Radioactive Decay

# Abstract

In this exercise, we measure and model the rates of emission over time from the radioactive decay of Barium (Ba-137m). We have chosen two models, one where the emission rate I(t) = a\*exp(b\*t), and one where ln(I(t)) = a\*t+b, where a and b are parameters of each model to be fitted to our data, t is time, and ln is the natural logarithm. After fitting these models to our data, we graphically plot and compare these model curves along with the theoretical curve of emission rate, and our data with error bars. To evaluate the quality of our models, we calculate and discuss the reduced Chi squared values of our models and data, and confirm that the theoretically predicted half-life of Ba-137m lies within the estimated half life with uncertainty of our two models. This analysis was done in Python by use of the numpy, scipy and matplotlib modules.

# Introduction

In this exercise, we measure and model the rates of emission over time caused by radioactive decay from a sample of Barium (Ba-137m). The corresponding theoretically predicted relationship between emission rate (I) and time (t) is:

I(t) = I\_0\*exp(−t/τ) = I\_0 \* (1/2)^(t/t\_(1/2)). Where I\_0 is the initial emission rate, τ is the mean lifetime of the isotope, and t\_(1/2) is the half-life of the isotope. In our experiment, t is the independent variable, and I is our dependent variable.

We are modelling this relationship by using two models, one where the emission rate I(t) = a\*exp(b\*t), and one where ln(I(t)) = a\*t+b, where a and b are parameters of each model to be fitted to our data, t is time, and ln is the natural logarithm.

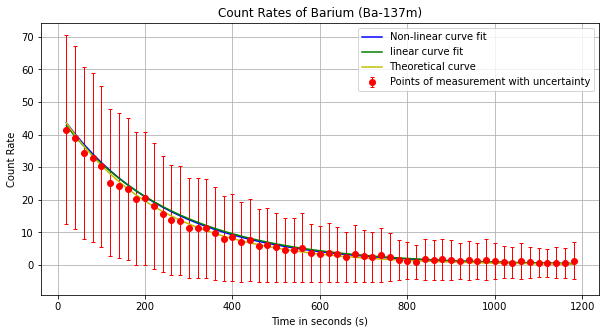
# Methods, Materials and Experimental Procedure

We successfully followed the procedures as described in the exercise3.pdf document.

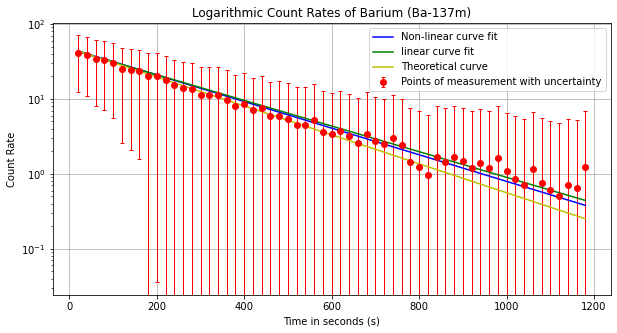
The points of data for which we based this analysis on, was downloaded from Quercus as instructed by the TAs. The uncertainties of the measured data were calculated as described in the exercise2.pdf document.

# Results

Below in Figure 1 and 2, we see our data from the experiment plotted as points with corresponding error bars. Furthermore, we see the theoretical curve as described in the introduction, along with the curves corresponding to our two models best fitted to our data.



*Figure 1: Relationship between count rates and time. We have plotted the points of measurement   
 with corresponding error bars, the curve of theoretical emission rate, and the two models   
 of emission rate best fitted to our data.*



*Figure 2: Equal to Figure 1, though with a logarithmic y-axis.*

The estimated optimal parameters with uncertainty by scipy optimize curve fit are:

[a,b] = [-0.00393908 3.75408883] +- [0.00090733 0.29589009] for the linear model, and

[a,b] = [ 4.36112915e+01 -4.08767747e-03] +- [1.35411958e+01 1.00398911e-03] for the non linear model.

The reduced Chi squared values for the nonlinear and linear model, respectively, are 0.0069 and 0.0076.

The Half-life of the Barium, predicted by the non-linear model, is: 170.0 +- -42.0 seconds.

The Half-life of the Barium, predicted by the linear model, is: 176.0 +- -41.0 seconds.

# Discussion

The reduced Chi squared values we found were very low. This means that our models fit our data very well. Thus, the Euclidian distance between the data points and our curves is in general very low. However, this is not necessarily a good sign. Our reduced Chi squared values should ideally both be equal to one. That we have extremely low reduced Chi squared values implies that we do not have enough data. It means that we are in risk of overfitting our models to our data.

