Exercise

DMPy

DATA MODELLING WITH PYTHON

AIMS

(a) Impose physical models on real data to extract meaningful parameters

(b) Use the **Python** to perform least squares fitting, linear regression and report errors

(c) Use **Python** to integrate data and compare to analytical model

(d) Use **Python** to plot data

SKILLS USED IN THIS EXERCISE

- Data analysis, including linear and non-linear fitting
- Critical evaluation of a physical model
- Extraction of physical parameters using a physical model

INTRODUCTION

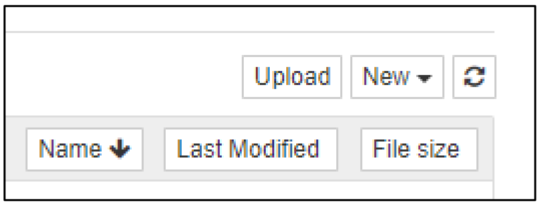
Many of the experiments that you will perform in CHEM2011 will require you to be able to manipulate data using computer software. For this purpose we often use **Microsoft Excel**, as it is versatile and commonly found in the workplace. However, the programming language of **Python** has recently become one of the most powerful and popular tools available to analyse data. In this exercise you will learn how to use Python in a web browser to analyse experimental data.

Part 1: Download and plot your data

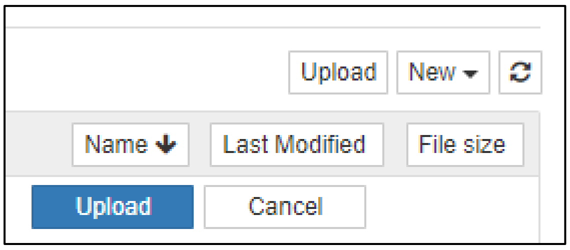
You will find several data files available on Moodle. Navigate and open the files labeled **data1.csv** and **data2.csv**. **Microsoft Excel** should automatically recognize and open these files. Save these locally.

1. In your web browser, navigate to https://jupyter.org/try
2. Click on **Try Classic Notebook**.

3. Open a new notebook through the File menu: **File > New Notebook > Python 3**
4. Upload the .csv files into the browser by **File** > **Open**, then **Upload**



Click on the blue Upload button to complete the process. You should now see the file in your binder.



The code for this Exercise is available on Moodle. Using this text will prevent errors due to character conversion using **Copy + Paste**.

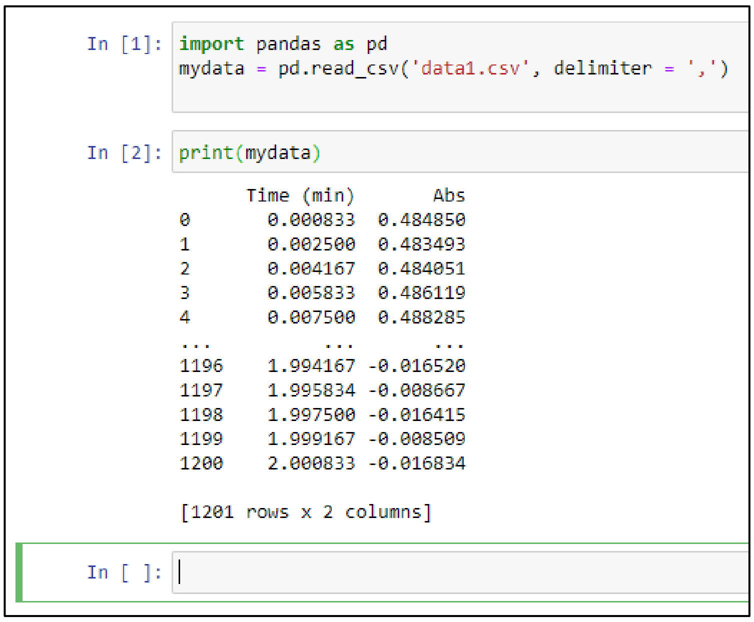
5. Import your data into the notebook with

```
import pandas as pd

mydata = pd.read_csv('data1.csv', delimiter = ',')

print(mydata)
```

6. Click Run. Your data should now be uploaded into a structure called **mydata**.

We check that the data has been uploaded correctly by running the **print** command.

