Numerical Recipes Project Report

Declan Mathews
s1610357
B103565

November 2018

Abstract

In this project the decay of an unknown nuclei to an unknown daughter nuclei is simulated and data is fit. The observed values are the angle of decay, θ in $[0, 2\pi]$ rads, and time of decay, t in $[0, 10]$ seconds. The probability of decay is given by 2 components with 3 parameters τ_1 ,τ_2 and F, and the observed values. Data is generated by a box method, and plotted as histograms so show the probability density function(PDF) for given values of $\tau_1 = 1$, $\tau_2 = 2$ and F at 0.5, 0 and 1. For each F value the histogram is plotted for the time data and theta data in separately.

A given dataset is then fitted to find the best values of τ_1 ,τ_2 and F. This is done for t only, at $\theta = 2$ rads, and for the full t and θ data. Each set is done by minimizing the negative log likelihood(NLL) and the errors are then found by graphing and extrapolating. The minimisation is done first simply then by the proper method. 30 points were used to graph the NLL data.

For t only with F in $[0, 2 \times 10^{-5}]$, τ_1 in $[0.90, 0.91]$ and τ_2 in $[1.920, 1.935]$ for the graphing; after the simplistic minimization the best values of the parameters are found to be $F = 0.0(9.6389 \times 10^{-6})$, $\tau_1 = 0.90778(\infty)$, $\tau_2 = 1.9275(2.5555 \times 10^{-3})$. When proper minimization is done, with an array of 5 points for the parameter kept constant, $F = 0.0(9.2709 \times 10^{-6})$, $\tau_1 = 0.90778(\infty)$ and $\tau_2 = 1.9275(3.1426 \times 10^{-3})$.

However, when the full data is used and the ranges for the error plots now; F in $[0.542, 0.550]$, τ_1 in $[1.37, 1.42]$ and τ_2 in $[2.66, 2.76]$, the simple method gives: $F = 0.54594(1.8005 \times 10^{-3})$, $\tau_1 = 1.3952(3.7029 \times 10^{-3})$ and $\tau_2 = 2.7088(9.0524 \times 10^{-3})$. By proper method: $F = 0.54594(2.6144 \times 10^{-3})$, $\tau_1 = 1.3952(0.898\,02 \times 10^{-3})$ and $\tau_2 = 2.7088(3.8238 \times 10^{-3})$.

Introduction

The probability density function(PDF) for the decay has two components. Each component has a characteristic decay time, τ, and there is a contribution factor, F, determining the contribution of each component . Thus the total PDF is determined by:

$$PDF = F \times P_1 + (1 - F) \times P_2 \tag{1}$$

where P_1 and P_2 are the given components;

$$P_1(t, \theta; \tau_1) = \frac{1}{N_1} \ (1 + cos^2\theta) \ e^{\frac{-t}{\tau_1}} \tag{2}$$

$$P_2(t, \theta; \tau_2) = \frac{1}{N_2} \ 3sin^2\theta \ e^{\frac{-t}{\tau_2}} \tag{3}$$

Here N_1 and N_2 are normalisation factors that are to be determined. This PDF describes the probability of a decay occurring in a given time. It is given that θ in in the range of $[0, 2\pi]$ rads and t is in the range $[0, 10]$ seconds.

If values for F, τ_1 and τ_2 are provided then the PDF can be plotted as a histogram. This is done by using 'Monte Carlo' methods to generate values and then binning them and plotting the histogram. An example Monte Carlo method to do this is the 'Box method'. This involves generating random values for t and θ, scaling them, finding the PDF at this value and seeing if it is within the accepted range by comparison to a randomly generated value scaled by the max value. If it is, it can be used for the plot. This is repeated for the required number of data points.

However, if data is recorded for unknown F, τ_1 and τ_2 values, fitting methods can be used to determine what are the most likely(i.e. 'best fitting') values for these parameters. A common method to do this is by minimising the negative log likelihood(NLL). This involves summing the negative log values of the PDF for each set of values of t and θ and finding the values of F, τ_1 and τ_2 which give the lowest NLL. The error on the value is found by extrapolating the value of the parameter where NLL is 0.5 above the minimum.

There are many packages which do minimisation which allows passing of the parameters to be minimised and it does the minimisation and returns the minimum values. However, this is a simplistic method and to properly minimize and calculate the proper errors each parameter must be held constant in turn and the others minimised. This is done for a range of values about the minimum and is then repeated for the next parameter to be held constant until the NLL rises by 0.5 at which point you calculate the errors.

By taking the t only data into consideration and then the full t and θ data, differences in behaviour when fitting the data can be seen, especially if the data is correlated. This correlation refers to how the value of one parameter may affect the other, which can cause many difficulties in minimising and so leads to behavioural issues.

Methods

The PDF

By hand the PDF was normalised analytically, this was done for the given ranges of t in $[0, 10]$ and θ in $[0, 2\pi]$, giving:

$$N_1 = 3\pi\ \tau_1(1 - e^{\frac{-10}{\tau_1}}) \tag{4}$$

$$N_2 = 3\pi\ \tau_2(1 - e^{\frac{-10}{\tau_2}}) \tag{5}$$

A class was created for the PDF with two methods implemented, one for each of equation (2) and (3) with the appropriate N_i, which were called on by a third method to calculate and return the total PDF for given t, θ and an array of F, τ_1 and τ_2. For each component of the PDF a catch method was implemented that made any value of $\tau_i \leq 0$ to a negligible value of 1×10^{-6} to prevent maths errors when dividing by 0 or unphysical values of negative τ_i.

Part 1

This section required the distribution to be generated and plotted for various F values. Three classes were implemented to govern this; a class to generate the data(using PDF), a class to plot the data and a class that ran the section, governing iterations and calling the required methods in order. A set number of points, a range for the t and θ values and set F, τ_1 and τ_2 values were prescribed and so were initialised into the `RunPart1` class.

The data was generated for the first set of values using the `GenerateData` class, which implemented the 'box method' as described in the introduction. Firstly random scaled t and θ values were used to find the PDF(for the given F and τ_i) and this was compared to a random value scaled by the maximum value(found analytically to be $\frac{1}{2\pi}$). If the PDF value was lower, it was accepted and this process repeated until the required 10,000 data points were found.

The data was then returned and passed to `GraphPlot` which plotted histograms of the PDF separately for the t and θ data. This entire process was looped over in `RunPart1` for each value of F required.

Minimiser

The minimisation was done by `NLLMinimiser`, a class initialised with the observed data. This contained two methods; `find_nll` and `minimiser`. The latter is provided with a the first guest at the variables and the bounds on which they could be, which is fed into scipy.optimize's minimiser with the former method to minimise the NLL and return the values of the parameters at that point. `find_nll` loops over the entire dataset, finding the log of

the PDF at each point with the supplied parameters from the minimiser and summing them, then makes the total negative and returns the NLL value. There is a catch method if the PDF is not greater than 0, to make it negligibly small(1×10^{-5}) to avoid maths errors. The passing of bounds to the minimiser prevents unphysical results like F not in the range $[0, 1]$ and τ_i could not be negative. `minimiser` returns an array of the minimum values for the parameters and also the minimum NLL.

