**Computational Physics – Exercise 1 – Report**

*Caspar Fuller 1310006*

**Problem 1 – Series expansion of a function**.

The aim of this exercise was to write a program to compute a Taylor series expansion of the arc-tangent function using the following Taylor series:

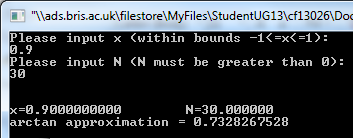
Where x and N are arguments inputted by the user. Comparisons can be made with the built in arc-tangent function to test the efficiency and accuracy of the Taylor series by varying the x and N values within the code and graphically comparing the outputs. It is also possible to investigate the functional limits of the program, finding the limit of N after which the program will not converge. Finally the program can be used to approximate Pi to varying degrees of accuracy, firstly by expanding 4\*arctan(1) and then by using the identity:

Comparisons can then be made between the methods to find the most efficient one.

To solve this problem a program was set-up to take x and N as user inputted arguments, then a loop was called to carry out successive iterations of equation (1) summing each iteration to give an approximation for the arc-tangent(x). When x was varied a similar loop was used however the loop ran between a range of x’s, given by the condition using the previous inputted N as the argument for N. Varying N again used a similar loop, which kept x constant as the inputted value, and varied between 1≤N≤500. Both sets of results were written to a file which also contained results for the difference between the built in arctan(x) function and the Taylor expansion. To find the value of N the gives arctan(x) to 7 significant figures was carried with the use of a double for loop if an if condition, which cycled between the previously stated range of x values and 1≤N≤1000. When the difference was below the required accuracy the results were written to a file. It was found however that the limit on N was too small to find an N value for x=1 or x=-1. To do this a separate bit of code had to be written, this was in the form of a while loop which incremented N by a much larger increment than 1. Once the required accuracy was found at it was necessary to loop again beginning at with the increment halved in value. The conditions on the while loop forced this process to continue until the increment became 1 therefore finding an exact value for N. The fact that the arctan(x) function is an odd function was used to argue that the N value for x=1 is the same for x=-1. This saved the program from running through the lengthy loop again increasing efficiency.

To approximate Pi using the identity 4\*arctan(x) a while loop setup much like the previous one, checking the difference between the approximation and the built in Pi value for required accuracy. Evaluating Pi using equation (2) involved a simpler while loop which called the arctan(x) approximation three times incrementing N by one each iteration until the required accuracy was found.

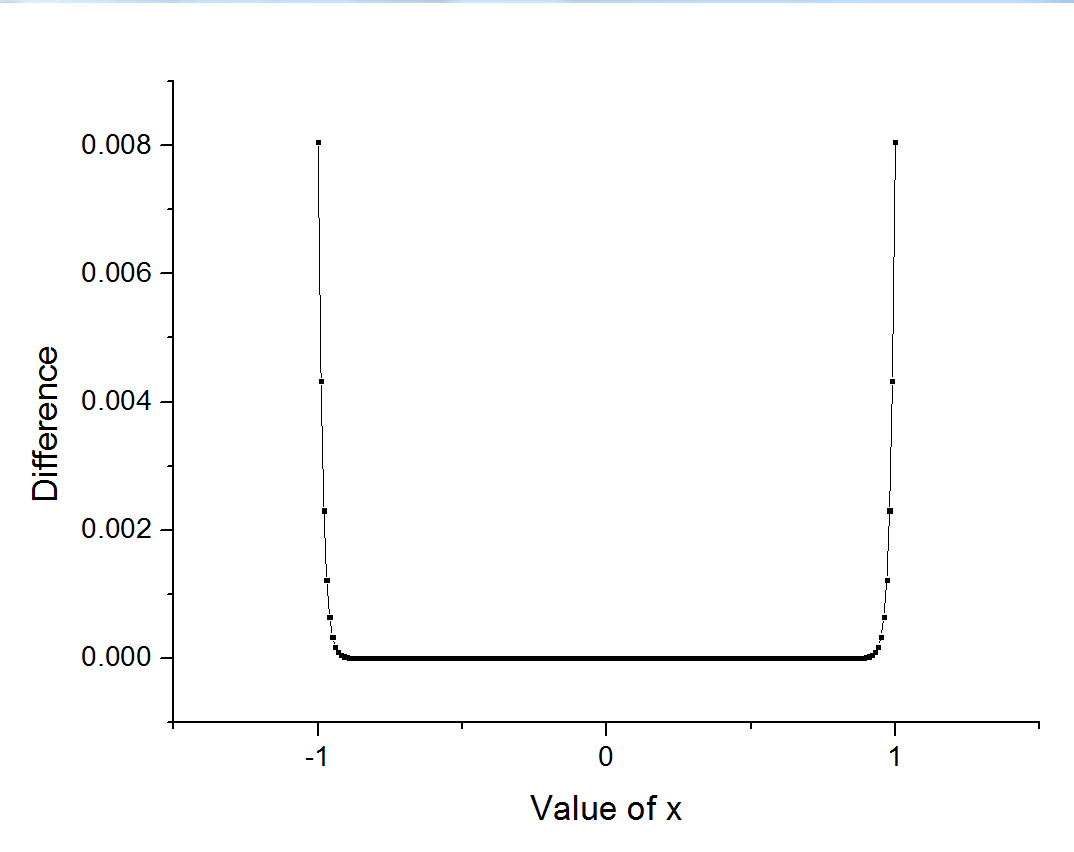
The program approximated correctly for user inputted values as shown in the figure below.



