** button, then the **Files** button again. You should see the new data file in the folder and a tab with the file name should be available if you want to edit it more.

---

1 Almost everything you need to know about plotting is in the IPython notebook at this link:

http://nbviewer.ipython.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb.

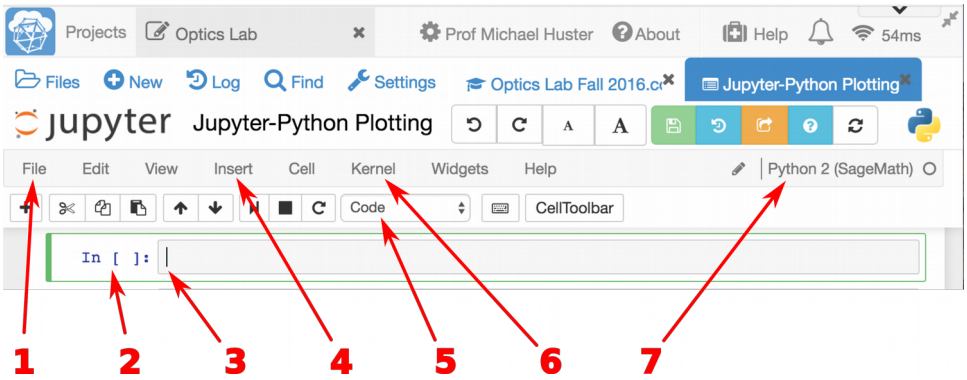
THE JUPYTER NOTEBOOK – FIRST TWO ROWS



1) This take you to your home page. SMC accounts are organized in *projects*.
2) The file viewer for your project.
3) Lets you create new files, folders and programs.
4) Home tab for *Optics Lab*.
5) Your account tab.
6) The currently active tab. A Jupyter notebook in this case.

Make sure you familiarize yourself with these tabs and know how to navigate around.

JUPYTER NOTEBOOK BUTTONS



1) The **`File`** menu. Mostly use this to save and halt the notebook.
2) This item shows the current cell (highlighted in green) is a code input cell. As you work the execution number shows here.
3) The code window. You can enter multiple lines of a python program in the *cell*. When you press **`<Ctrl><Enter>`** the code executes. Type

```
print "Hello, world!"
```

Then type **`<Ctrl><Enter>`** to run your first sample of python code. Yeah! Then you can delete the cell by clicking on it and either clicking the scissors icon, of using the menu and selecting **`Edit`** → **`Delete Cells`**.

4) The **`Insert`** menu allows to add cells above or below the active cell.
5) This button allows you to change a cell from code to *markdown* text for commenting. Markdown is a simple way to format text. Go to an empty cell, click on this button and select **`Markdown`**. Type

```
# This is a Heading 1 style
## This is a Heading 2 style
Finally this is normal text with an equation $f(x) = x^2 + 4x + 4$.
```

Then type **`<Ctrl><Enter>`** to format your first markdown code. You will bold headers and a nive formula.

6) This menu item lets you interrupt a program or make the notebook completely start over. You can also run a completely different program than python.
7) This cell shows the program you are currently running. We will use a **`Python 2`** kernel.

PLOTTING WITH PYTHON & MATPLOTLIB

First you need some import some libraries and enable the Jupyter notebook to show plots. Enter the code below in first code cell in your notebook, then type **<Ctrl><Shift>** to run that cell. (You can cut and paste from this document.) You will see that the code runs, then the next cell is active. (if an asterisk appears in the cell label and doesn't go away, you should restart the kernel by clicking **Kernel** → **Restart**.) If you are in the last cell, a new cell will be created when you execute that cell. The **numpy** library defines data arrays and defines most of the math functions you will need. You will need the curve fit library later.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

Now you need to read your data in from your data file. Cut and paste these lines into the next cell and tun it. The output should print the data.

```
tmp = np.loadtxt("run1.csv", comments='#', delimiter=',')
theta = tmp[:,0]
Intensity = tmp[:,1]
print theta
print Intensity
```

You next have to create a *figure* and make *axes* in that figure, then use an axis object method to plot the data. The following code plots the data as red circles. The string **'ro'** is shorthand for this choice.

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(theta, Intensity, 'o')
```

You can run this cell to see your first plot. Click on the cell to make it active, because you are going to enter more code in the cell.

Let's draw a model line that could fit this data. You can create an x-axis for your model with the numpy function linspace(min, max, number_of_points. Then you can use the cos function, square it with the ****** operator. I guessed the values to fit this data. Enter this code in the same cell as the after the code above and execute the cell. *Note*: *I use the intensity value 2500. Change that to the maximum value of your data*.

