Newton Raphson Rootfinder Algorithm

1635585

November 29, 2017
Abstract

This report contains information and observations made while executing the Newton Raphson Rootfinder Algorithm. This algorithm is used to find roots of equations in single variables. The algorithm iterates from an initial guess value to reach the root of the equation. It makes use of the evaluation of function on the iterated values as well as the derivative of the function.

The report refers to the implementation of the algorithm using Fortran and Python. The routines for calculating the roots are written in Python and the functions and routines are called using scripting done in Python. The plots included in the report for solution and error convergence of the algorithm are generated using Matplotlib package in Python.
0.1 Introduction

The idea behind the Newton method is as follows: We start with an initial guess which we think is reasonably close to the true root. After this the function is approximated by it’s tangent line (which we will be computing using the tools of calculus). The x-intercept of this tangent line is then computed. The value of the x-intercept is usually a better approximation to the function’s root than the original guess. This method can be further iterated to reach the true solution.

The implementation of the algorithm is done using 2 programming languages:

- Fortran (Fortran 90)
- Python (Python 3.x)

The implementation is currently designed to find roots of the below two functions:

1. \( f = (x - 3) \times (x - 4) \)
2. \( f = (x - 1) \times (x - 2) \)

The report contains findings and observations after the executing the code for the function \( f = (x - 3) \times (x - 4) \). The true solutions of the function are values 3 and 4. The code is basically executed for three different threshold values of \( 10^{-4}, 10^{-6} \) and \( 10^{-8} \). It is also executed for two different initial guess values. One that is close to the true solution (5 in our implementation) and the other that is farther away from the true solution of the function (8 in our implementation).

0.2 Observations when initial guess is near to the true solution

The function \( f = (x - 3) \times (x - 4) \) was passed to the code, with an initial guess of 5. This value was chosen because it is near to one of the true solution of the function (i.e. root: 4). The code is able to correctly identify the root. It takes different number of iterations depending on the threshold values, but the calculated values at each iteration keep converging towards the real solution and do not go out of bounds, which might occur in the Newton method if the initial guess is not chosen carefully. The threshold values refer to the value of the difference between two the root values in consecutive iteration at which we stop further execution of the code. So that no unnecessary executions of the code are performed.

The plots and findings from the code execution are as below:

0.2.1 Threshold Value = \( 1.0 \times 10^{-4} \)

The code converges to the true solution in 5 iterations. In each iteration we calculate the value of the function and it’s derivative at the initial guess and
corresponding x-intercept of the tangent line calculated in each iteration. The plots for solution convergence and error convergence is shown below\(^1\).

0.2.2 Threshold Value = \(1.e^{-6}\)

The code converges to the true solution in 6 iterations. The plots for solution convergence and error convergence is shown below.

0.2.3 Threshold Value = \(1.e^{-8}\)

The code converges to the true solution in 6 iterations. This is the same number of iterations as we got with the threshold value as \(1.e^{-6}\). This is because in the sixth iteration the residual value we get is smaller than both \(1.e^{-6}\) as well as \(1.e^{-8}\). Upon further decreasing the threshold value we might observe more iterations to converge towards the solution. The plots for solution convergence and error convergence is shown below.

\(^1\)I tried having the images right below this point in code and I even used override [H!] for that. It worked initially, but I don’t know why it’s not working in the final execution. I think it is because there was too much images or text in the document.
Figure 2: Threshold $= 1.e-6$, Initial Value $= 5$

Figure 3: Threshold $= 1.e-8$, Initial Value $= 5$
0.3 Observations when initial guess is far from the true solution

The function \( f = (x - 3) \cdot (x - 4) \) was passed to the code, with an initial guess of 8. This value is far from the true solutions of the function. Even with a farther initial guess, the code is correctly able to identify the root of the function. It takes different number of iterations depending on the threshold values, but the calculated values at each iteration keep converging towards the real solution and do not go out of bounds. The threshold values refer to the value of the difference between two the root values in consecutive iteration at which we stop further execution of the code. So that no unnecessary executions of the code are performed.

The plots and findings from the code execution are as below:

0.3.1 Threshold Value = \( 1.e^{-4} \)

The code converges to the true solution in 7 iterations. The plots for solution convergence and error convergence is shown below.

![Convergence plots for threshold 1.e-4](image.png)

Figure 4: Threshold = \( 1.e^{-4} \), Initial Value = 8

0.3.2 Threshold Value = \( 1.e^{-6} \)

The code converges to the true solution in 7 iterations. The residual value after seventh iteration seems smaller than both \( 1.e^{-4} \) and \( 1.e^{-4} \). The plots for solution convergence and error convergence is shown below.
The code converges to the true solution in 8 iterations. The plots for solution convergence and error convergence is shown below.

0.3.3 Threshold Value = $1.e^{-8}$

The code converges to the true solution in 8 iterations. The plots for solution convergence and error convergence is shown below.

0.4 Findings and Observations

From above observations we can see that if the initial guess is farther from the true solution of the function then it takes more number of code iterations to reach the true solution.

Also it can be observed that in these cases the new iteration values are always better than the previous iteration values and initial guess. We did not observe cases where the new values are farther than the previous value in our execution of the code, but depending upon the function and the initial guess value that phenomenon also occurs. It does not occur in the cases when it is a linear system and the root multiplicity is 1.
Figure 6: Threshold = 1.e-8, Initial Value = 8