Task 1

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

```
from Newton import newton_Method
from Secant import secant_Method
from FixedPoint import fixedPointIteration
from Bisection import bisection

fixedPointIteration(0.3, 0.0005, 200)

bisection(0.3, 0.7, 200)

newton_Method(0.5, 0.0005, 200)

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.

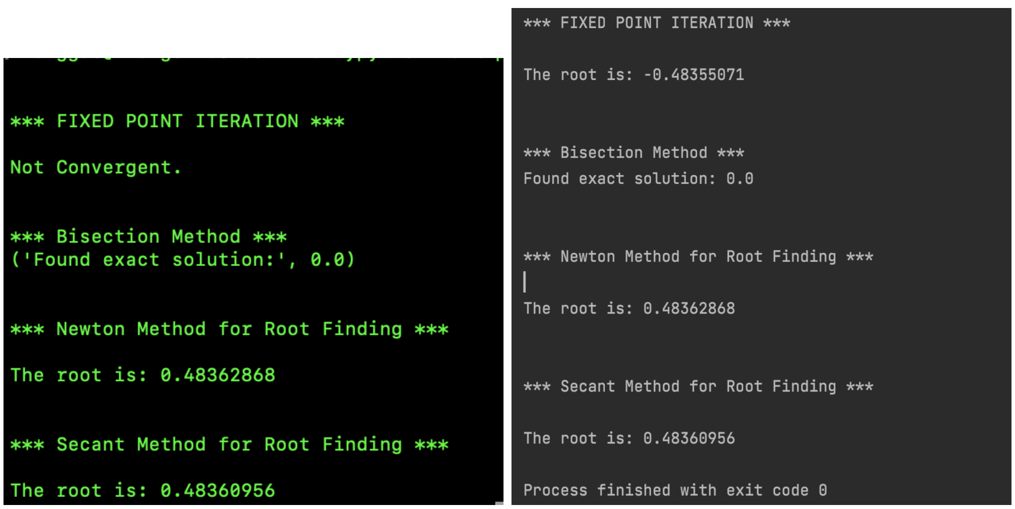

Figure 1. Running the Test Code from the Terminal vs. Running from the IDE.

Task 2

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

```
import numpy as np

def f(x):
    return np.exp(-np.power(x, 2)) * np.sin(4 * np.power(x,2) - 1) + 0.051

def df(x):
    return 8 * np.exp(-np.power(x, 2)) * x * np.cos(4 * np.power(x,2) - 1) - 2 * np.exp(-np.
    power(x, 2)) * x * np.sin(4 * np.power(x,2) - 1)

def newton_Method(x0, e, N):
    print('\n\n*** Newton Method for Root Finding ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        x1 = x0 - f(x0)/df(x0)
        x0 = x1 # update the the old x1 to be the new x0
        step = step + 1
```

```python
        if step > N:
            flag = 0
            break
        condition = np.abs(f(x1)) > e
    if flag == 1:
        print('\nThe root is: %0.8f' % x1)
    else:
        print('\nCannot find a root.')

newton_Method(6, 0.005, 200)
```

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

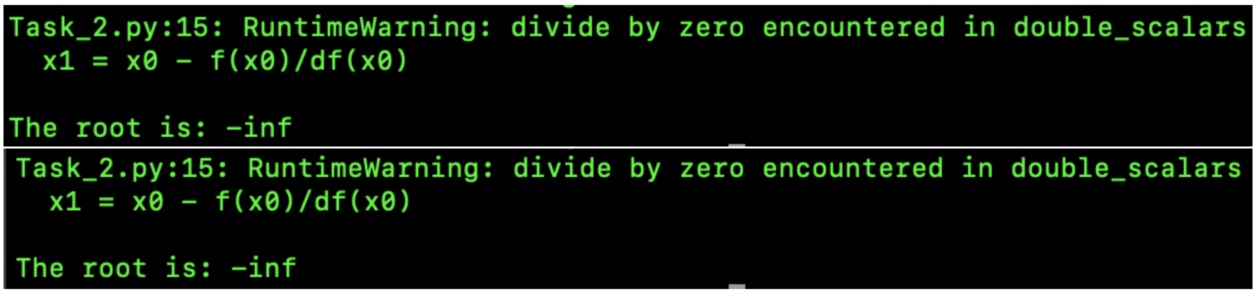

Figure 2. The Results Using Newton Method With $x_0 = -5.0$ and $x_0 = 6$, Respectively.

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

Task 3

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