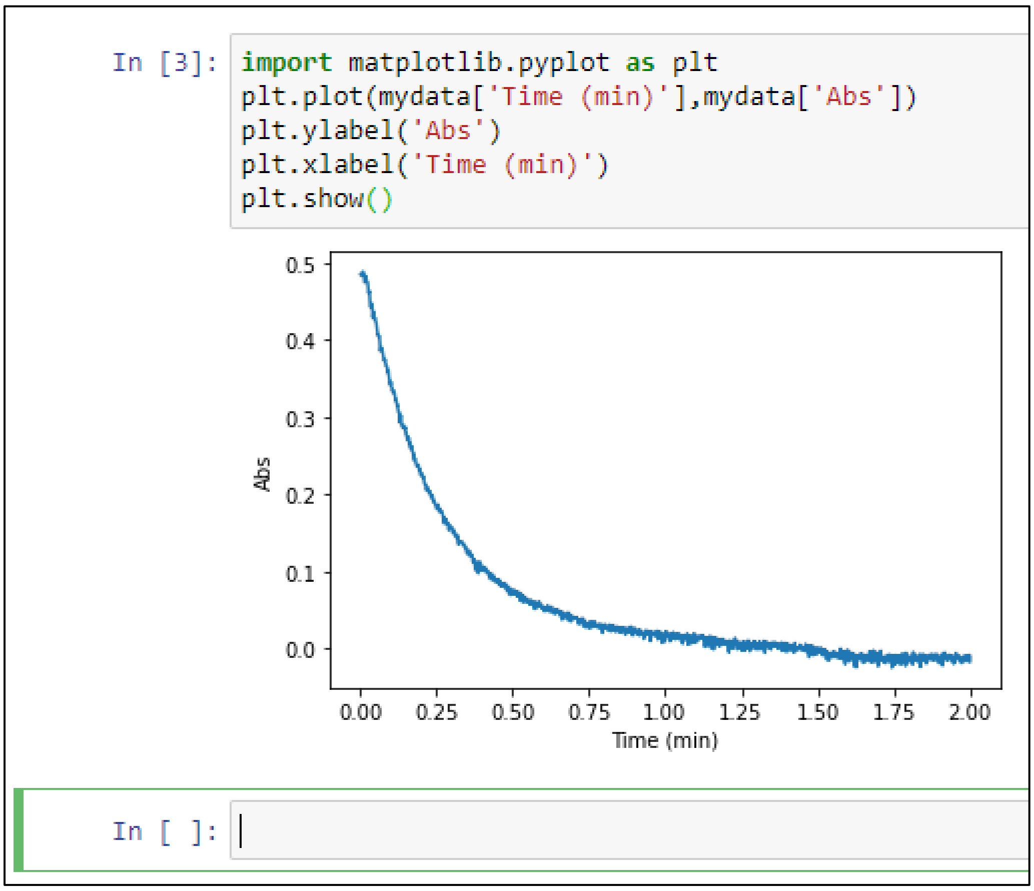
You will see that the data is loaded into a structure. The header rows serve automatically as labels for the columns of data.



```
In [1]: import pandas as pd
        mydata = pd.read_csv('data1.csv', delimiter = ',')

In [2]: print(mydata)

           Time (min)       Abs
        0    0.000833  0.484850
        1    0.002500  0.483493
        2    0.004167  0.484051
        3    0.005833  0.486119
        4    0.007500  0.488285
        ...       ...       ...
        1196 1.994167 -0.016520
        1197 1.995834 -0.008667
        1198 1.997500 -0.016415
        1199 1.999167 -0.008509
        1200 2.000833 -0.016834

        [1201 rows x 2 columns]

In [ ]:
```

7. Plot the data using the following code.

```
import matplotlib.pyplot as plt
plt.plot(mydata['Time (min)'],mydata['Abs'])
plt.ylabel('Abs')
plt.xlabel('Time (min)')
plt.show()
```



