

Task 1

I have created the following testing python file for this test. And the python module folder can be found here.

```
1 from Newton import newton_Method
2 from Secant import secant_Method
3 from FixedPoint import fixedPointIteration
4 from Bisection import bisection
5
6 fixedPointIteration(0.3, 0.0005, 200)
7
8 bisection(0.3, 0.7, 200)
9
10 newton_Method(0.5, 0.0005, 200)
11
12 secant_Method(0.3, 0.7, 0.0005, 200)
```

These are the results I have found using the four different methods that are closed to zero. I have included two results, since when I ran the code from terminal, the fixed point method didn't converge while as running from the IDE shows it does converge.

```

*** FIXED POINT ITERATION ***
Not Convergent.

*** Bisection Method ***
('Found exact solution:', 0.0)

*** Newton Method for Root Finding ***
The root is: 0.48362868

*** Secant Method for Root Finding ***
The root is: 0.48360956

```

```

*** FIXED POINT ITERATION ***
The root is: -0.48355071

*** Bisection Method ***
Found exact solution: 0.0

*** Newton Method for Root Finding ***
|
The root is: 0.48362868

*** Secant Method for Root Finding ***
The root is: 0.48360956

Process finished with exit code 0

```

Figure 1. Running the Test Code from the Terminal vs. Running from the IDE.

Task 2

Using my Newton method with $x_0 = -5.0$ and $x_0 = 6$, given the following code:

```
1 import numpy as np
2
3 def f(x):
4     return np.exp(-np.power(x, 2)) * np.sin(4 * np.power(x,2) - 1) + 0.051
5
6 def df(x):
7     return 8 * np.exp(-np.power(x, 2)) * x * np.cos(4 * np.power(x,2) - 1) - 2 * np.exp(-np.
8         power(x, 2)) * x * np.sin(4 * np.power(x,2) - 1)
9
10 def newton_Method(x0, e, N):
11     print('\n\n*** Newton Method for Root Finding ***')
12     step = 1
13     flag = 1
14     condition = True
15     while condition:
16         x1 = x0 - f(x0)/df(x0)
17         x0 = x1 # update the the old x1 to be the new x0
18         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 newton_Method(6, 0.005, 200)

```

We have the following two results, for $x_0 = -5.0$ and $x_0 = 6$, respectively.

```

Task_2.py:15: RuntimeWarning: divide by zero encountered in double_scalars
  x1 = x0 - f(x0)/df(x0)

The root is: -inf

Task_2.py:15: RuntimeWarning: divide by zero encountered in double_scalars
  x1 = x0 - f(x0)/df(x0)

The root is: -inf

```

Figure 2. The Results Using Newton Method With $x_0 = -5.0$ and $x_0 = 6$, Respectively.

Comparing to Task 1, the problem comes with the large initial guess.

Task 3

Using my updated hybrid method with the initial interval $[-5, 6]$,

```

1 import numpy as np
2
3 def f(x):
4     return np.exp(-np.power(x, 2)) * np.sin(4 * np.power(x,2) - 1) + 0.051
5
6 def df(x):
7     return 8 * np.exp(-np.power(x, 2)) * x * np.cos(4 * np.power(x,2) - 1) - 2 * np.exp(-np.
8         power(x, 2)) * x * np.sin(4 * np.power(x,2) - 1)
9
10 def hybrid_Method(a, b, x0, e, N):
11     print('\n\n*** Hybrid Method for Root Finding ***')
12     step = 1
13     flag = 1
14     condition = True
15     while condition:
16         x1 = x0 - f(x0) / df(x0)
17         x0 = x1 # update the the old x1 to be the new x0
18         step = step + 1
19         e1 = np.abs(x1 - x0) # the error from the Newton method
20         while e1 > e: # the condition needed to start the bisection loop
21             a_n = a
22             b_n = b
23             x0 = m_n # print("The OG guessed root has become", m_n)
24             e2 = np.abs(-f(m_n) / df(m_n))
25             if e2 < e: # check the error to get back to Newton's method
26                 break
27             # recall from the lecture a good rule of thumb for n (bisection) is 4
28             for n in range(0, 3):
29                 m_n = (a_n + b_n) / 2
30                 if f(a_n) * f(m_n) < 0:
31                     a_n = a_n
32                     b_n = m_n
33                 elif f(b_n) * f(m_n) < 0:
34                     a_n = m_n
35                     b_n = b_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 hybrid_Method(-5, 6, 0.3, 0.0005, 200)

```

I get the following result

```

[xianggao@Xiangs-MacBook-Pro Tasksheet_06 % python Task_3.py

*** Hybird Method for Root Finding ***

Found solution: 0.48361081

```

Figure 3. The Results Using Hybird Method with the Initial Interval $[-5, 6]$.

Task 4

The following code is the hybrid method using the secant method, however, I wasn't able to fix the code in time to before the due time.

```

1 import numpy as np
2
3 def f(x):
4     return np.exp(-np.power(x, 2)) * np.sin(4 * np.power(x, 2) - 1) + 0.051
5
6 def hybrid_Method(a, b, x0, x1, e, N):
7     print('\n\n*** Hybird 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        e1 = np.abs(x1 - x0) # the error from the Newton method
17        while e1 > e: # the condition needed to start the bisection loop
18            a_n = a
19            b_n = b
20            x0 = m_n # print("The OG guessed root has become", m_n)
21            e2 = np.abs(x0 - (x1 - x0) * f(x0) / (f(x1) - f(x0)))
22            if e2 < e: # check the error to get back to Newton's method
23                break
24            # recall from the lecture a good rule of thumb for n (bisection) is 4
25            for n in range(0, 3):
26                m_n = (a_n + b_n) / 2
27                if f(a_n) * f(m_n) < 0:
28                    a_n = a_n
29                    b_n = m_n
30                elif f(b_n) * f(m_n) < 0:
31                    a_n = m_n
32                    b_n = b_n
33                elif f(m_n) == 0:

```

```
34         print("Found exact solution using Bisection steps:", f(m_n))
35     if step > N:
36         flag = 0
37         break
38     condition = np.abs(f(x1)) > e
39     if flag == 1:
40         print('\nFound solution: %0.8f' % x1)
41     else:
42         print("We don't have a root in this interval")
43
44 hybrid_Method(-5, 6, 0.3, 0.6, 0.0005, 200)
```

Task 6

From this website¹ I found, when it comes to finding multiple roots of a function. It's really hard to come up with a structural algorithmic approach.

However, with the information from this website². The Broyden's method is apparently one approach for function with multiple roots.

¹<http://jwilson.coe.uga.edu/EMT669/Student.Folders/Frietag.Mark/Homepage/roots/roots.html>

²<https://www.ams.org/journals/mcom/1965-19-092/S0025-5718-1965-0198670-6/>