Simple Error Calculation

For the error calculations of parts 2 and 3, the class `NLLData` was used. It was passed the data to plot and the limits of the plot for the parameters to generate NLL values about the minimum, which could be narrowed by eye until a graph clearly showing the rise of the NLL by 0.5 could be found. A method was used to gather all the NLL data for each point in the array of each of the parameters, with the other two held constant at their minimum. Methods were then used to plot each of these parameters against the NLL and scipy.interpolate was used to extrapolate the value of the parameter at 0.5 above the minimum NLL, and the difference in this and the minimum was returned as the error.

Part 2

From this part on a supplied dataset of t and corresponding θ was used to find the best fitting parameters by minimisation of the NLL(as described in the introduction). In this section, only the t data was to be analysed by simple minimisation. To achieve this, a constant θ value was used, and was chosen as 2 radians as holding this constant ensured that any changes in the values were only due to the time component. The `RunPart2` class was initialised with the data, limits and number of graph points for error calculation and ran the minimisation and error calculation through the classes described above. The bounds for the minimiser ensured only plausible values for the parameters could be found i.e. $F : [0, 1]$ $\tau_i = [0, 1 \times 10^5]$.

The result of minimisation found F to be 0. This caused the error on τ_1 to become infinite and posed a problem for calculating the errors due the NLL not changing as when $F = 0$ the PDF is not affected by P_1 and thus not by τ_1. The method for plotting and extrapolating the error on τ_1 was not called for part 2 and instead a print statement stating the error is infinite was used. Why F became 0 and what this reveals will be discussed in the results section.

Part 3

The full supplied dataset was used here and the exact same steps followed for part 2, except there was no issue with τ_1's error. F was not found to be

0 and so the error could be calculated on all three parameters. `RunPart3` was initialised with the full data, same number of graph points and different limits, which were also found and narrowed by eye. This class then ran the minimisation and error calculation using the classes described above with the same bounds on minimisation as before to ensure plausibility also.

Part 4

The proper error calculation used the same minimiser and graphing classes to minimise and extrapolate the errors. However, `ProperMins` was used to to loop over the array of each parameter and minimise the others while holding it constant. The goal was to get the NLL as low as possible and so for any decrease the values of the parameters and NLL were stored. A while loop was used to prevent excessive run times by only allowing a given number of iterations. An array of the parameter to be held constant was made for a supplied number of points, in the range around the previous minimum(0.03 either side). For the issue when F was minimised to 0, an `if` statement ensured it was only 0.03 above. There was several `if` statements which governed which parameter was being held constant and so the others minimised, this was inside a loop for the list of parameters around the minimum.

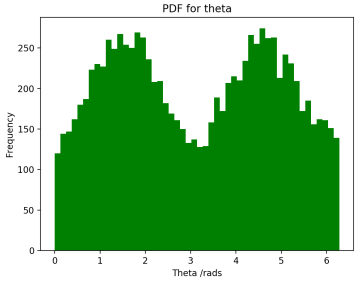
As this was to be run for t only and the full data, there were some `if` statements to skip τ_1 as it had an infinite error due to F being minimised to 0. These were only used when a passed value `part` told the method it was for part 2. To ensure each variable was done at least once a check value was used which if it did not increase then the loops were exited. This is also governed when the proper values were found. After the first iteration of all parameters, if the NLL did not decrease for any point in the array for the held parameter then the loop was exited and the minimum values returned to have the errors found.

Results

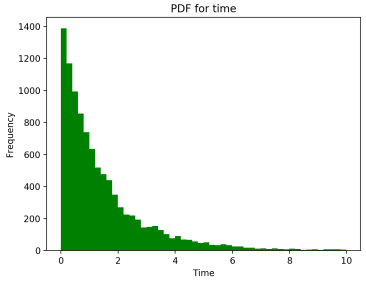
Part 1

Discussion

The plots for t and θ are very easily distinguished between due to the difference in axis and also the entirely different shape. For different F values, the t graphs are much harder to distinguish between as they have a similar shape with varying decay rate. For θ, the graphs for different F values are still easily distinguishable as they have peaks and troughs in different locations, or have much smaller troughs or bigger peaks. However, it can be noticed that whenever the PDF for $F = 1$ rises, the PDF for $F = 0$ falls, suggesting an anti-correlation between P_1 and P_2.

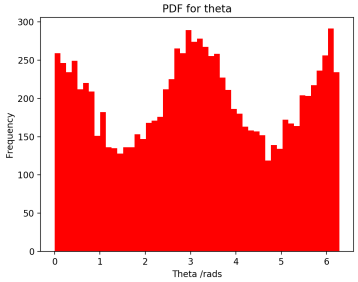


(a) Histogram of PDF for θ

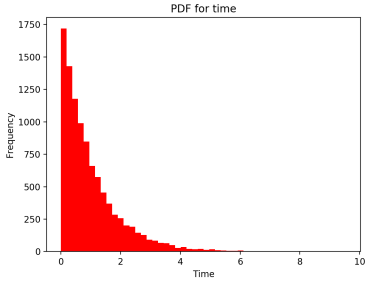


(b) Histogram of PDF for t

Figure 1: PDFs for values at; F = 0.5, $\tau_1 = 1$, $\tau_2 = 2$

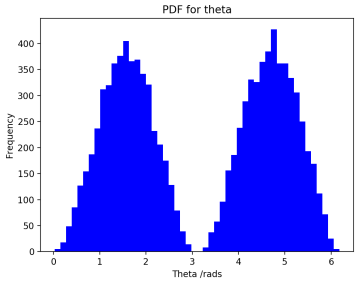


(a) Histogram of PDF for θ

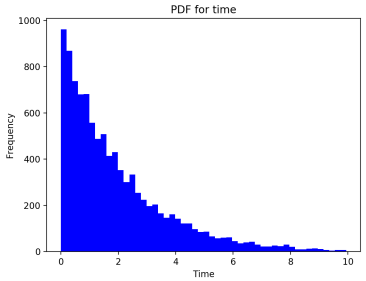


(b) Histogram of PDF for t

Figure 2: PDFs for values at; F = 1, $\tau_1 = 1$, $\tau_2 = 2$



(a) Histogram of PDF for θ



(b) Histogram of PDF for t

Figure 3: PDFs for values at; F = 0, $\tau_1 = 1$, $\tau_2 = 2$

Part 2

Discussion

Due to F being minimised to 0, this meant that there was no contribution to the overall PDF from P_1, which contained all the τ_1 terms. This meant that the error on τ_1 became infinite. When the constant value θ was held

	Minimum Value	Simple Error
F	0.0	9.6389×10^{-6}
τ_1	0.90778	∞
τ_2	1.9275	2.5555×10^{-3}

Table 1: The minimised values and their errors for part 2; the time only data

at was changed, sometimes F became 1 and so τ_2 error became infinite. It was decided that holding the value constant was better than removing all contribution for θ from this section by ignoring the terms as it did not change the PDF terms but merely held them constant so that only the changes due to time were studied. If more time was available, this could be re done by either removing θ terms or integrating over all θ to remove it. It could then be seen if this is due to badly behaved data or an error in the approach to using only the t data.

Part 3

	Min. Val. Part 2	Min. Val. Part 3	Simple Error Part 2	Simple Error Part 3
F	0.0	0.54594	9.6389×10^{-6}	1.8005×10^{-3}
τ_1	0.90778	1.3952	∞	3.7029×10^{-3}
τ_2	1.9275	2.7088	2.5555×10^{-3}	9.0524×10^{-3}

Table 2: The minimised values and their errors for part 2 and 3; the time only and full data sets, in comparison

Discussion

The results from the full data set are much more well behaved. The minimised values are quite different for the two parts and only the error on τ_2 from part 2 is roughly in agreement with part 3. There is no infinite errors and all the values look close to what was expected by the initial guesses for the full data, with errors that are all in agreement. Clearly the data fitting is much more effective when the results are well behaved and give a representative and clear picture of the study.

