Math 4610 Assignement 2 - Kaden Taylor A02257212

Task 1: Newton's Method for Root Finding.

Here is a link to the software manual that includes documentation about the code, how to use it, examples, and the code itself: doc/newton.md

My code for the newton functin is here:

```
def newton(f, df, x0, tol, v):
    if v == 0:
        if abs(f(x0[-1])) < tol:
            return x0
        else:
            x0_next = x0[-1] - f(x0[-1])/df(x0[-1])
            x0[-1] = x0_next
            return newton(f, df, x0, tol, 0)
    else:
        if abs(f(x0[-1])) < tol:
            return x0
        else:
            x0_next = x0[-1] - f(x0[-1])/df(x0[-1])
            x0.append(x0_next)
            return newton(f, df, x0, tol, 1)
```

The code is tested with the function, `f(x) = x*math.e**-x`

The command line entry looks like this:

```
$ python rootfinding.py x*math.e**-x math.e**-x-x*math.e**-x x-x*math.e**-x -2 1 .1 .2 .000001 20 1
```

The output for the Newton method is here:

```
----------Newton-------------------------------------------------------------------
iter |              x               |             f(x)             |      |Error|
----------------------------------------------------------------------------------
  0  |  -0.011111111111111113       |  -0.011235256319625977       |  0.11111111111111112
  1  |  -0.0001221001221001234      |  -0.00012211503145013845     |  0.01098901098901099
  2  |  -1.4906619716777104e-08     |  -1.490661993898442e-08      |  0.00012208521548040662
----------------------------------------------------------------------------------
```

Task 2: Secant Method for Root Finding.

Here is a link to the software manual that includes documentation about the code, how to use it, examples, and the code itself: doc/secant.md

My code for the Secant method is here:

```
def secant(f, x0, tol, v):
    if v == 0:
        if abs(f(x0[-1])) < tol:
            return x0
        else:
            x0_next = x0[-2] - ((x0[-1]-x0[-2]) * f(x0[-2])) / (f(x0[-1]) - f(x0[-2]))
            x0[-2] = x0[-1]
            x0[-1] = x0_next
            return secant(f, x0, tol, 0)
    else:
        if abs(f(x0[-1])) < tol:
            return x0
        else:
            x0_next = x0[-2] - ((x0[-1]-x0[-2]) * f(x0[-2])) / (f(x0[-1]) - f(x0[-2]))
            x0.append(x0_next)
            return secant(f, x0, tol, 1)
```

The code is tested with the function, `f(x) = x*math.e**-x`

The command line entry looks like this:

```
$ python rootfinding.py x*math.e**-x math.e**-x-x*math.e**-x x-x*math.e**-x -2 1 .1 .2 .000001 20 1
```

The output for the Secant method is here:

```
----------Secant-----------------------------------------------------------------------
iter |              x              |             f(x)            |       |Error|
-------------------------------------------------------------------------------------------
  0  |  0.2                        |  0.1637461506155964         |  0.1
  1  |  -0.023506370143776475      |  -0.024065464982500492      |  0.2235063701437765
  2  |  0.005132884712382435       |  0.005106605708157981       |  0.02863925485615891
  3  |  0.00011954905045136535     |  0.00011953475933016354     |  0.00501333566193107
  4  |  -6.152459741965571e-07     |  -6.152463527242824e-07     |  0.00012016429642556191
-------------------------------------------------------------------------------------------
```

Task 3: Tabulation of Results

All the programs accept a `v` flag as their last argument. `v = 0` prints a single result while `v=1` prints all the iterations.

Task 4: Hybrid Method - Bisection/Newton's Method

Here is a link to the software manual that includes documentation about the code, how to use it, examples, and the code itself: doc/hybrid_newton.md

My code for the Hybrid Bisection and Newton Method is here:

```
def hybrid_newton(f, df, a, b, tol, maxIter, v):
    error = 10.0*tol
    iteration = 0
    x0 = .5*(a+b)
    array = []
    while error > tol and iteration < maxIter:
        x1 = x0 - f(x0)/df(x0)
        newton_error = abs(x1 - x0)
        if newton_error > error:
            fa = f(a)
            fb = f(b)
            for i in range(1, 4):
                c = 0.5*(a+b)
                fc = f(c)
                if fa*fc < 0:
                    b = c
                    fb = fc
                else:
                    a = c
                    fa = fc
            error = abs(b-a)
            x0 = .5*(a + b)
        else:
            x0 = x1
            error = newton_error
        iteration = iteration + 1
        if v == 1:
            array.append(x0)
            array.append(error)
    if v == 0:

        array.append(x0)
        array.append(error)
    return array
```

The code is tested with the function, `f(x) = 10.14*((math.e**x)**2)*math.cos(math.pi/x)` , on an interval `[-3,7]` .

