**Computational Physics Exercise 2**

*Caspar Fuller 1310006*

**Numerical Differentiation**

This exercise was focused on comparing the functionality of two methods of differentiation, the forward (1) and central (2) difference formulae.

(1)

(2)

The function being differentiated is shown in equation (3), the first and second derivatives will be found and an investigation into the error and accuracy of the method can be undertaken to find an optimal value for *h* for both derivatives.

*(3)*

To solve this problem code was written that initially iterated for a range of x-values for a user inputted value of h. X-values ranged between 0≤*x≤*2 and results, including an error value found by taking the absolute difference between the approximation and the true value calculated using the derivative shown in equation (4), were written to a file. The calculation of the approximations were carried out with the use of auxiliary functions to compute the forward and central formulae which were called into the file function when required.

(4)

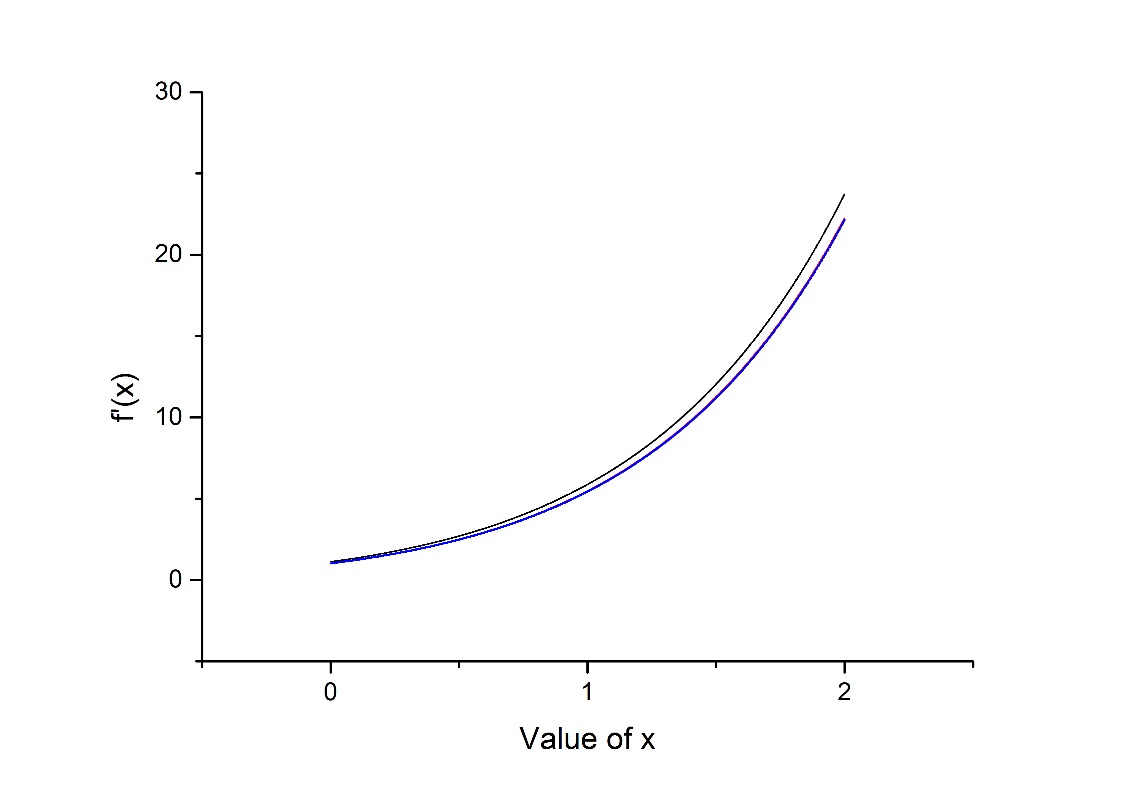
A second file was then created in which results were stored from varying the h-value used for a given x, this was done in a similar method to before. After some analysis it was found that the error values calculated in each iteration had to be initialised after they had been printed to the file otherwise they summed for the whole range causing inaccurate data. Additionally when printing the file data for varying h-values it was important to give a high level of accuracy (e.g 15 decimal places). If the level of accuracy was not sufficient then the computer would simply print zero to the file making results hard to interpret correctly.