Part 4

Discussion

The proper errors are all fairly similar to the simple ones. For each part some parameter has an increase in error and others a decrease. Part 3 sees

slightly bigger differences in the errors, especially for both τ errors. The increase in some and decrease in others makes sense as the data is known to be correlated. However, it was thought that all the values errors would increase slightly. With more time the code for the calculating part 4 may need to checked and re evaluated. Their values all still seem reasonable and so they would need to be compared to proper errors produced by a known working programme to be certain if they are correct.

	Simple Error Part 2	Proper Error Part 2	Simple Error Part 3	Proper Error Part 3
F	9.6389×10^{-6}	9.2709×10^{-6}	1.8005×10^{-3}	2.6144×10^{-3}
τ_1	∞	∞	3.7029×10^{-3}	$0.898\,02 \times 10^{-3}$
τ_2	2.5555×10^{-3}	3.1426×10^{-3}	9.0524×10^{-3}	3.8238×10^{-3}

Table 3: The simple errors for part 2 and 3; the time only and full data sets, in comparison with the proper errors

Summary

Part 1 presented graphs that fit the trend to be expected and showed how each component contributed. This also showed that there is anti-correlation which is expected to give trouble to any minimiser. Part 2 gave $F = 0(9.6389 \times 10^{-6})$, $\tau_1 = 0.90778(\infty)$ and $\tau_2 = 1.9275(2.5555 \times 10^{-3})$ which seems reasonable for τ_2 but for the other parameters raised doubts due to the value for F and error on τ_1). This may be due to choosing a constant θ rather than removing it, and so would need to be checked. This could also just be an example of how badly behaved data gives trouble when fitting, especially when combined with anti-correlation. However, part 3 behaved very well and gave $F = 0.54594(1.8005 \times 10^{-3})$, $\tau_1 = 1.3952(3.7029 \times 10^{-3})$ and $\tau_2 = 2.7088(9.0524 \times 10^{-3})$ which seems very reasonable for all parameters. This shows the disparity in ability to fit data when it well behaved rather than not. Part 4 also showed this disparity with errors for t only: $F = 9.2709 \times 10^{-6}$, $\tau_1 = \infty$, $\tau_2 = 3.1426 \times 10^{-3}$ and full data: $F = 2.6144 \times 10^{-3}$, $\tau_1 = 0.898\,02 \times 10^{-3}$, $\tau_2 = 3.8238 \times 10^{-3}$. The t only data continued to have an infinite error while the full data gave errors of reasonable value. Some of the errors decreased while others increased but all remained around the same order of magnitude. It was expected that they'd all increase slightly and so the algorithm for calculating them may need to be checked further to ensure its correctness.