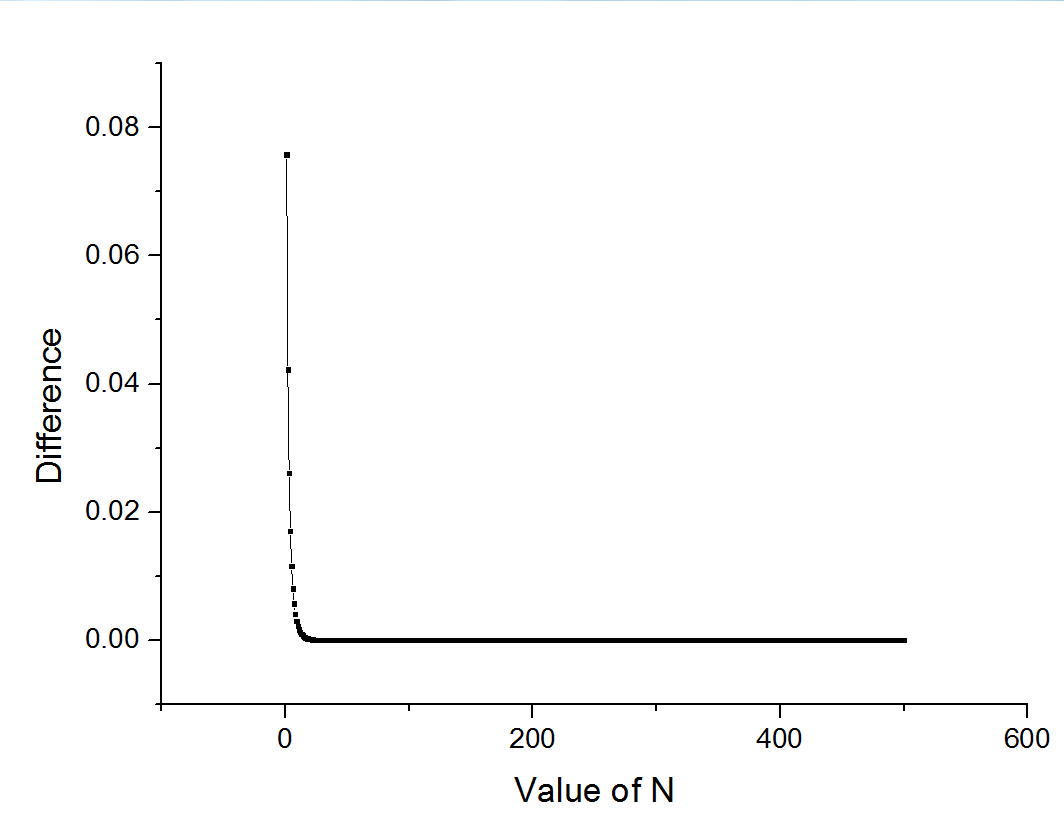
*Figure (1) – Program output for user inputted values.*

Error checks on the input value using a while loop allowed the program to refuse invalid inputs and prompt for an input again. This stopped the program from having to be restarted each time making it more user friendly.

