Objective

We want to fit a linear function

$$y(x) = mx + b$$

to data that has linear-like behavior. The parameters of the function are:

- $m =$ slope of the line
- $b =$ the y-axis intercept of the line (at $x = 0$)

Least squares fitting

By "fitting" the data, we mean that the parameters of the model are allowed to vary until the curve (line) makes the best description of the data. We define "best" to be the set of parameters that minimize χ^2, which is the sum of the squared y-distances between the data and the curve. The squared sum χ^2 is defined as

$$\chi^2 = \sum_i \frac{(y_i - y(x_i))^2}{\sigma_{y_i}^2}$$

where the sum over i is over all the data points. The variables are the following:

- y_i are the individual data points.
- σ_{y_i} are the errors on the individual data points.
- $y(x_i)$ is the value of the linear function at the data point with $x = x_i$.

Minimizing χ^2

We will use the "curve_fit" command in the **scipy** library to minimize χ^2. Conveniently, this command automatically returns the best fit values for m and b, as well as the errors on them. These are 1-σ errors, which means that if you repeat the process with new data , i.e., repeat experiment, then you will get m and b within this error bar range 67% of the time.

Quality of the fit

We evaluate the "quality" of the fit by looking at the "reduced χ^2", χ^2_ν, which is given by

$$\chi^2_\nu = \frac{\chi^2}{\text{degrees of freedom}}$$

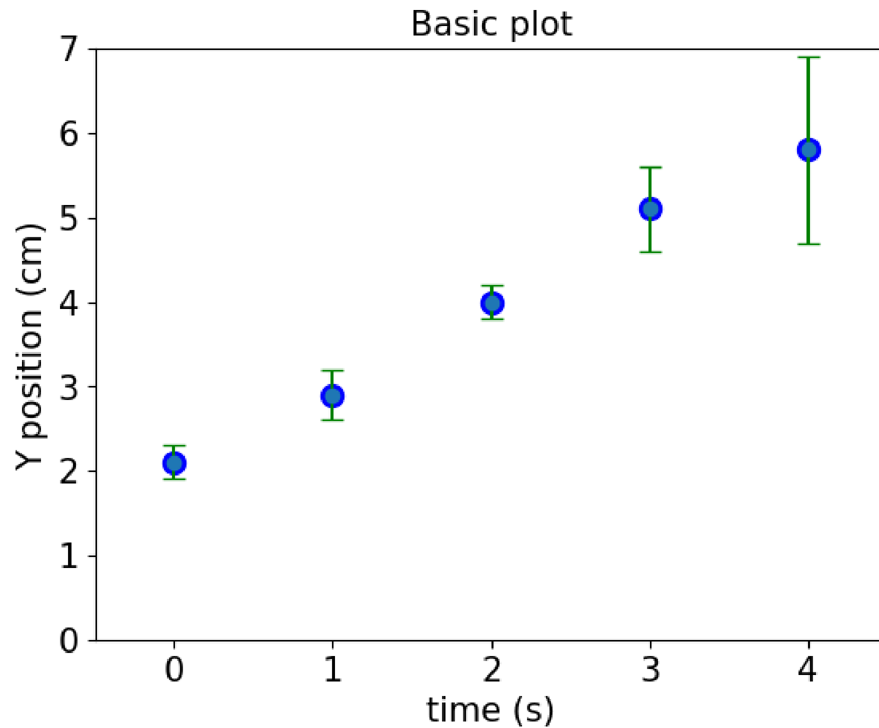
The *degrees of freedom* are given by

$$\nu = \text{degrees of freedom} = \# \text{ of data points} - \# \text{ of fit parameters}$$

In general, you want $\chi^2_\nu \simeq 1$. If this is not the case, then there are two main possibilities:

- If $\chi^2_\nu \gg 1$, then the errors bars are likely to be **underestimated**, *or* the model function **does not** describe the data well (statistically speaking).
- If $\chi^2_\nu \ll 1$, then the error bars are likely to be **overestimated**, meaning that the model function describe the data better than it should (statistically speaking).

```
# 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

# Data to be plotted and fitted
x_data=np.array([0, 1, 2, 3, 4])                       # time in seconds
y_data=np.array([2.1, 2.9, 4, 5.1, 5.8])               # Y-position in cm

y_errorbars=np.array([0.2, 0.3, 0.2, 0.5, 1.1])       # Error on y_data

# 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=10)  # make the plot with round markers
plt.title('Basic plot', fontsize=15)                            # Add the plot title

plt.ylabel('Y position (cm)', 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('time (s)', 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(-0.5, 4.5)                                             # Set the range of the plot's x-axis
plt.ylim(0.0, 7.0)                                              # Set the range of the plot's y-axis

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

fig1.show()
```