Appendix

Error Plots

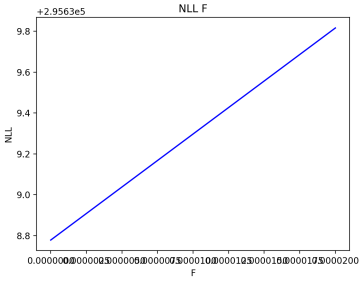


Figure 4: Figure 4: NLL error plot for F for part 2

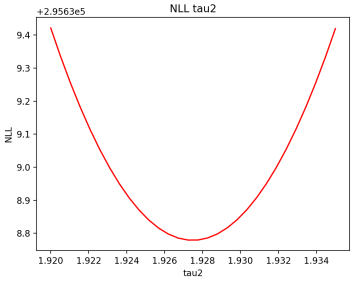


Figure 5: Figure 5: NLL error plot for τ_2 for part 2

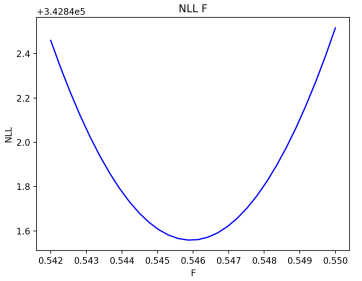


Figure 6: Figure 6: NLL error plot for F for part 3

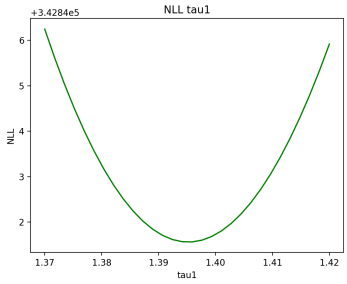


Figure 7: Figure 7: NLL error plot for τ_1 for part 3

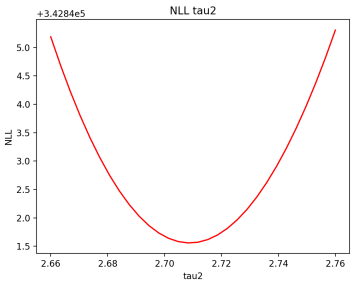


Figure 8: Figure 8: NLL error plot for τ_2 for part 3

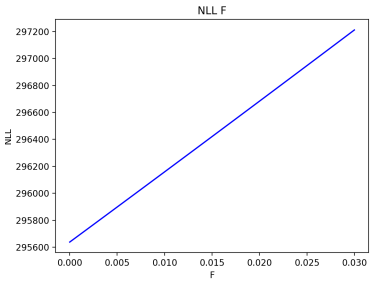


Figure 9: Figure 9: NLL error plot for F for part 4a

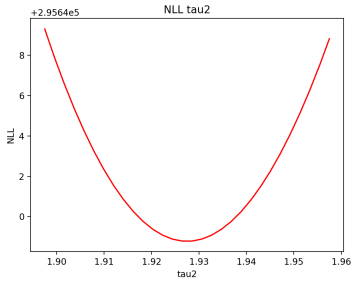


Figure 10: Figure 10: NLL error plot for τ_2 for part 4a

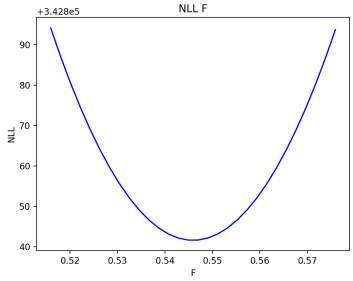


Figure 11: Figure 11: NLL error plot for F for part 4b

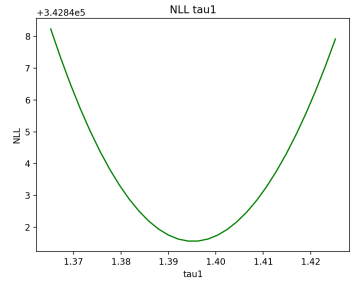


Figure 12: Figure 12: NLL error plot for τ_1 for part 4b

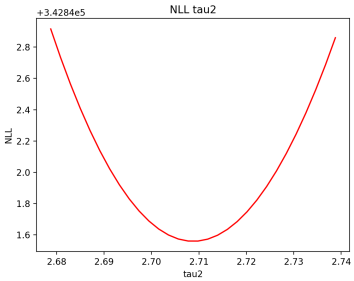


Figure 13: Figure 13: NLL error plot for τ_2 for part 4b

Code

Each class and method has comments included to describe its purpose. There are also the inline comments to detail less clear points. These have been included in the screenshots to explain the code. I will include any further points as captions.

```
#=============================================================================#
#                            Class for PDF                                    #
#=============================================================================#

class PDF(object):

    def __init__(self):
        self.tripi = 3.0 * m.pi   #common constant for each PDF

    '''Finds P1; given time, theta and tau1'''
    def find_P1(self, t, theta, tau1):
        if not tau1 > 0:   #catch method to prevent dividing by 0; made small enough to be negligible but prevent math errors
            tau1 = 1e-6
        P1 = (1.0/(self.tripi * tau1 *(1.0 - m.exp(-10.0 / tau1)))) * (1+m.cos(theta)**2.0) * (m.exp(-t/tau1))
        return P1

    '''Finds P2; given time, theta and tau2'''
    def find_P2(self, t, theta, tau2):
        if not tau2 > 0:  #catch method to prevent dividing by 0; made small enough to be negligible but prevent math errors
            tau2 = 1e-6
        P2 = (1.0/(self.tripi * tau2 *(1.0 - m.exp(-10.0 / tau2)))) * 3.0 * (m.sin(theta)**2.0) * (m.exp(-t/tau2))
        return P2

    '''Finds the total PDF; given time, theta and variable containg Fraction, Tau1, and Tau2'''
    def find_PDF(self, t, theta, var):   #var = [F, tau1, tau2]
        P1 = self.find_P1(t, theta, var[1])
        P2 = self.find_P2(t, theta, var[2])
        PDF = (var[0] * P1) + ((1.0 - var[0])*P2)   #Calculation of fraction of each PDF required and totalled
        return PDF

```

Figure 14: This is the class that returns a value for the PDF for the supplied parameters and variables

```python
#=============================================================================#
#                            Classes for Part 1                               #
#=============================================================================#

#----------Class for Plotting the Generated PDF data as Histograms----------#

class GraphPlot(object):

    def __init__(self, tdata, thetadata, PDFdata, colour):
        self.tdata = tdata
        self.thetadata = thetadata
        self.PDFdata = PDFdata

        '''Gives a different colour for each Fraction'''
        if colour == 0:
            self.colour = 'g'   # For fraction = 0.5
        elif colour == 1:
            self.colour = 'r'   # For fraction = 1.0
        elif colour == 2:
            self.colour = 'b'   # For fraction = 0.0

    def plot(self):
        '''Plot for PDF of time as histogram'''
        f = plt.figure(1)
        plt.hist(self.tdata, bins=50, color = self.colour)
        plt.xlabel('Time')
        plt.ylabel('Frequency')
        plt.title('PDF for time')

        '''Plot of PDF of theta as histogram'''
        g = plt.figure(2)
        plt.hist(self.thetadata, bins=50, color = self.colour)
        plt.xlabel('Theta /rads')
        plt.ylabel('Frequency')
        plt.title('PDF for theta')

        plt.show()

```

Figure 15: The class to plot the generated PDF data against t and θ

```python
#--------------------Class for Generating PDF data--------------------#

class GenerateData(object):

    def __init__(self, var, numpoints, timemax, thetamax):
        self.pdf = PDF()   #Initialises PDF class
        self.var = var   #supplied [F, tau1, tau2] in problem
        self.points = numpoints   #number of decays needed
        self.tmax = timemax   #Supplied maximum time in problem
        self.thetamax = thetamax   #Supplied maximum theta in problem
        self.fmax = 1.0/(2.0*m.pi) + 0.0000001   #Max value of PDF, derived by hand, used for deciding if point is accepted or not
        '''Initialised arrays for generated data'''
        self.tdata = np.array([])
        self.thetadata = np.array([])
        self.PDFdata = np.array([])

    '''Returns Time data'''
    def get_tdata(self):
        return self.tdata

    '''Returns Theta data'''
    def get_thetadata(self):
        return self.thetadata

    '''Generates and scales a random time'''
    def rand_t(self):
        t = np.random.uniform() * self.tmax
        return t

    '''Generates and scales a random theta'''
    def rand_theta(self):
        theta = np.random.uniform() * self.thetamax
        return theta
```

Figure 16: The class to generate the data for the PDF plots, with the methods for generating and scaling random values and returning the t and θ data

```
122
123     '''Finds an allowed PDF value, box-method; If random generated value is less
124        than the max value it is returned'''
125     def evaluate(self):
126         PDF = 0
127         testmax = 10   #these are initialised to begin while loop
128         while testmax > PDF:   #loop continues until acceptable value found by if it is smaller than the test max
129             t = self.rand_t()
130             theta = self.rand_theta()
131             PDF = self.pdf.find_PDF(t, theta, self.var)
132             testmax = self.fmax * np.random.uniform()
133
134         self.tdata = np.append(self.tdata, t)    #acceptable time and theta values stored in array
135         self.thetadata = np.append(self.thetadata, theta)
136         return PDF
137
138     '''Loop to find the required number of data points and returns them as an array'''
139     def find_data(self):
140         for i in range(self.points):
141             PDF1 = self.evaluate()
142             self.PDFdata = np.append(self.PDFdata, PDF1)
143
144         return self.PDFdata
145
```

Figure 17: A continuation of the same class showing the method to find all the data, `find_data`, and the method to find individual points, `evaluate`

```
#=============================================================================#
#                          Classes for Parts 2, 3 and 4                       #
#=============================================================================#

#--------Class for Minimising the Paramaters for Negative Log Likelihood--------#

class NLLMinimiser(object):

    def __init__(self, timedata, thetadata):
        self.pdf = PDF()   #initialises the PDF class
        self.tdata = timedata
        self.theta = thetadata

    '''Finds the negative log likelihood'''
    def find_nll(self, var):
        logsum = 0
        for i in range(len(self.tdata)):   #loop finds indivual log likelihoods and sums them
            PDF = self.pdf.find_PDF(self.tdata[i], self.theta[i], var)

            '''This is a catch method that prevents the log(0) by making PDF = 0 values
               to very small PDF values of 1e-5, preventing math errors'''
            if not PDF > 0:
                PDF = 1e-5

            log = m.log(PDF)
            logsum += log
        nll = -1.0 * logsum   #makes it negative as it is for negative log likelihood
        return nll

    def minimiser(self, guess, bounds):
        [F, tau1, tau2] = guess   #first guess at best values of the variables to be minimised
        var = [F, tau1, tau2]
        minval = sc.minimize(self.find_nll, var, bounds = bounds)
        minis = [minval.x[0], minval.x[1], minval.x[2], minval.fun]
        return minis   #the minimum values for the variables and the NLL returned as array

