

Task 1

I have created a code that will search for the root of the given function

$$f(x) = xe^{3x^2} - 7x \quad (1)$$

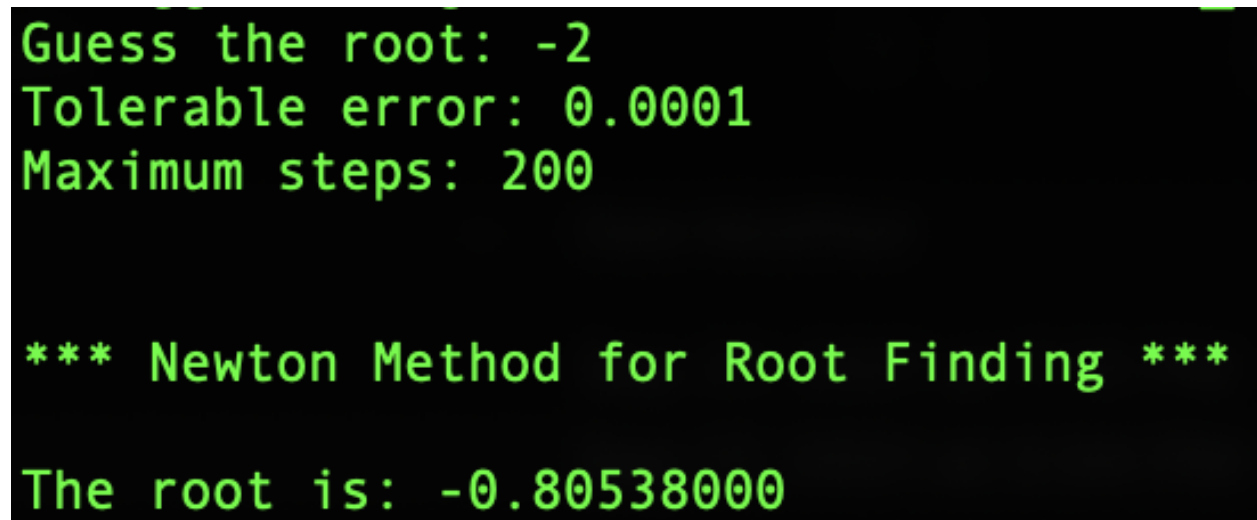
using Newton's method. I have also added this code to my software manual with the example results.

```

1 import numpy as np
2
3 def f(x):
4     return x * np.exp(3*np.power(x, 2)) - 7*x
5
6 def df(x):
7     return np.exp(3*np.power(x, 2)) + 6 * np.exp(3*np.power(x, 2)) * np.power(x, 2) - 7
8
9 def newton_Method(x0, e, N):
10    print('\n\n*** Newton Method for Root Finding ***')
11    step = 1
12    flag = 1
13    condition = True
14    while condition:
15        x1 = x0 - f(x0)/df(x0)
16        x0 = x1 # update the the old x1 to be the new x0
17        step = step + 1
18        if step > N:
19            flag = 0
20            break
21        condition = np.abs(f(x1)) > e
22    if flag == 1:
23        print('\nThe root is: %0.8f' % x1)
24    else:
25        print('\nCannot find a root.')
26
27 x0 = float(input('Guess the root: '))
28 e = float(input('Tolerable error: '))
29 N = int(input('Maximum steps: '))
30
31 newton_Method(x0, e, N)

```

This is the result I have with initial guess of -2 , tolerable error of 0.0001 , and maximum steps of 200 using function (1):



```

Guess the root: -2
Tolerable error: 0.0001
Maximum steps: 200

*** Newton Method for Root Finding ***

The root is: -0.80538000

```

Figure 1. Newton's Method of Root Finding Result.