```python
import numpy as np

def f(x):
    return np.exp(-np.power(x, 2)) * np.sin(4 * np.power(x,2) - 1) + 0.051

def df(x):
    return 8 * np.exp(-np.power(x, 2)) * x * np.cos(4 * np.power(x,2) - 1) - 2 * np.exp(-np.power(x, 2)) * x * np.sin(4 * np.power(x,2) - 1)

def hybrid_Method(a, b, x0, e, N):
    print('\n\n*** Hybird Method for Root Finding ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        x1 = x0 - f(x0) / df(x0)
        x0 = x1  # update the the old x1 to be the new x0
        step = step + 1
        e1 = np.abs(x1 - x0)  # the error from the Newton method
        while e1 > e:  # the condition needed to start the bisection loop
            a_n = a
            b_n = b
            x0 = m_n  # print("The OG guessed root has become", m_n)
            e2 = np.abs(-f(m_n) / df(m_n))
            if e2 < e:  # check the error to get back to Newton's method
                break
            # recall from the lecture a good rule of thumb for n (bisection) is 4
            for n in range(0, 3):
                m_n = (a_n + b_n) / 2
                if f(a_n) * f(m_n) < 0:
                    a_n = a_n
                    b_n = m_n
                elif f(b_n) * f(m_n) < 0:
                    a_n = m_n
```

```
                    b_n = b_n
                elif f(m_n) == 0:
                    print("Found exact solution using Bisection steps:", f(m_n))
        if step > N:
            flag = 0
            break
        condition = np.abs(f(x1)) > e
    if flag == 1:
        print('\nFound solution: %0.8f' % x1)
    else:
        print("We don't have a root in this interval")

hybrid_Method(-5, 6, 0.3, 0.0005, 200)
```

I get the following result

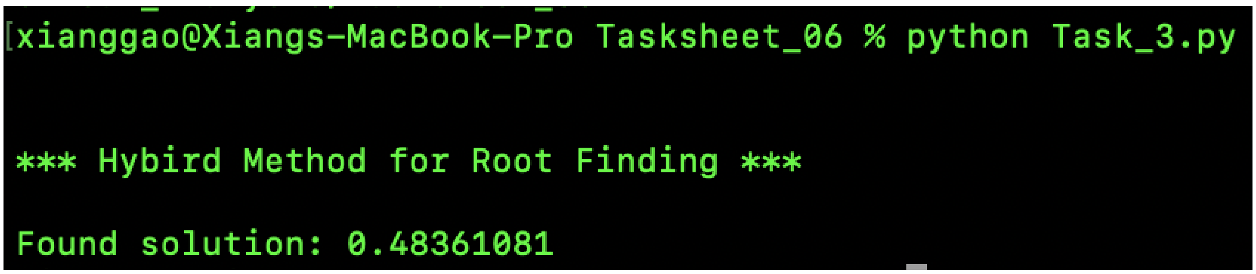

Figure 3. The Results Using Hybird Method with the Initial Interval $[-5, 6]$.

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.

```
import numpy as np

def f(x):
    return np.exp(-np.power(x, 2)) * np.sin(4 * np.power(x,2) - 1) + 0.051

def hybrid_Method(a, b, x0, x1, e, N):
    print('\n\n*** Hybird Method for Root Finding ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        x2 = x0 - (x1 - x0) * f(x0) / (f(x1) - f(x0))
        x0 = x1  # update the the old x1 to be the new x0
        x1 = x2  # update the the old x2 to be the new x1
        step = step + 1
        e1 = np.abs(x1 - x0)  # the error from the Newton method
        while e1 > e:  # the condition needed to start the bisection loop
            a_n = a
            b_n = b
            x0 = m_n  # print("The OG guessed root has become", m_n)
            e2 = np.abs(x0 - (x1 - x0) * f(x0) / (f(x1) - f(x0)))
            if e2 < e:  # check the error to get back to Newton's method
                break
            # recall from the lecture a good rule of thumb for n (bisection) is 4
            for n in range(0, 3):
                m_n = (a_n + b_n) / 2
                if f(a_n) * f(m_n) < 0:
                    a_n = a_n
                    b_n = m_n
                elif f(b_n) * f(m_n) < 0:
                    a_n = m_n
                    b_n = b_n
                elif f(m_n) == 0:
```

```
                    print("Found exact solution using Bisection steps:", f(m_n))
        if step > N:
            flag = 0
            break
        condition = np.abs(f(x1)) > e
    if flag == 1:
        print('\nFound solution: %0.8f' % x1)
    else:
        print("We don't have a root in this interval")

hybrid_Method(-5, 6, 0.3, 0.6, 0.0005, 200)
```

Task 6

From this website[1] 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 website[2]. 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/