```

Figure 18: The minimising class with the method to find the NLL which is provided the the minimiser in `minimiser`

```python
#-------------Gathers and Plots NLL data and Determines Errors-------------#

class NLLData(object):

    def __init__(self, timedata, thetadata, minVar, numpoints, Flims, tau1lims, tau2lims):
        self.p = numpoints   #number of graph points
        self.thetadata = thetadata
        self.tdata = timedata
        '''Minimum values of variables and NLL'''
        self.minF = minVar[0]
        self.mintau1 = minVar[1]
        self.mintau2 = minVar[2]
        self.minnll = minVar[3]
        '''This initilaises the limits of the values to find data around for the minimum NLL'''
        self.Fdata = np.linspace(Flims[0], Flims[1], numpoints)
        self.tau1data = np.linspace(tau1lims[0], tau1lims[1], numpoints)
        self.tau2data = np.linspace(tau2lims[0], tau2lims[1], numpoints)
        '''This initialises the arrays for the NLL data to be found and stored in'''
        self.nlldata_F = np.array([])
        self.nlldata_tau1 = np.array([])
        self.nlldata_tau2 = np.array([])

    '''This gathers the NLL data for the variables in the given range for the given number of points'''
    def gather_nlldata(self):
        foo = NLLMinimiser(self.tdata, self.thetadata)
        for i in range(self.p):
            '''NLL data for F'''
            varF = [self.Fdata[i], self.mintau1, self.mintau2]
            self.nlldata_F = np.append(self.nlldata_F, foo.find_nll(varF))
            '''NLL data for tau1'''
            vartau1 = [self.minF, self.tau1data[i], self.mintau2]
            self.nlldata_tau1 = np.append(self.nlldata_tau1, foo.find_nll(vartau1))
            '''NLL data for tau2'''
            vartau2 = [self.minF, self.mintau1, self.tau2data[i]]
            self.nlldata_tau2 = np.append(self.nlldata_tau2, foo.find_nll(vartau2))
```

Figure 19: Class for generating the NLL data to be plot, plotting it and extrapolating errors. The data generation method is shown

```
    '''Plots the generated NLL data against the F and returns the error on F'''
    def plot_nll_F(self):
        plt.plot(self.Fdata, self.nlldata_F, 'b')
        plt.title('NLL F')
        plt.xlabel('F')
        plt.ylabel('NLL')
        F_ex = interp1d(self.nlldata_F, self.Fdata, fill_value = 'extrapolation')   #creates F = F(NLL)
        dF = np.absolute(self.minF - F_ex(self.minnll + 0.5))   #extrapolates error of F at 0.5 above minNLL
        plt.show()   #comment out to allow running from start to finish without plots
        return dF

        '''Plots the generated NLL data against the tau1 and returns the error on tau1'''
    def plot_nll_tau1(self):
        plt.plot(self.tau1data, self.nlldata_tau1, 'g')
        plt.title('NLL tau1')
        plt.xlabel('tau1')
        plt.ylabel('NLL')
        tau1_ex = interp1d(self.nlldata_tau1, self.tau1data, fill_value = 'extrapolation')   #creates tau1 = tau1(NLL)
        dtau1 = np.absolute(self.mintau1 - tau1_ex(self.minnll + 0.5))   #extrapolates error of tau1 at 0.5 above minNLL
        plt.show()   #comment out to allow running from start to finish without plots
        return dtau1

        '''Plots the generated NLL data against the tau2 and returns the error on tau2'''
    def plot_nll_tau2(self):
        plt.plot(self.tau2data, self.nlldata_tau2, 'r')
        plt.title('NLL tau2')
        plt.xlabel('tau2')
        plt.ylabel('NLL')
        tau2_ex = interp1d(self.nlldata_tau2, self.tau2data, fill_value = 'extrapolation')   #creates tau2 = tau2(NLL)
        dtau2 = np.absolute(self.mintau2 - tau2_ex(self.minnll + 0.5))   #extrapolates error of F at 0.5 aopve minNLL
        plt.show()   #comment out to allow running from start to finish without plots
        return dtau2

```

Figure 20: The same class, showing the plotting and extrapolation methods

```
#=====================================================================#
#                          Classes for Part 4                         #
#=====================================================================#

#----------------------Proper Minimums Calculation---------------------#

class ProperMins(object):

    def __init__(self, timedata, thetadata, minVar, heldpoints, maxits):
        self.tdata = timedata
        self.thetadata = thetadata
        self.mini = NLLMinimiser(self.tdata, self.thetadata)
        self.minVar = minVar    #[F, tau1, tau2, NLL]
        self.its = 0   #iterations counter
        self.maxits = maxits   #maximum iterations for cases where may run for too long
        self.p = heldpoints   #number of points to iterate over for the variable held constant  while minimising

    def find_properMins(self, part):
        bool = 0   #this is used to exit the nested loops when a new minimum is not found
        while self.its <= self.maxits:
            self.its += 1   #increase iterations
            print('Currently on iteration ' + str(self.its))
            for i in range(len(self.minVar) - 1):   #-1 as last value in array is NLL
                helddata = np.linspace(self.minVar[i]-0.03, self.minVar[i]+0.03, self.p)   #array for constant variable
                '''This is for F in part 2, as its min is 0 the range cannot have the -0.03 or it is negative, which is unphysical'''
                if (part == 2) & (i == 0):
                    helddata = np.linspace(self.minVar[i], self.minVar[i]+0.03, self.p)

                check = 0   #this is used to check if a new minimum has not been found and so the minimisation is finished
                for j in range(len(helddata)):
                    '''This section is for when F is constant'''
                    if i == 0:
                        print('Currently minimising over F please wait...')
                        if not helddata[j] >= 0:   #this is a catch method incase F is made negative, if so it is just made to be 0
                            helddata[j] = 0
                        #the bounds for the constant term are the same so it is held at that value by the minimiser
                        bounds = [(helddata[j], helddata[j]), (0, 1e5), (0, 1e5)]
                        guess = [helddata[j], 1.0, 2.0]   #the first guess at the minimums
                        newMins = self.mini.minimiser(guess, bounds)
```

Figure 21: A class to find the lowest possible NLL and return the values of the parameters

```
                    if newMins[3] < self.minVar[3]:
                        #This is the checking method if a new minimum is found then the minimiser must continue but with
                        #the values of the minimum stored, if one isn't then it is complete if one full iteration over each varaible has been done
                        check += 1
                        self.minVar[3] = newMins[3]
                        self.minVar[0] = newMins[0]
                        self.minVar[1] = newMins[1]
                        self.minVar[2] = newMins[2]
                        print('NLL minimum was updated for F')

                    if (part == 2) & (self.its == 1):  #Ensures that the minimisation continues on the first iteration of each variable
                                                       #so each variable gets held constant and checked for a new minimum once
                        check += 1

                elif (i == 1) & (part == 3):   #For minimisation of tau1 for the full data
                    print('Currently minimising over tau1 please wait...')
                    bounds = [(0, 1), (helddata[j], helddata[j]), (0, 1e5)]   #bounds hold tau1 constant by same method again
                    guess = [0.5, helddata[j], 2.0]   #first guess
                    newMins = self.mini.minimiser(guess, bounds)

                    if newMins[3] < self.minVar[3]:   #This is the same checking method for continuation of minimisation
                                                      #and saving of minimum values as before
                        check += 1
                        self.minVar[3] = newMins[3]
                        self.minVar[0] = newMins[0]
                        self.minVar[1] = newMins[1]
                        self.minVar[2] = newMins[2]
                        print('NLL minimum was updated for tau1')

                    if self.its == 1:   #Ensures that each variable gets held constant for minimisation on the first loop round
                        check += 1

                elif (i == 1) & (part == 2):   #Used for the time only data as its error is infinite it cannot be minimised and so
                                               #is skipped for run time
                    check += 1

                elif i == 2:   #Used for tau2
                    print('Currently minimising over tau2 please wait...')
                    bounds = [(0, 1), (0, 1e5), (helddata[j], helddata[j])]   #tau2 held constant
                    guess = [0.5, 1.0, helddata[j]]   #first guess
                    newMins = self.mini.minimiser(guess, bounds)
```