The non-linear regression method gave a half-life closer to the expected half-life of 2.6 minutes = 156 seconds. 170 seconds is closer to 156 seconds, than 176 seconds, see half-life results in the Results section. Do however note that the theoretical half life falls within both of our models’ estimates of the half-lives, with associated uncertainties.

Though it is hard to distinguish the two models in the non-linear plot, in the logarithmic, linear plot, you can more easily see that the non-linear model returns a curve closer to the theoretical curve than the curve returned by the linear model. Both models do however fit the theoretical curve quite well. Furthermore, both models are well within the uncertainties of our measurements, see Figure 1 and 2 in the Results section.

# Conclusion

In this exercise we estimated the emission rate caused by radioactive decay of Barium (Ba-137m). We successfully followed the instructions for the experiment written in the exercise2.pdf document without issues. Though both of our models of emission rate over time returned quite similar curves, the non-linear model returned a curve closest to the theoretical curve. We have plotted our data with error bars, our models, and the theoretical curve in a normal plot, and a plot with logarithmic y-axis. Furthermore, we have calculated and discussed each models reduced Chi squared value, and confirmed that the theoretical half-life of Ba-137m lies within each of our models’ predicted half-lives with uncertainties.

# References

Exercise2.pdf

# Appendices

The codes and data we wrote and downloaded for this experiment is shown below in Code 1, Code 2, Data 1 and Data 2. Code 1 is the code which plots and calculates, Code 2 contains many of the functions I call in Code 1, Data 1 contains the radioactive decay of Barium data, and Data 2 contains the data of the background radiation.

#Importing modules

import numpy as np

import matplotlib.pyplot as plt

import Functions as F

#Defining Constants

Sample\_time = 20 #seconds

Experiment\_length = 20\*60 #Seconds. = 20 min

#Specifying modelling function

def non\_linear\_model(x,a,b):

return a\*np.exp(b\*x)

def linear\_model(x,a,b):

return a\*x+b

#Importing data

Sample\_number\_barium, Number\_of\_counts\_barium = F.read\_data('Barium.txt',

None,2)

Sample\_number\_bar\_background, Number\_of\_counts\_bar\_background = F.read\_data(

'Barium\_Background.txt', None,2)

#However, background radiation is not measured at the same time as the

# radiation from the Barium. Thus I subtract the mean number of counts from

# the background radiation, from the number of counts for each Barium sample.

Number\_of\_counts\_bar\_background = np.mean(Number\_of\_counts\_bar\_background)

Number\_of\_counts = Number\_of\_counts\_barium - Number\_of\_counts\_bar\_background

#Now we remove the latter part of the data, where the radiation from the

# barium is at the same level as the background radiation. When this happens

# the Number\_of\_counts value above may be negative.

list\_of\_negative\_values = []

for i in range(len(Number\_of\_counts)):

if Number\_of\_counts[i]<0:

list\_of\_negative\_values.append(i)

if len(list\_of\_negative\_values)==0:

list\_of\_negative\_values.append(-1)

#Now that we know at what index this happen, we shorten all the arrays we will

# use later on

Number\_of\_counts = Number\_of\_counts[0:list\_of\_negative\_values[0]]

Number\_of\_counts\_barium = Number\_of\_counts\_barium[0:list\_of\_negative\_values[0]]

Sample\_number\_barium = Sample\_number\_barium[0:list\_of\_negative\_values[0]]

#Standard deviation for each point, for derivation, see exercise 2 document.

Count\_rate\_uncertainty = np.sqrt(

Number\_of\_counts\_barium + Number\_of\_counts\_bar\_background)

Count\_rate = Number\_of\_counts/Sample\_time

#Now we fit the parameters of the two models to our data:

popt\_linear, pstd\_linear = F.fit\_data(linear\_model,

Sample\_number\_barium\*Sample\_time,

np.log(Count\_rate),

Count\_rate\_uncertainty/Count\_rate,

[-0.00393908, 3.75408883])

popt\_non\_linear, pstd\_non\_linear = F.fit\_data(non\_linear\_model,

Sample\_number\_barium\*Sample\_time,

Count\_rate,

Count\_rate\_uncertainty,

[4.36112953e+01, -4.08767785e-03])

print("The estimated optimal parameters with uncertainty by scipy optimize",

"curve fit are:",popt\_linear, "+-",pstd\_linear," for the linear, and:",

popt\_non\_linear, "+-", pstd\_non\_linear, "for the non linear model.")

