Assignment 2

Ayden S. McCann

March 2021

Code Listing at the end

Introduction
The purpose of this assignment is to test various aspects of root finding algorithms. These include the Bisection method, the secant method, and the Newton-Raphson method. These algorithms are first implemented in Python, on various equations and then compared to analyse their efficiency and accuracy given various parameters.

1 Question 1

1.1

By using the bisect_root function in 1.1py to find the solution of (1) over the range $[0, 1]$ to 6 significant figures the root is determined to be 0.609375. Substituting this result into (1) yields ~ 0.00398 which, given the significant figures used, validates this methods accuracy.

$$sin(x) + cos(2x)^3 = x \tag{1}$$

As seen below in figure 1, when comparing the observed numerical convergence it clearly follows the same convergence as given in the lecture slides (2), with the exception of being restricted to integer values.

$$\frac{log(b_0 - a_0)log(t)}{log(2)} \tag{2}$$

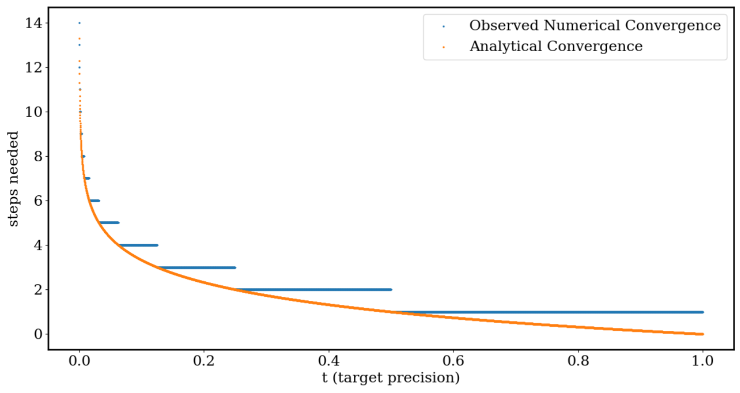

Figure 1: The bisect root function iterated from target precisions of 0.0001 to 1 to observe its convergence.

1.2

By using 1.2.py to evaluate the root of $f(x) = x$ over the interval $[-0.1, 0.1]$ the number of steps is 1. This is in fact expected. Given that from inspection that $f(0) = 0$. Since this root is equidistant from either side of the interval it is immediately picked up by the lines:

```
n += 1
c = (a + b) / 2
fc = f(c)
if fc == 0:
    return c, n
```

Since $a = -0.1$ and $b = 0.1$, $\frac{a+b}{2} = 0$ and without further effort the root has been located.

1.3

The bisect_root function cannot find the root of the function $f(x) = x^2$

This is due to a line in the original function stating:

```
if not (a < b and fa * fb < 0):
    raise ValueError('invalid input parameters or function')
```

Where fa and fb is the function evaluated at points a and b respectively. Given that $f(x)$ has no negative values for all x $fa * fb$ can never be < 0.

This can be circumvented however by simply commenting out this line which leads the function to the same procedure as in 1.2, where given the root is equidistant from either bound in the range $\frac{a+b}{2} = 0$ and the root is located on the first iteration.

2 Question 2

The code bisect_root was altered to use the Secant method (2.py)

The Secant method works in the following way:

for $n = 1, 2, 3...$:

1. Use secant line to get next guess (3)

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1}} \tag{3}$$

2. Is $|x_{n+1} - x_n| < c$? If yes, EXIT
3. Return to step 1.

By using the same range $[-0.1, 0.1]$, the two methods can then be compared. As seen in Figure 2 the Secant method converges faster than the Bisection method for in essence the entire range of target precision values. It is possible but not confirmed that this is merely due to the fact that the mathematical function dictating this methods convergence has been forced into integers, and the analytical values would be below those of (2) for the Bisection method for all t.

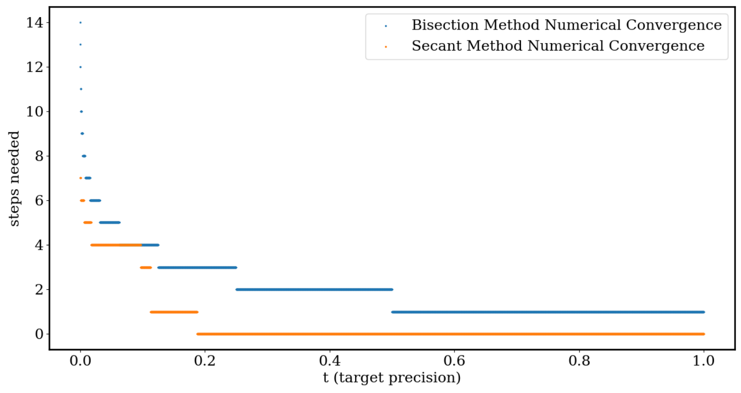

Figure 2: The bisect root function iterated from target precisions of 0.0001 to 1 to observe its convergence.

