ZEEMAN EFFECT CODES FOR BOHR MAGNETON ANALYSIS

The Nuclear Physics summEr school for undEr Represented Students (NuPEERS) 2024

Written by Faith Cherop

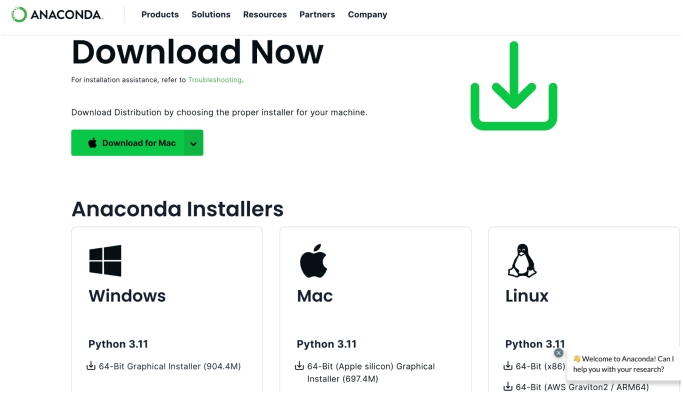
Edited by Marie Ange Ntivuguruzwa

Introduction:

The following are the steps one can follow in order to successfully analyze the Zeeman Effect data. The codes will calculate the C0 constants and Bohr Magneton when various parameters are altered.

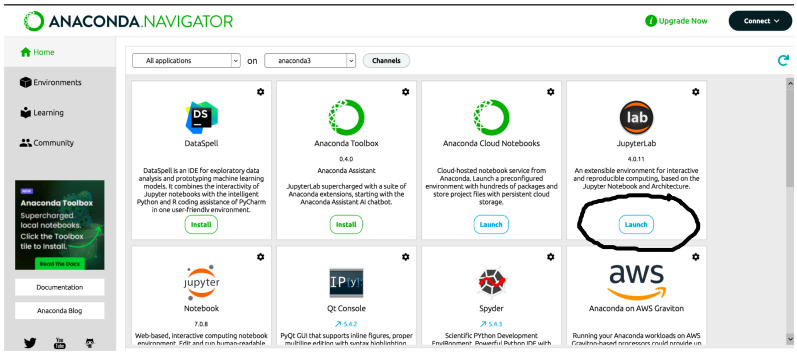
1. Accessing JupyterLab Notebook.

- Downloading anaconda
- To access the Notebook, you will first have to download the Ancadonda
- Use the link to download anaconda that works on your device. Make sure you download the correct app as downloading the wrong one might not work with your device and you won't have access to the Notebook.
- https://www.anaconda.com/download/success (if you can't access it)



- Create an account. Using the code sent to your email, verify the account to continue with the installation. You can use either of your emails but school email might be preferred.
- Install it. The installation might take some time(at least 15 mins). Kindly be patient.

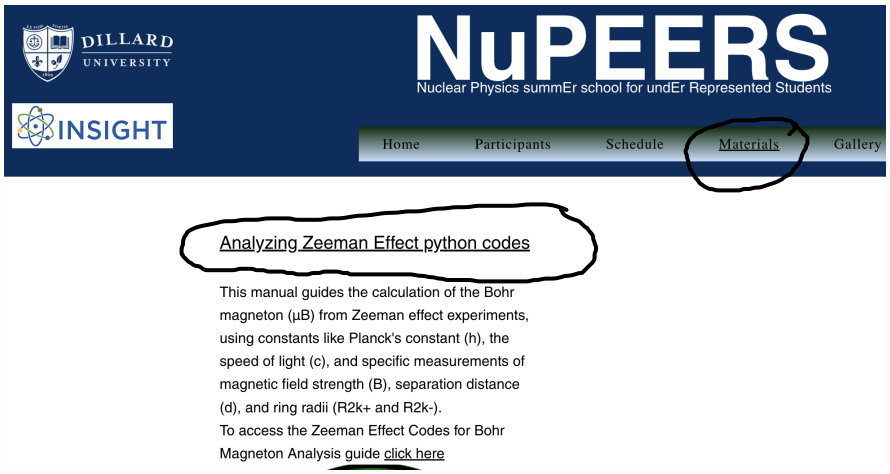
- Click on the **‘Launch’ JupyterLab**. A tab will open on your default browser. (Chrome, safari, firefox, brave, etc)