Task 2

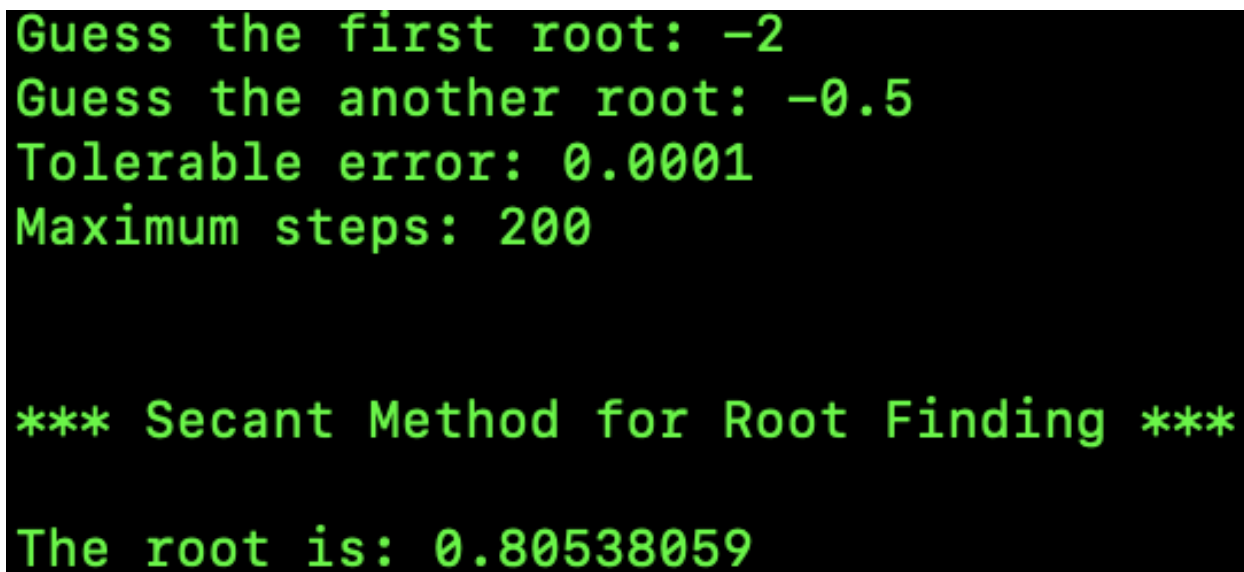
I have repeated Task 1 for the secant method. The code is provided below.

```

1 import numpy as np
2
3 def f(x):
4     return x * np.exp(3*np.power(x, 2)) - 7*x
5
6 def secant_Method(x0, x1, e, N):
7     print('\n\n*** Secant Method for Root Finding ***')
8     step = 1
9     flag = 1
10    condition = True
11    while condition:
12        x2 = x0 - (x1-x0)*f(x0)/(f(x1) - f(x0))
13        x0 = x1 # update the the old x1 to be the new x0
14        x1 = x2 # update the the old x2 to be the new x1
15        step = step + 1
16        if step > N:
17            flag = 0
18            break
19        condition = np.abs(f(x1)) > e
20    if flag == 1:
21        print('\nThe root is: %0.8f' % x1)
22    else:
23        print('\nCannot find a root.')
24
25 x0 = float(input('Guess the first root: '))
26 x1 = float(input('Guess the another root: '))
27 # because of the approximation of the derivative, we need a second guess
28 e = float(input('Tolerable error: '))
29 N = int(input('Maximum steps: '))
30
31 secant_Method(x0, x1, e, N)

```

This is the result I have with initial guess of -2 , second guess of -0.5 , tolerable error of 0.001 , and maximum steps of 200 using function (1):



```

Guess the first root: -2
Guess the another root: -0.5
Tolerable error: 0.0001
Maximum steps: 200

*** Secant Method for Root Finding ***

The root is: 0.80538059

```

Figure 2. Secant Method of Root Finding Result.