When x was varied between 1 and -1 (figure 2) it was seen that in general the difference between the built in function and the approximation was zero, except when the function neared 1 and -1. The difference increased rapidly here. This is due to the fact that the Taylor series fails to converge at these limits and above them so the difference continues to increase to infinity.

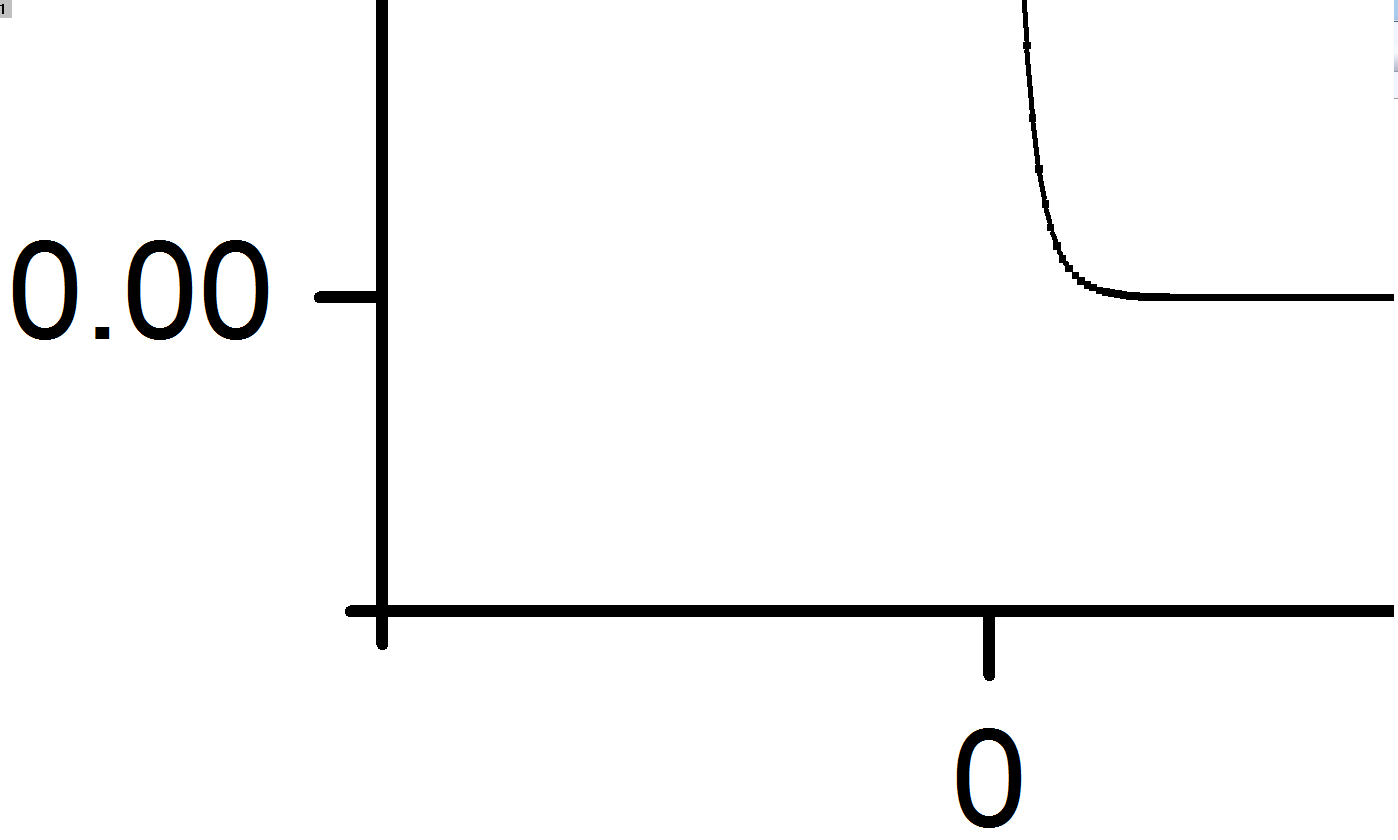


*Figure (2) – Plot of Difference when x is varied (N was held constant at 30)*

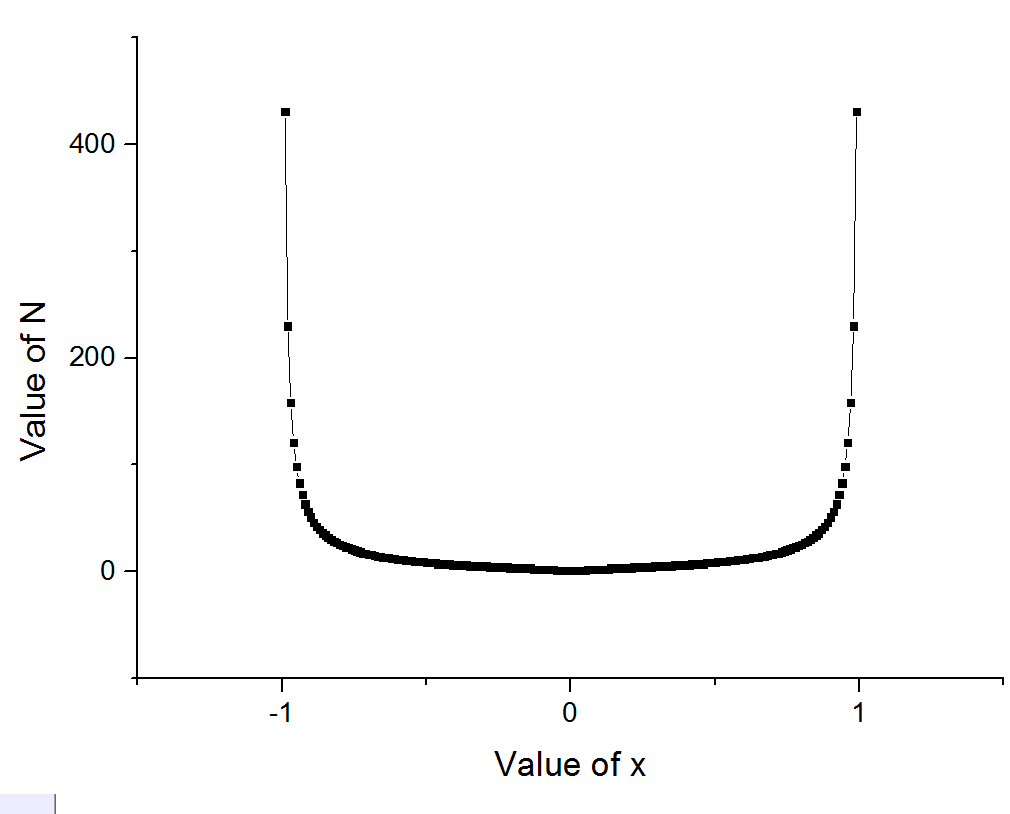


*Figure 3 – Plot of Difference as N is varied (x was held constant at x=0.9)*

When N was varied it was seen that at larger N’s the approximation converged well onto the result shown in figure (3). However when N was below approximately 80 there was still a small difference between the approximation and true value. Figure 4 shows this rapid increase in difference as N decreased, fewer iterations restricted the expansion from converging on the true value meaning it became more of an approximation.



*Figure 4 – Zoomed in version of figure (3) on the range 0≤N≤100*

Figure 5 shows the results for finding N to 7 significant figures. The plot only includes -0.99≤x≤0.99 as the values for x=1 and x=-1 were found to be N=2499998. This would have rendered the plot worth less as it would scale down the other results significantly. This large N value is due to the difficulty the expansion has converging when reaching the bounds of x. The calculation of N for x=1 took