Importing data into Python

We will use CSV formatted data files:

- csv = comma separated values
- You can save data in Excel or Google Sheets to a ".csv" file

Uploading a data file into the Colab environment

You can upload your .csv file into Colab with the following steps:

- Click on the "folder" symbol on the vertical left menu.
- A list of files & folders available to Colab will appear (e.g. "sample_data").
- Right-click in this area and select the "upload" option from the menu.
- Select your file from your computer and upload it.
- Alternative: Drag-and-drop your file into this part of the browser.
- The file should appear in the list of files available to Colab.

Note: Colab will erase this file after a period of inactivity.

Importing data from a file into your Python program

You can import CSV data into Python using the following code:

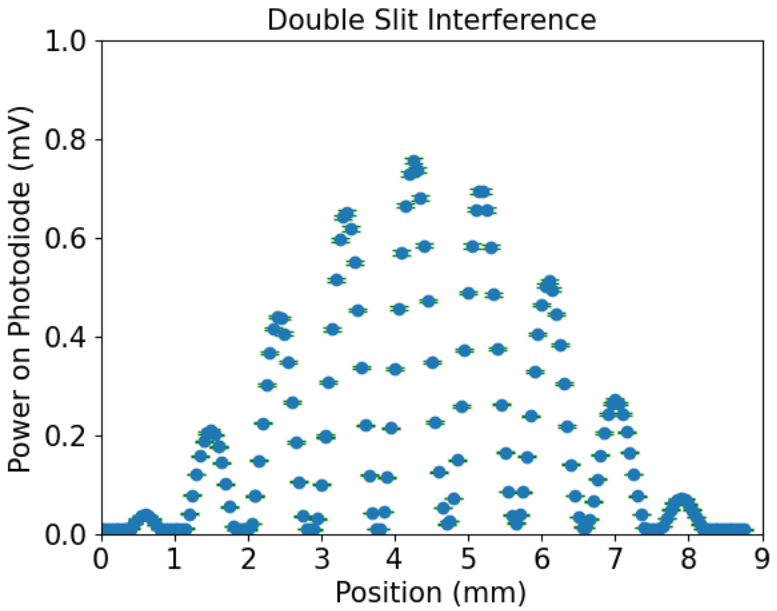
```
# Load data
import pandas as pd
df = pd.read_csv('data_doubleslit_2.csv')     # Read data file
x=df.get('position').to_numpy()       # Put data in the 'position' column into the x array
y=df.get('power').to_numpy()         # Put data in the 'counts' column into the y array
e=df.get('error').to_numpy()          # Put data in the 'error' column into the e array

# NOTE: The text header is automatically removed.

# Convert data to float format
x_data=x*1.0
y_data=y*1.0
y_error=e*1.0

# Optional print statement to see if loaded data are present
# print("x = ", x_data[0], "y = ", y_data[0], "error = ", y_error[0])
# print("x = ", x_data[1], "y = ", y_data[1], "error = ", y_error[1])
```

Plot the data

We plot the data to make sure that it loaded properly and to estimate parameters for curve fitting.

```
# import library modules needed for plotting and data analysis
import numpy as np
import matplotlib.pyplot as plt

# import library module for curve fitting
from scipy.optimize import curve_fit

# Plot the data on a new figure
fig1=plt.figure(1)                                          # create new figure for plotting
plt.plot(x_data, y_data, 'blue', linestyle='none', marker='o', markerfacecolor='blue', markersize=3)  # make the plot with round markers
plt.title('Double Slit Interference', fontsize=15)                          # Add the plot title

plt.ylabel('Power on Photodiode (mV)', fontsize=15)                    # Add the y-axis label, "fontsize" is optional
plt.yticks(size=15)                                         # adjust the size of the y-axis tick number labels

plt.xlabel('Position (mm)', fontsize=15)                         # Add the x-axis label, "fontsize" is optional
plt.xticks(size=15)                                         # adjust the size of the x-axis tick number labels

plt.xlim(0.0, 9.0)                                         # Set the range of the plot's x-axis
plt.ylim(0.0, 1.0)                                          # Set the range of the plot's y-axis
```

```
plt.errorbar(x_data, y_data, y_error, fmt='o', ecolor='green', capsize=5)   # Add error bars

fig1.show()
```



Double-Slit Interference Pattern Fit

We want to fit the data to sinusoid function for double-slit interference (including the single diffraction envelope):

$$y(x) = B + A\cos^2(c_d(x - x_0))\left[\frac{\sin(c_s(x - x_0))}{c_s(x - x_0)}\right]^2$$