Once again to solve for the second derivative a method akin to the previous ones was employed. By once again varying h and using the user inputted x-value the second derivative could be found using the following formula:

(5)

A separate auxiliary function was used to compute equation (5) and the data was written to a third file for analysis.

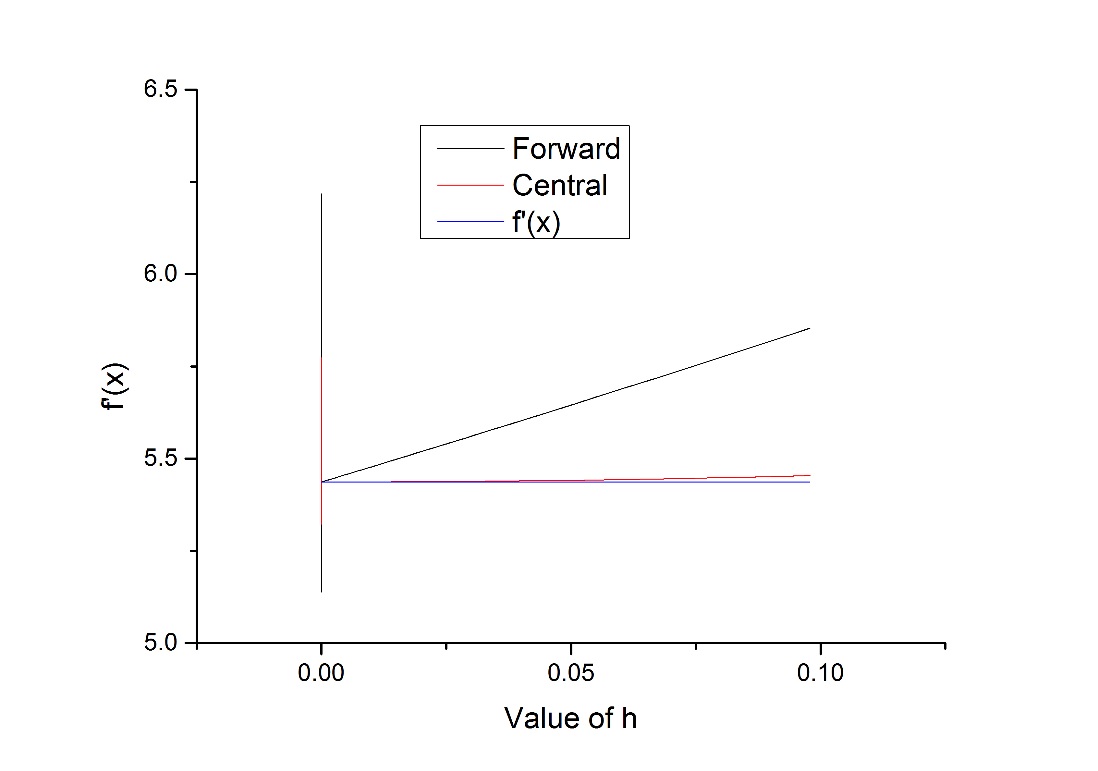
Figure 1 shows the plot of the forward and central formulae approximations compared to the true value of the derivative computed in excel. The h value used here was chosen as 0.1 to show the difference in approximations clearly.