a long time for the program to compute. By analysing the code, one reason this took so long may be due to the fact the arctan(x) approximation function was called during each iteration. However if it is was put directly into the loop and not called the computation time would theoretically decrease as the computer would have many less sums to carry out.

*Figure 5 – Plot of varying x against the corresponding value of N to give an accuracy of 7 significant figures*

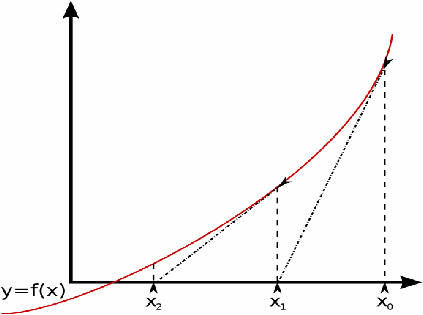
As the function reaches the bounds of x the program requires larger N’s to converge. However due to the limits of the program it is possible to find a maximum value for N, after which the program will fail to converge at all. This was found using the DBL\_MAX code in the <float.h> library to show the maximum double value possible (as N was set as a double.). As the expansion has the expression it is appropriate to say the maximum value for N corresponds to:

This computes to approximately N=8.998, this value is very large so the user will not need to worry about the maximum value of N if it is typed as a double. However if it was cast as an integer this limit becomes N=1073741823. It is worth noting at this point that the program will not carry out the Taylor series expansion explicitly. Instead it will use a combination of Chebyshev polynomials to compute the calculation. This method is far quicker than computing the Taylor expansion and allows the program to run at a greatly increased rate.