The parameters of the function are:

- $B =$ y-axis offset (background)
- $A =$ amplitude of the double slit interference pattern
- $c_d =$ "frequency" constant for the double-slit pattern
- $c_s =$ "frequency" constant for the single-slit pattern
- $x_0 =$ x-axis offset, i.e. the center position of double-slit pattern

The "frequency" constants for the single-slit and double-slit sinusoids are defined as:

$$c_d = \frac{\pi d}{\lambda l}$$

$$c_s = \frac{\pi a}{\lambda l}$$

Here the additional physical parameters for the double-slit interference experiment are the following:

- $\lambda =$ wavelength of laser or lamp
- $l =$ distance between slits and detector
- $a =$ width of the slits
- $d =$ distance between the two slits

Non-Linear Least squares fitting

We use the standard non-linear least squares method that we have used throughout the course.

```
# Estimate the c_s and c_d "frequencies" for the sinusoid functions
pi = np.pi              # pi=3.1415926
Lambda = 650.0e-9       # Lambda = 650 nm, note: "lambda" (lowercase l) is a protected name
l = 0.5                 # l = 50 cm (slits-to-detector distance)
```

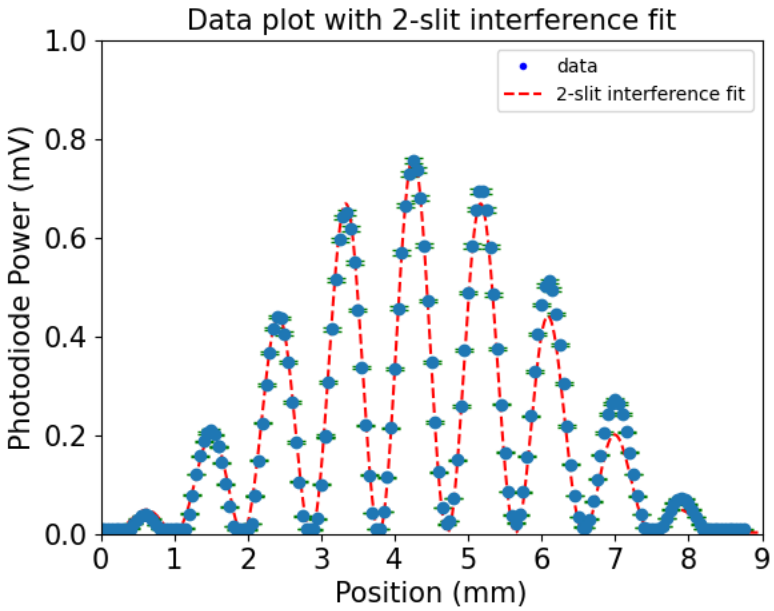
```
a = 0.1e-3              # a = 0.1 mm (slit width)
d = 0.4e-3              # d = 0.4 mm (slit separation)

c_s_guess = pi*a/(Lambda*l)
c_d_guess = pi*d/(Lambda*l)

print("c_s_guess =", c_s_guess)
print("c_d_guess =", c_d_guess)
```

```
c_s_guess = 966.643893412244
c_d_guess = 3866.575573648976
```

```
# import library module for curve fitting
from scipy.optimize import curve_fit

# Non-linear function with parameters a = y-offset, b = amplitude, c = "frequency", d = x-offset
def yModel(x, Background, Amplitude, c_s, c_d, X0):
  x_shift_meters = (x - X0)/1000.0       # center x on origin and convert from mm to meters
  cos_squared = np.power(np.cos(c_d*x_shift_meters),2)
  sin_squared = np.power(np.sin(c_s*x_shift_meters)/(c_s*x_shift_meters),2)
  return Background + Amplitude*cos_squared*sin_squared

# Fit the data with the linear model
initialParameters = [0.01, 0.8, 900.0, 3500.0, 4.2111]           # initial guess for fit parameters [B, A, c_s, c_d, X0]
finalParameters, finalParameterErrors = curve_fit(yModel, x_data, y_data, initialParameters, y_error, True)

# Extract Fit values and errors for a=y-offset, b=amplitude, and c=x-offset
B_fit = finalParameters[0]
B_error_squared = finalParameterErrors[0,0]
B_error = np.sqrt(B_error_squared)

A_fit = finalParameters[1]
A_error_squared = finalParameterErrors[1,1]
A_error = np.sqrt(A_error_squared)

c_s_fit = finalParameters[2]
c_s_error_squared = finalParameterErrors[2,2]
c_s_error = np.sqrt(c_s_error_squared)

c_d_fit = finalParameters[3]
c_d_error_squared = finalParameterErrors[3,3]
c_d_error = np.sqrt(c_d_error_squared)

X0_fit = finalParameters[4]
X0_error_squared = finalParameterErrors[4,4]
X0_error = np.sqrt(X0_error_squared)

# Evaluate the quality of the fit
yModel_i = yModel(x_data, B_fit, A_fit, c_s_fit, c_d_fit, X0_fit)              # Calculate the Y-array (yModel_i) for the values of the fit at the x_data po
residuals_y = y_data - yModel_i                                 # Calculate the difference between data and model (linear fit), i.e. the residuals
residuals_y_normalized = residuals_y/y_error            # Normalize the residuals to the Y-error on each data point

Chi_squared = np.sum(residuals_y_normalized**2)             # Calculate the Chi^2 for the data and fit
DOF = len(y_data)-len(finalParameters)                      # Calculate the degrees of freedom: DOF = number of data points - number of fit parameters