*Figure 1 – Plot of f’(x) for various approximation methods over a range of x.*

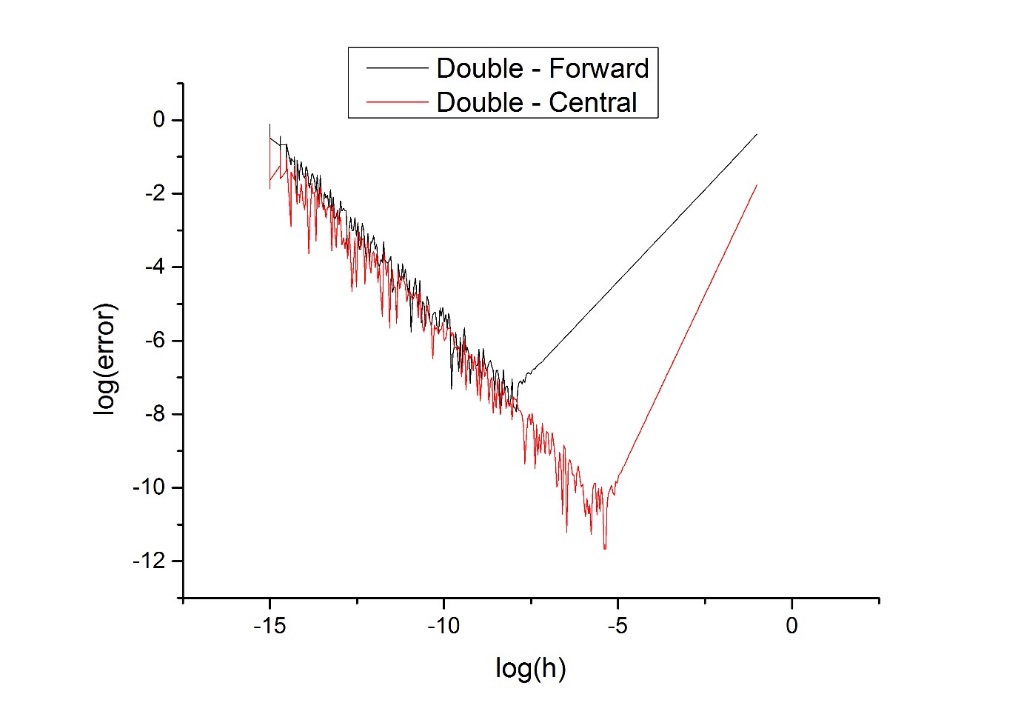
The line for the central formula (red) cannot be seen as it is covered directly by the plot of the true value for the derivative (blue) showing a good level of accuracy for the central formula. The forward formula (black) is seen to be accurate at low x’s however as x increases the approximation becomes less accurate. Figure 2 shows a plot for varying h values whilst keeping x constant at 1. Although the graph is unclear at very low values of h a clear trend can be seen, where the approximations are more accurate at lower h values and as h increases the accuracy gets worse. The f’(x) plot shows the true value for the derivative and so it can be graphically seen here how much more accurate the central formula is. The plot as h tends towards zero results from the limitations in the computer for storing such low h values. It cannot handle the required accuracy so gives inaccurate data as a result. This is shown also in figure 3.

The error in these plots agree well with the error equation used to find the error in the forward and central formula. For x=1.5 at h=0.1 the error or difference in the forward formula is 0.78430 which correlates well to the difference between the true value and approximation of 0.81896. For the central formula the difference is found to be 0.03361 from the formula and the difference given by the program is 0.03364, this result is very similar showing a good level of accuracy with both the approximation of error and the program itself.



