Palmer Edholm
Dr. Joe Koebbe
MATH 4610
October 22, 2021

Tasksheet 5

1. The code for Newton's method is as follows

```
import numpy as np

def newton(x0, f, df, tol, max_iter):
    # Initialize needed variables
    f0 = f(x0)
    df0 = df(x0)
    error = 10*tol
    icnt = 0
    # Create loop to perform newton's method
    while error > tol and icnt < max_iter:
        x1 = x0 - f0/df0
        error = abs(x1-x0)
        icnt += 1
        x0 = x1
        f0 = f(x0)
        df0 = df(x0)
    return x1
```

When using it to approximate the roots of $xe^{3x^2} - 7x$ with a tolerance of 0.0001 and an initial guess of 1 it gives me the value 0.8053798692388104 after 5 iterations. I will add the code for the Newton's method, the Secant method, and the hybrid method to a Python package at the end.

2. The code for the Secant method is as follows

```
import numpy as np

def secant(x0, x1, f, tol, max_iter):
    # Initialize needed variables
    f0 = f(x0)
    f1 = f(x1)
    error = 10*tol
    icnt = 0
    # Create loop to perform secant method
    while error > tol and icnt < max_iter:
        x2 = x1 - f1*(x1 - x0)/(f1 - f0)
        error = abs(x2-x1)
        icnt += 1
        x0 = x1
        x1 = x2
```

```
        f0 = f(x0)
        f1 = f(x1)
    return x2
```

When using it to approximate the roots of $xe^{3x^2} - 7x$ with a tolerance of 0.0001 and the interval $[0.5, 1]$ it gives me the value 0.8053798245521222 after 14 iterations.

3. For a computational convergence analysis, I'll perform a simple log-log linear regression where I regress the error of the approximation on the successive approximations. The following is the code I used to perform the regression on Newton's method

```
from scipy import stats
import numpy as np
from newton import newton

approx = []
for i in range(1, 7):
    approx.append(newton(1, lambda x: x * np.e ** (3 * x ** 2) -
            7 * x, lambda x: np.e ** (3 * x ** 2) + 6 * x ** 2 *
            np.e ** (3 * x ** 2) - 7, 0.00001, i))

errors = []
for i in range(len(approx)-1):
    errors.append(approx[i]-approx[len(approx)-1])

lin = [[],[]]

for i in range(len(errors)-1):
    lin[0].append(np.log(errors[i]))
    lin[1].append(np.log(errors[i+1]))

print(stats.linregress(lin[0], lin[1]).slope)
```

The code tells us that the slope of the simple regression is 1.9680537609138742. We can see that the rate of convergence is approaching 2.

4. The code for the regression that I did for the secant method is as follows

```
from scipy import stats
import numpy as np
from secant import secant

approx = []
for i in range(1, 17):
    approx.append(secant(0.5, 1, lambda x: x*np.e**(3*x**2)-7*x,
        0.00000001, i))

errors = []
for i in range(len(approx)-1):
```

```
    errors.append(abs(approx[i]-approx[len(approx)-1]))

lin = [[],[]]

for i in range(len(errors)-1):
    lin[0].append(np.log(errors[i]))
    lin[1].append(np.log(errors[i+1]))

print(stats.linregress(lin[0], lin[1]).slope)
```

The slope of this simple regression is 1.6056871745337682 which is very close to 1.65.

5. The code for the hybrid Newton-bisection method is as follows:

```
import numpy as np

def hybrid(a, b, x0, f, df, tol, max_iter):
    fa = f(a)
    fb = f(b)
    if fa * fb >0:
        raise Exception('Root not in interval')
    if x0 < a or x0 > b:
        raise Exception('Invalid initial guess')
    f0 = f(x0)
    df0 = df(x0)
    error = 10*tol
    icnt=0
    while error > tol and icnt < max_iter:
        x1 = x0-f0/df0
        errnewt = abs(x1-x0)
        if errnewt > error:
            for i in range(4):
                c = 0.5*(a+b)
                fc = f(c)
                if fa*fc<0:
                    b=c
                else:
                    a=c
        error = abs(x1-x0)
        x0 = x1
        f0 = f(x0)
        df0 = df(x0)
        icnt += 1
    return x1
```

The result we get for the same function we've been using with the interval $[0.5, 1]$, initial guess 0.9, and a tolerance level of 0.0001 is 0.8053798665551777 after 4 iterations. Now that I have all my code, I'll go ahead and build my Python package. As seen in figure 1, I now have my tar file and my whl file.

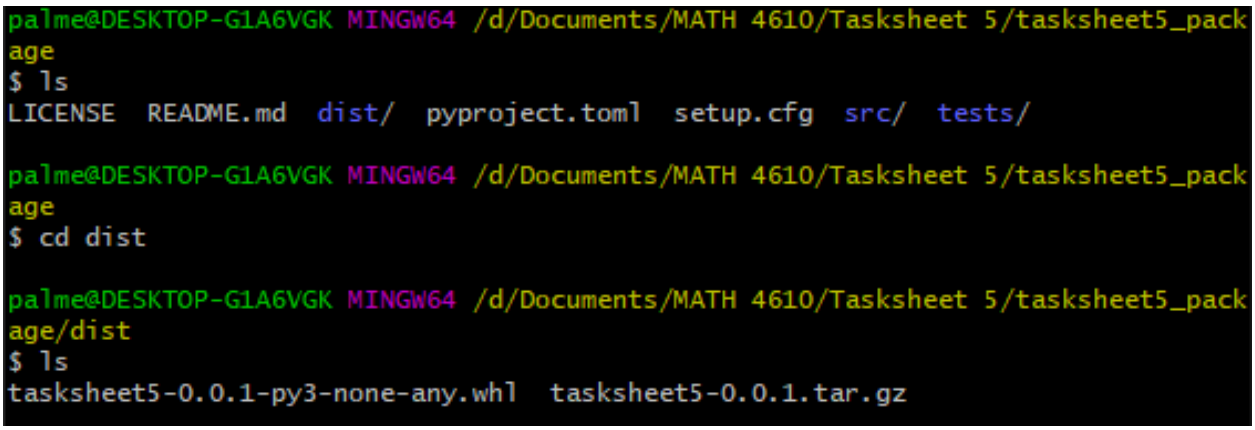

Figure 1: Python package

6. In [1], I found a nice encapsulation of the three methods. Newton's method requires the most in terms of inputs. Newton's method is also the fastest with a rate of convergence of 2. Rate of convergence of the secant method and bisection is approximately 1.62 (super linear) and 1 (linear) respectively. Bisection has the least amount of conditions for convergence, there just needs to be one root within the interval passed in. The secant method has the most restrictions to convergence as both x_0 and x_1 need to be close enough to the root x^*.

References

[1] https://arnold.hosted.uark.edu/NA/Pages/BisectSecantNewton.pdf