The rootfinding.py script couldn't take the function as an argument so the function and derivative were hard coded into rootfinding.py as:

```
def f(x):
    return 10.14*((math.e**x)**2)*math.cos(math.pi/x)


def df(x):
    return 10.14*((2*((math.e**x)**2)*x*math.cos(math.pi/x))+((math.pi*((math.e**x)**2)*math.sin(math.
```

The values on the command line were:

```
$ python rootfinding.py 1 1 1 -3 7 .1 .2 .000001 20 1
```

See the results below:

```
----------Hybrid Newton-------------------------------------------------------------------------
iter |              x               |             f(x)            |      |Error|
-------------------------------------------------------------------------------------------
  0  |  2.0                         | 3.3899768986470717e-14      |  0.0
-------------------------------------------------------------------------------------------
```

Task 5: Another Hybrid Method - Bisection/Secant Method

Here is a link to the software manual that includes documentation about the code, how to use it, examples, and the code itself: doc/hybrid_secant.md

My code for the Hybrid Bisection and Secant Method is here:

```
def hybrid_secant(f, a, b, tol, maxIter, v):
    error = 10.0*tol
    iteration = 0
    x0 = .49*(a+b)
    x1 = .51*(a+b)
    array =[]
    while error > tol and iteration < maxIter:
        x2 = x0 - ((x1-x0) * f(x0)) / (f(x1) - f(x0))
        secant_error = abs(x2 - x0)
        if secant_error > error:
            fa = f(a)
            fb = f(b)
            for i in range(1, 4):
                c = 0.5*(a+b)
                fc = f(c)
                if fa*fc < 0:
                    b = c
                    fb = fc
                else:
```

```
                a = c
                fa = fc
            error = abs(b-a)
            x0 = .49*(a+b)
            x1 = .51*(a+b)
        else:
            x0 = x1
            x1 = x2
            error = secant_error
        iteration = iteration + 1
        if v == 1:
            array.append(x1)
            array.append(error)
    if v == 0:
        array.append(x1)
        array.append(error)
    return array
```

The code is tested with the function, `f(x) = 10.14*((math.e**x)**2)*math.cos(math.pi/x)` , on an interval `[-3,7]` .

The rootfinding.py script couldn't take the function as an argument so the function was hard coded into rootfinding.py as:

```
def f(x):
    return 10.14*((math.e**x)**2)*math.cos(math.pi/x)
```

The values on the command line were:

```
$ python rootfinding.py 1 1 1 -3 7 .1 .2 .000001 20 1
```

See the results below:

```
----------Hybrid Secant-----------------------------------------------------------------------
iter |               x              |             f(x)            |      |Error|
------------------------------------------------------------------------------------------
  0  |  -2.48625                    |  0.021236866297075437       |  1.125
  1  |  -1.9842187500000001         |  -0.002394571874499232      |  0.140625
  2  |  -1.9967585912324382         |  -0.0004766538732894278     |  0.0903523412324383
  3  |  -1.9998750768643085         |  -1.8227555084960276e-05    |  0.0156563268643084
  4  |  -1.9999989918998782         |  -1.470464927420363e-07     |  0.0032404006674400243
  5  |  -1.9999999996852116         |  -4.5916473408971796e-11    |  0.00012492282090303952
  6  |  -1.9999999999999991         |  -1.1234265173688567e-16    |  1.0081001209361062e-06
  7  |  -1.9999999999999998         |  -2.986614685263548e-17     |  3.147881955101184e-10
------------------------------------------------------------------------------------------
```