When approximating Pi by using Pi=4\*arctan(1) many iterations were performed leading to a long calculation time. This was due to the same reason as when arctan(1) was evaluated so a similar improvement could be used to increase the efficiency of the program. Pi was approximated by this method in 6,535,944 iterations. Initial the value was thought to be 10,000,004 iterations however when the rounding method of the program was investigated it was found that this value was an over estimate for the value of N. The accuracy was initially set at 0.0000001 however due to the truncation of the program this accuracy resulted in the over estimated value. If the accuracy was set to 0.000000153 then the program converged on the correct value for N as the program now finds the first value of N that satisfies Pi to 7 significant figures π=3.141593. This accuracy was found as it corresponds to the Pi value of 3.141562501 which is the first approximation that rounds to Pi correct to 7 significant figures. When using the identity in equation (2) the method converged far quicker, in 18 iterations giving Pi to 12 significant figures, π=3.141592653590 . This identity uses Fibonacci numbers to approximate Pi. It is more efficient than the previous method as the x values used in the expansions are not near the bounds of x causing the function to converge quicker. There are an infinite number of these Fibonacci identities which could be used, so as an improvement different identities could be tested to see if they are more effective, two such identities are listed below.

**Problem 2 – Newton Raphson Method**

The aim of this exercise was to write a program to solve a quartic equation by employing the Newton Raphson method by linear approximation. By guessing a value for the root, , the program will converge on the root by expanding the quartic though a Taylor series and truncating this at the first derivative then iterating until the root is found with required accuracy. This is done using equation (4) and graphically represented in figure (5.1):



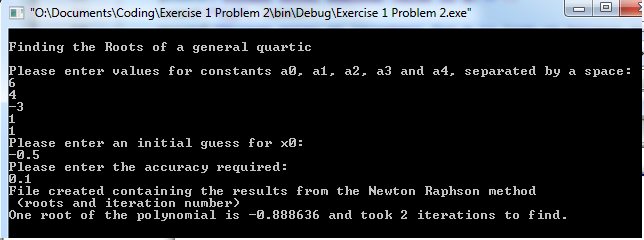
*[1] Figure 6 – Graphically representation of the Newton –Raphson method*