Figure 22: This is a continuation of the class, showing the `if` statements to govern which parameter is being minimised and if it is successful updating the values

```
332
333                     if newMins[3] < self.minVar[3]:   #Check for new minimum and saving the values at minimum
334                         check += 1
335                         self.minVar[3] = newMins[3]
336                         self.minVar[0] = newMins[0]
337                         self.minVar[1] = newMins[1]
338                         self.minVar[2] = newMins[2]
339                         print('NLL minimum was updated for tau2')
340
341                 if check == 0:    #If no new minimum found, or not first iteration, check is still at 0 and so minimisation is
342                                   #complete and the loop can be exited
343                     bool += 1   #needed as this is a nested loop that requires a double break
344                     break
345             if bool == 1:
346                 break
347
348         print('The total iterations for the proper minimisation was: ' + str(self.its))
349
350         return self.minVar
351
```

Figure 23: The final part of this class showing the exit condition and returning the values

```
#=============================================================================#
#                        Classes for Running All Parts                        #
#=============================================================================#

#-------------------------Class for Running Part 1----------------------------#

class RunPart1(object):

    def __init__(self, numpoints, timerange, thetarange, var):
        self.numdatapoints = numpoints   #number of data points
        self.timerange = timerange
        self.thetarange = thetarange
        self.var = var   #[F, tau1, tau2]

    def run(self):
        starttime = time.time()
        print('\nNow finding the data for the PDF plots as histograms...')

        for i in range(len(self.var[0])):   #this loop is to repeat for each given fraction
            nexttime = time.time()
            passvar = [self.var[0][i], self.var[1], self.var[2]]
            Gen = GenerateData(passvar, self.numdatapoints, self.timerange[1], self.thetarange[1])
            PDFdata = Gen.find_data()
            tdata = Gen.get_tdata()
            thetadata = Gen.get_thetadata()

            print('The data for this fraction took ' + str(time.time()-nexttime) + ' s to collect')

            Plots = GraphPlot(tdata, thetadata, PDFdata, i)   #comment these out to run from start to finish with no break to show plots
            Plots.plot()

```

Figure 24: This class runs part 1, calling the required classes to generate the data and plot it, for the varying F value

```
#-------------------------Class for Running Part 2-------------------------#

class RunPart2(object):

    def __init__(self, data, lims, points):
        self.tdata = data[0]   #time data
        self.thetadata = data[1]   #theta data
        self.NLL =  NLLMinimiser(self.tdata, self.thetadata)   #Initialises minimiser
        '''These are found and altered to give the best picture of the NLL in a reasonable time'''
        self.Flims = lims[0]
        self.tau1lims = lims[1]
        self.tau2lims = lims[2]
        self.numpoints = points

    def run(self):
        starttime = time.time()
        bounds = [(0, 1), (0, 1e5), (0, 1e5)]   #prevents F going outside allowed range of [0,1] and tau being negative
        guess = [0.5, 1.0, 2.0]   #first guess
        #minVar = [0, 0.9077752640044711, 1.9274864323044965, 295638.7786384368]   #Can be used to save time re running the minimiser
        print('\nNow minimising for the time only data...')
        minVar = self.NLL.minimiser(guess, bounds)
        print('The minimisation of time only data took ' + str(time.time() - starttime) + ' s to complete')

        print('\nThe minimised value of F for the time only data is: ' + str(minVar[0]))
        print('The minimised value of tau1 for the time only data is: ' + str(minVar[1]) + ' s')
        print('The minimised value of tau2 for the time only data is: ' + str(minVar[2]) + ' s')

        print('\nCurrently finding the NLL data for the time only data errors...')
        NLLGraphing = NLLData(self.tdata, self.thetadata, minVar, self.numpoints, self.Flims, self.tau1lims, self.tau2lims)   #for plotting minimum
                                                                                                                                #curve to extrapolate for error
```

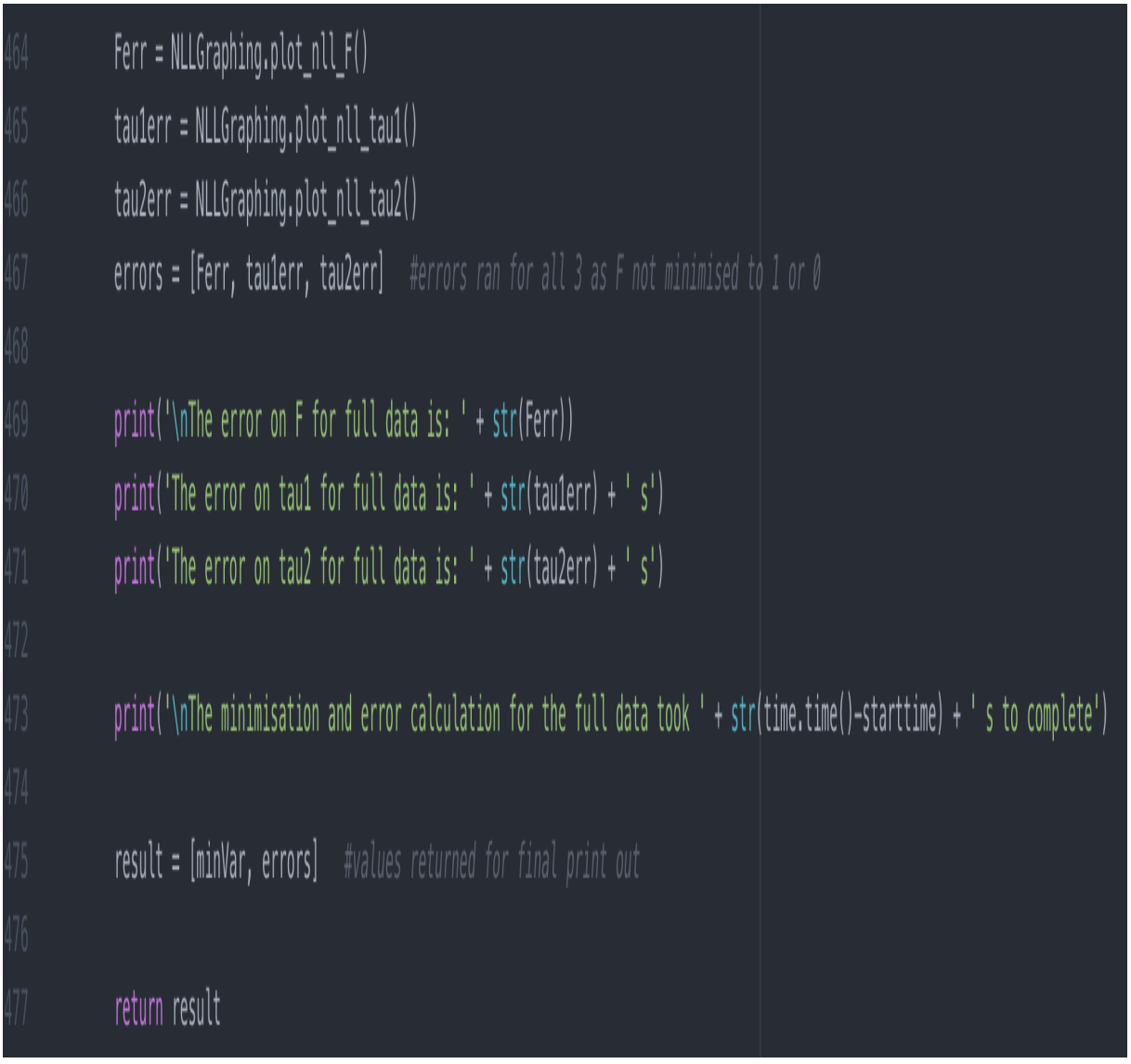
Figure 25: This class runs part 2, calling the minimisation of the parameters and then graphing and finding their errors

```
413         nexttime = time.time()
414         NLLGraphing.gather_nlldata()
415         print('The NLL data for time only data took ' + str(time.time() - nexttime) + ' s to gather')
416 
417         Ferr = NLLGraphing.plot_nll_F()
418         tau2err = NLLGraphing.plot_nll_tau2()
419         errors = [Ferr, 'Infinity', tau2err]   #as F is minimised to 0, tau1 error is infinity and so the NLL graph will give straight line
420 
421         print('\nThe error on F for time only data is: ' + str(Ferr))
422         print('The error on tau1 for time only data is infinite due to the 0 value for fraction')
423         print('The error on tau2 for time only data is: ' + str(tau2err) + ' s')
424 
425         print('\nThe minimisation and error calculation for the time only data took ' + str(time.time()-starttime) + ' s to complete')
426 
427         result = [minVar, errors]   #passes back results for final print out
428 
429         return result
430 
```