Reduced_Chi_squared = Chi_squared/DOF                       # Calculate the reduced Chi^2, which determines the quality of the fit


# Output the fit parameter values and errors
print("Model parameters:")
print("Background = ", B_fit, "+/-", B_error)
print("Amplitude = ", A_fit, "+/-", A_error)
print("c_s = ", c_s_fit, "+/-", c_s_error)
print("c_d = ", c_d_fit, "+/-", c_d_error)
print("X0 = ", X0_fit, "+/-", X0_error)
print("")
print("Fit quality")
print("Chi^2 = ", Chi_squared)
print("Reduced Chi^2 =", Reduced_Chi_squared)
```

```
Model parameters:
Background =  0.002773836001183536 +/- 8.397688891048001e-05
Amplitude =  0.7607261585033165 +/- 0.0008189568160720726
c_s =  676.5388474011796 +/- 0.3805486647910954
c_d =  3377.720680046194 +/- 0.2604227568738885
X0 =  4.252733513537623 +/- 0.00011938966147984833

Fit quality
Chi^2 =  54587.75845933077
Reduced Chi^2 = 313.7227497662688
```

```
# Make a new plot with the original data and the non-linear fit

fig3=plt.figure(3)                                          # create new figure for plotting

# Original code for plotting data points with error bars, but with legend label added
#############################################################################################
plt.plot(x_data, y_data, 'blue', linestyle='none', marker='o', markerfacecolor='blue', markersize=3, label='data')  # make the plot with round markers
plt.title('Data plot with 2-slit interference fit', fontsize=15)                          # Add the plot title

plt.ylabel('Photodiode Power (mV)', fontsize=15)                    # Add the y-axis label, "fontsize" is optional
plt.yticks(size=15)                                                 # adjust the size of the y-axis tick number labels

plt.xlabel('Position (mm)', fontsize=15)                           # Add the x-axis label, "fontsize" is optional
plt.xticks(size=15)                                                 # adjust the size of the x-axis tick number labels

plt.xlim(0.0, 9.0)                                                 # Set the range of the plot's x-axis
plt.ylim(0.0, 1.0)                                                  # Set the range of the plot's y-axis

plt.errorbar(x_data, y_data, y_error, fmt='o', ecolor='green', capsize=5)   # Add error bars
#############################################################################################

# Generate the points for the fitting curve (linear function in this case)
x_fit = np.linspace(0.0,9.0,1000)                   # Generate an ordered array of 100 x-values between -0.3 and 4.3
y_fit = yModel(x_fit, B_fit, A_fit, c_s_fit, c_d_fit, X0_fit)                 # Generate the y-values for the fitting curve (linear in this case)

plt.plot(x_fit,y_fit, 'r--', label='2-slit interference fit')                        # Plot the fit as a dashed red line on top of the original plot

# Add in legend
plt.legend()

plt.show()
```




```
# Save the plot to your computer
from google.colab import files               # import the library for loading/saving files to/from Google Colab environment

figure_filename='DoubleSlitInterference_Laser_withFit_v2.pdf'   # string variable that contains the filename, including the extension.
fig3.savefig(figure_filename,bbox_inches='tight')               # Saves the fig2 figure to the Google colab environment. Extension defines the file format.
files.download(figure_filename)                                 # Download the file "figure_filename" from the Google colab environment to your computer.
```