- It will pop up showing all the folders on your Laptop.
- **A window might pop up asking you to download git. Yes, download/ install it. It might take some time to download. You can do other tasks while installing it.**

2. Downloading the folder from the Website

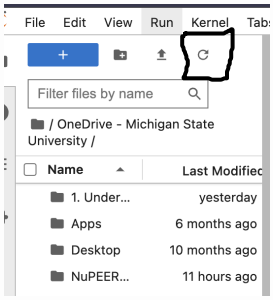
- Using the link to access the website where you can download the Folders.
- Since you can’t edit the folder, download it to your computer and unzip the file.



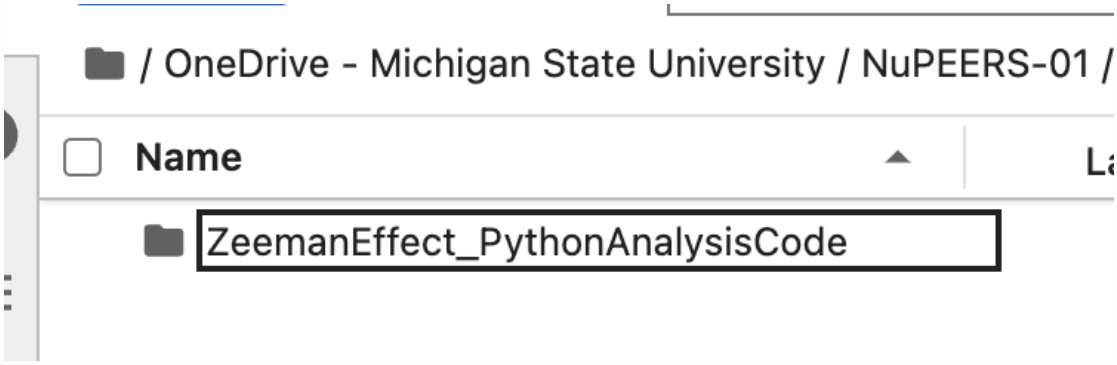
- You can also rename the unzipped folder for easier access i.e NuPEERS. **Note** the location of the folder.

3. Accessing the Folders on the Notebook

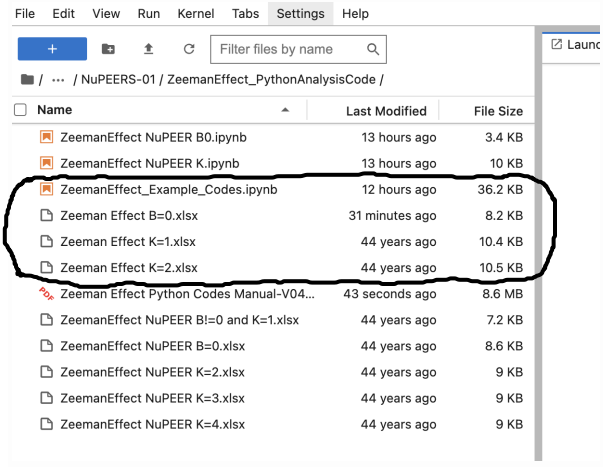
- Locate the folder you just downloaded. This can be done by navigating through the various folders/ directories you might have.
- If you can't see the folder, refresh the page by clicking on the refresh icon.



