An Investigation of Error Analysis with a Focus on Various Methods of Linear Fitting

By

Conor Lawton

A thesis submitted in partial fulfilment for the
degree of *double major* of experimental physics

Maynooth University
National University of Ireland Maynooth

Faculty of Science and Engineering
Experimental Physics Department

April 2021

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of experimental qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed:  Date: 22/04/2021

Acknowledgements

I would like to express great appreciation to my project supervisor, Frank Mulligan for much helpful advice and unending support during the project.

I would also like to thank all my family and friends for all their support and some useful discussion which contributed immensely to the evolution of my ideas for the project.

Abstract

One aim of this project was to determine which method of linear fitting was superior: least squares or point-by-point. Using experimental and Monte Carlo simulation data, it was found that least squares linear fitting provided greater accuracy. When using the method to estimate the true resistance (33.2Ω) for 1000 simulated circuits the least squares provided a distribution of estimates fitting the distribution with mean of 33.227 and standard distribution 0.785 while the point-by-point method gave a distribution of estimates with mean 33.169 and standard distribution 1.27. Least squares mean was closer to the true value and had less variance from the mean than point-by-point.

An interactive python program was developed in the Jupyter programming environment to provide an educational aid to assist with undergraduates understanding of error analysis and curve fitting. The program allowed the user to change parameters of a plot of data points using interactive sliders. When the user changes the position and error on the points, an ordinary and weighted least squares fit was continuously updated on the graph. Using this software, the effects of using a weighted least squares approach was explored.

Nine values of the cross section of a 121.6 nm photon following a collision of an electron with a hydrogen molecule recorded by nine separate laboratories were used to calculate a more accurate value. Using a method closely related to weighted least squares; weighted mean arithmetic provided a value which was closer to the true cross section with less error than any of the values used to calculate it. The cross section was refined from the data as $(8.0 \pm 0.3) \times 10^{-18} cm^2$.

Overall, the project aimed to show that after measurements have been taken, the analysis of the data collected is also of huge importance and should be given sufficient thought and effort.

Chapter One: Introduction

During experimentation, the physical designing of experiments and taking measurements is only part of the process. Sufficient effort should be taken in deciding what do with experimental data after it has been collected, meaning in what way should it be processed or interpretated to extract correct, accurate and unbiased information from it. This project aimed to investigate some methods of extracting this information from data sets.

1.1. Error analysis

Error analysis involves the study of the uncertainty in measurement which affects the accuracy to which an experimental result can be known. This uncertainty is not simply a mistake in taking the measurements but is an intrinsic part of the measurement as it comes from the external noise or the nature of the measurement itself. Uncertainty in measurements can be reduced but never eliminated, so to fully state the results of an experiment, the uncertainty on measured values must be accounted for. Furthermore, any values calculated from these measurements have an error attached to them as they were calculated from an inexact value. There are two main types of error: systematic and random error (Kirkup and Frenkel, 2006).

1.1.1. Systematic error

Systematic error arises due to imperfect equipment and is a consistent bias in a measurement. When measuring the distance between two points a measuring tape may be used but say that during manufacturing the tape was cut one millimetre short so that all measurements it takes are one millimetre shorter than reality. This inaccuracy in the measurement is on every measurement that is taken i.e., systematic error. To try to find the exact error caused by the tape a laser telemeter which measures the distance to an object by timing how long it takes a laser to travel from it to the object and back might be used. A well calibrated telemeter might find that the measuring tape was in fact falling short of the true distance by one millimetre each time. Knowing that this is the systematic error for the measurements, the error could be accounted for by adding one millimetre. The problem here is that the telemeter itself may have been calibrated marginally wrong and could have a systematic error attached to it of 0.01 mm. The only way to find this error is to have more accurate, better calibrated instruments which themselves can never perfectly measure the distance. Therefore, there can never be zero or perfectly known systematic error (Kirkup and Frenkel, 2006).

1.1.2. Random error

Random error is caused by unpredictable changes. One such error may arise during an experiment using electronic circuits. There is an unpredictable, small amount of noise in circuits due to electric fields in the environment. To find the range in which the noise alters the data many data points must be recorded so that the standard deviation of the data can be calculated. From this the standard error can be calculated which is a measure of the uncertainty attached to each measurement. This allows the measurement to be written as a mean value with a range of error above or below this average (equation 1.1) (Kirkup and Frenkel, 2006).

$$Measurement = mean \pm standard\ error \tag{1.1}$$

1.1.3. Outliers

An outlier can be identified as a member of a data set which lies - usually by itself - quite far from the other values. An experimenter should put thought into how the outlier is dealt with. In some cases, it is warranted to simply ignore the abnormal data but not always. In the case of data, which is normally distributed, the probability that the outlier could randomly appear at a certain distance from the mean can be explored. Chauvenet's criterion states that if the probability of the point lying at a certain distance from the mean (p), multiplied by the total number of data values (n) is less than or equal to 0.5 then the value should be discarded (equation 1.2) (Kirkup, 2012).

$$Chauvenet's\ criterion:\ ignore\ outlier\ if\ (p \times n) \leq 0.5 \tag{1.2}$$

Another way, and often a better way of dealing with outliers is to use a weighted least squares approach. Instead of ignoring outliers as data, this method assigns a weight value to each point so that it affects the line of fit proportional to the accuracy with which each point is known. Weighted least squares is explained in more detail in later sections but this is how it relates to outliers.

1.2. Curve fitting

Curve fitting involves calculating a line which represents the relationship between two variables from several data points corresponding to the variables and how they change together. These data points are usually plotted on an x and y plane where the x axis represents the independent variable, and the y axis represents the dependant variable. The independent variable is the variable which is being manipulated in an experiment and the dependant changes as a result.

The relationship between the variables can be of many types including, polynomial, exponential and most importantly for this project linear relationships. The prominent least squares method used for calculating a line of fit for linear data can be extended to other relationships, but this project will instead go into detail on linear fitting. Linear data is a common relationship between physical values and as such is an important area of study (Squires, 2001).

1.2.1. Linear fitting

Linear regression aims to model the relationship between two variables, x, and y with a straight line. From data points a line of best fit of the form $y=mx+c$ is calculated where m is the slope of the line and c is the y-axis intercept. For many relationships, the slope gives an estimate of a value which links the two variables, for example the slope of the voltage and current in a circuit gives the resistance of the

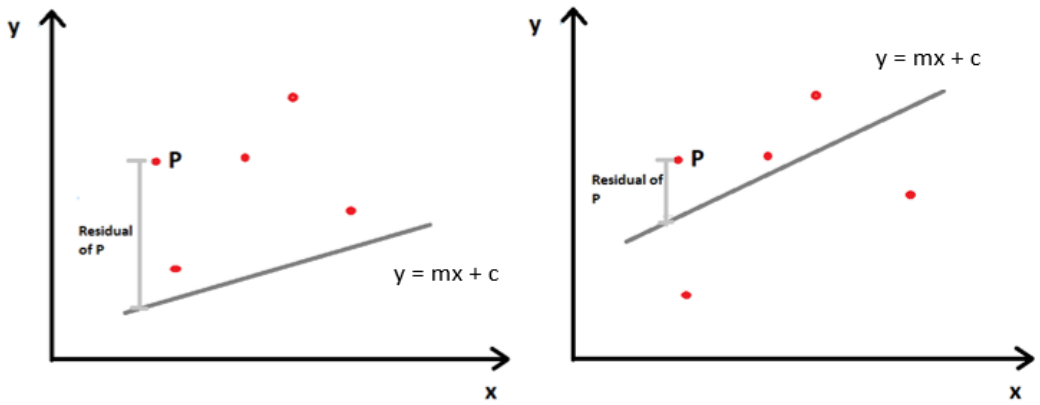

Figure 1.1 - Graph on right shows a better line of fit than the graph on the left for the data points shown. Sum of the residuals are less on the right.

circuit. The distance from the line of best fit to a data point is that point's residual, and a good model of a linear relationship minimises the total of the residuals as in Figure 1.1. As mentioned before, there is an error associated with these measured variables. Therefore, there is an error in this predicted relationship. Often the error which an experimenter is concerned with is on the dependent variable, but the error associated with the independent variable can also be dealt with. There are various linear regression methods, and this project aims to investigate some of these (Squires, 2001).

1.2.2. Point-by-point method

The point-by-point method tends to be an attractive approach for undergraduate students in physics due to its perceived simplicity as opposed to other methods, especially the commonly used least squares. The slope of the best fit line is obtained by simply finding the mean of the slopes of each line leading from each point p_i to p_{i+1}. Although this a relatively effortless form of regression, it has disadvantages when it comes to an error in the relationship, as outlined by Kirkup (Kirkup and Frenkel, 2019).

1.2.3. Ordinary least squares method

The least squares method is the most used method. Ordinary least squares assumes that the uncertainty on each measurement is the same. It involves minimising the total *squared* residuals, hence the least *squares* as in Figure 1.2. In practice, when calculating the line of fit, the y-axis intercept and the slope of the line are calculated. This gives the necessary information to fit the equation of a line $y=mx+c$. This intercept and slope have error associated with them which is of a direct result of the error on the measurements of the data points (Taylor, 1982).

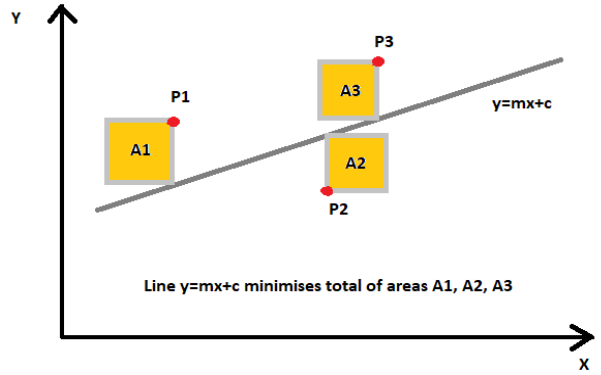

Figure 1.2 – For the least squares fit, the sum of A1, A2 and A3 are minimised where areas are all square of residuals.

1.2.4. Weighted least squares method

To deal with a set of experimental data which is heteroscedastic (discontinuous uncertainty on measurements), an alternate version of least squares can be used, that is called weighted least squares. The underlying idea is that if a measurement has a higher uncertainty than other points in a data set, it should have less influence on the regression line. The reason this is needed is because ordinary least squares assumes continuous error in the data set. This method introduces a weight factor to each data point based on its error, which scales the input to the final line of fit (Kirkup, 2012). A similar method for a single variable value rather than x and y data can be used called weighted mean arithmetic. This finds a mean of a data set while considering the error on the data. This is especially useful for compiling many laboratories values found in similar experiments. An example of this will be demonstrated later.

1.2.5. Students difficulty with error and linear fitting

In a study by Allie (Allie, Buffler, Campbell and Lubben, 1998), on first year undergraduate's perceptions of experimental measurements there is clear evidence that a large proportion of students do not understand the underlying theory for error analysis or at least have difficulty with it. The study categorised students based on their understanding with emphasis on their idea of how a measurement should be taken. The students were separated into categories of varying degrees of understanding. The students with better understanding had some knowledge of the importance of getting the mean of several measurements rather than a single measurement. The paper included some of the student's conversations which outlined their perceptions on outliers for which some students thought they should be excluded while others did not. The study also investigated the students thoughts on experimental values agreeing or not. The students were asked if two means from the same experiment agreed which each other. Some students understood that there was uncertainty in the means which must be considered whereas others did not and concluded that the means were different and so did not agree with each other.