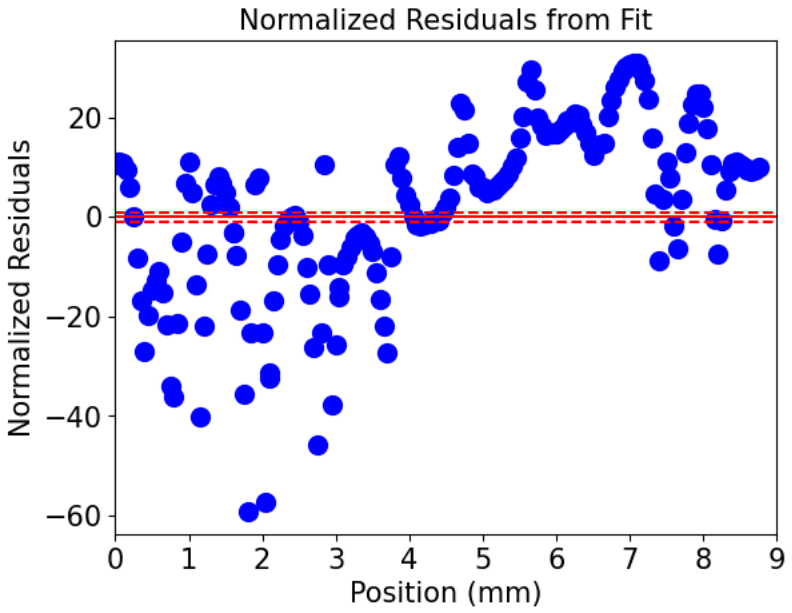
Residuals

Last, we plot the **normalized residuals** to inspect the quality of the fit for all x-values

```
# Plot the normalized residuals from the non-linear fit

fig4=plt.figure(4)                                          # create new figure for plotting
```

```
plt.plot(x_data, residuals_y_normalized, 'blue', linestyle='none', marker='o', markerfacecolor='blue', markersize=10, label='data label')  # make the plot w
plt.title('Normalized Residuals from Fit', fontsize=15)                          # Add the plot title

plt.ylabel('Normalized Residuals', fontsize=15)                    # Add the y-axis label, "fontsize" is optional
plt.yticks(size=15)                                               # adjust the size of the y-axis tick number labels

plt.xlabel('Position (mm)', fontsize=15)                           # Add the x-axis label, "fontsize" is optional
plt.xticks(size=15)                                               # adjust the size of the x-axis tick number labels

plt.xlim(0.0, 9.0)                                                # Set the range of the plot's x-axis
#plt.ylim(-75.0, 75.0)                                                 # Set the range of the plot's y-axis

# Add in the horizontal red line guides
x_lines = np.array([0.0, 9.0])
y_lines = np.array([1.0, 1.0])
plt.plot(x_lines,0.0*y_lines, 'r-')
plt.plot(x_lines,y_lines, 'r--')
plt.plot(x_lines,-y_lines, 'r--')
```

```
[<matplotlib.lines.Line2D at 0x7aa55416a260>]
```




```
# Plot the plain residuals from the non-linear fit

fig5=plt.figure(5)                                                # create new figure for plotting

plt.plot(x_data, residuals_y, 'blue', linestyle='none', marker='o', markerfacecolor='blue', markersize=10, label='data label')  # ma
plt.title('Residuals from Fit', fontsize=15)                             # Add the plot title

plt.ylabel('Residuals', fontsize=15)                       # Add the y-axis label, "fontsize" is optional
plt.yticks(size=15)                                               # adjust the size of the y-axis tick number labels

plt.xlabel('Position (mm)', fontsize=15)                           # Add the x-axis label, "fontsize" is optional
plt.xticks(size=15)                                               # adjust the size of the x-axis tick number labels

plt.xlim(0.0, 9.0)                                                # Set the range of the plot's x-axis
#plt.ylim(-75.0, 75.0)                                                 # Set the range of the plot's y-axis

plt.errorbar(x_data, residuals_y, y_error, fmt='o', ecolor='green', capsize=5)   # Add error bars

# Add in the horizontal red line guides
x_lines = np.array([0.0, 9.0])
y_lines = np.array([1.0, 1.0])
plt.plot(x_lines,0.0*y_lines, 'r-')
#plt.plot(x_lines,y_lines, 'r--')
#plt.plot(x_lines,-y_lines, 'r--')
```

```
[<matplotlib.lines.Line2D at 0x7aa554053f70>]
```