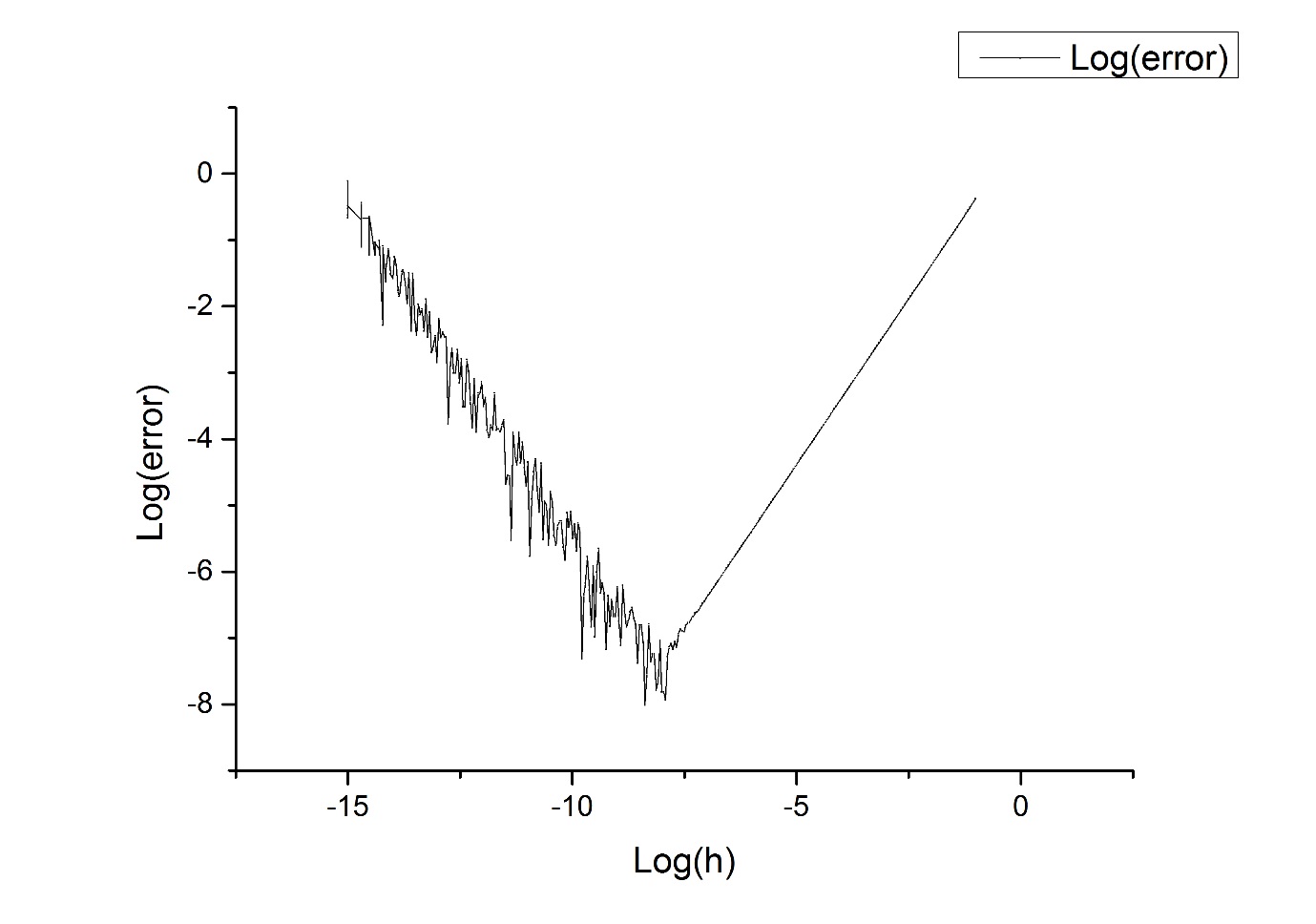
*Figure 2 – Plot of f’(x) by various approximation methods for a range of h values.*

Figure 3 shows a plot for the log of the modulus of the error for the forward and central formulae against the log of the modulus of h, as h varies. This allows us to quantify how much more accurate the central method is compared to the forward formula. This difference in accuracy was found to be of the order of 1,000. Optimal values for h can also be found from figure 3, there is a clear point where the computer can handle the numbers involved accurately again and stops the random varying as well as the error being at a minimum. This was found to be approximately 1.52336for the central formula and 1.42725for the forward formula. This value for the forward formula varies from the suggested optimal h-vale for a double precision point of 3.16228 however this could be explained by systematic error in accuracy when calling the separate functions.

****

*Figure 3 – Plot of log(error) against log(h)*

To improve this part of the program I would carry out investigations around the approximate optimal h values to find a more accurate value for h. Another extension would be to use single point precision variables instead of double and compare the difference in accuracy then.

Similar analysis was used to find an optimal value for h for the second derivative with figure 4 showing the resultant plot. This gives an optimum h value of approximately h=1.17896. Once again the error was seen to correspond badly with the formula given to find an approximate of the error. For h= 0.0099380 and x= 1.5 the difference was found from the program to be 0.078277 however from the formula the error was calculated at 0.00020287. This agreement is inaccurate because the value’s being used in the calculations are very large, however the errors are very small leading to inaccuracies.