From the findings of this study, it was decided that a part of this project would focus on developing a program which would be a useful tool in furthering undergraduates understanding of error analysis and linear fitting

1.3. Monte Carlo simulation

Software can be developed which models a specific experiment several times by using various equations and algorithms, this is called Monte Carlo simulation. A big advantage of such a method is that an experiment can be repeatedly ran and a large amount of data can be gathered in a short time. This data then represents far greater statistical significance than a smaller data set collected in the same or less amount of time in real life. The number of resources and time spent collecting data can be far less and so can save money in the process. One must take care when developing a model of a system as a small error can cause a largely different result than that expected in real life (Landau and Binder, 2009).

1.3.1. Parameters of a simulation

When developing a Monte Carlo model, some measurements of a system must be used to accurately assign correct parameters and equations to the simulation. Two important values for many systems (including models used in this project) are the mean and the standard deviation of the systems. These values allow the model to accurately simulate the randomness seen in a physical system. A Gaussian distribution of values can be used to model most physical randomness in a system. In an example which will be looked at in more detail later, a mean value of resistance and a standard deviation on voltage in a circuit is used to generate voltage data with simulated noise/error added to them corresponding to current values in the same circuit. This data could be run many times and then analysed to draw statistically significant results.

1.3.2. Random numbers

Monte Carlo simulation relies heavily on the ability to generate random numbers that attempts to be as close to truly random as possible. Most random generators use an algorithm or function which takes in a number which is used as the seed and generates a sequence of numbers which is seemingly unpredictable. It is difficult to prove that a random number generator is truly random and so this must be considered when developing a model. Random number generation is a huge area of study, with one paper exploring the use of quantum computing to generate random numbers (Yang and Zhao, 2016).

However, most coding languages have random number generating libraries which give numbers that are, for the purposes of this project, indistinguishable from true random numbers so are random enough for almost all models.

1.3.3. An application of Monte Carlo simulation

An example of a monte model is a percolation simulation. This involves simulating physical phenomena such as the likelihood of the spread of a forest fire and the permeability of rock but for this example, the latter will be explained. A grid of certain size is set up and is filled with 'holes' or 'filled' spaces which in terms of permeability represents an area where water can flow or cannot flow, respectively. The grid coordinates are assigned based on a certain probability to represent the holes in the rock. When this grid is populated, an analysis can be done to find whether such a rock would be permeable to liquid. Further simulations can be run while altering the probability to find a statistical point at which the commonness of holes causes permeability in rock (critical point). This tends to be a probability of about 0.6. Some instances of these simulations are shown in Figure 1.3, with probabilities of a hole in each cell displayed in the titles.

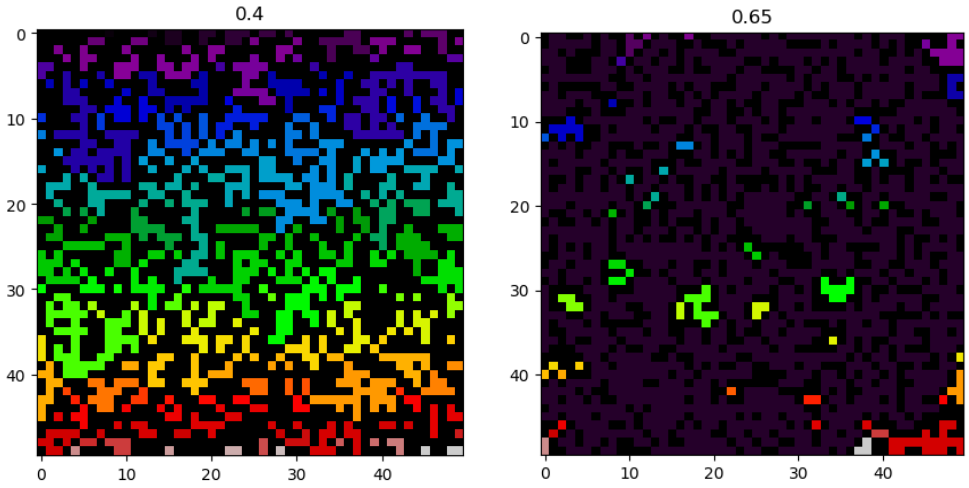

Figure 1.3 - 50x50 Percolation simulations. Probability of hole appearing in each cell is shown in graph titles. Black cells show solid rock and colours show groups of holes. The left graph represents rock which is not permeable as no group of holes goes from top to bottom. The right graph represents rock which is permeable as the dark purple hole group spans the grid, allowing water to flow.

1.4. Computation

1.4.1 Tuples

Tuples in Python are a way of storing multiple items in a single variable. A tuple which stores two items is a 2-tuple, three items is a 3-tuple, and so on. This is made use of in the project for functions which calculate more than one value, for which the function returns a tuple of the appropriate size.

1.4.2 Interactive curve fitting program

The University of Colorado Boulder hosts a website which includes many educational simulations, of which one simulates curve fitting as shown in Figure 1.4. It allows the user to place, move and change the error on points on a graph and then fits a curve to the data (Fitting, 2021). This type of simulation was deemed very effective and useful as an educational tool for undergraduates. As a result, it was

decided that a similar program would be developed during this project which would be helpful in improving undergraduate understanding of the subjects as discussed in section 1.2.5.

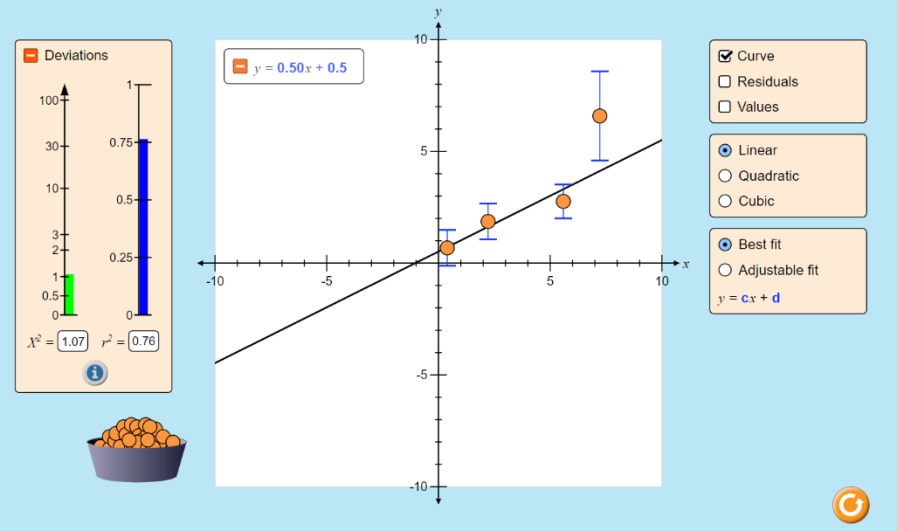

Figure 1.4 – Interactive simulation on the PHET interactive simulations website. It fits a dynamic line to movable data points(Fitting, 2021).

1.5. Probability and statistics

1.5.1. Samples

When studying an aspect of a whole group, it is common not to look at the entire group but just to study a certain percentage of the group as it may be unrealistic to study the entire group. This percentage of the population is called a sample. For example, if a biologist wants to study the height of lions on the African savannah, they will not find and measure every lion. Say there are 50,000 lions, the biologist may find 500 lions to study. From this sample, they can infer the information about the whole population with a particular level of certainty depending on the size of the sample. The information will have a lower uncertainty the larger the sample size (Peck, Short and Olsen, 2008).

1.5.2. Confidence intervals

The mean of a population sample will not necessarily be equal to the actual mean of the population. We can know to a level of certainty if the true mean will lie within a certain distance of the sample mean. The distance either side of the sample mean which we expect the true mean to be is called the confidence interval (CI). This expectation is not exact however, we only know the percentage chance of it being here. The percentage certainty used in this project is 95%. For 95% the confidence interval is calculated by finding 95% of the area under the curve centred on the mean and the bounds of the interval lie at the bounds of this area. This turns out to be 1.966 standard deviations to either side of the mean. We are 95% certain that the true mean lies in these bounds, visually represented in Figure 1.5. A confidence interval will get narrower with larger sample sizes or with low variance in the population. Equation (1.3) outlines the formula for the 95% CI interval boundaries and Figure 2.1 provides a visual representation of this. Using confidence intervals is a good way of stating complete statistical results (Peck, Short and Olsen, 2008).

$$CI = \bar{x} \pm \frac{1.966}{\sqrt{n}} \tag{1.3}$$

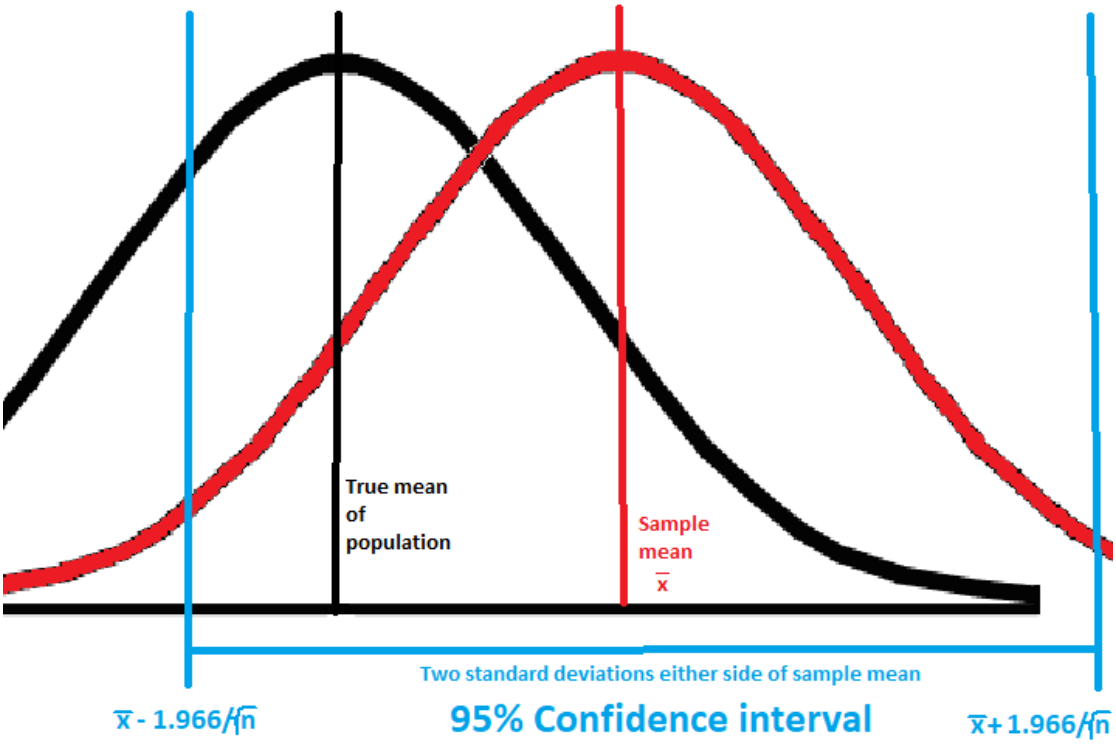

Figure 1.5 – True population mean within the 95% confidence interval calculated from mean of a sample of the population. For many samples, the number of 95% intervals which contain the true mean will tend to 95% of samples.