Task 3

As we discussed, the Newton's method is given by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

In order to do a computational convergence analysis, we need to assume the real root of the function to be x_r . Let error be defined as the difference between each iteration and the real root. Then the error analysis would be

$$\begin{aligned} x_{k+1} - x_r &= x_k - \frac{f(x_k)}{f'(x_k)} - x_r \\ e_{k+1} &= e_k - \frac{f(x_k)}{f'(x_k)} \\ &= \frac{e_k f'(x_k) - f(x_k)}{f'(x_k)} \end{aligned}$$

Now consider the Taylor Series expansion for the error, that is

$$\begin{aligned} 0 &= f(x_r) = f(x_k - e_k) = f(x_k) + f'(x_k)(x_k - x_r) + \frac{1}{2}f''(x_k)(x_k - x_r)^2 + \dots \\ &= f(x_k) + f'(x_k)(e_k) + \underbrace{\frac{1}{2}f''(\xi)e_k^2}_{\text{negligible}} + \dots \\ \implies -f(x_k) - e_k f'(x_k) &= \frac{1}{2}f''(\xi)e_k^2 \end{aligned}$$

Substitute this back to the previous error analysis result of e_{k+1} , we have

$$\begin{aligned} |e_{k+1}| &= \left| \frac{\frac{1}{2}f''(\xi)e_k^2}{f'(x_k)} \right| \\ &= \underbrace{\left| \frac{\frac{1}{2}f''(\xi)}{f'(x_k)} \right|}_C |e_k^2| \end{aligned}$$

Therefore, the prior result reduces down to $|e_{k+1}| = C|e_k|^2$, hence showing us Newton's method is quadratically convergent.

Using the example from Tasksheet 4, that is

$$f(x) = xe^{3x^2} - 7x$$

we have known at this point that one of the real root is -0.8053 , we can make a guess of -1 , which gives an error of

$$e_k = |-0.194699999| \approx 0.194699999$$

Using the initial guess, x_{k+1} will be

$$e_{k+1} = \left| \left(-1 - \frac{f(-1)}{f'(-1)} \right) - 0.8053 \right| \approx 0.096753454$$

And we can see that

$$e_{k+1} = Ce_k^2 = 0.194699999^2 \approx 0.096753454$$

for $C \approx 3$. Hence, it is suffice to say that Newton's method satisfies quadratic convergence.

Task 4

As we discussed, the Secant's method is given by

$$x_{k+1} = x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

In order to do a computational convergence analysis, we need to assume the real root of the function to be x_r . Let error be defined as the difference between each iteration and the real root. Then the error analysis would be

$$\begin{aligned} x_{k+1} - x_r &= x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} - x_r \\ e_{k+1} &= e_k - f(x_r + e_k) \frac{e_k - e_{k-1}}{f(x_k + e_k) - f(x_r + x_{k-1})} \\ &= e_k - \frac{f(x_r + e_k)(e_k - e_{k-1})}{f(x_k + e_k) - f(x_r + x_{k-1})} \end{aligned}$$

We can consider the Taylor Series expansion again for $f(x_r + e_k)$, since $f(x_r) = 0$, we have,

$$\begin{aligned} f(x_r + e_k) &\approx f'(x_r)e_k + \frac{f''(x_r)}{2}e_k^2 \\ &= e_k f'(x_r) \left(1 + \frac{f''(x_r)}{2f'(x_r)}e_k\right) \\ &= e_k f'(x_r) (1 + Me_k) \end{aligned}$$

where $M = \frac{f''(x_r)}{2f'(x_r)}$ for simplicity sake. Now, we can substitute it back to the error analysis equation, and get

$$\begin{aligned} e_{k+1} &\approx e_k - \frac{e_k f'(x_r) (1 + Me_k) (e_k - e_{k-1})}{f'(x_r) (e_k - e_{k-1}) (1 + M(e_k + e_{k-1}))} \\ &= e_k - \frac{e_k (1 + Me_k)}{1 + M(e_k + e_{k-1})} \\ &= \frac{e_{k-1} e_k M}{1 + M(e_k + e_{k-1})} \\ &\approx e_{k-1} e_k M = \frac{f''(x_r)}{2f'(x_r)} e_{k-1} e_k \end{aligned}$$

which we can see not linear but not quadratically either as the Newton's Method. Let assume:

$$|e_{k-1}| \approx C |e_k|^p$$

Then,

$$\begin{aligned} C |x_k|^p &\approx |M| |e_{k-1}| |e_k| \\ C |x_k|^{p-1} &\approx |M| |e_{k-1}| \\ |x_k|^{p-1} &\approx \frac{|M|}{C} |e_{k-1}| \\ |x_k| &\approx \left(\frac{|M|}{C}\right)^{\frac{1}{p-1}} |e_{k-1}|^{\frac{1}{p-1}} \end{aligned}$$