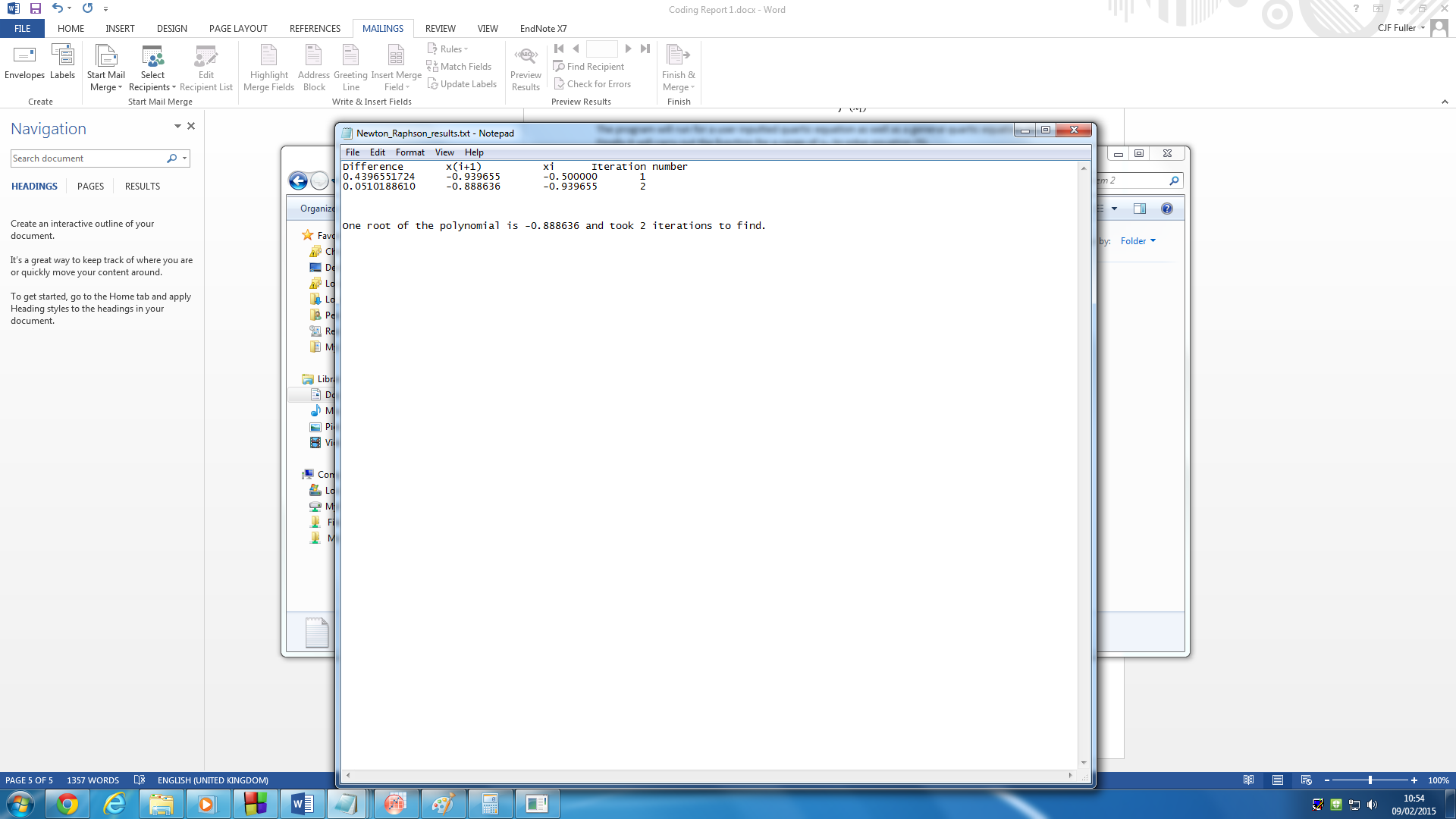
The program will run for a user inputted quartic equation as well as a general quartic equation (5). Finally it will carry out the function for a range of to solve equation (5).

The program was created to take user inputs for the co-efficients of each term as well as an initial guess for and the required accuracy, σ. A file is then created in which the iteration results will be written. A for loop with an if condition was used to iterate equation (4) until a maximum number of iterations (100) is reached incrementing each time by making= if the difference is above σ . The for loop computes the function and finds the difference between and each time checking to see with the required accuracy has been met. Here it is important to declare f(x) and f’(x) in separate functions and call them into the loop. This means the argument of f(x) and f’(x) will change correctly in each iterations, if this is not done the argument will remain as causing to function to not converge.

When looping over a range of to solve equation (5) the code was written to carry out the iterative equation (4) for each x using a double for loop with an if condition to check whether the required accuracy has been met (6 significant figures). The argument of x for the first for loop was assigned a different name to the x value in the second loop. These are both the same during the first iteration but when the second for loop iterates this ‘x’ value will change which causes the outer loop to not run correctly if the values are not assigned differently. The results were then written to a file from which the Newton-Raphson method can be analysed, showing how different values correspond to different roots.