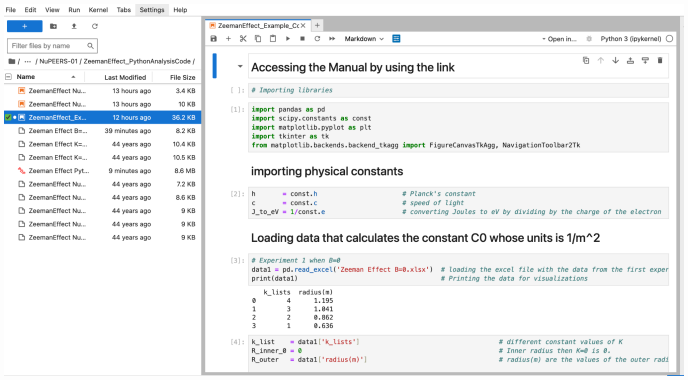
- Once you locate the folder, click on the folder to access the folder named ‘ ZeemanEffect_PythonAnalysisCode.’



- Clicking on the Folder to access the files. The following list of files will pop up. The file names with ‘NuPEERS’ are the files you will save your data in and the codes you will be working with.
- The files circled are the files showing the sample data analysis codes.
- The Python codes manual saved as PDF can also be accessed.



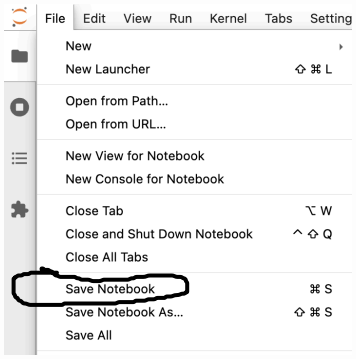
- Click on the **‘ZeemanEffect_Example_Codes.ipynb**’ file to open it in the notebook. The figure below shows the expected results.



4. Running the codes

- To run the codes, use the play icon. Run each cell with the codes.

- Remember to always save your Notebook in order to save any changes made. To do so, under File section, click on save notebook option.



- For the **'ZeemanEffect_Example_Codes.ipynb**', run each cell of codes to see the expected results.
- The following snapshots show the various expected results at various sections of the codes.

5. Zeeman codes calculating C_0 values

- In the first part of the codes calculate the C_0 values.

- **Importing libraries**

```
import pandas as pd
import scipy.constants as const
import matplotlib.pyplot as plt
import tkinter as tk
```

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
```

Results:

```
import pandas as pd
import scipy.constants as const
import matplotlib.pyplot as plt
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
```

- **Loading the dataset used in calculating the constants C_0 values**

```
# Experiment 1 when B=0
data1 = pd.read_excel('Zeeman Effect B=0.xlsx')  # loading the
excel file with the data from the first experiement.
print(data1)
```

Results:

```
# Experiment 1 when B=0
data1 = pd.read_excel('Zeeman Effect B=0.xlsx')  # loading the excel file with the data from the first experiement.
print(data1)                                     # Printing the data for visualizations
```

```
   k_lists  radius(m)
0        4      1.195
1        3      1.041
2        2      0.862
3        1      0.636
```

- **Importing constants**

```
h       = const.h                    # Planck's constant
c       = const.c                    # speed of light
J_to_eV = 1/const.e                   # converting Joules to eV
by dividing by the charge of the electron
```

Results:

```
h       = const.h                    # Planck's constant
c       = const.c                    # speed of light
J_to_eV = 1/const.e                  # converting Joules to eV by dividing by the charge of the electron
```

```
k_list                                  =        data1['k_lists']
# different constant values of K
R_inner_0                               =                       0
# Inner radius then K=0 is 0.
R_outer                                 =         data1['radius(m)']
# radius(m) are the values of the outer radius measured.
```

```
C0_list                                    =                                    []
# Creating an empty list that stores the C0 Constants
for i in range(len(R_outer)):
      C0_list.append(k_list[i] / ((R_outer[i] -R_inner_0) ** 2))
# Calculate C0 values.
print(C0_list)
```

Results:

```
k_list    = data1['k_lists']                                    # different constant values of K
R_inner_0 = 0                                                   # Inner radius then K=0 is 0.
R_outer   = data1['radius(m)']                                  # radius(m) are the values of the outer radius measured.