*Figure 4 – Plot of log(error) against log(h) for the second derivative*

**Numerical Integration via the Simpsons Rule**

The purpose of this exercise was to solve the equations of motion for the period, T, of a simple pendulum for large angle of oscillations. Code was needed to execute the Simpsons Rule (6) to compute the integral shown in equation (7) . The Simpsons Rule is similar to the trapezium rule however a sum of parabolic approximations is taken instead of straight line approximations, the formula is:

(6)

where the distance between A and B is divided into N intervals of even width where N is an even number. The width of each interval or step height, Δh is found using the formula. The integral being computed is an elliptic integral of the first kind and has the following form:

(6)

I will be evaluating the period for a range of initial oscillation angles and for different values of N to test whether the approximation improves or otherwise. I will then compare the results from the Simpsons rule to the small angle approximation (7) finding the angle limit for when the approximation differs by 10%. The small angle approximation is:

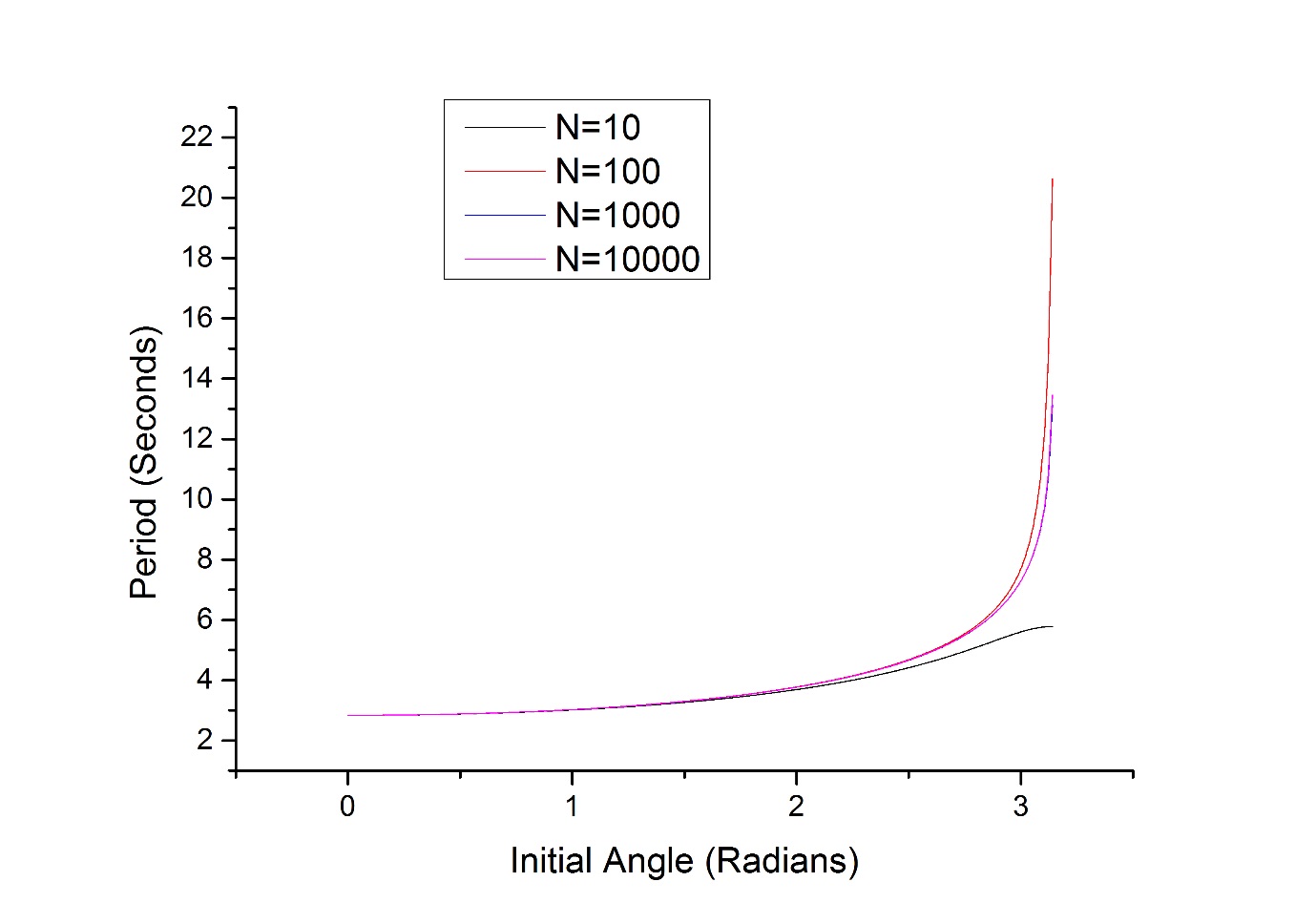
(7)