Chapter two: Theory

2.1. Curve Fitting

2.1.1. Point-by-point

When fitting a line to a data set by the point-by-point method equation (2.1) is used and the uncertainty in this measurement is given by equation (2.2). This method assumes a line which intercepts the y-axis at the origin.

$$m = \frac{\sum_{i=0}^{n-1} y_i x_i}{n} \tag{2.1}$$

$$\Delta m = \frac{\Delta\left(\frac{y_i}{x_i}\right)}{\sqrt{n}} \tag{2.2}$$

(Kirkup and Frenkel, 2019).

2.1.2. Ordinary least squares

When fitting a line to a data set by the ordinary least squares method the full set of equations used are outlined on equations (2.3)-(2.8). These equations result in a slope of a line (m) and the coordinate of its y-axis intercept (c). The error on the slope is given by equation (2.10) and the error on the intercept is according to equation (2.11).

$$S = \sum_{i=0}^{n-1} (y_i - mx_i - c)^2 \tag{2.3}$$

$$x' = \frac{1}{n} \sum_{i=0}^{n-1} x_i \tag{2.4}$$

$$y' = \frac{1}{n} \sum_{i=0}^{n-1} y_i \tag{2.5}$$

$$D = \sum_{i=0}^{n-1} (x_i - x')^2 \tag{2.6}$$

$$m = \frac{1}{D} \sum_{i=0}^{n} (x_i - x') y_i \tag{2.7}$$

$$c = y' - mx' \tag{2.8}$$

$$d_i = y_i - mx_i - c \tag{2.9}$$

$$(\Delta m)^2 = \frac{1}{D}\frac{\sum_{i=0}^{n} d_i^2}{n-2} \tag{2.10}$$

$$(\Delta c)^2 = \left(\frac{1}{n} + \frac{(x')^2}{D}\right)\frac{\sum_{i=0}^{n} d_i^2}{n-2} \tag{2.11}$$

(Five-minute physics, 2018).

2.1.3. Weighted least squares

The weighted least squares method operates much the same except for a weight factor outlined in equation (2.12) that scales certain values during the calculation of the slope and y-intercept. This is fully described in equations (2.12) - (2.17). The error on the slope is given by equation (2.18) and the error on the y-intercept is given by equation (2.19).

$$\text{Weight factor: } w_i = \frac{1}{(\Delta y_i)^2} \quad where\ \Delta y_i\ is\ error\ on\ y_i \tag{2.12}$$

$$x' = \frac{1}{n}\sum_{i=0}^{n}\frac{w_i x_i}{w_i} \tag{2.13}$$

$$y' = \frac{1}{n}\sum_{i=0}^{n}\frac{w_i y_i}{w_i} \tag{2.14}$$

$$D = \sum_{i=0}^{n} w_i(x_i - x')y_i \tag{2.15}$$

$$Slope\ of\ regression\ line\text{: } m = \frac{1}{D}\sum_{i=0}^{n} w_i(x_i - x')y_i \tag{2.16}$$

$$Y - intercept\ of\ regression\ line\text{: } c = y' - mx' \tag{2.17}$$

$$(\Delta m)^2 = \frac{1}{D}\frac{\sum_{i=0}^{n} w_i d_i^2}{n-2} \tag{2.18}$$

$$(\Delta c)^2 = \left(\frac{1}{w_i} + \frac{(x')^2}{D}\right)\frac{\sum_{i=0}^{n} w_i d_i^2}{n-2} \tag{2.19}$$

(Five-minute Physics, 2018).

2.1.4. Weighted mean arithmetic

Weighted mean arithmetic is used to compile measurements to get a more accurate value. This method differs from a normal mean calculation as the errors on each value differs. Calculations for the mean are as equations (2.20)-(2.21).

$$\text{Weight factor: } w_i = \frac{1}{(\Delta x_i)^2} \quad where\ \Delta x_i\ is\ error\ on\ x_i$$

$$Weighted\ mean: \bar{x} = \frac{\sum_{i=0}^{n} w_i x_i}{\sum_{i=0}^{n} w_i}$$

$$Error\ on\ mean: \Delta\bar{x} = \frac{1}{\sqrt{\sum_{i=0}^{n} w_i}}$$

Chapter three: Instrumentation

3.1. Python

3.1.1. Spyder

The Spyder IDE (Scientific Python Development Environment) was used to explore and compare the accuracies of the linear fitting methods investigated in this project. This environment with the help of some open-source function libraries, allowed for editing and running of python code in a single package. The debugger included in the Spyder environment was also made use of for finding bugs in the code which was developed(Team, 2021).

3.1.2. Jupyter

The Jupyter IDE was used to develop an interactive graph which displays and demonstrates a weighted least squares linear fitting method for a data set (Project Jupyter, 2021). The environment was highly suitable for the interactive nature of this part of the project as it allows for interactive elements called widgets. Widgets allowed for parameters of data to be changed quickly and any effects of these changes would be immediately displayed. These interactive widgets can come in many forms but the type which was most useful for this project were the slider widgets. They allowed for values to be changed in an easy and understandable way for the user. The sliders are set up with minimum, maximum, default value and step size values to govern their operation. They can also be set to only update the display when the slider stopped moving. There is a function called interactive_output which can be used with widgets which pass values to another function which does something with these values. In the case of this project the slider widgets pass a value for data points y values and error to a function which plots the points and fits lines to them.

The widgets and interactive_output function are used as follows:

widget1=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,value=0.1)

ipywidgets.interactive_output(plotting_function,widget1,widget2…)

3.1.3. Libraries

One library of python functions used during the project was NumPy (NumPy, 2021). This is a scientific computing library which allows the user to call premade functions which may be complicated to implement from scratch, therefore streamlining code and simplifying certain processes. One such function is the mean function which takes an array of values and returns the mean of the values. This is typed as one line in the users code.

Another important library used was the matplotlib library (Matplotlib: Python plotting — Matplotlib 3.4.1 documentation, 2021). This allows the simple creation of visual plots of data. Many of its functions take in sets of data and print different graphs and plots such as histograms, scatter plots or line plots. This library was very important for linear fitting and visual representations of the methods explored in the project.

Chapter four: Experimental procedure

4.1. Comparison of point-by-point and least squares linear fitting

One goal of the project was to repeat and build upon work done by Kirkup in a paper which outlined the superiority of the least squares linear fitting method over the point-by-point method (Kirkup and Frenkel, 2019). Kirkup did this by using the methods on experimental and simulated Monte Carlo data. This was largely repeated in this project with some alterations and extensions.

4.1.1. Developing code to calculate fit by point-by-point method

A function in python was developed (Figure 4.1) which takes a data set as parameters in the form of an array of x-values and an array of y-values, assumed to be the independent and dependent variables respectively, obtained from experimentation as in Figure 4.3. The function used the equations defined in the theory chapter (equation (2.1),(2.2)) to calculate the slope and the error on it which it then returns (2-tuple/couple) as the result of the linear fitting. This method assumes a y-intercept of zero.

```
def pbp(x,y):
    """
    Takes x variable array(independent) and y variabel array(dependent).
    Returns slope of point-by-point linear fit for data as well as
    error on slope.
    """
    m=sum((y/x)/len(x))
    z=Z(x,y)
    err=np.std(z)/np.sqrt(len(z))
    return m,err
```

Figure 4.1 – Python function which calculates slope and slope error of point-by-point fit for data set consisting of x and y variables.

4.1.2. Developing code to calculate fit by ordinary least squares method

A python function was developed (Figure 4.2) which takes array parameters of x and y variables as in Figure 4.2. Using the equations defined (equations (2.3)-(2.11)) the function then calculates the slope and y-intercept of the least squares line along with the errors on these values. These calculated values are returned as a 4-tuple as the result of the linear fitting for this data set.

```
def lsq(x,y):
    """
    Takes x variable array(independent) and y variabel array(dependent).
    Returns parameters of least squares linear fit for data as well as
    errors on these.
    """
    n = len(x)
    x_bar = (1/n)*sum(x)
    y_bar = (1/n)*sum(y)
    D = sum((x-x_bar)**2)
    m = (1/D)*sum((x-x_bar)*y)
    c = y_bar-(m*x_bar)
    d = y - (m*x) - c
    m_err = (1/D)*((sum(d**2))/(n-2))
    c_err = ((1/n)+(x**2/D))*(sum(d**2)/(n-2))
    return m,c,np.sqrt(m_err),np.sqrt(c_err)
```

Figure 4.2 – Python function which calculates the parameters of a least squares fit for a data set consisting x and y variables.

4.1.3. Comparing methods with experimental data

Using the code developed, experimental data could be analysed to get a picture of the linear fits achieved by the methods. The experimental data used was the same as that in Kirkup's paper which considers the relationship between the voltage and current in a circuit. The relationship in question is Ohm's law and the slope of a graph of the two should give an estimate of the resistance. The aim is to show which method of linear fitting gives a better estimate. The data is displayed in Figure 4.3 After fitting lines by both methods, they could be compared by the values of their slopes/resistance estimates and by their errors. The lines were plotted on a graph with the data points to verify visually that the lines were fitting the data to a certain degree of accuracy as expected.

I_i (A)	V_i (V)	$R_i = \frac{V_i}{I_i}$ (Ω)
0.005	0.17	34.00
0.010	0.33	33.00
0.015	0.50	33.33
0.020	0.66	33.00
0.025	0.83	33.20
0.030	1.00	33.33
0.035	1.16	33.14
0.040	1.33	33.25
0.045	1.50	33.33
0.050	1.66	33.20
0.055	1.83	33.27
0.060	1.99	33.17
0.065	2.16	33.23
0.070	2.32	33.14
0.075	2.49	33.20
0.080	2.66	33.25
0.085	2.82	33.18
0.090	2.99	33.22
0.095	3.15	33.16
0.100	3.32	33.20

Figure 4.3- Experimental data for which estimates of resistance was calculated by point-by-point and least squares methods.

4.1.4. Comparing methods with Monte Carlo simulation

Using Monte Carlo simulations, similar data was collected but for 1000 data sets instead of just one. Using the same current values as for the experimental data corresponding voltage values were generated in the form of a gaussian distributed. These values were generated assuming a true value of resistance in the circuit of 33.2 Ω and a standard distribution of 0.01 V. This was achieved by calculating the voltage from the current values by Ohm's law ($V = IR$). These were values of voltage without noise in the circuit so each of these values had random noise added to them to represent a real physical system. For each iteration of the simulation this was repeated to give different sets of voltage values for each. For each iteration, the two methods were applied, and the lines of fit were gotten.

With a large set of linear fits for many data sets the methods could be analysed further. The slope of the line gotten by the methods was the estimate of the resistance in the circuit. For both methods, the percentage difference between these values and the true value of resistance (equation (4.1)) was plotted on the same graph to show the differences in the accuracy of the estimates between the methods.

To further compare the methods the standard error in the slopes/best estimates of the resistance was plotted for each iteration of the simulation.

$$\%Difference = \frac{(Estimate\ of\ resistance) - (True\ resistance)}{(True\ resistance)} \times 100\% \qquad (4.1)$$

The uncertainty of the methods was also investigated. The uncertainty on the slopes is one value which is calculated in the point-by-point and least squares methods and returned. These values were graphed to show any visual difference.