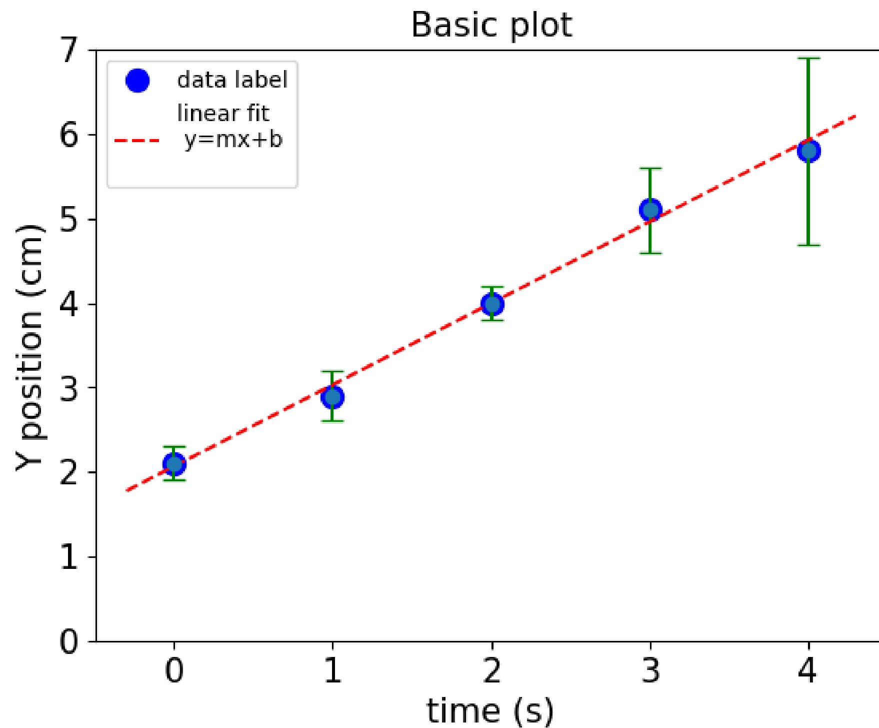

```
# Linear function with parameters m=slope and b=intercept
def yModel(x, m, b):
  return m * np.float_(x) + b


# Fit the data with the linear model
initialParameters = [1.5,2]           # initial guess for fit parameters [m,b]
finalParameters, finalParameterErrors = curve_fit(yModel, x_data, y_data, initialParameters, y_errorbars, True)

# Extract Fit values and errors for m=slope and b=intercept
m_fit = finalParameters[0]
m_error_squared = finalParameterErrors[0,0]
m_error = np.sqrt(m_error_squared)

b_fit = finalParameters[1]
b_error_squared = finalParameterErrors[1,1]
b_error = np.sqrt(b_error_squared)

m_b_covariance_error2 = finalParameterErrors[1,0]

# Evaluate the quality of the fit
yModel_i = yModel(x_data, m_fit, b_fit)                     # Calculate the Y-array (yModel_i) for the values of the fit at the x_data points
```

```
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_errorbars           # 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("m =", m_fit, "+/-", m_error)
print("b = ", b_fit,"+/-", b_error)
print("")
print("covariance error (squared) =", m_b_covariance_error2)
print("")
print("Fit quality")
print("Reduced Chi^2 =", Reduced_Chi_squared)
```

```
Model parameters:
m = 0.9658100007040309 +/- 0.11803771685912197
b =  2.0642470457916993 +/- 0.18406343736110187

covariance error (squared) = -0.01614723409364721

Fit quality
Reduced Chi^2 = 0.10343054504530728
```

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

fig2=plt.figure(2)                                            # 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=10, label='data label')  # make the plot with round marker
plt.title('Basic plot', fontsize=15)                          # Add the plot title

plt.ylabel('Y position (cm)', 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('time (s)', 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(-0.5, 4.5)                                           # Set the range of the plot's x-axis
plt.ylim(0.0, 7.0)                                            # Set the range of the plot's y-axis

plt.errorbar(x_data, y_data, y_errorbars, 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(-0.3,4.3,100)                    # Generate an ordered array of 100 x-values between -0.3 and 4.3
y_fit = yModel(x_fit, m_fit, b_fit)                  # Generate the y-values for the fitting curve (linear in this case)

plt.plot(x_fit,y_fit, 'r--', label='linear fit\n y=mx+b\n ')                         # 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='Python_plot_withFit_v3.pdf'        # string variable that contains the filename, including the extension.
fig2.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.
```

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

```
fig2=plt.figure(2)                                              # create new figure for plotting

plt.plot(x_data, residuals_y_normalized, 'blue', linestyle='none', marker='o', markerfacecolor='blue', mar
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
plt.yticks(size=15)                                             # adjust the size of the y-axis tick number

plt.xlabel('time (s)', fontsize=15)                             # Add the x-axis label, "fontsize" is optic
plt.xticks(size=15)                                             # adjust the size of the x-axis tick number

plt.xlim(-0.5, 4.5)                                             # Set the range of the plot's x-axis
plt.ylim(-2.0, 2.0)                                              # Set the range of the plot's y-axis

#plt.errorbar(x_data, y_data, y_errorbars, fmt='o', ecolor='green', capsize=5)   # Add error bars
x_lines = np.array([-0.5, 4.5])
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 0x7a89aa5eb280>]
```