To solve this problem code was written to input the length and initial oscillation angle of the pendulum. Conditions were set on both these inputs however during the initial angle input the user is given an option to enter ‘1234’ which will vary the initial angle between the limits 0≤x≤π. The check statement for the value of ‘1234’ is put before the check to see if the initial angle is within limits (as 1234>π) so the program diverges here to prevent the error message from being seen. This works because two exit points have been used within the program allowing the user to choose which part of the problem to carry out.

If the user enters a specific initial angle a function is called to employ Simpsons rule for that value with a user inputted value of N which is checked to be even. The code works by iterating over each N value between 1≤n≤(N-1), checking each time to see if N is odd. If it is odd it is summed to an ‘odd sum’ variable with the correct coefficient. If it is not found to be odd it is summed to an ‘even sum’ with the even coefficient of 2. Once the program has finished the loop the first and last terms are added on with no coefficients, and the full sum, multiplied by is printed onscreen. It was important to initialise each sum before the iterations began otherwise they were assigned random values which corrupted the data.

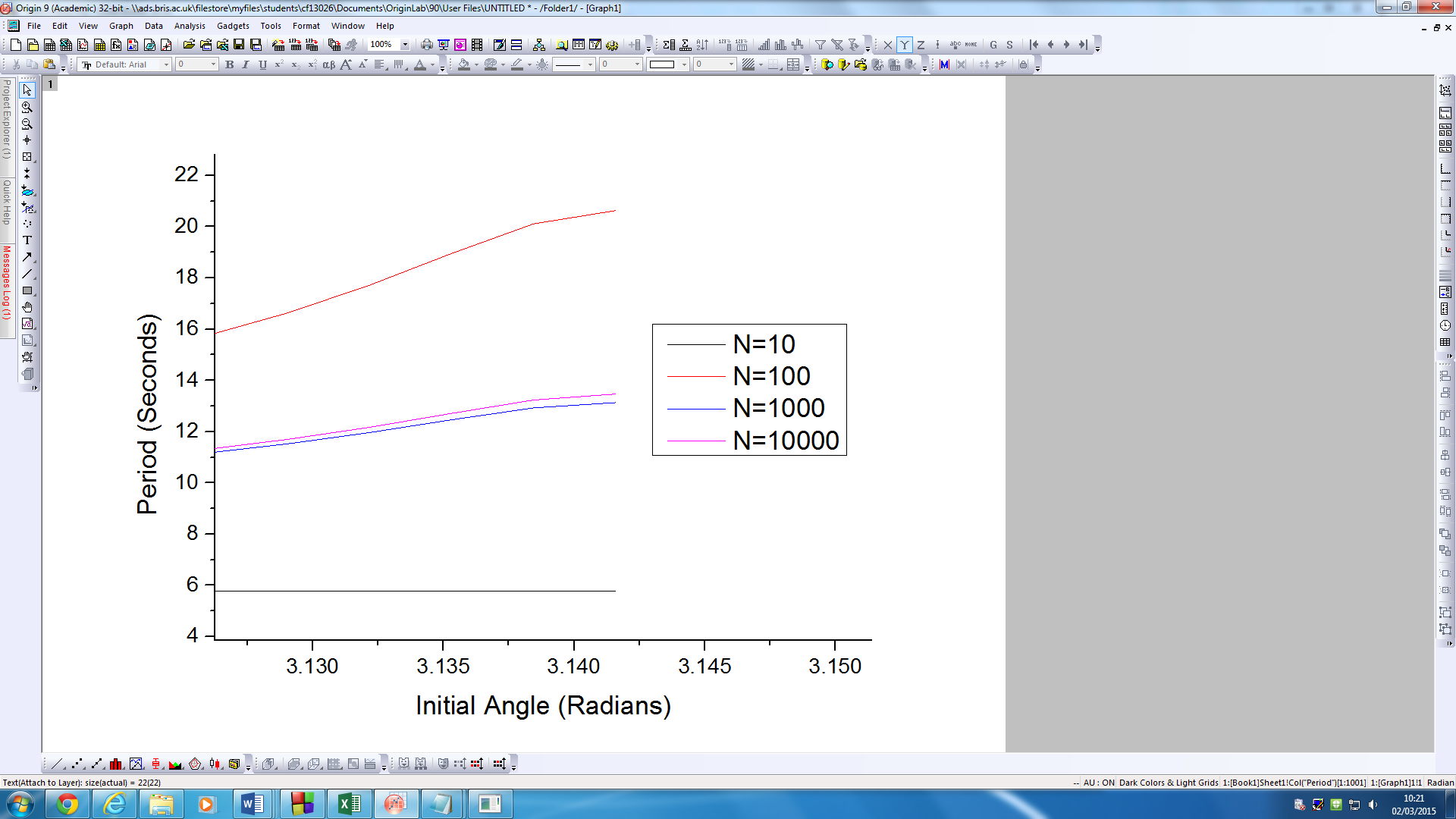
If the user chose to vary the values for the initial angles a separate function was called to perform this. A number of for loops were used to carry this out, iterating over many initial angles between the limits. For each angle a range of N values were used 10≤N≤10,000 where N is always even and is increased by =\*10 , and for each N value the same code as before was used to employ the Simpsons rule. The results were written to a file for analysis.

The code was tested initially for a pendulum with a length of 2 metres and initial angle of oscillation of 0.5 radians. The number of approximations was set to 20 and the results was found to be T=2.881013 seconds. An online large amplitude simulator [1] was used to compare results to, this gave a result of T=2.881969 seconds so it can be stated that the code was sufficiently accurate.