4.2. Comparison of ordinary least squares and weighted least squares linear fitting methods

The ordinary least squares method assumes that the y-error on each data point is the same so in cases when this is not true (heteroscedastic data), the estimate of the line of fit can be very inaccurate. The weighted least squares fit was investigated as an alternative to ordinary least squares.

4.2.1. Developing code to calculate fit by a weighted least squares method

A python function was developed to calculate the parameters of a line of fit by weighted least squares (Figure 4.4). The function takes arrays of x and y variables as well an array of the error on the y values. The function then calculates the slope, y-intercept, and the errors on these values for a weighted least squares fit. The function uses the equations defined in the theory section (equations (2.12)-(2.19)) to calculate these values. These parameters are then returned as a 4-tuple.

```
def weighted(x,y,y_err):
    """
    Takes x variable array(independent),y variable array(dependent) and
    y error array.
    Returns parameters of weighted least squares linear fit for data as well
    as errors on these.
    """
    n = len(x)
    w = 1/(y_err**2)
    x_bar = sum(w*x)/sum(w)
    y_bar = sum(w*y)/sum(w)
    D = sum(w*((x-x_bar)**2))
    m = (1/D)*sum(w*(x-x_bar)*y)
    c = y_bar-(m*x_bar)
    d = y - (m*x)
    m_err = (1/D)*((sum(w*(d**2)))/(n-2))
    c_err = ((1/sum(w))+(x**2/D))*(sum(w*(d**2))/(n-2))
    return m,c,m_err**(1/2),c_err**(1/2)
```

Figure 4.4 - Python function which calculates the parameters of a weighted least squares fit for a data set with different errors on the y variables.

4.2.2 Fitting real data with ordinary and weighted least squares

The functions developed for ordinary and weighted least squares fitting was used for a set of real experimental data which was displayed in the second-year laboratory manual in Maynooth university. The data used was as shown in Figure 4.5 which is measurements taken by different laboratories of cross section of a 121.6 nm photon following a collision of an electron with a hydrogen molecule. The error on the cross sections is heteroscedastic which is suitable for weighted least squares fitting. The data points and both fits were graphed together to show the results of the two methods.

Laboratory	Cross section (10^{-18} cm^2)	Error on cross section(10^{-18} cm^2)
1	13.1	4.1
2	11	4.3
3	10.3	2.2
4	12.1	1.3
5	14.6	1.3
6	7.3	1.5
7	6.7	0.5
8	8.2	1.3
9	7.2	0.7

Figure 4.5 – Data for cross sections of a 121.6 nm photon after collision with hydrogen electron recorded by various laboratories.

4.3. Interactive weighted least squares fitting

A program which displays data points which can be varied in value and error and then displayed the ordinary and weighted least squares fit lines on the same graph was developed. The program was an extension of the code for comparing ordinary and weighted least squares. Sliders (Figure 4.6) were incorporated into the program so that a user could change the y-value and y-error of each data point. The lines of fit were calculated as below. Using the interactive output function from ipywidgets a function which plotted the points and fits was called every time one of the sliders was move, which appeared as though the points and fits moved in real time.

```
#Sliders
y0=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y0: ',value=0.1)
e0=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y0: ')
y1=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y1: ',value=1.5)
e1=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y1: ')
y2=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y2: ',value=1.8)
e2=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y2: ')
y3=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y3: ',value=3.3)
e3=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y3: ')
y4=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y4: ',value=4.6)
e4=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y4: ')
y5=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y5: ',value=4.7)
e5=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y5: ')
y6=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y6: ',value=6.2)
e6=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y6: ')
y7=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y7: ',value=7)
e7=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y7: ')
y8=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y8: ',value=8.9)
e8=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y8: ')
#Containers for sliders
box0 = widgets.HBox([y0,e0])
box1 = widgets.HBox([y1,e1])
box2 = widgets.HBox([y2,e2])
box3 = widgets.HBox([y3,e3])
box4 = widgets.HBox([y4,e4])
box5 = widgets.HBox([y5,e5])
box6 = widgets.HBox([y6,e6])
box7 = widgets.HBox([y7,e7])
box8 = widgets.HBox([y8,e8])
ui = widgets.VBox([box0,box1,box2,box3,box4,box5,box6,box7,box8])
```

```
#Displays sliders and graph
out = widgets.interactive_output(weighted_unweighted,{'e0':e0,'y0':y0,'e1':e1,'y1':y1,'e2':e2,'y2':y2,'e3':e3,'y3':y3,'e4':e4,'y4
display(ui,out)
```

Figure 4.6 – Slider setup for interactive program. Y position sliders are contained in a box to the left and error sliders are contained in a box to the right.

Chapter five: Results and discussion

5.1. Comparison of point-by-point and least squares linear fitting

5.1.1. Comparing methods with experimental data

The best guesses for the resistance of the circuit were as follows:

Point-by-point: 33.241±0.044 Ω

Least squares: 33.183±0.024 Ω

These values suggest that the least squares method is superior in accuracy as the uncertainty for the estimate by point-by-point is nearly twice that of least squares. These calculations were carried out in the paper by Kirkup and the value for least squares differs here by a small amount due to this project accounting for the possibility of the y-intercept not being zero. Kirkup states a value of 33.203±0.012 Ω (Kirkup and Frenkel, 2019). We know that in the case of this experiment that a zero intercept is expected but in general this not always the case.

The values obtained for each method are relatively similar, which could be due to small variance in the voltage values measured as seen in Figure 5.1. This low variance may result in the true scope of the accuracies of the methods not being seen. This is because this only accounts for one set of measurements, so this is further explored with the use of many Monte Carlo simulations.

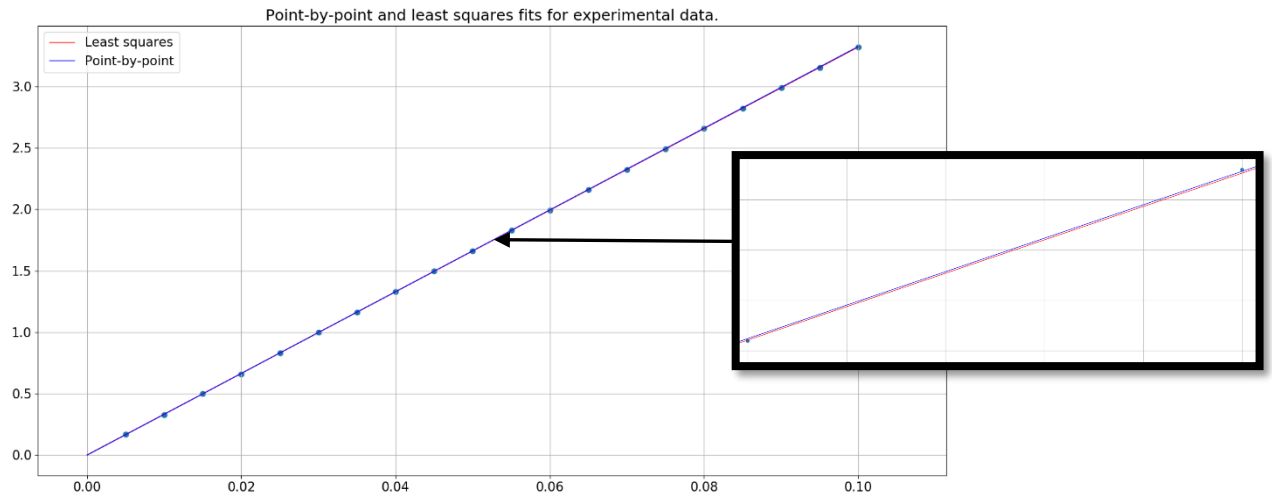

Figure 5.1 – Point-by-point and least squares fit to experimental data showing small difference in lines.

5.1.2. Comparing methods with Monte Carlo simulation

When the 1000 simulations were run and lines of fit were gotten, varying values for slopes were obtained. When running this once the mean and standard deviation of the 1000 best estimates for both methods were as follows.

Least squares mean: 33.227

Least squares standard deviation: 0.785

Point-by-point mean: 33.169

Point-by-point standard deviation: 1.27

This data shows that the mean of the least squares is closer to the true value of resistance of 33.2 with it being within 0.027 of this value while the point-by-point estimate is out by 0.031. This is a relatively small difference but the important information here is the difference in standard deviations of the values, these give an idea of the certainty with which these methods give their estimates. The relative size of the standard deviation of the point-by-point vs least squares estimate shows that the uncertainty for least squares is considerably lower and this is shown by the visual distribution of the values in Figure 5.2. The 95% confidence intervals are shown on the same figure where we are 95% certain that the true mean of 33.2 Ω will lie within them. Both confidence intervals have the same level of confidence but the least squares confidence interval is narrower so this gives a narrower band in which the true mean should be and so is more certain.

A further demonstrate of the consistent superiority of the least squares method is the percentage difference from the true value of resistance for every estimate plotted in Figure 5.3. The point-by-point method reached values which were about 10% different to the true value apart from less than 10 values higher than this (Although these may seem like outliers these are still statistically valid). while least squares reached a maximum of about 5-6% from the true value except for less than 10 values exceeding this. This shows that the point-by point estimate is on average further from the true resistance of the circuit.

Additionally, to contrast the stability of the methods, when comparing the direct uncertainty of each estimate there is consistently higher uncertainty for point-by-point as seen in Figure 5.4. There is a substantially larger mean and even more substantial standard deviation of error on point-by-point than least squares. For 1000 iterations the average error for both was as follows:

Least squares mean error: 0.772

Least squares standard deviation: 0.132

Point-by-point mean error: 1.128

Point-by-point standard deviation: 0.465

The error for point-by-point was not only on average larger, but it is also much less stable and has a potential of giving a value with error of over 3 Ω either side of the true 33.2 Ω value. In contrast, the least squares was far more stable with maximum values of uncertainty reaching only about 1.2 Ω.

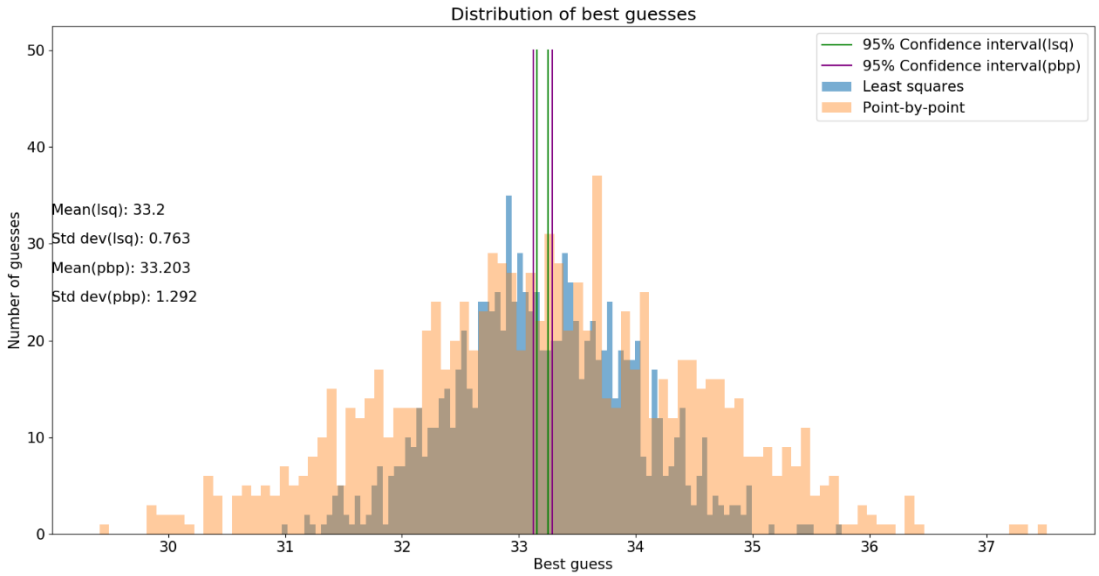

Figure 5.2 – Histograms of 1000 simulations of point-by-point and least squares best guesses of resistance of simulated circuits. 95% confidence intervals for both are overlaid. Wider interval for point-by-point showing that least squares gives an estimate of the true mean with more certainty.