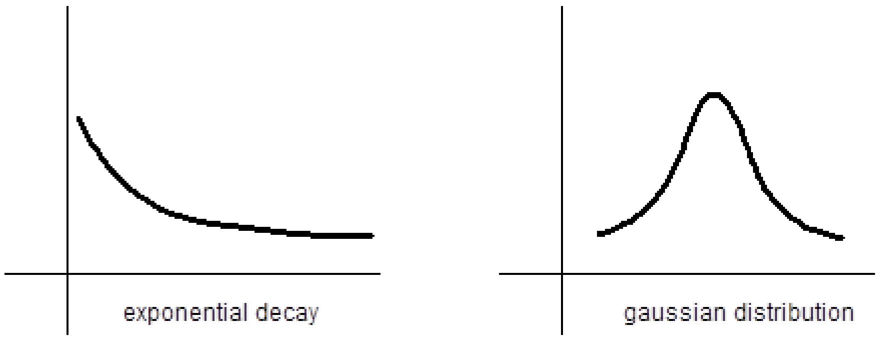
```
In [3]: import matplotlib.pyplot as plt
        plt.plot(mydata['Time (min)'],mydata['Abs'])
        plt.ylabel('Abs')
        plt.xlabel('Time (min)')
        plt.show()

In [ ]:
```

Do not continue until you are comfortable with plotting data.

The following notes are for **data1.csv**, so stick with this to complete the **Quiz**. You may use data2.csv to practise data modelling.

Part 2: Data modelling

Many forms of measured data can be modelled using simple functions. Two examples of simple mathematical functions exhibited by nature are the exponential decay, and the Gaussian distribution, drawn below.



Of these, what type of curve does your data most resemble?

An exponential decay may be displaced vertically, such that the asymptotic value is non-zero. Likewise, the rate of decay and initial value can vary. The Gaussian distribution may also vary in its background value, its height, its width and central value.

