Instruction to use the Visualizer

Introduction:

The goal was to better visualize Rosenbluth Separations, which are measurements of electromagnetic distributions in a nucleon called form factors. It's done by making a cross section measurement of electron-nuclear scattering at a specific four-momentum transfer. In layman's terms, we measure the scattering of electrons following a collision at a specific energy, and we use the data to derive the distribution of electric and magnetic forces. Now, normally, this data would be visualized by plotting the reduced cross-section divided by the magnetic form factor against the linear-polarization parameter. From there you would fit a line to the data points and derive the line's equation to get the form factors. There is a much simpler way of displaying the data. Robert Hofstadter suggested plotting the form factors against each other, with each data point being a band. Add in the uncertainty bands and we can see the form factors where the bands overlap. Color each band based on it's energy and the plots are very easy to read.

Before you start please make sure you have downloaded the following:

- Python
- MatPlotLib Package
- Numpy Package
- Software for editing Python

Be sure to also have the following information from an elastic scattering experiment. The corresponding variables that will be used in the program have also been included:

Information	**Corresponding Variable**
Q^2	q2
Particle/Beam Energy	E
Particle/Beam Energy after scattering	E1
Angle of Spectrometer	angle
Cross-section of the Experiment	S
Error of the data point	Err

Walkthrough:

The file we will be using is titled Visualizer.py. This is the latest version of the visualizer as of writing. The program has had many versions before this and this is the most advanced and also simplest to use. This is designed to plot scattering data for electron-proton collisions.

At the top of the file, we find the import functions. Numpy and Math are imported to help with calculations. Matplotlib is imported to serve as the graphing program. All of these packages are necessary and this program will not work without them.

From this point, we find the calculation functions. Each one corresponds to a variable in the equations wea re plotting. All of these can be more clearly explained in the published paper, so I will not go into too much detail here.

First, we see the Tau function, T(q2). Tau is dependent on the mass of the proton and Q^2. The variable for the proton's mass is written after this. Then we see the parameter Epsilon (eps(q2, angle)) which is dependent on angle of scattering and Q^2. After that, we find a conversion function, GeVtonb(x). It is necessary to convert from Giga-electron Volts Squared (GeV^2) to Nanobarns (nb). The next function is sigmaMott(E,E1,angle) which corresponds to Sigma no-structure (σ_{ns}) in the paper. This function requires the Beam Energy, the Beam Energy after collision, and the angle of scattering.

Following these functions, we find the array GmList. The value of the magnetic form factor is known to be between values of 0 and 2.7. Please note that on the plot, the error bars might exceed 2.7 or go below 0. This is normal.

Now we find the main function of the program: plotGe(q2, E, E1, angle, S, Err, Color, Alpha, left, right, down, up). This function records a data point to the plot. Each point will be represented as a band. The first six inputs are data from the scattering experiments, please consult the table at the beginning of this paper for more information. The next inputs are to control the display of the graph. Color is used to control the band color for that data point. Alpha is used to control the alpha level, when you have many bands overlapping it is hard to control these variables. The final inputs are to control the display limits on the graph.

You will notice that there is a function which is very similar. This function, plotGeAdjusted(q2, E, E1, angle, S, Err, Color, Alpha, left, right, down, up), was used for a very specific purpose. During my project, I compared plots done by SLAC, some of which had normalized data (which simply means that it had been adjusted based on previously obtained data, which is done quite commonly). This function plots points with this normalization factored in. Unless you are working on the same paper as I did, this function will likely serve no purpose to you.

After these functions you will see another piece of code that is specific to my project. This is a plot of a global data fit. I have left it in as a useful example and labeled it's start and finish.

At the end of all of these previous sections, you will see where the functions are called and the data is plotted. In the next section I will explain step-by-step how to do this. This code was left as an example and can be deleted if you are adapting the visualizer to your own personal project.

Step-by-step

1. Acquire all data as detailed at the beginning of the instructions.
2. Go into Python editor and open visualizer.py.
3. Scroll to the bottom of the page where the plotGe is called. Clear example files as needed. The line where the fundamental functions of the code stops is seen below:

```
    plt.xlim(left,right)
    plt.ylim(down,up)
    plt.fill_betweenx(GmList,UpGeList,DownGeList,color=Color,edgecolor=None,alpha=Alpha)
    plt.tight_layout()
    plt.xlabel("G$_E^2$",fontsize=30)
    plt.ylabel("G$_M^2$",fontsize=30)
```