The `rootfinding.py` Program

This is the program that compiles all the methods together and formats all the prints of the results. Here is the code for the program:

```
import fixedPointIteration as fp
import bisection as bi
import newton as newton
import secant as secant
import hybrid_newton as hybrid_newton
import hybrid_secant as hybrid_secant
import math
import sys

if len(sys.argv) > 8:
    f_expression = sys.argv[1]
    df_expression = sys.argv[2]
    g_expression = sys.argv[3]
    a = sys.argv[4]
    b = sys.argv[5]
    x0 = sys.argv[6]
    x1 = sys.argv[7]
    tol = sys.argv[8]
    maxIter = sys.argv[9]
    v = sys.argv[10]
else:
    print("Don't add any parenthesis to the equations")
    f_expression = input("Enter a value for f(x) = ")
    df_expression = input("Enter a value for f'(x) = ")
    g_expression = input("Enter a value for g(x) = ")
    a = input("Enter a value for a = ")
    b = input("Enter a value for b = ")
    x0 = input("Enter a value for x0: ")
    x1 = input("Enter a value for x1: ")
    tol = input("Enter a value for tol: ")
    maxIter = input("Enter a value for maxIter: ")
    v = input("Print full table? 1 = Yes, 0 = No: ")


def f(x):
    return 10.14*((math.e**x)**2)*math.cos(math.pi/x)


def df(x):
    return 10.14*((2*((math.e**x)**2)*x*math.cos(math.pi/x))+((math.pi*((math.e**x)**2)*math.sin(math.


def g(x):
    return x - f(x)


# f = lambda x: eval(f_expression)
# df = lambda x: eval(df_expression)
# g = lambda x: eval(g_expression)
a = float(a)
b = float(b)
x0 = float(x0)
```

```python
x1 = float(x1)
tol = float(tol)
maxIter = int(maxIter)
v = int(v)

fixed = fp.fixedPointIter(g, x0, tol, maxIter, v)
bisec = bi.bisection(f, a, b, tol, maxIter, v)

x0_array = [x0]
newt = newton.newton(f, df, x0_array, tol, v)

x0_array = [x0, x1]
sec = secant.secant(f, x0_array, tol, v)

hnewt = hybrid_newton.hybrid_newton(f, df, a, b, tol, maxIter, v)
hsec = hybrid_secant.hybrid_secant(f, a, b, tol, maxIter,v)

print("\nIf the full table is not requested, the final iteration will show as iteration 0\n")

print("----Fixed Point Iteration-------------------------------------------------------------------")
print("iter |              x               |              f(x)             |       |Error|  ")
print("--------------------------------------------------------------------------------------------")
for i in range(0, int(len(fixed)/2)):
    print(f"{i:>3}  |  {fixed[i*2]:<25}  |  {f(fixed[i*2]):<25}  |  {fixed[i*2 + 1]:<25}")
print("--------------------------------------------------------------------------------------------\n\n

print("----------Bisection-------------------------------------------------------------------------")
print("iter |              x               |              f(x)             |       |Error|  ")
print("--------------------------------------------------------------------------------------------")
for i in range(0, int(len(bisec)/2)):
    print(f"{i:>3}  |  {bisec[i*2]:<25}  |  {f(bisec[i*2]):<25}  |  {bisec[i*2 + 1]:<25}")
print("--------------------------------------------------------------------------------------------\n\n

if len(newt) < 2:
    newt.append(newt[0])

print("----------Newton-------------------------------------------------------------------------")
print("iter |              x               |              f(x)             |       |Error|  ")
print("--------------------------------------------------------------------------------------------")
for i in range(1, len(newt)):
    print(f"{i-1:>3}  |  {newt[i]:<25}  |  {f(newt[i]):<25}  |  {(abs(newt[i] - newt[i-1])):<25}")
print("--------------------------------------------------------------------------------------------\n\n

print("----------Secant-------------------------------------------------------------------------")
print("iter |              x               |              f(x)             |       |Error|  ")
print("--------------------------------------------------------------------------------------------")
for i in range(1, len(sec)):
    print(f"{i-1:>3}  |  {sec[i]:<25}  |  {f(sec[i]):<25}  |  {(abs(sec[i] - sec[i-1])):<25}")
print("--------------------------------------------------------------------------------------------\n\n

print("----------Hybrid Newton-------------------------------------------------------------------------
print("iter |              x               |              f(x)             |       |Error|  ")
print("--------------------------------------------------------------------------------------------")
for i in range(0, int(len(hnewt)/2)):
    print(f"{i:>3}  |  {hnewt[i*2]:<25}  |  {f(hnewt[i*2]):<25}  |  {hnewt[i*2 + 1]:<25}")
print("--------------------------------------------------------------------------------------------\n\n
```

```
print("----------Hybrid Secant-------------------------------------------------------------------------
print("iter |             x               |            f(x)             |      |Error|  ")
print("--------------------------------------------------------------------------------------------")
for i in range(0, int(len(hsec)/2)):
    print(f"{i:>3}  |  {hsec[i*2]:<25}  |  {f(hsec[i*2]):<25}  |  {hsec[i*2 + 1]:<25}")
print("--------------------------------------------------------------------------------------------\n\n
```