Next the initial angle was varied for various N values to study the effect of N with a pendulum of 2 metres. Figure 5 shows these results. The graph shows that for the majority of angles the approximated for the period is generally the same for different N’s however when the upper limit is reached this changes. For low N the result is an under estimate. If N is increased slightly (N=100) then this approximation becomes an over estimate. Only when N is made even larger do we see the approximation beginning to converge on the true value, this is shown more clearly in figure 6. These results seem reasonable as the Simpsons rule converges on the true value as N tends to infinity.

*Figure 5 – Plot of Theta against period for different N values*

To find the angle for which T varies from the small angle limit (7) by 10% the value of T needs to exceed 3.120711 seconds. By plotting this value as a straight line in figure (5) and finding the intersect It can be found that for N=10,000 the first to vary by 10% is 1.212655 radians.



*Figure 6 – Zoom on figure 5 to show difference in accuracy*

The code worked sufficiently in this exercise however one improvement that was made would was the formatting of the file the data is read into. The file’s data initially had the results for each N value for each one after another, making analysis of just N=10 results for example quite complex. A solution was found on excel by separating out the different N values, however if the loop ran for each N first, then looped over the range of values then the results could be easily analysed.

A second improvement that could be used was when inputting and checking the user inputted values. It was found that if a character was entered instead of a number then the do while loop would enter an infinite loop, crashing the program. No solution was implemented for this unfortunately but the method to solve this was studied. It required to take the input argument as a stream of characters and check to see if I is an integer or double. If the computer finds it is it will set it as required and if not it will throw out the input and ask for another.