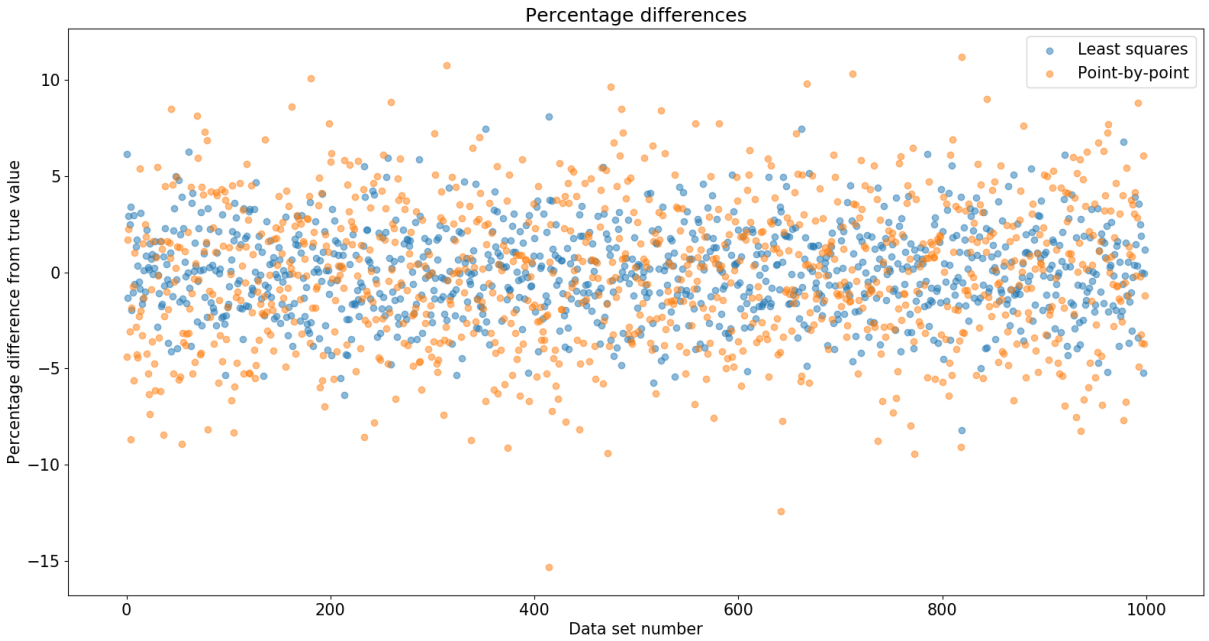

Figure 5.3 – Graph showing percentage differences of best estimates of resistance by least squares and point-by-point methods. 1000 simulated systems are shown.

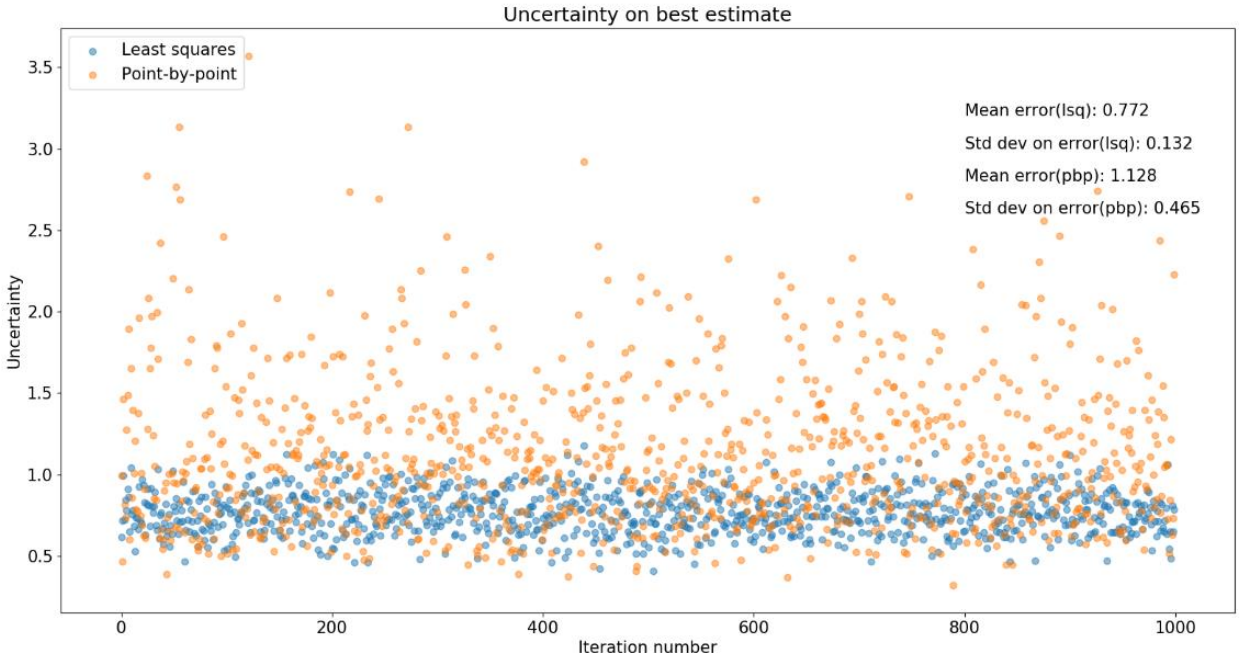

Figure 5.4 – Graph showing the uncertainties on the best estimates of resistance by least squares and point-by-point methods. 1000 simulated systems are shown. The average uncertainty on least squares is lower than point-by-point.

5.2. Comparison of ordinary least squares and weighted least squares linear fitting

Having established least squares as a far superior method of linear fitting than point-by-point this section will aim to show that it can be improved upon in certain situations. When the error on data is not the same across the board weighted least squares can be used effectively. The development of an interactive program for this is also outlined.

5.2.1 Interactive program

An interactive program was developed as both a way to explore the differences quickly and easily between ordinary and weighted least squares methods and as an effective way for someone to gain intuitive knowledge of the methods by changing parameters of the data themselves. The primary reason for this program was the difficulty undergraduate students tend to have with the subjects of error analysis and linear fitting. It was thought that an interactive program which students could change the parameters and see the effects then they could gain a broader understanding of the ideas behind fitting a line to data.

The program graphs 9 data points with linear fits for both ordinary least squares and weighted least squares. As is seen in Figure 5.5 (left) the program consists of 9 sliders for the y-position of the points, 9 sliders for the error on the points, and a graph of the points with linear fits. The points are originally positioned in a way that they are representing a linear relationship when the program is started. When the sliders for the y-positions are moved the relevant point on the graph also moves which causes both the lines of fit to alter accordingly. When the sliders for the errors are moved the error on the relevant point changes which causes both the points error bar to change and consequently for the weighted fit to alter.

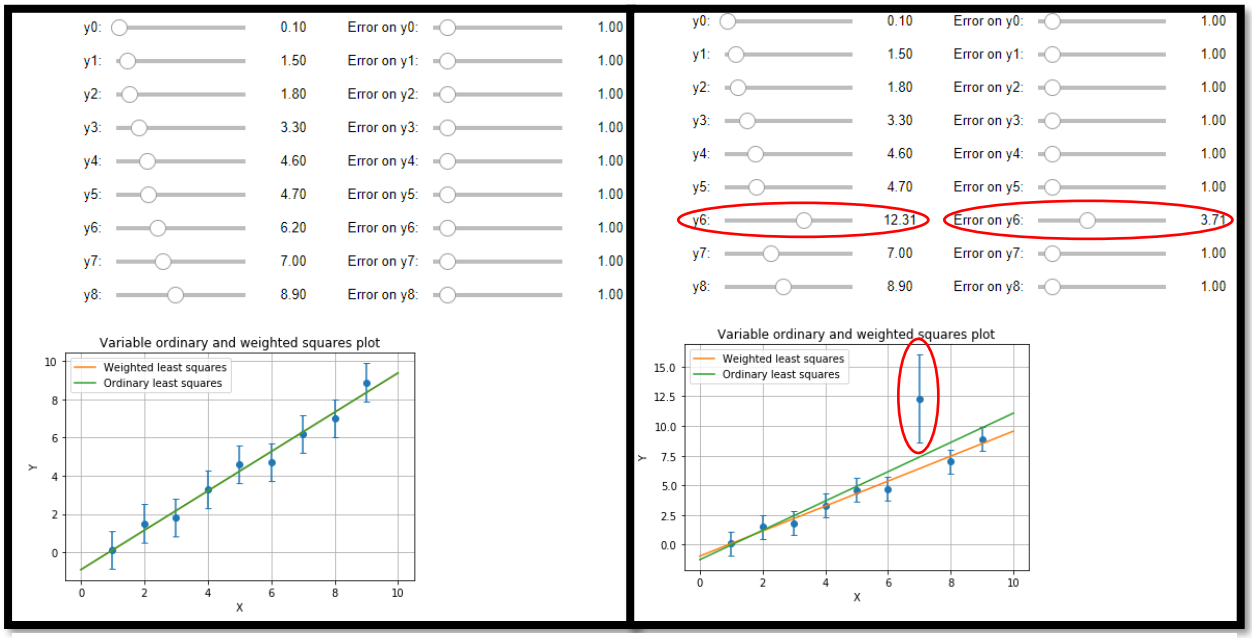

Figure 5.5 – User interface for interactive weighted least squares program. Left shows unaltered program. Right shows the position and error of data point 6 altered and the resulting lines of fit for unweighted and weighted least squares.

5.2.2. Demonstration of benefits of a weighted least squares method

Using the interactive program developed various data was configured to illustrate the weighted least squares in different situations. One such figure; Figure 5.6 there are 9 data points which seem to be well behaved in that they seem to fit a line without much deviation. The errors on these points are identical so we do not yet see any difference from the two methods of fitting yet.

Moving to Figure 5.7 there is a data point far from the rest. Looking at all the points this point appears to be an outlier. In this instance the error is assumed to be the same throughout and so the ordinary and weighted least squares gives the same lines due to the weights being equal for the weighted method. The line of fit is affected by this outlier and depending on the data set this could be changing the fit so that accuracy of the relationship is lost.

In Figure 5.8 we see the same positions for the data points but there is a higher error on the apparent outlier. This higher error is justified as in experiments a point this far from the general trend in the data is likely due to an additional error which is not on the other points. This higher error effects the weighted line by putting less 'weight' on the last point due to its relatively high error. This is where we see the weighted least squares come into effect. The weighted line is brought down closer to all the first eight points and away from the last point as it should have less influence. The ordinary squares line stays where it was as varying error does not affect it.