#Calculating predicted y-values of models:

Count\_rate\_predicted\_non\_linear = np.zeros(len(Sample\_number\_barium))

Count\_rate\_predicted\_linear = np.zeros(len(Sample\_number\_barium))

#And now we also calculate the y-values predicted by the linear model,

# for the non logarithmic scale. I will use this later on in the plot

Count\_rate\_predicted\_linear\_non\_linear = np.zeros(len(Sample\_number\_barium))

for i in range(len(Sample\_number\_barium)):

Count\_rate\_predicted\_non\_linear[i] = popt\_non\_linear[0]\*np.exp(

popt\_non\_linear[1]\*i\*Sample\_time)

Count\_rate\_predicted\_linear[i]= popt\_linear[0]\*i\*Sample\_time+popt\_linear[1]

Count\_rate\_predicted\_linear\_non\_linear[i] = np.exp(popt\_linear[1])\*np.exp(

popt\_linear[0]\*i\*Sample\_time)

#The Chi squared values for these models are:

chi2\_non\_linear = F.chi2reduced(Count\_rate,Count\_rate\_predicted\_non\_linear,

Count\_rate\_uncertainty,2)

chi2\_linear = F.chi2reduced(np.log(Count\_rate),Count\_rate\_predicted\_linear,

Count\_rate\_uncertainty/Count\_rate,2)

print("\nThe reduced Chi squared values for the non linear and linear",

"model respectively",

"are", np.round(chi2\_non\_linear,4), "and", np.round(chi2\_linear,4))

print("These values are very low. This means that our models fit our data",

"very well. Thus, the Euclidian distance between the data points and our",

"curves are in general low.")

print("However, this is not necessarily a good sign. Our reduced Chi squared",

"values should ideally both be equal to one. That we have extremely low",

"reduced Chi squared values implies that we have to little data.",

"That we are in risk of overfitting our models to our data.")

#Now we calculate the estimate of the half-life, for our two models:

# We know half life = (mean lifetime)\*ln(2)

# Furthermore, we know that the parameter b in the non-linear model, is equal

# to -1/mean\_lifetime. Thus, for the non-linear model, we have:

b\_non\_linear = popt\_non\_linear[1]

mean\_lifetime\_non\_linear = -1/b\_non\_linear

Half\_life\_non\_linear = mean\_lifetime\_non\_linear\*np.log(2)

#Now we calculate the uncertainty of the half-life. multiplying the quantity

# by scalars, means that we must also multiply the uncertainties by these

# scalars. However, when we take the quantity 1/a, then the uncertainty of

# 1/a is equal to the uncertainty of a divided by a^2:

Half\_life\_non\_linear\_uncertainty = -np.log(2) \* (

pstd\_non\_linear[1]/popt\_non\_linear[1]\*\*2)

print("\nThe Half-life of the Barium, predicted by the non-linear model, is:",

np.round(Half\_life\_non\_linear,0), "+-", np.round(

Half\_life\_non\_linear\_uncertainty,0))

#For the linear model, we have:

a\_linear = popt\_linear[0]

mean\_lifetime\_linear = -1/a\_linear

Half\_life\_linear = mean\_lifetime\_linear\*np.log(2)

Half\_life\_linear\_uncertainty = -np.log(2) \* (pstd\_linear[0]/popt\_linear[0]\*\*2)

print("The Half-life of the Barium, predicted by the linear model, is:",

np.round(Half\_life\_linear,0), "+-", np.round(

Half\_life\_linear\_uncertainty,0))

print("The non-linear model gave a half-life closer to the expected",

"half-life of 2.6 minutes/156 seconds.")

print("\nThe non-linear regression method gave a half-life closer to the",

"expected half-life of 2.6 minutes. 170 seconds is closer to 156",

"seconds, than 176 seconds. However, note that the theoretical half",

"life falls within both of our estimates of the half-lives with",

"associated uncertainties.")