```
thModel = np.linspace(0., 180, 181)
IntenseModel = 2500 * np.cos(thModel * np.pi / 180) ** 2
ax.plot(thModel, IntenseModel, 'b-')
```

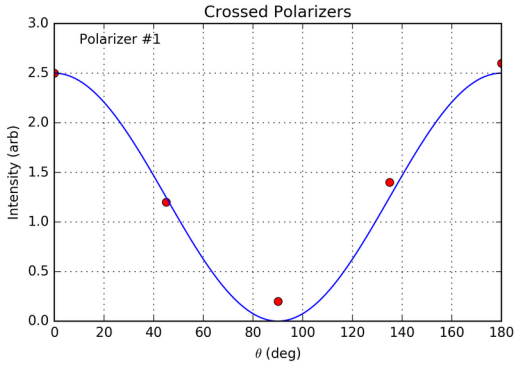
Now you need to add some normal things to the plot. First label the x & y axes. Note that the xlabel has an r in front of the string to make the string *raw*. Then the pair of **$**'s indicated the next characters should be interpreted as a *LaTex* string. The string **\theta** will be drawn as a Greek character. The next few lines give the plot a title, inserts some text in the plot, turns on grid lines.

```
ax.set_xlabel(r'$\theta$ (deg)')
ax.set_ylabel('Intensity (arb)')
ax.set_title('Crossed Polarizers')
ax.text(10, 2.8, 'Polarizer #1')
ax.grid(True)
```

The plot should look like the figure to the left. If you want to save the figure put in a line like

```
fig.savefig('PolMeas1.png', dpi=300)
```

There are many other features like logarithmic axes, many more symbols. You can find these by googling the website **http://matplotlib.org**.



FITTING A MODEL TO YOUR DATA

You often want to fit a model to your data. We are going beyond the simple cases of linear or polynomial models. First you have to import the curve fitting library. The first line below does that. Next you need to define the function you are going to fit your data to. The first argument of the function are your x values, the next arguments are the function parameters. Here is a function for this polarization data. I made the parameters **`IMax`**, **`theta0`**, **`period`**, and **`IMin`**. The fit will tell us the best values for our data.

```
def fitCos2(theta, IMax, theta0, period, IMin):
    return IMax * np.cos(2 * np.pi * (theta - theta0) / period)**2 + IMin
```

Before doing the fit, you must put your estimates in an array. I expect the **`IMax`** will be about 2.5, **`IMin`** about 0.5, **`theta0`** about 0, and **`period`** about **`180`**. Usually these initial estimates do not need to be very close.

```
paramsInitial = np.array((2.0, 0, 360, 0.))
```

Next, you call the actual function that fits your model to your data. It returns some values that describe the best fit. The code also prints out the important parts of the fit.

```
popt, pcov = curve_fit(fitCos2, theta, Intensity, p0=paramsInitial)
print "Best Parameters = ", popt
perr = np.sqrt(np.diag(pcov))
print "Uncertainty in parameters = ", perr
```

The results print out after the cell is executed. They should be something like

```
Best Parameters =  [  2.3138e+00  -7.7043e+00   3.7898e+02   2.1470e-01]
Uncertainty in parameters =  [  0.14659554   7.69453671  28.85787459   0.12157423]
```

Sweet! Now we want to plot our *best fit* model with our data. I will also plot our original guess at the model. First you have to calculate the model using the optimal parameters. The code below does this.

```
(IMaxOpt, theta0Opt, periodOpt, IMinOpt) = popt
yModelOpt = fitCos2(thModel, IMaxOpt, theta0Opt, periodOpt, IMinOpt)
```

In this last figure, notice that I added a label to **`plot`** command. This allows me to add a *legend* at the end which labels each plot symbol.

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(theta, Intensity, 'ro', label='Data')
thModel = np.linspace(0., 180, 181)
IntenseModel = 2.5 * np.cos(thModel * np.pi / 180) ** 2
ax.plot(thModel, IntenseModel, 'b-', label='Guess')
ax.plot(thModel, yModelOpt, 'g--', label='Fit')

ax.set_xlabel(r'$\theta$ (deg)')
ax.set_ylabel('Intensity (arb)')
ax.set_title('Crossed Polarizers')
ax.text(10, 2.8, 'Polarizer #1')
ax.grid(True)
ax.legend(loc=3)
```

Here is my final figure. Your figure will use your data.