Name: Andy Skabelund
Task 1: The link to my software manuals can be found at the following address:
https://github.com/JetzNation/math4610/tree/main/Software%20Manuals
My Newton's method code is as follows:

```
import numpy as np


def f_of_x(x):
    return x * np.exp(-x)


def d_of_x(x):
    return -1 * (x - 1) * np.exp(-x)

def newtonsMethod(func, fderiv, x, maxIter):
    iteration = 0

    while (iteration < maxIter):
        i = x - (f_of_x(x) / d_of_x(x))
        x = i
        iteration += 1
    print(f"The root is found at {x} after {iteration} iterations")

newtonsMethod(f_of_x, d_of_x, .5, 10)
```

Output:

```
The root is found at 0.0 after 10 iterations
```

Task 2:
My Secant method code is as follows:

```
import numpy as np


def f_of_x(x):
    return x * np.exp(-x)

def secantMethod(func, x0, x1, maxIter):
    iteration = 0
    tol = .00000001
    error = 10 * tol

    while (iteration < maxIter and error > tol):
        fx0 = f_of_x(x0)
        fx1 = f_of_x(x1)
        xi = x1 - (fx1 * (x1 - x0)) / (fx1 - fx0)
```

```
        error = np.abs(xi - x1)
        x0 = x1
        x1 = xi
        fx0 = fx1
        iteration += 1
    print(f"The root is found at {x1} after {iteration} iterations")

secantMethod(f_of_x, -1, 1, 25)
```

Output:

```
The root is found at 1.549984207858443e-15 after 19 iterations
```

Task 3:
My code for task 3 is relatively the same as task 2 but I added this line to Newton's method to create the table:

```
print(f"iter: {iteration}, approximation: {x}")
```

Output:

```
iter: 0, approximation: -0.5
iter: 1, approximation: -0.16666666666666669
iter: 2, approximation: -0.023809523809523836
iter: 3, approximation: -0.0005537098560354364
iter: 4, approximation: -3.0642493416461764e-07
iter: 5, approximation: -9.389621148813321e-14
iter: 6, approximation: -8.80999858950826e-27
iter: 7, approximation: 0.0
iter: 8, approximation: 0.0
iter: 9, approximation: 0.0
The root is found at 0.0 after 10 iterations
```

I added this line to the Secant method to create a table:

```
print(f"iter: {iteration}, x0= {x0}, x1= {x1}, approximation: {x1}")
```

Output:

```
iter: 1, x0= 1, x1= 0.7615941559557649, approximation: 0.7615941559557649
iter: 2, x0= 0.7615941559557649, x1= -6.145115192053641, approximation: -6.145115192053641
iter: 3, x0= -6.145115192053641, x1= 0.7607373842425833, approximation: 0.7607373842425833
iter: 4, x0= 0.7607373842425833, x1= 0.7598809490516972, approximation: 0.7598809490516972
iter: 5, x0= 0.7598809490516972, x1= -2.4117305197619667, approximation: -2.411730519761966
iter: 6, x0= -2.4117305197619667, x1= 0.7185207502140756, approximation: 0.7185207502140756
iter: 7, x0= 0.7185207502140756, x1= 0.6782842464737974, approximation: 0.6782842464737974
iter: 8, x0= 0.6782842464737974, x1= -1.6152103158907716, approximation: -1.615210315890771
iter: 9, x0= -1.6152103158907716, x1= 0.5850439821307694, approximation: 0.5850439821307694
iter: 10, x0= 0.5850439821307694, x1= 0.5001671785336054, approximation: 0.5001671785336054
iter: 11, x0= 0.5001671785336054, x1= -0.6389152674058604, approximation: -0.638915267405860
iter: 12, x0= -0.6389152674058604, x1= 0.2719161660239241, approximation: 0.271916166023924
```

```
iter: 13, x0= 0.2719161660239241, x1= 0.13879672313573155, approximation: 0.138796723135731
iter: 14, x0= 0.13879672313573155, x1= -0.04740627021736338, approximation: -0.047406270217
iter: 15, x0= -0.04740627021736338, x1= 0.00687409861557263, approximation: 0.0068740986155
iter: 16, x0= 0.00687409861557263, x1= 0.0003193254984228877, approximation: 0.000319325498
iter: 17, x0= 0.0003193254984228877, x1= -2.202990601857084e-06, approximation: -2.20299060
iter: 18, x0= -2.202990601857084e-06, x1= 7.035826268411442e-10, approximation: 7.03582626
iter: 19, x0= 7.035826268411442e-10, x1= 1.549984207858443e-15, approximation: 1.54998420785
The root is found at 1.549984207858443e-15 after 19 iterations
```

