ME EN 2450 | Kyan Malloy u08[illegible]07 | HW2

Exercise 1

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)h + \frac{f''(x_i)h^2}{2!}$$

$f(x) = e^{-x}$ $f''(x) = e^{-x}$ $x_{i+1} = 1$ $x_i = 0.25$ $h = 0.75$

$f'(x) = -e^{-x}$

$$f(1) \approx f(.25) + f'(.25)(0.75) + \frac{f''(0.25)(0.75)^2}{2!}$$

$$f(1) \approx 0.41373716$$

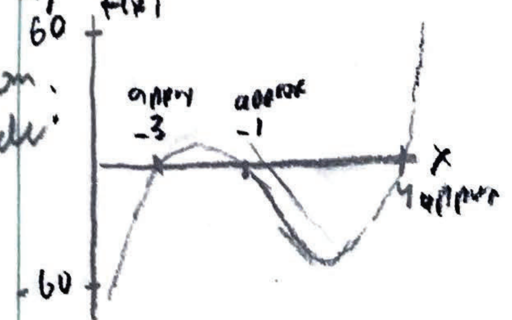
$$f(1) = e^{-1} = 0.3678794412$$

$$E_r = \overset{\text{true}}{T} - \overset{\text{Approx}}{A} = 0.3678794412 - 0.41373716$$

$$= \boxed{-0.045858}$$

Exercise 2 a)

From code:



approximate roots: -3, -1, 4

Exercise 3

$$y(x) = \frac{w_0}{(120EIL)}(-x^5 + 2L^2x^3 - L^4x)$$

$E = 29 \times 10^4$ psi
$I = 723 \text{ in}^4$
$w_0 = 3000 \frac{\text{lbs}}{\text{ft}}$
$L = 15 \text{ ft}$

lbs

a) $$\frac{dy}{dx} = \frac{w_0}{(120EIL)}(-5x^4 + 6L^2x^2 - L^4)$$

b) $\frac{dy}{dx} = 0$ when at max deflection

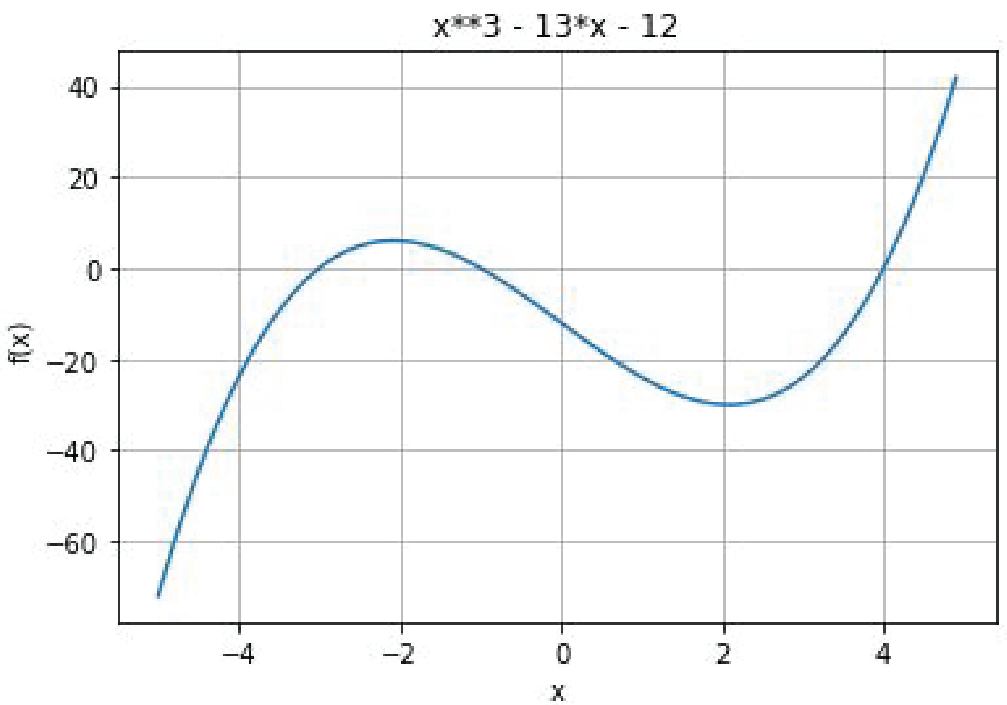
say $\frac{dy}{dx} = f(x) = \frac{w_0}{(120EIL)}(-5x^4 + 6L^2x^2 - L^4) = 0$

$$f'(x) = \frac{w_0}{(120EIL)}(-20x^3 + 12L^2x)$$

→ Using Newton Raphson solve for x value when y = max deflection

ME EN 2450
Ryan Dalby u0848407
HW2

2a)



b)

Iteration	Root estimate	Approximate Relative Error(%)
1	4.5	
2	3.25	-38.46153846153847
3	3.875	16.129032258064516
4	4.1875	7.462686567164178
5	4.03125	-3.875968992248062

c) Comparing Müller’s method to the bisection method that I created I can see that Müller’s method converges to a value quicker than the bisection method. This can be seen as after 5 iterations the approximate relative error of Müller’s method is .0000119% while the bisection method is 3.876%. Since the error of Müller’s method is less after the same iterations we can see convergence is quicker and thus for this problem Müller’s method converges quicker than the bisection method.