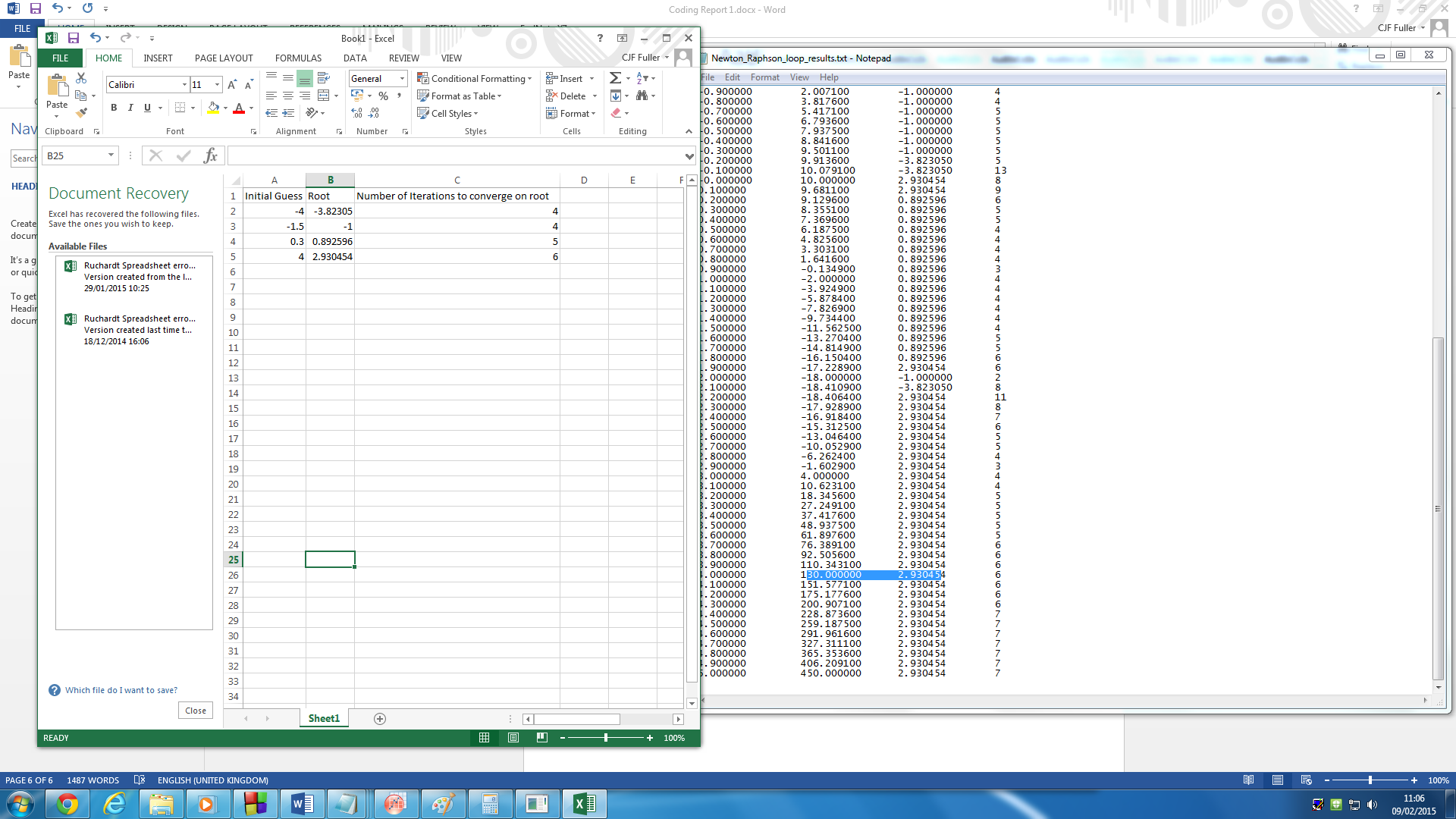
An example result from the user inputted quartic equation is shown in figures (6) and (7). It converges efficiently if an appropriate guess is made. However it was found that if the coefficient’s on the higher order x terms were large the program would often exceed the maximum number of iterations and fail to converge. This could be corrected by having the user input the maximum number of iterations to carry out so the program does not fail if large values are put in for the higher order x terms.