The exponential decay has a general formula,

$$f(x) = a + b\exp(-kx)$$

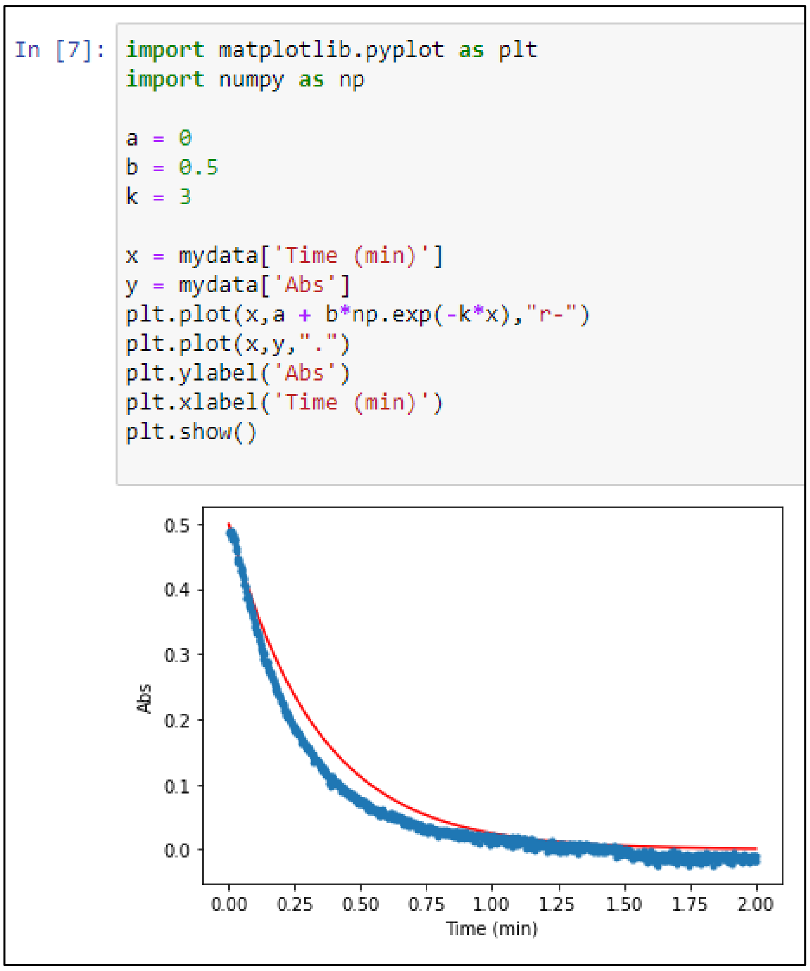
while the Gaussian is

$$g(x) = a + b\exp\left(-k(x - x_0)^2\right).$$

To plot an exponential decay, we create some variables a, b and k, and plot the function on the same plot as the data with the following code. To make it easier, we create new vectors x and y containing the downloaded data. To use the exponential function, we import “numpy”, call it “np”, and use mathematical functions as np.log or np.exp.

```
import matplotlib.pyplot as plt
import numpy as np

a = 0
b = 1
k = 1

x = mydata['Time (min)']
y = mydata['Abs']
```

```
plt.plot(x,a + b*np.exp(-k*x),"r-")
plt.plot(x,y,".")
plt.ylabel('Abs')
plt.xlabel('Time (min)')
plt.show()
```



```
In [7]: import matplotlib.pyplot as plt
        import numpy as np

        a = 0
        b = 0.5
        k = 3

        x = mydata['Time (min)']
        y = mydata['Abs']
        plt.plot(x,a + b*np.exp(-k*x),"r-")
        plt.plot(x,y,".")
        plt.ylabel('Abs')
        plt.xlabel('Time (min)')
        plt.show()
```

8. Manipulate the parameters and fit the data “by hand”.

What is the effect of each parameter on the shape of your modelled plot?