This demonstration shows that it is important to consider the error from each point to treat more accurate information with more importance rather than allowing data which may be an outlier to influence the line of fit by a disproportionate amount. Weighted least squares is shown here to fit the line closely to the more accurate data but it also considers the less accurate data with diminishing importance. This is superior as the information which the less accurate data point lends to the trendline is still being used and is not entirely ignored. If an approach where outliers are thrown away was used any information from them would be void, therefore there would be a smaller effective sample size and overall accuracy of the line of fit would be lessened. In addition, for data sets in which the uncertainty differs, the line of fit will be more accurate overall as each point is accounted for with weight relative to all the others i.e., data is treated which as much significance as its relative uncertainty suggests as in Figure 5.9. This figure outlines that for ordinary least squares the three points with the highest error drag the line higher than it should be whereas for the weighted least squares the line is brought up with the correct proportion relative to the errors between the points.

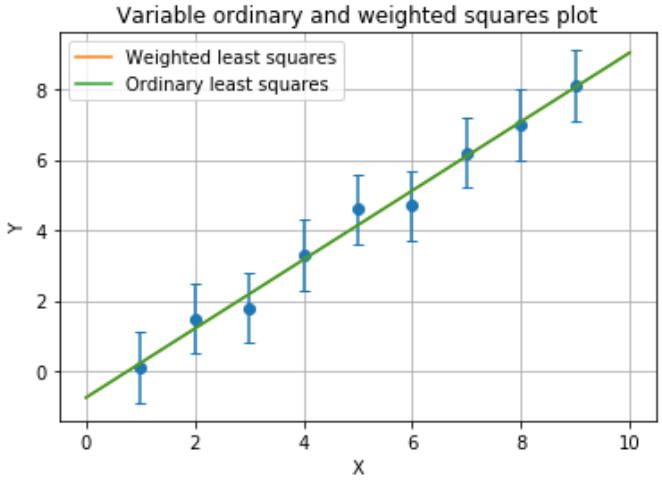

Figure 5.6 – Weighted and unweighted least squares fit with identical error on each data point. Lines of fit are identical also.

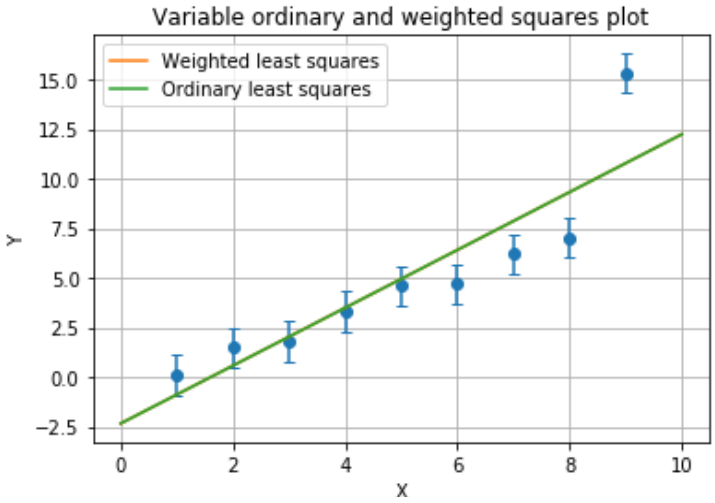

Figure 5.7 - Weighted and unweighted least squares fit with identical error on each data point. Seems to be an outlier in the data.

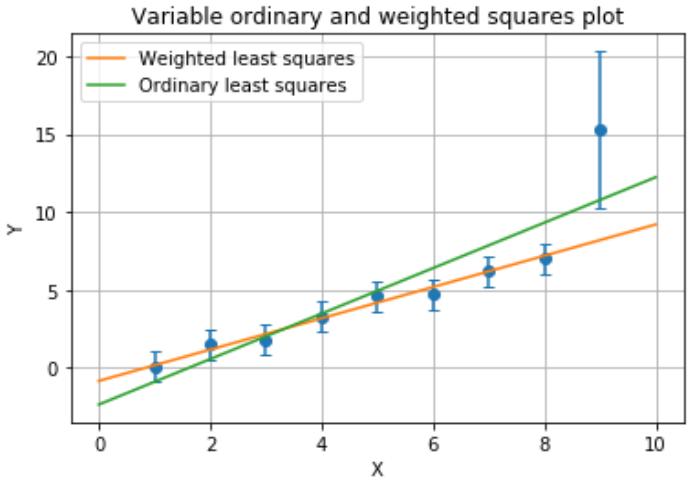

Figure 5.8 - Weighted and unweighted least squares fit. There is larger error on the last data point, so it does not influence the weighted line as much as the other points. Ordinary least squares is not affected by the errors

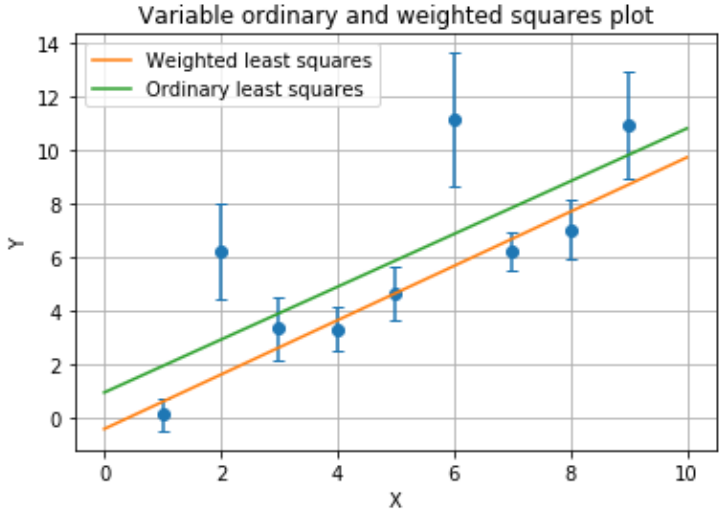

Figure 5.9 - Weighted and unweighted least squares fit. Error is different on each point, so each point lends differing amounts of information to the final line of fit. Smaller error means more significance for a point and vice versa.

5.2.3. Linear fitting real heteroscedastic data

The graph in Figure 5.10 is data from a nine different laboratories. The values plotted are the cross sections of a 121 nm photon after they have collided with a hydrogen atoms electron. These cross sections can be used to infer information about the size of the electron but for this example we will focus on the direct values. Each laboratory measured the cross section and published their measure of the value along with the error on the value which was subject to the method and sophistication of the equipment that was used. The values are all within the same order of magnitude with large overlap of their errors so we can assume that they all agree to a certain degree and all provide useful information in estimating the exact value. Linear fitting this data can provide us with an even better estimate for cross section than any one measurement does.

Laboratories may have bias in their data due to equipment or methods used so when using each measurement to find a better value it is worth using a weighted approach. The data in this example does not consist of an x-y relationship so weighted least squares is not appropriate here although a visual representation is presented in Figure 5.9 using it. To find a better value a weighted mean arithmetic approach should be used as in equations (2.20)-(2.22). The mean and error calculated were as follows:

Weighted mean: 8.0

Weighted mean error: 0.3

This value was plotted in purple with the data on Figure 5.9. It appears to be close to the values with lower mean which is to be expected due to them being more accurate. The mean also has a smaller error than any of the single values, showing how powerful the method can be. Using many estimates of a value an ever more accurate value with less error has been achieved. Furthermore, it provides a better approach to outliers as it allows us to keep them in a data set without incorrectly scaling the resulting value. Assuming each laboratory involved carried out the experiment as objectively as possible and with correct error values stated this value is closer to the true value of the cross section.

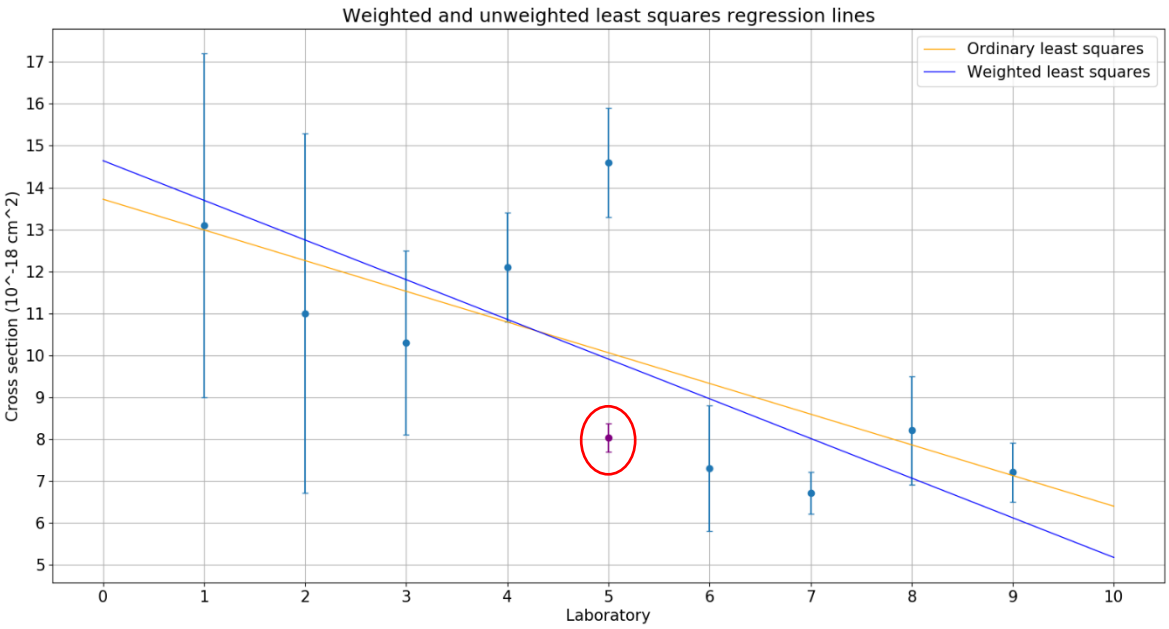

Figure 5.10 – Graph of measurements taken by different laboratories of cross section of a 121.6 nm photon following a collision of an electron with a hydrogen molecule. Ordinary and weighted least squares fits are overlayed. Purple error bar shows the final weighted mean of all the values.

Chapter six: Conclusion

6.1. Comparison of point-by-point and least squares linear fitting

The project aimed to show which linear fitting method was more precise between least squares fitting and point-by-point fitting. Various aspects of the two methods were investigated and least squares was shown to be superior to point-by-point linear fitting.

When experimental data was fitted by both methods the results suggested that least squares provided a better fit. The estimate of resistance of the circuit provided by the point-by-point method included a value for uncertainty which was almost twice that of the least squares estimate, which indicated a value which was known to within a smaller interval. Although this insinuated a superior least squares method it was not definitive proof for it so further investigation was carried out.

Monte Carlo simulation played a large role in showing the precedence for the use of least squares. Far more statistical significance was shown with the 1000 simulations that were run rather than the singular set of data for the experimental section. A distribution of best guesses for each method in every simulation was shown on a histogram. The narrower 95% confidence interval calculated for the distribution of the least squares estimates paired with the standard deviations of the distributions provided evidence that the uncertainty for it was less than that of point-by-point. The percentage difference between the true value of resistance and the estimates from the fits showed that the values provided by least squares was on average much better than that of point-by-point. The max percentage difference from the true value on least squares was approximately half that of point-by-point. A final representation of the superiority of least squares showed that the stability of least squares outshone that of point-by-point. Although the point-by-point method sometimes had estimates close to the true resistance, it's worst estimates were over two and a half times least squares worst estimates which is a clear indicator that its estimate can be stated with far less accuracy.