C0_list = []                                                     # Creating an empty list that stores the C0 Constants
for i in range(len(R_outer)):
     C0_list.append(k_list[i] / ((R_outer[i] -R_inner_0) ** 2))  # Calculate C0 values.
print(C0_list)
```

```
[2.801071409814254, 2.7683423442876642, 2.6916306436765525, 2.4722123333728887]
```

- **Adding C_0 values to the initially loaded data.**
- Adding the C_0 values to initially loaded data helps for easier visualizations.
- These C_0 values will be used in the next calculations of the Bohr Magneton.

```
new_data = {'C0_values': C0_list}
data1 = data1.assign(**new_data)
print(data1)
```

Results:

```
new_data = {'C0_values': C0_list}
data1 = data1.assign(**new_data)
print(data1)
```

```
   k_lists  radius(m)  C0_values
0        4      1.195   2.801071
1        3      1.041   2.768342
2        2      0.862   2.691631
3        1      0.636   2.472212
```

6. Calculating the bohr magneton at different k values and C_0 values

- The next code will calculate the Bohr Magneton. The snapshots show the expected results of the each cell of codes

- **Import libraries used for data analysis and visualizations**

```
import pandas as pd
import scipy.constants as const
```

```
import matplotlib.pyplot as plt
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
```

Results:

```
import pandas as pd
import scipy.constants as const
import matplotlib.pyplot as plt
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
```

- **Load some data and print for visualization**
- Delta R is the difference between the R+ and R-.
- Trial no. is the number of experiments done when K=1

```
#Experiment 2 when k=1
data2 = pd.read_excel('Zeeman Effect K=1.xlsx')
print(data2)
```

Results:

```
#Experiment 2 when k=1
data2 = pd.read_excel('Zeeman Effect K=1.xlsx')
print(data2)
```

	Trial_no	R+	R-	Delta_R
0	1	0.661	0.606	0.055
1	2	0.661	0.606	0.055
2	3	0.661	0.603	0.058
3	4	0.660	0.604	0.056
4	5	0.660	0.607	0.053

- **Given variables and importing constants.**

```
#Constants
h       = const.h                       # Planck's constant
c       = const.c                       # speed of light
J_to_eV = 1/const.e# converting Joules to eV by dividing by the
charge of the electron


k  = 1
C0  =  2.472212        # 1/m^2, value obtained from the previous
calculations
```

```
d  = 1.995e-3      # Separation in meters
##
B  = 1.1           # Magnetic field in T

R_inner_minus = data2['R-']                  # R- inner radius from
the loaded data
R_outer_plus   = data2['R+']                      # R+ outer radius
from the loaded data
```

Results:

```
#Constants
h       = const.h                    # Planck's constant
c       = const.c                    # speed of light
J_to_eV = 1/const.e                  # converting Joules to eV by dividing by the charge of the electron


k  = 1
C0 = 2.472212                         # 1/m^2, value obtained from the previous calculations
d  = 1.995e-3                         # Separation in meters
B  = 1.1                              # Magnetic field in T

R_inner_minus = data2['R-']             # R- inner radius from the loaded data
R_outer_plus  = data2['R+']              # R+ outer radius from the loaded data
```

- **Calculating bohr magneton in J/T**

```
Bohr_magneton = [] # in J per Tesla
for j in range(len(R_inner_minus)):
      Bohr_magneton.append(((h * c * C0) / (2 * B * d)) *
((R_outer_plus[j] ** 2) - (R_inner_minus[j] ** 2)))   # Bohr
magneton calculation

print('The bohr magneton values in J/T is:', Bohr_magneton)
```

Results:

```
Bohr_magneton = [] # in J per Tesla

for j in range(len(R_inner_minus)):
    Bohr_magneton.append(((h * c * C0) / (2 * B * d)) * ((R_outer_plus[j] ** 2) - (R_inner_minus[j] ** 2)))  # Bohr magneton cal

print('The bohr magneton values in J/T is:', Bohr_magneton)
```

```
The bohr magneton values in J/T is: [7.797154970669215e-24, 7.797154970669215e-24, 8.202985222209972e-24, 7.920123662823418e-2
4, 7.513622062644883e-24]
```

- **Changing the Bohr Magneton Values in ev/T**