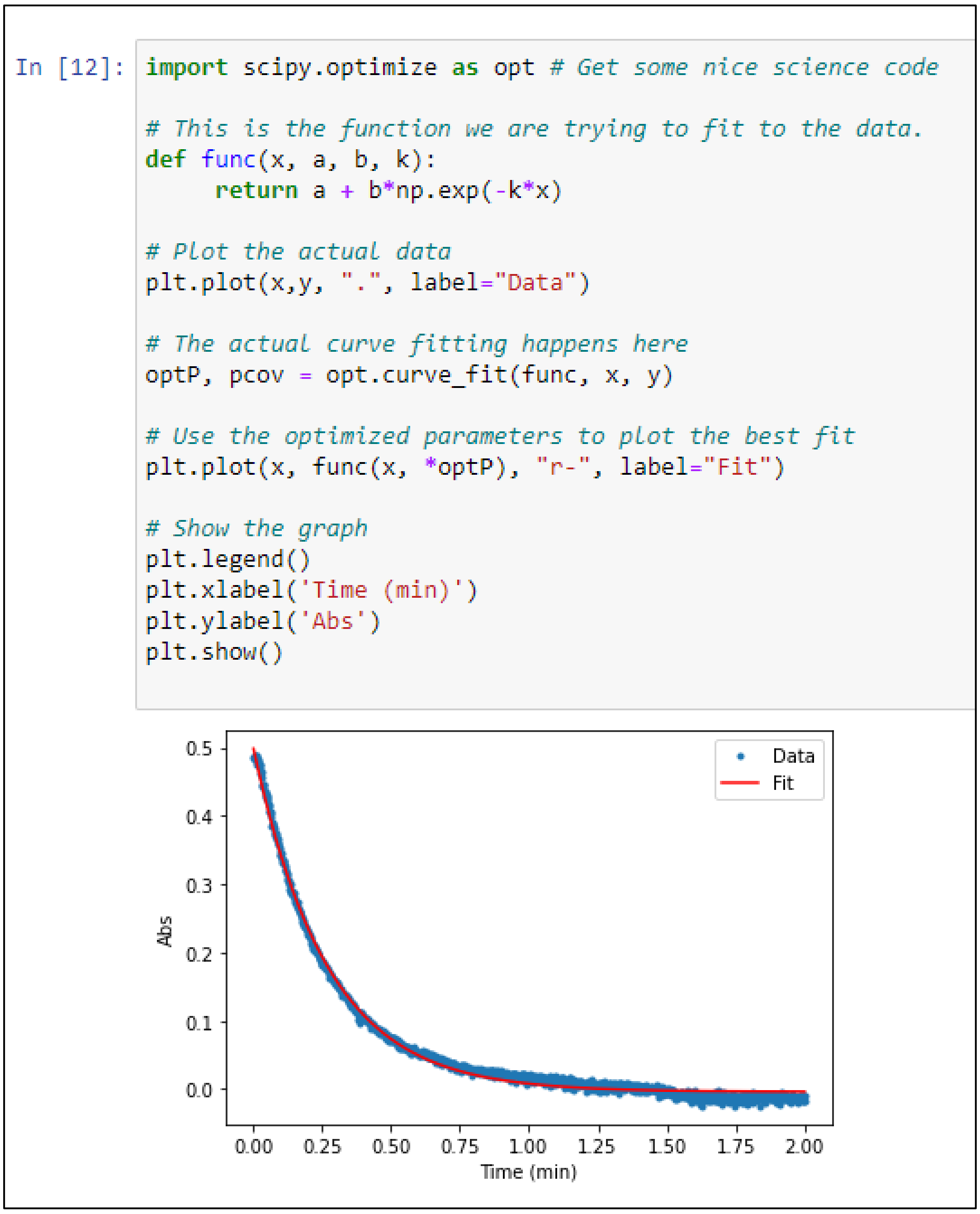
Part 3: Curve fitting

The great value of Python is that there are many functions and processes freely available. Curve fitting is very easily accomplished.

9. Use the following code to design a function to fit the data with, optimize the parameters and plot the data and the fit together.

In **Python**, the code may be commented using the hashtag. Commenting code makes it more readable and transparent to other users, and is considered good practice.

```
import scipy.optimize as opt # Get some nice science code

# This is the function we are trying to fit to the data.
def func(x, a, b, k):
     return a + b*np.exp(-k*x)

# Plot the actual data
plt.plot(x,y, ".", label="Data")

# The actual curve fitting happens here
optP, pcov = opt.curve_fit(func, x, y)

# Use the optimized parameters to plot the best fit
plt.plot(x, func(x, *optP), "r-", label="Fit")

# Show the graph
plt.legend()
plt.ylabel('Time (min)')
plt.xlabel('Abs')
plt.show()
```



```
In [12]: import scipy.optimize as opt # Get some nice science code

         # This is the function we are trying to fit to the data.
         def func(x, a, b, k):
             return a + b*np.exp(-k*x)

         # Plot the actual data
         plt.plot(x,y, ".", label="Data")

         # The actual curve fitting happens here
         optP, pcov = opt.curve_fit(func, x, y)

         # Use the optimized parameters to plot the best fit
         plt.plot(x, func(x, *optP), "r-", label="Fit")

         # Show the graph
         plt.legend()
         plt.xlabel('Time (min)')
         plt.ylabel('Abs')
         plt.show()
```

What are the optimized parameters? You can extract these from the data structure optP.

```
print(optP)
```

This will print the optimized parameters in the order they are defined in the function func.

Quiz: What are the optimized parameters of the fit?

The uncertainties are extracted from the matrix pcov. Print this matrix using

```
print(pcov)
```

The diagonal elements are the squared uncertainties of the parameters. You can use

```
print(pcov**0.5)
```

to print the square roots. See that exponents are set using two asterisks. Ignore the errors from square-rooting negative off-diagonal elements. You can print the optimized parameters by writing commands such as

```
print('a =',optP[0],'+/-',2*pcov[0][0]**0.5)
print('b =',optP[1],'+/-',2*pcov[1][1]**0.5)
print('k =',optP[2],'+/-',2*pcov[2][2]**0.5,’min-1’)
```

Here we have multiplied the uncertainties by 2. This is the so-called two-sigma error, encompassing 95% of the uncertainty range.

Python will give you these numbers to a lot of precision. Usually we do not report uncertainties to more than two significant figures, and if the first significant figure is a 3 or greater, one significant figure is enough. The parameter itself should not be quoted to higher precision than the uncertainty.

Quiz: What are the 2σ uncertainties of the parameters?

Propagating uncertainties is something we must often do. In the exponential decay example, the lifetime is the inverse of the decay rate,

$$\tau = \frac{1}{k}$$

$$d\tau = \frac{-1}{k^2}dk$$

By using calculus, we see that the uncertainty in the lifetime τ (tau) can be related to the uncertainty in *k*, and its value.