Compiling all these results the least squares method is clearly superior. Although the real-life experimental results was not a large enough sample to definitively say which method was more precise, but paired with Monte Carlo simulation, the method was shown to be consistently better in every way. Kirkup makes the point that the reason students use the point-by-point method is because they are taught about means as an estimate from data rather than perhaps more complex methods. This makes sense but it has been shown that it is not warranted.

6.2. Comparison of ordinary least squares and weighted least squares linear fitting

The interactive program developed as an educational aid for furthering the understanding of error analysis and curve fitting of undergraduate student operated as required. The points and error on the graph could be adjusted and the lines then immediately changed in response. This was deemed an effective way of showing an undergraduate student the way in which ordinary and weighted least squares change with different data and it makes it very visually clear in what way the weighted least squares gives a different line than ordinary least squares.

Using the program developed ordinary and weighted least squares were compared. The program made it possible to easily and effectively show the different circumstances in which a weighted approach was effective. It was shown that for data which the same error on it throughout the data set, the weighted method operated identically to the ordinary least squares method which is to be expected

as if all points have the same weight they should all effect the linear fit equally. When heteroscedastic data was introduced the methods were contrasted. For this type of data, the ordinary least squares falls short as it treats each point with the same importance. Weighted least squares treats each point with importance proportional to its uncertainty which gives a linear fit that represents the relationship between the x and y variables accurately.

The real data which was plotted outlined a suitable situation for the use of weighted mean arithmetic. Using this method, a value which was more accurate than any of the values used to calculate it was achieved. This gave insight into one possible way in which the quality of results can be improved without doing any extra experimentation but by using existing results in an effective way.

6.3. Final thoughts

The importance of the physical part of experimentation is not in question but this project aimed to show that the work does not stop after this. The correct analysis of results recorded is paramount in improving the experimentation process. As shown, the use of the correct methods, understanding the principles and pushing the limits of accuracy of measurement should be in the back of any physicist's mind during every experiment.

References

1. Kirkup, L. and Frenkel, R., 2006. *An introduction to uncertainty in measurement using the GUM (guide to the expression of uncertainty in measurement)*. Cambridge: Cambridge University Press.

2. Kirkup, L., 2012. *Data analysis for physical scientists*. 2nd ed. Cambridge: Cambridge University Press.

3. Squires, G. L. (2001) "Practical Physics" 4th ed (Cambridge: Cambridge University Press)

4. Kirkup, L. and Frenkel, B., 2019. "Comparison of parameter estimates determined using least squares versus a point-by-point analysis of experimental and simulated data." Physics Education, [online] 55(1), p.015018. Available at: <https://iopscience.iop.org/article/10.1088/1361-6552/ab555a> [Accessed 11 March 2021].

5. Taylor, J. R. (1982) "An Introduction to Error Analysis – The Study of Uncertainties in Physical Measurements" 2nd ed., University Science Books.

6. Allie, S., Buffler, A., Campbell, B. and Lubben, F., 1998. First-year physics students' perceptions of the quality of experimental measurements. *International Journal of Science Education*, 20(4), pp.447-459.

7. Landau, D., & Binder, K. (2009). *A Guide to Monte Carlo Simulations in Statistical Physics* (3rd ed.). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511994944

8. Yang, Y. and Zhao, Q., 2016. Novel pseudo-random number generator based on quantum random walks. *Scientific Reports*, 6(1).

9. Fitting, C., 2021. *Curve Fitting*. [online] PhET. Available at: <https://phet.colorado.edu/en/simulation/curve-fitting> [Accessed 21 April 2021].

10. Peck, R., Short, T. and Olsen, C., 2008. *Introduction to statistics and data analysis*. 3rd ed. Thomson Brooks Cole, pp.445-461,482-495.

11. Teaching.smp.uq.edu.au. 2018. Five-minute physics. [online] Available at: http://teaching.smp.uq.edu.au/fiveminutephysics/index.html#course=physlabs&lecture=LinearRegression

12. Team, S., 2021. *Home — Spyder IDE*. [online] Spyder-ide.org. Available at: <https://www.spyder-ide.org/> [Accessed 21 April 2021].

13. Jupyter.org. 2021. *Project Jupyter*. [online] Available at: <https://jupyter.org/> [Accessed 21 April 2021].

14. Numpy.org. 2021. *NumPy*. [online] Available at: <https://numpy.org/> [Accessed 21 April 2021].

15. Matplotlib.org. 2021. *Matplotlib: Python plotting — Matplotlib 3.4.1 documentation*. [online] Available at: <https://matplotlib.org/> [Accessed 21 April 2021].

Appendix

Code for least squares and point-by-point comparison

```
"""
Created on Sat Mar  6 13:17:33 2021

@author: Conor Lawton
"""

import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rnd
plt.rc('font', size=15,style='normal')

################################################################################

def Y(x,z):
    return x*z
def X(y,z):
    return y/z
def Z(x,y):
    return y/x

################################################################################

def read_file(f_name):
    """
    Reads data file and returns 3 seperated
    lists of columns
    """
    #Open file
    file = open(f_name)
    #read in lines and split data into lists
    file.readline()
    lines = file.readlines()
    column1=np.zeros(len(lines))
    column2=np.zeros(len(lines))
```

```
    column3=np.zeros(len(lines))

    for i in range(len(lines)):
        line = lines[i].split()
        column1[i]=float(line[0])
        column2[i]=float(line[1])
        column3[i]=float(line[2])

    return column1, column2, column3

###############################################################################

def lsq(x,y):
    """
    Takes x variable array(independent) and y variable array(dependent).
    Returns parameters of least squares linear fit for data as well as
    errors on these.
    """
    n = len(x)
    x_bar = (1/n)*sum(x)
    y_bar = (1/n)*sum(y)
    D = sum((x-x_bar)**2)
    m = (1/D)*sum((x-x_bar)*y)
    c = y_bar-(m*x_bar)
    d = y - (m*x) - c
    m_err = (1/D)*((sum(d**2))/(n-2))
    c_err = ((1/n)+(x_bar**2/D))*(sum(d**2)/(n-2))
    return m,c,np.sqrt(m_err),np.sqrt(c_err)

###############################################################################

def pbp(x,y):
    """
    Takes x variable array(independent) and y variabel array(dependent).
    Returns slope of point-by-point linear fit for data as well as
    error on slope.
```

```
    """
    m=sum((y/x)/len(x))
    z=Z(x,y)
    err=np.std(z)/np.sqrt(len(z))
    return m,err

################################################################################

def plotLine(m,c,x_len=11.0,colour='red',label_plot='',width='1'):
    """
    plots line of slope 'slope' with intercept c
    of x length 'x_len'
    """
    x=[]
    y=[]
    x.append(0)
    y.append(c)
    x.append(x_len)
    y.append((m*x_len) + c)
    plt.errorbar(x,y,color=colour,label=label_plot,linewidth=width)
    plt.legend()
    plt.grid(True)

################################################################################

def gaussDist(std,y):
    """
    returns gaussian distributed y values with standard
    deviation 'std' and mean of 'y'
    """
    ar = np.zeros(len(y))
    for i in range(len(y)):
        ar[i]=rnd.normal(y[i],std)
    return ar

################################################################################
```

```
def percent_diff(slope, true_val):
    """
    Returns percentage difference of slope from true_val
    """
    return ((slope-true_val)/true_val)*100

################################################################################

def plot_percent_diff(iterations,x_sim,true_val):
    """
    Plots percentage difference in best estimate and true value for
    least squares and point by point. 'iterations' number of simulations
    """
    lsq_percent=[]
    pbp_percent=[]
    lsqs=[]
    pbps=[]
    sim_iter = []
    for i in range(iterations):
        y_no_noise = Y(true_val,x_sim)
        y_sim = gaussDist(0.1,y_no_noise)
        m_lsq,_,_,_=lsq(x_sim,y_sim)
        m_pbp,_=pbp(x_sim,y_sim)
        lsq_percent.append(percent_diff(m_lsq,true_val))
        pbp_percent.append(percent_diff(m_pbp,true_val))
        lsqs.append(m_lsq)
        pbps.append(m_pbp)
        sim_iter.append(i)

    print('\n\nMean of least squares (1000)',np.mean(lsqs))
    print('Standard deviation of least squares (1000)',np.std(lsqs))

    print('\n\nMean of point-by-point (1000)',np.mean(pbps))
    print('Standard deviation of point-by-point (1000)',np.std(pbps))
```

```
plt.figure()
plt.scatter(sim_iter,lsqs,label='Least squares',alpha=0.5)
plt.scatter(sim_iter,pbps,label='Point-by-point',alpha=0.5)
plt.xlabel('Data set number')
plt.ylabel('Best guesses')
plt.title('Best guesses for different iterations')
plt.legend()

fig = plt.figure()
ax=fig.add_subplot(111)
ax.text(29,33,'Mean(lsq): '+str(round(np.mean(lsqs),3)))
ax.text(29,30,'Std dev(lsq): '+str(round(np.std(lsqs),3)))
plt.hist(lsqs,bins=100,alpha=0.6,label='Least squares')
ax.text(29,27,'Mean(pbp): '+str(round(np.mean(pbps),3)))
ax.text(29,24,'Std dev(pbp): '+str(round(np.std(pbps),3)))
plt.hist(pbps,bins=100,alpha=0.4,label='Point-by-point')

CI1=np.mean(lsqs)-(1.966*np.std(lsqs))/np.sqrt(len(lsqs))
CI2=np.mean(lsqs)+(1.966*np.std(lsqs))/np.sqrt(len(lsqs))
print('Left bound of least squares CI: ',CI1)
print('Right bound of least squares CI: ',CI2)
plt.plot([CI1,CI1],[0,50],color='green',label='95% Confidence interval(lsq)')
plt.plot([CI2,CI2],[0,50],color='green')

CI1=np.mean(pbps)-(1.966*np.std(pbps))/np.sqrt(len(pbps))
CI2=np.mean(pbps)+(1.966*np.std(pbps))/np.sqrt(len(pbps))
print('Left bound of point-by-point CI: ',CI1)
print('Right bound of point-by-point CI: ',CI2)
plt.plot([CI1,CI1],[0,50],color='purple',label='95% Confidence interval(pbp)')
plt.plot([CI2,CI2],[0,50],color='purple')

plt.title('Distribution of best guesses')
plt.xlabel('Best guess')
plt.ylabel('Number of guesses')
plt.legend(loc='best')
```

```
    plt.figure()
    plt.scatter(sim_iter,lsq_percent,label='Least squares',alpha=0.5)
    plt.scatter(sim_iter,pbp_percent,label='Point-by-point',alpha=0.5)

    plt.xlabel('Data set number')
    plt.ylabel('Percentage difference from true value')
    plt.title('Percentage differences')
    plt.legend()

################################################################################

def std_dev(y,slope):
    var = (1/(len(y)-1))*sum((y-slope)**2)
    return(np.sqrt(var))

################################################################################

def plot_std_mean(iterations,x_sim,true_val):
    """
    Plots the uncertainty of slope for 'iterations' number
    of simulations for both least squares and point by point to compare
    """
    lsq_std=[]
    pbp_std=[]
    sim_iter = []
    for i in range(iterations):
        y_no_noise = Y(true_val,x_sim)
        y_sim = gaussDist(0.1,y_no_noise)
        _,_,mErr_lsq,_=lsq(x_sim,y_sim)
        _,mErr_pbp = pbp(x_sim,y_sim)
        lsq_std.append(mErr_lsq)
        pbp_std.append(mErr_pbp)
        sim_iter.append(i)

    fig=plt.figure()
    ax=fig.add_subplot(111)
```

```
    ax.text(800,3.2,'Mean error(lsq): '+str(round(np.mean(lsq_std),3)))
    ax.text(800,3.0,'Std dev on error(lsq): '+str(round(np.std(lsq_std),3)))
    ax.text(800,2.8,'Mean error(pbp): '+str(round(np.mean(pbp_std),3)))
    ax.text(800,2.6,'Std dev on error(pbp): '+str(round(np.std(pbp_std),3)))
    plt.scatter(sim_iter,lsq_std,label='Least squares',alpha=0.5)
    plt.scatter(sim_iter,pbp_std,label='Point-by-point',alpha=0.5)
    plt.xlabel('Iteration number')
    plt.ylabel('Uncertainty')
    plt.title('Uncertainty on best estimate')
    plt.legend(loc='upper left')

################################################################################

#--------------Experimental data-------------------------------------------#
Iexp,Vexp,Rexp=read_file("VIR_data.txt")

m_lsq_exp,c_lsq_exp,mErr_lsq_exp,cErr_lsq_exp = lsq(Iexp,Vexp)
m_pbp_exp,mErr_pbp_exp = pbp(Iexp,Vexp)

plt.scatter(Iexp,Vexp)
plt.title('Point-by-point and least squares fits for experimental data.')
plotLine(m_lsq_exp,c_lsq_exp,x_len=0.1,label_plot='Least squares')
plotLine(m_pbp_exp,0,x_len=0.1,colour='blue',label_plot='Point-by-point')

print('\nExperimental:')

print("\nSlope of least squares: ",round(m_lsq_exp,3))
print("Error on slope for least squares: ",round(mErr_lsq_exp,3))

print("\nSlope of point-by-point: ",round(m_pbp_exp,3))
print("Error on slope for point-by-point: ",round(mErr_pbp_exp,3))



#--------------Monte Carlo sims--------------------------------------------#

I_sim = Iexp
```

```
true_R=33.2
V_no_noise = Y(true_R,Iexp)
V_sim = gaussDist(0.1,V_no_noise)
R_sim = Z(I_sim,V_sim)

m_lsq_sim,c_lsq_sim,mErr_lsq_sim,cErr_lsq_sim = lsq(I_sim,V_sim)
m_pbp_sim,mErr_pbp_sim = pbp(I_sim,V_sim)

print('\n\nMonte Carlo:')

print("\nSlope of least squares: ",round(m_lsq_sim,3))
print("Error on slope for least squares: ",round(mErr_lsq_sim,3))

print("\nSlope of point-by-point: ",round(m_pbp_sim,3))
print("Error on slope for point-by-point: ",round(mErr_pbp_sim,3))

#Percentage differences from true value of resistance
iterations = 1000
plot_percent_diff(iterations,I_sim,true_R)

#Show difference in std_dev on the mean for least squares vs point by point
plot_std_mean(iterations,I_sim,true_R)
```

```
"""
Created on Sun Mar 28 14:29:11 2021

@author: Conor Lawton
"""
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rnd
plt.rc('font', size=15,style='normal')

################################################################################

def Y(x,z):
    return x*z
def X(y,z):
    return y/z
def Z(x,y):
    return y/x

################################################################################

def gaussDist(std,x):
    """
    returns gaussian distributed y values with standard
    deviation 'std' and mean of 'y'
    """
    ar = np.zeros(len(x))
    for i in range(len(x)):
        ar[i]=rnd.normal(x[i],std)
    return ar

################################################################################

def unweighted(x,y):
    n = len(x)
    x_bar = (1/n)*sum(x)
```

```
    y_bar = (1/n)*sum(y)
    D = sum((x-x_bar)**2)
    m = (1/D)*sum((x-x_bar)*y)
    c = y_bar-(m*x_bar)
    d = y - (m*x) - c
    m_err = (1/D)*((sum(d**2))/(n-2))
    c_err = ((1/n)+(x_bar**2/D))*(sum(d**2)/(n-2))
    return m,c,m_err**(1/2),c_err**(1/2)

################################################################################

def weighted(x,y,y_err):
    """
    Takes x variable array(independent),y variable array(dependent) and
    y error array.
    Returns parameters of weighted least squares linear fit for data as well
    as errors on these.
    """
    n = len(x)
    w = 1/(y_err**2)
    x_bar = sum(w*x)/sum(w)
    y_bar = sum(w*y)/sum(w)
    D = sum(w*((x-x_bar)**2))
    m = (1/D)*sum(w*(x-x_bar)*y)
    c = y_bar-(m*x_bar)
    d = y - (m*x)
    m_err = (1/D)*((sum(w*(d**2)))/(n-2))
    c_err = ((1/sum(w))+(x_bar**2/D))*(sum(w*(d**2))/(n-2))
    print(m_err)
    print(c_err)
    return m,c,m_err**(1/2),c_err**(1/2)

################################################################################

def plotLine(m,c,x_len=10,colour='red',label_plot='',width='1'):
    """
```