3b) Newton Raphson, root = 80.49844718999243inches iterations = 3 tolerence = 0.001% initial guess = 90inches

c)Scipy fsolve, root = 80.49844718999243inches

As we can see the Newton Raphson method gives the same answer as Scipy fsolve, both are: 80.49844718999243 inches.

d)Max displacement = -2.985433897788816inches
This number seems reasonable because for a 15 foot beam we see that we have approximate a maximum of 3 inches of deflection. This is not too large(depends on application though) since we likely want some compliance in our beam.

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Feb  2 18:21:39 2019
HW2 Bisection
@author: Ryan Dalby
"""
import numpy as np
import matplotlib.pyplot as plt

def bisection(func, a, b, iterations = 0):
    """
    Given a function and bounds(a and b) for which a root exists between will return the value of t
    """
    if(func(a)*func(b) >= 0):
        raise Exception("There is not root between a and b given(or root was given as a or b)")

    stringFormat = "{:<15}\t{:<15}\t{:<15}" #formatting string for output
    currentA = a
    currentB = b
    lastC = 0 #last iteration's root esitmate
    i = 1 # loop counter
    print(stringFormat.format("Iteration", "Root estimate", "Approximate Relative Error(%)"))
    while(i <= iterations):
        c = (currentA + currentB) / 2

        #here we move either a or b to c
        if(func(a)*func(c) > 0): #if the function at a and c are the same sign then move a to c
            currentA = c
        elif(func(a)*func(c) < 0): #if the function at a and c are different signs then move b to c
            currentB = c
        else: #else the function at c is 0 and c is the root
            return c
        Ea = ((c - lastC) / c) * 100 #this is the current approximate relative error in percent
        print(stringFormat.format(i, c, Ea))
        lastC = c #set lastC for next loop
        i += 1 #set counter value for next loop

    return lastC

f = lambda x: x**3 - 13*x - 12
bisection(f, 2, 7, 5)
xVals = np.arange(-5,5, .1)
plt.plot(xVals, f(xVals))
plt.grid()
plt.title("x**3 - 13*x - 12")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
```

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  4 18:34:16 2019
HW2 Newton-Raphson
@author: Ryan Dalby
"""

import scipy.optimize as sciOp

def newtonRaphson(f, dfdx, initialGuess, tolerence):
    """
    Given a function and its derivative, an initial guess for a root value, and a termination toler
    will attempt to find a root of the original function and return it along with the iterations it
    """
    UPPER_LIMIT_ON_ITERATIONS = 100000
    x = initialGuess #will hold current x value
    lastX = initialGuess #will hold last x value
    Ea = 100 #approximate relative error, in percent, start arbitrarily at 100 to get while loop to
    i = 0 #iteration counter, will incrment at bottom of loop and when it exits will be how many lo
    while(abs(Ea) > tolerence):
        if(f(x) == 0): #method cannot be computed if dervivative at point we are looking at is 0
            raise Exception("The derivative at a point was 0")
        x = x - (f(x) / dfdx(x)) #netwton-raphson
        Ea = ((x - lastX) / x) * 100 #calculate current approximate relative error
        lastX = x
        i += 1
        if(i == UPPER_LIMIT_ON_ITERATIONS): #sets limit on number of iterations to find root
            raise Exception("The root could not be found under the limit of iterations")

    return (x,i)

E = 29 * 10**4 #psi
w0 = 250 #3000 lbs/ft in lbs/in
I = 723 #in^4
L = 180 #15 ft in inches

y = lambda x: (w0/(120*E*I*L)) * (-x**5 + 2*L**2*x**3 - L**4*x)
f = lambda x: (w0/(120*E*I*L)) * (-5*x**4 + 6*L**2*x**2 - L**4) #dy/dx
dfdx = lambda x: (w0/(120*E*I*L)) * (-20*x**3 + 12*L**2*x)

tol = .001 #tolerence for newton raphson in percent
intialGuess = 90 #inches
newtonRaphsonInfo = newtonRaphson(f, dfdx, intialGuess, tol) #0 index is root, 1 is iterations to f
scipyRoot = sciOp.fsolve(f, 90)
print("Newton Raphson, root = {}inches iterations = {} tolerence = {}% initial guess = {}inches".fo
print("Scipy fsolve, root = {}inches".format(scipyRoot[0]))
maxDisplacement = y(newtonRaphsonInfo[0])
print("Max displacement = {}inches".format(maxDisplacement))
```