Figure 26: This is the section of the class that finds the errors

```
#----------------------------Class for Running Part 3----------------------------#

class RunPart3(object):

    def __init__(self, data, lims, points):
        self.tdata = data[:,0]   #time data is the left column
        self.thetadata = data[:,1]   #theta data is the right column
        self.NLL =  NLLMinimiser(self.tdata, self.thetadata)   #Initialises minimiser
        '''These are found and altered to give the best picture of the NLL in a reasonable time'''
        self.Flims = lims[0]
        self.tau1lims = lims[1]
        self.tau2lims = lims[2]
        self.numpoints = points

    def run(self):
        starttime = time.time()
        bounds = [(0, 1), (0, 1e5), (0, 1e5)]   #prevents F outside [0,1] and negative tau
        guess = [0.5, 1.0, 2.0]
        #minVar = [0.5459322023278369, 1.395158844491453, 2.708695633202384, 342841.558345506]   #Can be used to save time re running the minimiser
        print('\nMinimising for the full data...')
        minVar = self.NLL.minimiser(guess, bounds)
        print('The minimisation of full data took ' + str(time.time() - starttime) + ' s to complete')

        print('\nThe minimised value of F for the full data is: ' + str(minVar[0]))
        print('The minimised value of tau1 for the full data is: ' + str(minVar[1]) + ' s')
        print('The minimised value of tau2 for the full data is: ' + str(minVar[2]) + ' s')

        print('\nCurrently finding the NLL data for the full data errors...')
        NLLGraphing = NLLData(self.tdata, self.thetadata, minVar, self.numpoints, self.Flims, self.tau1lims, self.tau2lims)
        nexttime = time.time()
        NLLGraphing.gather_nlldata()
        print('The NLL data for the full data took ' + str(time.time() - nexttime) + ' s to gather')
```

Figure 27: This class runs part 3, doing the same as part 2 but now initialised with the full data



```
464         Ferr = NLLGraphing.plot_nll_F()
465         tau1err = NLLGraphing.plot_nll_tau1()
466         tau2err = NLLGraphing.plot_nll_tau2()
467         errors = [Ferr, tau1err, tau2err]   #errors ran for all 3 as F not minimised to 1 or 0
468
469         print('\nThe error on F for full data is: ' + str(Ferr))
470         print('The error on tau1 for full data is: ' + str(tau1err) + ' s')
471         print('The error on tau2 for full data is: ' + str(tau2err) + ' s')
472
473         print('\nThe minimisation and error calculation for the full data took ' + str(time.time()-starttime) + ' s to complete')
474
475         result = [minVar, errors]   #values returned for final print out
476
477         return result
```

Figure 28: This is same class showing the error calculation. Also calculates τ_1 errors now

```
#----------------------------Class for Running Part 4-------------------------#

class RunPart4(object):

    def __init__(self, timedata, thetadata, minVar, heldpoints, graphpoints, maxits):
        self.tdata = timedata
        self.thetadata = thetadata
        self.p = heldpoints   #number of points in array for the variable held constant
        self.graphp = graphpoints   #number of points for graphing of the minimum NLL to extrapolate for error
        self.minVar = minVar
        self.maxits = maxits   #maximum iterations incase of excessive runtime
        self.PropMins = ProperMins(self.tdata, self.thetadata, self.minVar, self.p, self.maxits)

    def run(self, part):
        starttime = time.time()

        '''If statements depend on if done for time only or full data, 2 being time only'''
        if part == 2:
            print('\nNow calculating the proper minimum values for part 2...')
        elif part == 3:
            print('\nNow calculating the proper minimum values for part 3...')

        propMins = self.PropMins.find_properMins(part)

        print('The calculation of the properly minimised values took: ' + str(time.time()-starttime) + ' s')

        '''If statements depend on if done for time only or full data, 2 being time only'''
        if part == 2:
            print('\nThe properly minimised value of F for the time only data is: ' + str(propMins[0]))
            print('The properly minimised value of tau1 for the time only data is: ' + str(propMins[1]) + ' s')
            print('The properly minimised value of tau2 for the time only data is: ' + str(propMins[2]) + ' s')

        elif part == 3:
            print('\nThe properly minimised value of F for the full data is: ' + str(propMins[0]))
            print('The properly minimised value of tau1 for the full data is: ' + str(propMins[1]) + ' s')
            print('The properly minimised value of tau2 for the full data is: ' + str(propMins[2]) + ' s')

        if part == 2:
            print('\nCurrently finding the NLL data for the time only data proper errors...')
        if part == 3:
            print('\nCurrently finding the NLL data for the full data proper errors...')
```

Figure 29: This class runs part 4, with `if` statements to determine if is for t only or the full data, based on the supplied part value. The proper minimums are found here

```
520
521     '''These if statements change the limits of F for plotting as for time only data(part = 2) F
522     is minimised at 0 and F cannot be lower than 0 so -0.03 on lower range not needed'''
523     if part == 2:
524         [Flims, tau1lims, tau2lims] = [[propMins[0], propMins[0]+0.03], [propMins[1]-0.03, propMins[1]+0.03], [propMins[2]-0.03, propMins[2]+0.03]]
525     if part == 3:
526         [Flims, tau1lims, tau2lims] = [[propMins[0]-0.03, propMins[0]+0.03], [propMins[1]-0.03, propMins[1]+0.03], [propMins[2]-0.03, propMins[2]+0.03]]
527     NLLGraphing = NLLData(self.tdata, self.thetadata, propMins, self.graphp, Flims, tau1lims, tau2lims)
528     nexttime = time.time()
529     NLLGraphing.gather_nlldata()
530
531     print('The NLL data for the full data took ' + str(time.time() - nexttime) + ' s to gather')
532
533     Ferr = NLLGraphing.plot_nll_F()
534     '''tau1 error cannot be found for time only as F is minimised to 0 so tau1 error is infinite => math error'''
535     if part == 3:
536         tau1err = NLLGraphing.plot_nll_tau1()
537     tau2err = NLLGraphing.plot_nll_tau2()
538
539     '''If statements just to set tau1 error due to it being infinity for time only data'''
540     if part == 2:
541         errors = [Ferr, 'Infinity', tau2err]
542     elif part == 3:
543         errors = [Ferr, tau1err, tau2err]
544
545     print('\nThe proper error on F for full data is: ' + str(Ferr))
546     '''If statements just to set tau1 error due to it being infinity for time only data'''
547     if part == 3:
548         print('The proper error on tau1 for full data is: ' + str(tau1err) + ' s')
549     elif part == 2:
550         print('The proper error on tau1 for full data is still infinite due to F being 0')
551     print('The proper error on tau2 for full data is: ' + str(tau2err) + ' s')
552
553     print('\nThe minimisation and proper error calculation for the full data took ' + str(time.time()-starttime) + ' s to complete')
554
555     result = [propMins, errors]   #result for final print out
556
557     return result
```