Python can perform mathematical manipulations for you. First, let’s make things a little neater.

```
k = optP[2]

dk = 2*pcov[2][2]**0.5
```

Now convert to a lifetime. Be careful with units! Here we were given the data in minutes, so if you want seconds, you’d better convert it.

```
tau = 1/k

dtau = -1/(k**2)*dk
```

By now you should have figured out how to print the values of parameters.

Quiz: What is the uncertainty in the lifetime?

Modelling data by fitting simple mathematical functions is widely applicable in science, engineering and finance. Make sure you understand how to use **Python** to accomplish this.

Part 4: Integration

Being able to integrate functions is something that will be of use in later experiments. Often we are interested in the area under a peak in a spectrum, or some other signal (for instance, the area under a velocity *vs* time curve should be distance travelled). The following code will integrate your data using the trapezoidal rule.

```
data_area = np.trapz(mydata['Abs'],mydata['Time (min)'])

print('data area is', data_area)
```

The area under an exponential function from x = 0 to some value x = t is given by

$$f(x) = a + b\exp(-kx)$$
$$\int_0^t f(x)dx = at + b(1-\exp(-kt))/k$$

The area under a Gaussian function from $x = -t$ to some value $x = t$ is given by

$$g(x) = a + b\exp\left(-k(x-x_0)^2\right)$$
$$\int_{-t}^{t} g(x)dx = 2at + b\sqrt{\frac{\pi}{k}}, \text{if } t >> 1/k$$

```
a = optP[0]
b = optP[1]
t = x[1200] #last measurement


fit_area = a*t + b*(1-np.exp(-k*t))/k
print('fitted area is',area)
```

Quiz: What is are the numerical and fitted areas under the curve?

Part 5: Linear relationships

The most common functional form fitted to data is the linear relationship, or "straight line". Indeed, sometimes a non-linear relationship is manipulated to yield a linear one for the purposes of ease-of-fitting. Open arrkin.csv[i] which contains kinetics data which you will model with the Arrhenius law,

$$k = A\exp(-E_a / k_B T),$$

where k is the rate, A is the frequency factor, E_a is the activation energy, k_B is Boltzmann's constant and T is the temperature in Kelvin. Manipulation gives

$$\log(k) = \log(A) - E_a / k_B T,$$

Select a data set and plot the natural logarithm of rate (dependent variable) against inverse temperature in Kelvin (0°C = 273.15K).

```
import pandas as pd
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('arrkin.csv')

x = 1/(data['T(C)'] + 273.15) #convert to Kelvin and invert
y = np.log(data['rate (s-1)']) #take logarithm of data

plt.plot(x,y, ".", label="Data")
plt.legend()
plt.ylabel('log(rate)')
plt.xlabel('1/T')
plt.show()
```

Quiz: What quantity is given by the slope and intercept of this plot respectively?

Now, we can use a statistics package to perform a linear regression analysis on the plot. The following code performs the analysis and reports the error on the slope only.

```
slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)

print('slope is',slope,'+/-',2*std_err)
print('intercept is', intercept)
```

Quiz: What is the activation energy of your system (include 2σ error estimate)?

Often we are interested in the intercept, too. A more general approach to fitting a straight line is to define a function, *y = mx + b*, with which we fit the data. Use the following code.

```
import scipy.optimize as opt

# This is the function we are trying to fit to the data.
def func(x, m, b):
     return m*x + b

# The actual curve fitting happens here
optP, pcov = opt.curve_fit(func, x, y)

print('slope is',optP[0],'+/-',2*pcov[0][0]**0.5)
print('interCept is',optP[1],'+/-',2*pcov[1][1]**0.5)

plt.plot(x,y, ".", label="Data") #plot the data
plt.plot(x, func(x, *optP), "r-", label="Fit") #plot the fitted line

plt.legend() #include a legend
plt.ylabel('log(rate)') #label the y-axis
plt.xlabel('1/T') #label the x-axis
plt.show() # Show the graph
```

Quiz: What is the frequency factor of your system (include 2σ error estimate)?

REPORT

Answer the ***online Quiz*** associated with this exercise.

[i] http://web.lemoyne.edu/~GIUNTA/classicalcs/arrkin.html