```
Bohrmagneton =[]
for val in range(len(Bohr_magneton)):
    Bohrmagneton.append( Bohr_magneton[val] *J_to_eV)

print('The bohr magneton values in ev/T is:', Bohrmagneton)
```

Results:

```
Bohrmagneton =[]
for val in range(len(Bohr_magneton)):
    Bohrmagneton.append( Bohr_magneton[val] *J_to_eV)
print('The bohr magneton values in ev/T is:', Bohrmagneton)
```

```
The bohr magneton values in ev/T is: [4.866601350440875e-05, 4.866601350440875e-05, 5.119900670209108e-05, 4.9433523712363785e-05, 4.689634028606663e-05]
```

- **Adding a new column to the initially loaded datasets.**

```
data = {'Bohr_Magneton': Bohr_magneton}
data2 = data2.assign(**data)
print(data2)
```

Results:

```
data = {'Bohr_Magneton': Bohr_magneton}
data2 = data2.assign(**data)
print(data2)
```

	Trial_no	R+	R-	Delta_R	Bohr_Magneton
0	1	0.661	0.606	0.055	7.797155e-24
1	2	0.661	0.606	0.055	7.797155e-24
2	3	0.661	0.603	0.058	8.202985e-24
3	4	0.660	0.604	0.056	7.920124e-24
4	5	0.660	0.607	0.053	7.513622e-24

- **Plotting using matplotlib and tkinter.**

```
def plot():
    x_var = x_variable.get()
    y_var = y_variable.get()

    fig, ax = plt.subplots()
    ax.scatter(data2[x_var], data2[y_var])
    ax.set_xlabel(x_var)
    ax.set_ylabel(y_var)
    ax.set_title(f'{y_var} vs {x_var}')

    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
          canvas.get_tk_widget().pack(side=tk.TOP,  fill=tk.BOTH, expand=1)

              toolbar   =   NavigationToolbar2Tk(canvas,   root, pack_toolbar=False)
```

```
    toolbar.update()
    toolbar.pack(side=tk.BOTTOM, fill=tk.X)
```

Results:

```
def plot():
    x_var = x_variable.get()
    y_var = y_variable.get()

    fig, ax = plt.subplots()
    ax.scatter(data2[x_var], data2[y_var])
    ax.set_xlabel(x_var)
    ax.set_ylabel(y_var)
    ax.set_title(f'{y_var} vs {x_var}')

    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

    toolbar = NavigationToolbar2Tk(canvas, root, pack_toolbar=False)
    toolbar.update()
    toolbar.pack(side=tk.BOTTOM, fill=tk.X)
```

- **Creating a window**

```
root = tk.Tk()
root.title("Bohr Magneton Plot")
```

Results:

```
root = tk.Tk()
root.title("Bohr Magneton Plot")
```

- **Dropdown menus for selecting x and y variables**

```
left_frame = tk.Frame(root)
left_frame.pack(side=tk.LEFT, anchor='n')

# X variable selection
tk.Label(left_frame, text="Select X Variable:").pack()
x_variable = tk.StringVar(root)
columns = list(data2.columns)
x_menu = tk.OptionMenu(left_frame, x_variable, *columns)
x_menu.pack()

# Label to display the selected X variable
```

```
x_selected_label = tk.Label(left_frame, text="Selected X Variable: ")
x_selected_label.pack()

# Y variable selection
tk.Label(left_frame, text="Select Y Variable:").pack()
y_variable = tk.StringVar(root)
y_menu = tk.OptionMenu(left_frame, y_variable, *columns)
y_menu.pack()

# Label to display the selected Y variable
y_selected_label = tk.Label(left_frame, text="Selected Y Variable: ")
y_selected_label.pack()
```

Results:

```
left_frame = tk.Frame(root)
left_frame.pack(side=tk.LEFT, anchor='n')