Code for weighted least squares fitted to real data

```
  plots line of slope 'slope' with intercept 'y_intercept'
    of x length 'x_len'
    """
    x=[]
    y=[]
    x.append(0)
    y.append(c)
    x.append(x_len)
    y.append((m*x_len) + c)
    plt.errorbar(x,y,color=colour,label=label_plot,linewidth=width)
    plt.xticks(np.arange(min(x)-1, max(x)+1, 1))
    plt.yticks(np.arange(int(min(y))-1, max(y)+3, 1))
    plt.legend()
    plt.grid(True)

################################################################################

def weighted_mean(x,err):
    """
    Calculates weighted mean and error of x with individual errors on x values
    """
    w = 1/(err**2)
    mean=(sum(x*w)/sum(w))
    std_err=1/np.sqrt(sum(w))
    return mean,std_err


################################################################################
x = np.array([1,2,3,4,5,6,7,8,9])
y = np.array([13.1,11,10.3,12.1,14.6,7.3,6.7,8.2,7.2])
y_error = np.array([4.1,4.3,2.2,1.3,1.3,1.5,0.5,1.3,0.7])

#Plot data points
plt.errorbar(x,y,yerr=y_error,fmt='o',capsize=3)
```

```
#Plot unweighted least squares fit
m1,c1,m_err1,c_err1 = unweighted(x,y)
plotLine(m1,c1,colour='orange',label_plot='Ordinary least squares')

print('Slope of unweighted least squares fit: ',m1)
print('Error on slope of unweighted least squares fit: ',m_err1)

#Plot weighted least squares fit
m2,c2,m_err2,c_err2 = weighted(x,y,y_error)
plotLine(m2,c2,colour='blue',label_plot='Weighted least squares')

print('Slope of weighted least squares fit: ',m2)
print('Error on slope of weighted least squares fit: ',m_err2)

weight_mean,weight_mean_err=weighted_mean(y,y_error)
print('Weighted mean:',weight_mean)
print('Weighted mean error: ',weight_mean_err)

plt.errorbar([5],[weight_mean],yerr=weight_mean_err,fmt='o',capsize=3,color='purple')

plt.xlabel('Laboratory')
plt.ylabel('Cross section (10^-18 cm^2)')
plt.title('Weighted and unweighted least squares regression lines')
```

Code for interactive program in Jupyter

```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import numpy as np
import matplotlib.pyplot as plt

#Sliders
y0=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y0: ',value=0.1)
e0=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y0: ')
y1=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y1: ',value=1.5)
e1=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y1: ')
y2=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y2: ',value=1.8)
e2=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y2: ')
y3=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y3: ',value=3.3)
e3=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y3: ')
y4=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y4: ',value=4.6)
e4=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y4: ')
y5=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y5: ',value=4.7)
e5=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y5: ')
y6=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y6: ',value=6.2)
e6=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y6: ')
y7=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y7: ',value=7)
e7=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y7: ')
y8=widgets.FloatSlider(min=0.01,max=20,step=0.1,continuous_update=False,description='y8: ',value=8.9)
e8=widgets.FloatSlider(min=0.01,max=10,step=0.1,continuous_update=False,value=1,description='Error on y8: ')
#Containers for sliders
box0 = widgets.HBox([y0,e0])
box1 = widgets.HBox([y1,e1])
box2 = widgets.HBox([y2,e2])
box3 = widgets.HBox([y3,e3])
box4 = widgets.HBox([y4,e4])
box5 = widgets.HBox([y5,e5])
box6 = widgets.HBox([y6,e6])
box7 = widgets.HBox([y7,e7])
box8 = widgets.HBox([y8,e8])
ui = widgets.VBox([box0,box1,box2,box3,box4,box5,box6,box7,box8])
```

```
def weighted_unweighted(e0,e1,e2,e3,e4,e5,e6,e7,e8,y0,y1,y2,y3,y4,y5,y6,y7,y8):
    """
    Plots 9 points of variable data and error which is controlled
    by interactive sliders in Jupyter
    """
    x_lim=10
    xs=np.array([1,2,3,4,5,6,7,8,9])
    y_err=np.array([e0,e1,e2,e3,e4,e5,e6,e7,e8])
    ys=np.array([y0,y1,y2,y3,y4,y5,y6,y7,y8])
    plt.errorbar(xs,ys,yerr=y_err,fmt='o',capsize=3)

    n = len(xs)
    w = 1/(y_err**2)
    x_bar = sum(w*xs)/sum(w)
    y_bar = sum(w*ys)/sum(w)
    D = sum(w*((xs-x_bar)**2))
    m = (1/D)*sum(w*(xs-x_bar)*ys)
    c = y_bar-(m*x_bar)
    d = ys - (m*xs)
    m_err = (1/D)*((sum(w*(d**2)))/(n-2))
    c_err = ((1/sum(w))+(xs**2/D))*(sum(w*(d**2))/(n-2))
    xplot=[]
    yplot=[]
    xplot.append(0)
    xplot.append(x_lim)
    yplot.append(c)
    yplot.append((m*x_lim) + c)
    plt.errorbar(xplot,yplot,label='Weighted least squares')

    x_bar = (1/n)*sum(xs)
    y_bar = (1/n)*sum(ys)
    D = sum((xs-x_bar)**2)
    m = (1/D)*sum((xs-x_bar)*ys)
    c = y_bar-(m*x_bar)
    d = ys - (m*xs) - c
```

```
    m_err = (1/D)*((sum(d**2))/(n-2))
    c_err = ((1/n)+(xs**2/D))*(sum(d**2)/(n-2))
    xplot=[]
    yplot=[]
    xplot.append(0)
    xplot.append(x_lim)
    yplot.append(c)
    yplot.append((m*x_lim) + c)
    plt.errorbar(xplot,yplot,label='Ordinary least squares')

    plt.title('Variable ordinary and weighted squares plot')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)
    plt.legend()
    plt.show()

#Displays sliders and graph
out =
widgets.interactive_output(weighted_unweighted,{'e0':e0,'y0':y0,'e1':e1,'y1':y1,'e2':e2,'y2':y2,'e3':e3,'y3':y3,'e4':e4,'y4':y4,'e5
':e5,'y5':y5,'e6':e6,'y6':y6,'e7':e7,'y7':y7,'e8':e8,'y8':y8})
display(ui,out)
```