#Now we plot these models, the data, and the theoretical curve:

#Now we create some data points on the model curve for plotting

#We calculate the predicted count rates by the theory:

# Note, theoretical half-life is 2.6 min

Count\_rate\_theory\_predicted = np.zeros(len(Sample\_number\_barium))

for i in range(len(Sample\_number\_barium)):

Count\_rate\_theory\_predicted[i] = popt\_non\_linear[0]\*np.exp(

-i\*Sample\_time/(2.6\*60)\*np.log(2))

#Now we plot the original data with error bars, along with the curve fit model

plt.figure(figsize=(10,5))

plt.errorbar(Sample\_number\_barium\*Sample\_time,

Count\_rate,Count\_rate\_uncertainty ,c='r', ls='',

marker='o',lw=1,capsize=2,

label = 'Points of measurement with uncertainty')

plt.plot(Sample\_number\_barium\*Sample\_time,

Count\_rate\_predicted\_non\_linear, c='b',

label = 'Non-linear curve fit')

plt.plot(Sample\_number\_barium\*Sample\_time,

Count\_rate\_predicted\_linear\_non\_linear, c='g',

label = 'linear curve fit')

plt.plot(Sample\_number\_barium\*Sample\_time,Count\_rate\_theory\_predicted, c='y',

label = 'Theoretical curve')

plt.title("Count Rates of Barium (Ba-137m)")

plt.xlabel("Time in seconds (s)")

plt.ylabel("Count Rate")

plt.legend()

plt.grid()

plt.savefig("Count Rates of Barium (Ba-137m)"+'.png')

plt.show()

#Now we plot the logarithmic version of this:

plt.figure(figsize=(10,5))

plt.errorbar(Sample\_number\_barium\*Sample\_time,

Count\_rate,Count\_rate\_uncertainty ,c='r', ls='',

marker='o',lw=1,capsize=2,

label = 'Points of measurement with uncertainty')

plt.plot(Sample\_number\_barium\*Sample\_time,

Count\_rate\_predicted\_non\_linear, c='b',

label = 'Non-linear curve fit')

plt.plot(Sample\_number\_barium\*Sample\_time,

Count\_rate\_predicted\_linear\_non\_linear, c='g',

label = 'linear curve fit')

plt.plot(Sample\_number\_barium\*Sample\_time,Count\_rate\_theory\_predicted, c='y',

label = 'Theoretical curve')

plt.title("Logarithmic Count Rates of Barium (Ba-137m)")

plt.xlabel("Time in seconds (s)")

plt.ylabel("Count Rate")

plt.legend()

plt.grid()

plt.yscale('log')

plt.savefig("Logarithmic Count Rates of Barium (Ba-137m)"+'.png')

plt.show()

print("\nThough it is hard to distinguish the two models in the non-linear",

"plot, in the logarithmic, linear plot, you can more easily see that",

"the non-linear model returns a curve closer to the theoretical curve,",

"than the curve returned by the linear model.")

print("Both models does however fit the theoretical curve quite good.",

"Furthermore, both models are well within the uncertainty of our",

"measurements, see plots above.")

*Code 1: Code used for this exercise.*

import numpy as np

import scipy.optimize as optim

import matplotlib.pyplot as plt

#Defining the function for curve fitting and plotting

def curve\_fit\_and\_plot(model,initial\_guess,xdata,ydata,y\_uncer,xunit,yunit,

plot\_title):

"""

This function uses the scipy curve\_fit function to estimate the parameters

of the model which will minimize the euclidian distance between our data

points, and the model curve.

We print these optimal model parameters along with their uncertainty, and

plot the original data with error bars, along with the curve fit model.

Parameters

----------

model : function to be used as model

model(x,a,b,c,...), where we are estimating a,b,c, etc.

initial\_guess : list of guesses for the parameters a,b,c, etc. eg. [2,4,254]

xdata : list of input points for the model, eg. [2,4,5,7,9,28]

ydata : list of output points for the model, eg [23,25,26,85,95,104]

y\_uncer : list of uncertaintes associated with the ydata, which the model

shall output

xunit : String describing the unit along the x-axis for label when plotting.

eg. 'Voltage (V)'

yunit : String describing the unit along the y-axis for label when plotting.

eg. 'Current (A)'

plot\_title : String describing the title of the plot.

eg. 'Current vs. Voltage'

Returns None

"""

#Using the scipy curve fit function to find our model parameters

p\_opt , p\_cov = optim.curve\_fit(model , xdata , ydata, p0 = initial\_guess,

sigma = y\_uncer, absolute\_sigma = True )

p\_std = np.sqrt( np.diag ( p\_cov ))

print("The optimal values for our curve fit model parameters, are:",np.round(p\_opt,2))

print("Their associated uncertainties are:", np.round(p\_std,2))

#Now we create some data points on the model curve for plotting

xvalues\_for\_plot = np.linspace(xdata[0],xdata[-1],1000)

yvalues\_for\_plot = []

for i in xvalues\_for\_plot:

yvalues\_for\_plot.append(model(i,p\_opt[0],p\_opt[1]))