4. Write code for data fit. Compile G_E and G_M positions into lists. An example is seen below:

```
#plot of global data fit (begin)
# list of Q2
Q2List = [1.75,2.5,3.25,4,5,6,7,8.83]

# Form-Factor lists
GeFitList = []
GmFitList = []
GeFitList2 = []
GmFitList2 = []

for q in Q2List:
    GeFitList.append(((1 + ((-0.21) * T(q))) / (1 + ((12.21) * T(q)) + ((12.6) * T(q) ** 2) + ((23) * T(q) ** 3)))**2)
    GmFitList.append((2.73 * (1 + ((0.058) * T(q)))) / (1 + ((10.85) * T(q)) + ((19.9) * T(q) ** 2) + ((4.4) * T(q) ** 3)))
    GeFitList2.append(((1 + ((-0.01) * T(q))) / (1 + ((12.16) * T(q)) + ((9.7) * T(q) ** 2) + ((37) * T(q) ** 3)))**2)
    GmFitList2.append((2.73 * (1 + ((0.093) * T(q)))) / (1 + ((11.07) * T(q)) + ((19.1) * T(q) ** 2) + ((5.6) * T(q) ** 3)))
#plot of global data fit (end)
```

5. Call plotGe, or plotGeAdjusted if you are accounting for SLAC's normalization.
6. Use called function for every data point, inputting data in this order:
   - Q^2 .
   - Beam Energy.
   - Beam Energy after collision.
   - Angle of Spectrometer.
   - Area of Cross-section.
   - Recorded Error of the data.
   - Desired color for this data point.
   - Desired intensity of color.
   - Left-bound limit.

- Right-bound limit.
- Downward limit.
- Upward limit.

```
plotGe(1.75,1.511,0.578,90.066,0.144,1.750*1E-3,"r",0.2,-0.01,0.05,0,0.3)
```

It will look like the line above if done correctly.

7. After every data point has been plotted, use the matplotlib package to plot the data fit. In this program it is imported as plt.plot. Plot the data like so:
   - plt.plot(G_E Fit List, G_M Fit List, Color/Symbol Code, label= "INSERT LABEL HERE", markersize= "INSERT MARKER SIZE HERE")

```
plt.plot(GeFitList, GmFitList, "ko", label="JLab Global Fit 1", markersize=10)
plt.plot(GeFitList2, GmFitList2, "rs", label="JLab Global Fit 2", markersize=8)
```

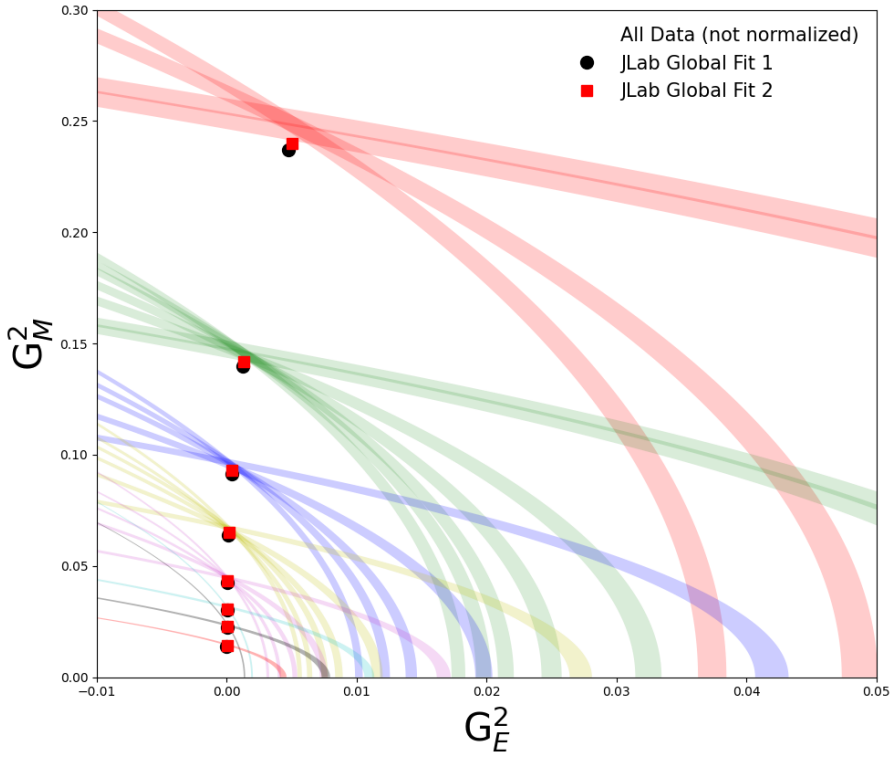
   Here's an example.
8. Add a legend using:
   - leg = plt.legend(loc=1,fontsize="INSERT FONT SIZE HERE")
   - leg.get_frame().set_linewidth(0.0)

```
leg = plt.legend(loc=1,fontsize=15)
leg.get_frame().set_linewidth(0.0)
```

9. Call either plt.show() to display the graph or plt.savefig("FILE NAME HERE . png")

```
plt.tight_layout()
plt.savefig("Q2-175.png")
#plt.show()
```

If done properly, the resulting image should look like this:


All Data (not normalized)
JLab Global Fit 1
JLab Global Fit 2
G_M^2
G_E^2
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.01
0.00
0.01
0.02
0.03
0.04
0.05