Figure 30: This is the error calculation from the graphing for the same class

```
#=============================================================================#
#                    Main Functions to Run Certain Sections                   #
#=============================================================================#

#-----------------Gathers the data provided from given file------------------#

def get_Filedata(filename):
    data = np.loadtxt('datafile-Xdecay.txt')
    return data

#----------------Displays the final results of the programme-----------------#

def final_data(result2, result3, result4_2, result4_3):
    print('\nThe final results are:')

    print('\nPart 2, Time only data: ')
    print(' Minimised Values for: ')
    print('  F: '+str(result2[0][0])+'\n  tau1: '+str(result2[0][1])+'\n  tau2: '+str(result2[0][2]))
    print(' The errors for: ')
    print('  F: '+str(result2[1][0])+'\n  tau1: '+str(result2[1][1])+'\n  tau2: '+str(result2[1][2]))

    print('\nPart 3, Full data: ')
    print(' Minimised Values for: ')
    print('  F: '+str(result3[0][0])+'\n  tau1: '+str(result3[0][1])+'\n  tau2: '+str(result3[0][2]))
    print(' The errors for: ')
    print('  F: '+str(result3[1][0])+'\n  tau1: '+str(result3[1][1])+'\n  tau2: '+str(result3[1][2]))

    print('\nPart 4, Time only data: ')
    print(' Properly minimised Values for: ')
    print('  F: '+str(result4_2[0][0])+'\n  tau1: '+str(result4_2[0][1])+'\n  tau2: '+str(result4_2[0][2]))
    print(' The proper errors for: ')
    print('  F: '+str(result4_2[1][0])+'\n  tau1: '+str(result4_2[1][1])+'\n  tau2: '+str(result4_2[1][2]))


    print('\nPart 4, Full data: ')
    print(' Properly minimised Values for: ')
    print('  F: '+str(result4_3[0][0])+'\n  tau1: '+str(result4_3[0][1])+'\n  tau2: '+str(result4_3[0][2]))
    print(' The proper errors for: ')
    print('  F: '+str(result4_3[1][0])+'\n  tau1: '+str(result4_3[1][1])+'\n  tau2: '+str(result4_3[1][2]))

```

Figure 31: This is a function to gather the data from the supplied file and a function to print out all the final results from the sections.

```
#------------The main function to provide input and run programme-----------#

'''
All of the values of any parameter that can be varied are changed here

For the given values there are results in arrays commented out, these can be commented in and the minimisations and error
calculations commented out to save time for checking part 4 or final print out etc.
'''
def main():
    initialtime = time.time()

    '''Part 1 - Generating and plotting PDFs'''
    numdatapoints = 10000   #problem asked for 10000 data points
    timerange = [0, 10]   #time range was given as [0,10]
    thetarange = [0, 2.0*m.pi]   #angle range can only be [0, 2pi] as this is full circle
    [F, tau1, tau2] = [[0.5, 1.0, 0.0], 1.0, 2.0]   #part 1 gave tau1 as 1 and tau2 as 2, and asked for plots when F = [0.5, 0, 1]
    variables = [F, tau1, tau2]   #stores variables as array

    Run1 = RunPart1(numdatapoints, timerange, thetarange, variables)
    Run1.run()

    '''Part 2 - Simple minimisation and error calculation of time only data'''
    filename = 'datafile-Xdecay.txt'   #generalised to allow for different files
    fulldata = get_Filedata(filename)   #data returned as per np.loadtxt
    numpoints = 30   #the number of points for the NLL plots

    timedata = fulldata[:,0]   #seperation of time data as this is for time only in part 2
    thetadata = [2.0]*len(timedata) #theta chosen as 2 due to first peak in plots from part 1 at 2 radians
    part2data = [timedata, thetadata]   #stored at one array

    Flims2 = [0, 2e-5]   #these are the limits I found from running the programme and by eye finding the minimum,
                         # then decreasing the limits to the point where the increase of 0.5 is easily found
    tau1lims2 = [0.9, 0.91]   #this will not matter as F is 0 but is included for completeness
    tau2lims2 = [1.92, 1.935]   #error may not be found and programme will break if these are changed in
                                #certain ways e.g. if minimum is not inside the range
    lims2 = [Flims2, tau1lims2, tau2lims2]

    Run2 = RunPart2(part2data, lims2, numpoints)
    result2 = Run2.run()
    #result2 = [[0.0, 0.9077752640044711, 1.9274864323044965, 295638.7786384368], [9.638941146106613e-06, 'Infinity', 0.0025554615308012174]]
    minVar2 = result2[0]
```

Figure 32: Here is the main which has all of the variables that are to be supplied, and runs each section of the assignment. The results are stored and commented out so each section can be run independently

```
    '''Part 3 - Simple minimisation and error calculation for full data'''
    Flims3 = [0.542, 0.55]   #these ranges were found by eye and show the NLL change for 0.5 clearly and allow error calculation
    tau1lims3 = [1.37, 1.42]   #error may not be found and programme will break if these are changed in certain
                               #ways e.g. if minimum is not inside the range
    tau2lims3 = [2.66, 2.76]
    lims3 = [Flims3, tau1lims3, tau2lims3]

    Run3 = RunPart3(fulldata, lims3, numpoints)
    result3 = Run3.run()
    #result3 = [[0.5459395723599973, 1.395176446104108, 2.7087544941754746, 342841.55831819074], [0.0018004541642643312, 0.003702886982983377, 0.00905
    minVar3 = result3[0]

    '''Part 4 - Proper minimisation and error calculation for time only data then full data'''
    helddatapoints = 5   #number of points in array for constant variable in proper error calculation; drastically affects run time
    maxits = 3   #maximum iterations
    part2 = 2   #tells proper error calculation this is for time only data; needed due to infinite error on tau1 for time only

    Run4 = RunPart4(part2data[0], part2data[1], minVar2, helddatapoints, numpoints, maxits)
    result4_2 = Run4.run(part2)
    #result4_2 = [[0.0, 0.9077752640044711, 1.9274864323044965, 295638.7786384368], [9.270858397442995e-06, 'Infinity', 0.0031426308792368296]]

    part3 = 3   #tells proper error calculation this is for full data; needed due to infinite error on tau1 for time only

    Run4 = RunPart4(fulldata[:,0], fulldata[:,1], minVar3, helddatapoints, numpoints, maxits)
    result4_3 = Run4.run(part3)
    #result4_3 = [[0.5459395723599973, 1.395198605038462, 2.708767892955916, 342841.55831150897], [0.002614404985655905, 0.00089802383462123
    final_data(result2, result3, result4_2, result4_3)   #prints final results

    timer = (time.time() - initialtime) / 60.0
    print('\nThe total time for this programme is ' + str(timer) + ' mins')
main()
```

Figure 33: This is the last section of the main, doing the same as before but this shows it for part 3 and part 4. The final results are also printed out