# X variable selection
tk.Label(left_frame, text="Select X Variable:").pack()
x_variable = tk.StringVar(root)
columns = list(data2.columns)
x_menu = tk.OptionMenu(left_frame, x_variable, *columns)
x_menu.pack()

# Label to display the selected X variable
x_selected_label = tk.Label(left_frame, text="Selected X Variable: ")
x_selected_label.pack()

# Y variable selection
tk.Label(left_frame, text="Select Y Variable:").pack()
y_variable = tk.StringVar(root)
y_menu = tk.OptionMenu(left_frame, y_variable, *columns)
y_menu.pack()

# Label to display the selected Y variable
y_selected_label = tk.Label(left_frame, text="Selected Y Variable: ")
y_selected_label.pack()
```

- **Add a button to trigger the plot and update the label variables**

```
button = tk.Button(left_frame, text="Plot", command=plot)
button.pack()

def update_selected_labels():
    x_selected_label.config(text=f"Selected X Variable: {x_variable.get()}")
    y_selected_label.config(text=f"Selected Y Variable: {y_variable.get()}")
```

```
# Update selected labels when variables change
x_variable.trace_add('write',              lambda              *args:
update_selected_labels())
y_variable.trace_add('write',              lambda              *args:
update_selected_labels())
```

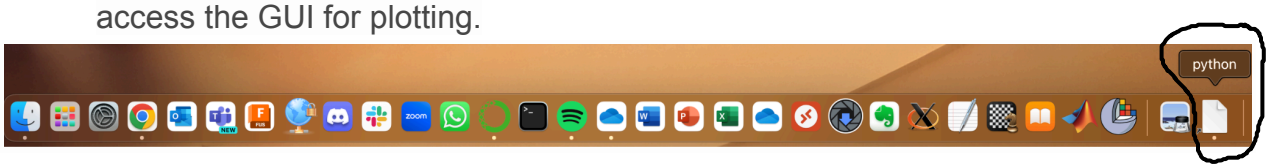
Results:

```
button = tk.Button(left_frame, text="Plot", command=plot)
button.pack()

def update_selected_labels():
    x_selected_label.config(text=f"Selected X Variable: {x_variable.get()}")
    y_selected_label.config(text=f"Selected Y Variable: {y_variable.get()}")

# Update selected labels when variables change
x_variable.trace_add('write', lambda *args: update_selected_labels())
y_variable.trace_add('write', lambda *args: update_selected_labels())
```

- **A text box with instructions to be displayed in the GUI and root.mainloop()**

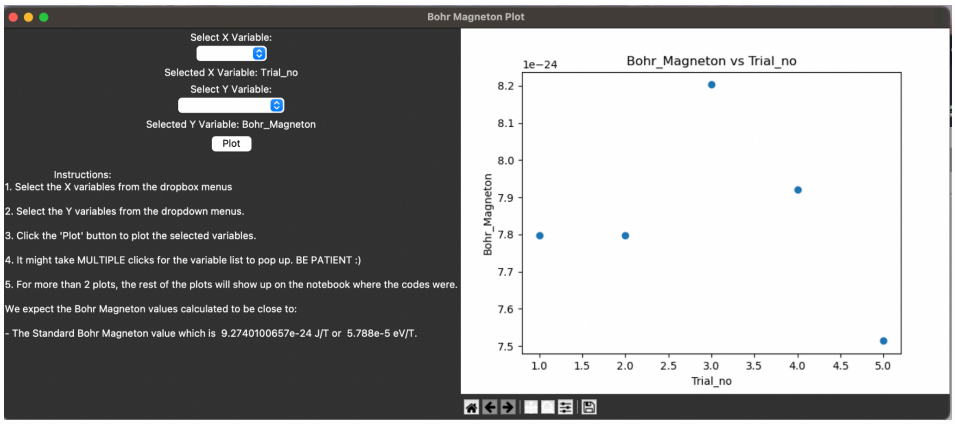
```
instructions_text = """
                Instructions:
1. Select the X variables from the dropbox menus