Since $p = \frac{1}{p-1}$, exactly how the Golden ratio is defined, that is

$$p = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

Now, we can conclude that, for the secant method,

$$|e_{k+1}| \approx \left| \frac{f''(x_r)}{2f'(x_r)} \right| \frac{\sqrt{5}-1}{2} |e_k| \frac{\sqrt{5}-1}{2}$$

Checking out conclusion against the example from Tasksheet 4, it is true.

Task 5

The code I created for the Hybrid method is provided below.

```

1 import numpy as np
2
3 def f(x):
4     return x * np.exp(3*np.power(x, 2)) - 7*x
5
6 def df(x):
7     return np.exp(3*np.power(x, 2)) + 6 * np.exp(3*np.power(x, 2)) * np.power(x, 2) - 7
8
9 def hybrid_Method(a, b, x0, e, N):
10     print('\n\n*** Hybrid Method for Root Finding ***')
11     step = 1
12     flag = 1
13     condition = True
14     while condition:
15         x1 = x0 - f(x0) / df(x0)
16         x0 = x1 # update the the old x1 to be the new x0
17         step = step + 1
18         e1 = np.abs(x1 - x0) # the error from the Newton method
19         while e1 > e: # the condition needed to start the bisection loop
20             a_n = a
21             b_n = b
22             x0 = m_n # print("The OG guessed root has become", m_n)
23             e2 = np.abs(-f(m_n) / df(m_n))
24             if e2 < e: # check the error to get back to Newton's method
25                 break
26             # recall from the lecture a good rule of thumb for n (bisection) is 4
27             for n in range(0, 3):
28                 m_n = (a_n + b_n) / 2
29                 if f(a_n) * f(m_n) < 0:
30                     a_n = a_n
31                     b_n = m_n
32                 elif f(b_n) * f(m_n) < 0:
33                     a_n = m_n
34                     b_n = b_n
35                 elif f(m_n) == 0:
36                     print("Found exact solution using Bisection steps:", f(m_n))
37             if step > N:
38                 flag = 0
39                 break
40             condition = np.abs(f(x1)) > e
41         if flag == 1:
42             print('\nFound solution: %0.8f' % x1)
43         else:
44             print("We don't have a root in this interval")
45
46 a = float(input('Guess the left end point: '))
47 b = float(input('Guess the right end point: '))
48 x0 = float(input('Guess the root: '))
49 e = float(input('Tolerable error: '))
50 N = int(input('Maximum steps: '))
51
52 hybrid_Method(a, b, x0, e, N)

```

This is the result I have with the interval $[-10, -0.5]$, initial guess of -5 , tolerable error of 0.0008 , and maximum steps of 80 :

```
Guess the left end point: -10
Guess the right end point: -0.5
Guess the root: -5
Tolerable error: 0.0008
Maximum steps: 80

*** Hybird Method for Root Finding ***

Found solution: -0.80538312
```

Figure 3. Hybrid Method of Root Finding Result.

Task 6

From this journal article¹ I found, the Bisection method seems to be the slowest of the three. With the Newton and Secant methods converging to the exact root with 0.000000 error at the 8^{th} and 6^{th} iteration respectively, the Bisection method only converges at the 52 second iteration.

Even though Secant method is in general slower than the Newton's method, a factor we need to consider is the cost of convergence. That is, the Secant method only requires one function per iteration while as the Newton's method requires both the function and its derivative.

¹<https://www.researchgate.net/publication/314387274-Comparative-Study-of-Bisection-Newton-Raphson-and-Secant-Methods-of-Root-Finding-Problems>