3 Question 3

3.1

As shown in Figure 3, it is difficult to quantitatively assess this methods convergence with comparison to the other two methods. However plotting its convergence against, for example, the analytical convergence of the bisection method, it can be more clearly seen that for values of target precision $\sim$

$0.05 - 0.5$ the Newton-Raphson method appears to require a lower number of steps to achieve the same precision.

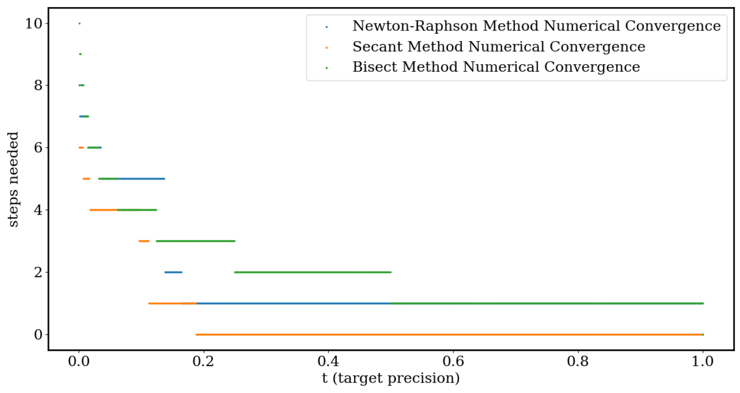

Figure 3: The three discussed root finding methods iterated from target precisions of 0.0001 to 1 to observe its convergence.

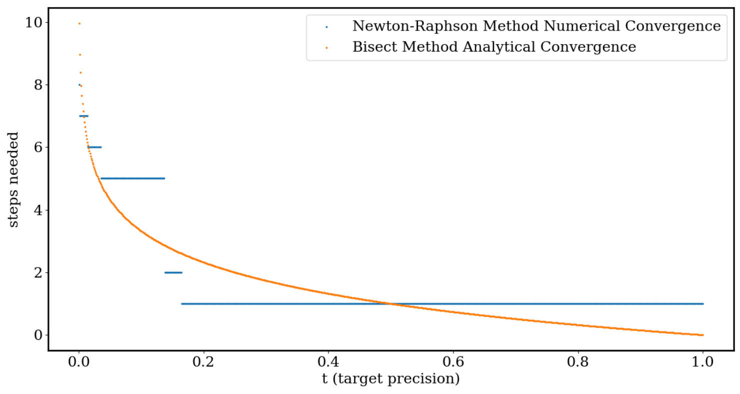

Figure 4: The analytical convergence of the bisection method (2) plotted with the numerical convergence of the Newton-Raphson method. Iterated from target precisions of 0.0001 to 1 to observe its convergence.

3.2

The Newton-Raphson method was utilised to find the first 100 roots of $J_(0)$. As seen in Figure 6 the accuracy has increased when compared to the inital approximate roots.

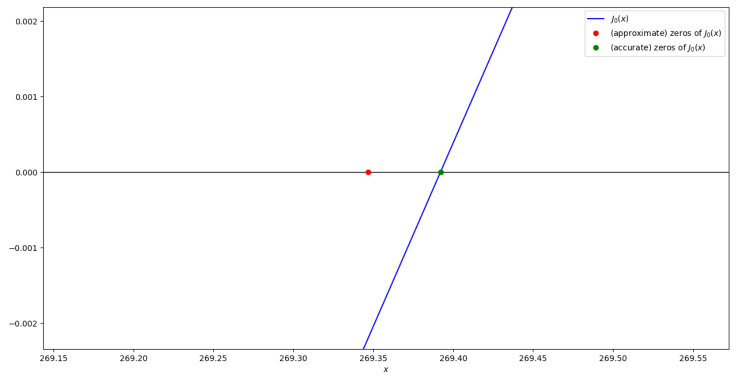

Figure 5: An example of the increased accuracy of the roots obtained using the Newton-Raphson method

After rounding the roots to 6 significant figures as requested the average error as calculated by (4) is 0.00017782990673883336 when compared to the scipy jn_zero function.

Discussion

As demonstrated in this assignment there are various methods for numerically obtaining roots to functions. As seen in Figures 2, 3 and 4 for the most part they all require a differing number of iterations to reach a certain level of precision as they converge differently.

4 Code Listing

Question 1.1 - 1.1.py
Question 1.2 - 1.2.py
Question 1.3 - 1.3.py
Question 2 - 2.py
Question 3.1 - 3.1.py
Question 3.2 - 3.2.py