Task 4:
The code for my Newton hybrid method is as follows:

```
import numpy as np


def f_of_x(x):
    return 10.14 * np.exp(x * x) * np.cos(np.pi / x)


def d_of_x(x):
    return 10.14 * (2 * x * np.exp(x * x) * np.cos(np.pi / x) + (np.pi * np.exp(x * x) * np.


def newtonHybrid(func, a, b, tol, maxIter):
    error = 10 * tol
    iteration = 0
    x0 = .5 * (a + b)
    while (error > tol and iteration < maxIter):
        x1 = x0 - (f_of_x(x0) / d_of_x(x0))
        newtError = np.abs(x1 - x0)
        if newtError > error:
            for i in range(1, 4):
                c = .5 * (a + b)
                fa = f_of_x(a)
                fb = f_of_x(b)
                fc = f_of_x(c)
                if fa * fc < 0:
                    b = c
                    fb = c
                elif fb * fc < 0:
                    a = c
                    fa = fc
            error = np.abs(b - a)
            x0 = .5 * (a + b)
        else:
            x0 = x1
            error = newtError
```

```
            iteration += 1
        print(f"iter: {iteration}, approximation: {x0}")
    print(f"The root is found at {x0} after {iteration} iterations")

newtonHybrid(f_of_x, -3, 7, .0000001, 25)
```

Output:

```
iter: 0, approximation: 2.4121249999999996
iter: 1, approximation: 2.264797283659556
iter: 2, approximation: 2.1370335641976634
iter: 3, approximation: 2.0467849219470153
iter: 4, approximation: 2.006737554998073
iter: 5, approximation: 2.000155818029528
iter: 6, approximation: 2.0000000849388115
iter: 7, approximation: 2.0000000000000253
The root is found at 2.0000000000000253 after 7 iterations
```

Task 5:
The code for my Secant hybrid method is as follows:

```
import numpy as np

def f_of_x(x):
    return 10.14 * np.exp(x * x) * np.cos(np.pi / x)

def secantHybrid(func, a, b, tol, maxIter):
    x0 = a
    x1 = b
    error = 10 * tol
    iteration = 0

    f0 = f_of_x(x0)
    f1 = f_of_x(x1)

    while (error > tol and iteration < maxIter):
        xi = x1 - ((f1 * (x1 - x0)) / (f1 - f0))
        secantError = np.abs(xi - x1)

        if (secantError > error):
            fa = f_of_x(a)
            fb = f_of_x(b)

            for i in range(1, 4):
                c = .5 * (a + b)
                fc = f_of_x(c)

                if (fa * fc < 0):
```

```
                b = c
                fb = fc

            elif (fb * fc < 0):
                a = c
                fa = fc
        error = np.abs(b - a)

        x0 = a
        x1 = b

    else:
        x0 = x1
        x1 = xi
        f0 = f1
        f1 = f_of_x(xi)
        error = secantError
        iteration += 1
print(f"The root is found at {x0} after {iteration} iterations")

secantHybrid(f_of_x, -3, 7, .0000001, 25)
```

Output:

```
The root is found at -2.9999999999999982 after 2 iterations
```