2. Select the Y variables from the dropdown menus.

3. Click the 'Plot' button to plot the selected variables.

4. It might take MULTIPLE clicks for the variable list to pop up. BE PATIENT :)

5. For more than 2 plots, the rest of the plots will show up on the notebook where the codes were.

We expect the Bohr Magneton values calculated to be close to:

- The Standard Bohr Magneton value which is  9.2740100657e-24 J/T or  5.788 x 10-5 eV/T-1.
"""
instructions_label = tk.Label(left_frame, text=instructions_text, justify=tk.LEFT)
instructions_label.pack()

root.mainloop()
```

- Keeps the Tkinter application window open and responsive for user interactions.
- Executes what we wish to execute in the application.
- A pop up window will appear on your screen. If you don't see it, you might see a small pop up on your toolbar on your computer('Python'). Click on it and you will access the GUI for plotting.



- You might also have to minimize your tabs so that you can get to see the pop up. The screenshot below shows an example of the expected window.

- Select the Y-variables and X-variables then click on 'Plot'. This is also shown on the Instructions bar.
- The following snapshot shows the expected graph.

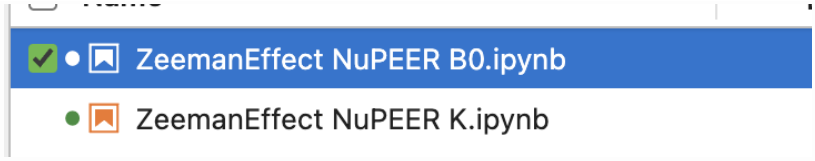


Zeeman Effect NuPEERS Experiment

Accessing the ZeemanEffect python codes

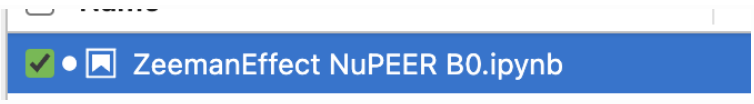
1. C_0 Calculations

- For NuPEER data analysis, you will use the following files:



- The notebook for calculating the C_0 values is different from the one used in calculating the Bohr Magneton values. Make sure you use the right Notebook.

- The file below has the codes that calculates the C_0 values.



- In the file you need to:
  - ☐ change the file name to match the expected file to be used.
  - ☐ Run each cell of codes.
  - ☐ Take note of the final C_0 values which will be used in the Bohr magneton calculation.

2. Bohr Magneton Calculations

- In this calculation, use the following file:

- In this file you need to:
  - ☐ Change the file name to match the file with the data
  - ☐ Run each cell of codes.
  - ☐ Access the GUI.
  - ☐ Plot the data.

3. Some Errors

- ★ When the Kernel isn't responding or when the cells don't show any output when they are supposed to, save the notebook then refresh the notebook.
- ★ If this persists for long, close the tab and 'Launch' the JupyterLab again.

Formulas used:

$$\mu_B = \frac{(h*c*C_0)}{2*B*d}\left(R^2_{k+} - R^2_{k-}\right)$$

$\mu_B = Bohr\ magneton$

h- planck's constant

c-speed of light

C_0 - constant ($\frac{1}{m^2}$)

B- magnetic field in Tesla

d- separation distance in meters

R^2_{k+}- Outer Ring Radius

R^2_{k-} - Inner Ring Radius

$$C_0 = \frac{k}{\left(R^2_k - R^2_0\right)}$$

$k - constant$

R^2_k - Outer ring radius

R_0 - = 0

References:

1. *Jupyter-pandas-Gui*. PyPI. (n.d.). https://pypi.org/project/jupyter-Pandas-GUI/
2. Anaconda. (2024, April 4). *Download Now | Anaconda*.

https://www.anaconda.com/download/success

3. GeeksforGeeks. (2024, May 9). *Python Tkinter*. GeeksforGeeks.

   https://www.geeksforgeeks.org/python-gui-tkinter/

★ We thank Y.G. for his contributions!