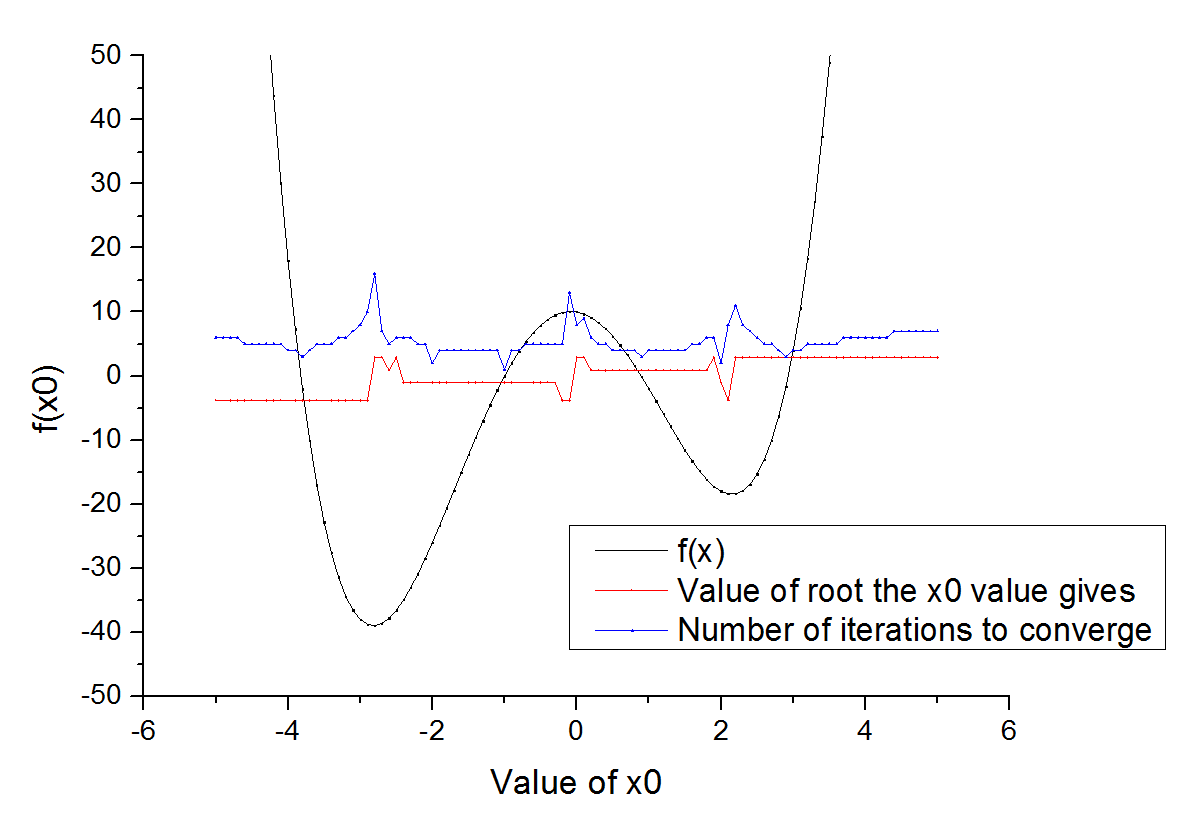
*Figures (6) and (7) – Output from the program for a user inputted quartic.*

The program can also be set-up to solve equation (5) shown in figure (8). Sensible initial guesses need to be used otherwise the program will fail to converge. Figure (9) shows the initial guess to achieve each root and the number of iterations needed to converge on this value.



*Figures (8) and (9) – The plot of equation (5) and results table.*

The code was then tested to find initial guesses that failed to converge on a root for equation (5). When equals a turning point on the graph, the program will fail to converge at all. This is due to the linear approximation of the first derivative not crossing the x-axis as f’(x)=0 and therefore the program is unable to give causing it to fail. By testing for different initial guesses it was found that the Newton-Raphson method maintained a great efficiency for large values of . For example converged in 25 iteration’s and when the function converged in just 49 iterations. Further testing may result in finding guesses that do not conserve, however the most likely guesses that won’t conserve will be guesses above the storage limit of found previously in the problem 1 discussion. For all other guesses it is expected to converge quickly and efficiently proving the power of the Newton-Raphson method.

Finally figure (10) shows graphically results for the loop over a range of values. It can be seen in general that the initial guess will likely converge on the nearest root, and the nearer the root the quicker the convergence. However when nears turning points of the function the Newton-Raphson method will converge on other roots. This is due to the inner workings of this method and how the truncation of the first derivative will place closer to a different root than the root near . Due to this significant jump between and the number of iterations need to converge on a root increases. A limitation of this program is the fact the user must define maximum and minimum values of x which would need to cover all the roots of the function. If this range was unknown then the program would not find all the roots. If an initial program was set-up to determine the range of roots within the function it would give an appropriate range of values to loop over.

*Figure 10 – Plot of x0 against f(x0) to show the roots and number of iterations needed to converge.*

References:

[1] http://astarmathsandphysics.com/a-level-maths-notes/FP1/a-level-maths-notes-fp1-the-newton-raphson-metod-of-finding-roots-of-equations.html