#Now we plot the original data with error bars, along with the curve fit model

plt.figure(figsize=(10,5))

plt.errorbar(xdata,ydata,y\_uncer,c='r', ls='', marker='o',lw=1,capsize=2,

label = 'Points of measurement with uncertainty')

plt.plot(xvalues\_for\_plot,yvalues\_for\_plot, c='b',

label = 'Scipy curve fit')

plt.title(plot\_title)

plt.xlabel(xunit)

plt.ylabel(yunit)

plt.legend()

plt.grid()

plt.savefig(plot\_title+'.png')

plt.show()

return None

def error\_plot(model,p\_opt,xdata,ydata,y\_uncer,xunit,yunit,

plot\_title):

#Now we create some data points on the model curve for plotting

xvalues\_for\_plot = np.linspace(xdata[0],xdata[-1],1000)

yvalues\_for\_plot = []

for i in xvalues\_for\_plot:

yvalues\_for\_plot.append(model(i,p\_opt[0],p\_opt[1]))

#Now we plot the original data with error bars, along with the curve fit model

plt.figure(figsize=(10,5))

plt.errorbar(xdata,ydata,y\_uncer,c='r', ls='', marker='o',lw=1,capsize=2,

label = 'Points of measurement with uncertainty')

plt.plot(xvalues\_for\_plot,yvalues\_for\_plot, c='b',

label = 'Scipy curve fit')

plt.title(plot\_title)

plt.xlabel(xunit)

plt.ylabel(yunit)

plt.legend()

plt.grid()

plt.savefig(plot\_title+'.png')

plt.show()

return None

def chi2(y\_measure,y\_predict,errors):

"""Calculate the chi squared value given a measurement with errors and

prediction"""

return np.sum( np.power(y\_measure - y\_predict, 2) / np.power(errors, 2) )

def chi2reduced(y\_measure, y\_predict, errors, number\_of\_parameters):

"""Calculate the reduced chi squared value given a measurement with errors

and prediction, and knowing the number of parameters in the model."""

return chi2(y\_measure, y\_predict, errors)/ \

(y\_measure.size - number\_of\_parameters)

def read\_data(filename, Del, skiprows, usecols=(0,1)):

"""Load give\n file as csv with given parameters,

returns the unpacked values"""

return np.loadtxt(filename,

skiprows=skiprows,

usecols=usecols,

delimiter=Del,

unpack=True)

def fit\_data(model\_func, xdata, ydata, yerrors, guess):

"""Utility function to call curve\_fit given x and y data with errors"""

popt, pcov = optim.curve\_fit(model\_func,

xdata,

ydata,

absolute\_sigma=True,

sigma=yerrors,

p0=guess)

pstd = np.sqrt(np.diag(pcov))

return popt, pstd

*Code 2: Code which defines functions which we use in Code 1.*

Radioactive Decay Experiment Sep 30, 2021 9:38 AM

Sample Number Number of Counts

1 834

2 785

3 691

4 663

5 609

6 508

7 491

8 472

9 411

10 413

11 365

12 314

13 280

14 275

15 232

16 231

17 228

18 199

19 165

20 174

21 145

22 155

23 121

24 124

25 111

26 93

27 93

28 109

29 77

30 71

31 79

32 68

33 55

34 73

35 59

36 54

37 64

38 52

39 33

40 29

41 23

42 37

43 33

44 37

45 34

46 28

47 32

48 28

49 36

50 26

51 21

52 18

53 27

54 19

55 16

56 14

57 18

58 17

59 29

60 14

*Data 1: Content of the Barium.txt file containing data for this experiment.*

Radioactive Decay Experiment Sep 30, 2021 9:12 AM

Sample Number Number of Counts

1 2

2 4

3 5

4 3

5 6

6 3

7 5

8 5

9 3

10 6

11 3

12 0

13 4

14 4

15 2

16 1

17 6

18 4

19 3

20 2

21 4

22 6

23 7

24 5

25 3

26 5

27 3

28 3

29 1

30 3

31 4

32 5

33 2

34 7

35 5

36 1

37 3

38 8

39 1

40 8

41 6

42 7

43 3

44 3

45 2

46 7

47 4

48 3

49 7

50 4

51 6

52 5

53 2

54 1

55 1

56 5

57 6

58 2

59 2

60 4

*Data 2: Content of the Barium\_Background.txt file containing data for this experiment.*