ME EN 2450 HW2b

Name: Christopher Wall

I declare that the assignment here submitted is original except for source material explicitly acknowledged.

I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the University website.

Christopher Wall

Name

Signature

u1467634

Student ID

Score

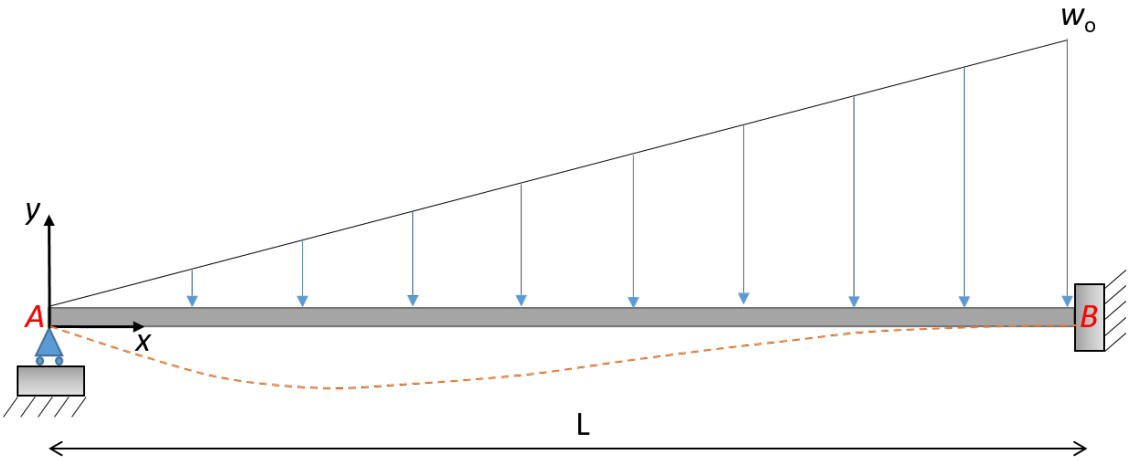
Total:

Q1 - 8 pts

Determine the maximum deflection of a beam. You will not need any previous knowledge of structural mechanics to solve this problem. The beam illustration and problem statement are only here to provide engineering context to the polynomial for which you will be determining roots.

A beam with uniform cross section is carrying a load that increases linearly from the pinned end (point A) to the fixed end (point B). The beam has the following properties:

- $E = 29 \times 10^4\ psi$ (Young's modulus)
- $I = 723\ in^4$ (Cross-section moment of inertia)
- $w_o = 3000\ \frac{lbs}{ft}$ (applied load, see figure)
- $L = 15\ ft$ (beam length, see figure)



The deformed shape of the beam, $y\,(x)$, (illustrated by the orange dashed line in the figure) is given by the following equation (note, the origin is placed at A):

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

a. Analytically derive the first derivative of $y\,(x)$ with respect to x, i.e. $\frac{dy(x)}{dx}$.

b. Next, the maximum deflection can be determined by setting $\frac{dy}{dx} = 0$ and solving for the root (the value of x where $\frac{dy}{dx} = 0$). Write a matlab or python script to determine the root using the Newton-Raphson method. Report the root in inches. Submit your code, along with what you used as the initial guess, tolerance, and how many iterations were required to achieve your tolerance.

c. Verify your obtained root using the built-in Matlab function `fzero`, Python function `scipy.optimize.fsolve` or `scipy.optimize.root`, or by solving analytically. Report a comparison of this result to that from your Newton-Raphson code.

Q2: (10 pts)

You now work on a new project to design compressed air tanks for energy storage:

Estimating Work Potential

Before creating machines that take advantage of sustainable energy sources, it is desirable to determine whether it would be feasible to do so. Such feasibility studies can take into account economic factors, such as setup or maintenance costs and expected profits, as well as more traditional engineer-ing factors, such as the safety of designs and the ease of replacing damaged parts. One such factor that should be considered is called exergy, which is an estimate of the maximum amount of useful work that we could expect to get from that source. (Exergy has the same units as energy, so in SI units it is in Joules, while in English units it may be in foot-pounds, calories, or Btu. Note: you should convert all measurements to SI units for your reported calculation.)

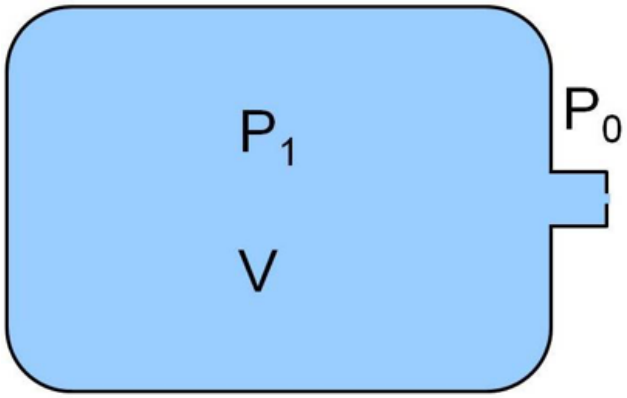


Figure 1: Schematic of a Compressed Air Tank. P1 indicates the pressure inside the tank, Po the atmospheric pressure outside of the tank, and V is the tank volume.

In the case of a tank of compressed air (Figure 1), exergy is the maximum amount of useful work we could expect to do by bringing the gas to the same state (i.e., pressure and temperature) as the environment. The work potential (exergy) of compressed air in a fixed-volume tank at the same temperature as its environment can be estimated using the formula:

$$X = P_1 V \left(\ln\left(\frac{P_1}{P_0}\right) + \frac{P_0}{P_1} - 1 \right) \tag{1}$$

where P_1 and P_0 are the pressure inside the tank and in the surrounding atmosphere, respectively, while V is the volume of the tank and X is the exergy. (Note that this is not a guarantee of the actual amount of energy you will be able to extract from the tank. It is an estimate of the most work that could be done if we used the gas in the most efficient way possible.)

Analyzing Error

Suppose that on a given day, the atmospheric pressure in Salt Lake City is $P_0 = 26.02$ inches Hg. You fill the tank with compressed air until the pressure gauge reads 50 psi (i.e., $P_1 = 50$ psi above the atmospheric pressure). For this calculation, choose a tank size of radius $r = 4.5$ inches, and a length $h = 0.55$ meters. Suppose you have a ruler capable of measuring the size of your tank with markings every $1/16$-inch - i.e., the measurement error is in the range of -1/16-inch to +1/16-inch. The pressure gauge on the pump you use to fill the tank has markings every 2 psi. The local atmospheric pressure is reported to the hundredths of an inch of mercury (in Hg).

(10 points) Perform an error propagation analysis on the exergy equation (using Equation 1) to determine (It would be much easier if you first convert everything into SI units):

- where the largest source of error is (1 pt), and
- if it's worth your time and money to invest in better sensors (1 pt).

Note that you need to first perform the symbolic analytical derivations by hand (4 pts) before calculating the numbers (4 pts) either by coding (recommended), or using calculators.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize

#deformed shape of beam
def y(x):
    #Beam properties:
    E = 29 * 10**4  #psi
    I = 723         #in^4
    w0 = 3000       #lbs/ft
    L = 15          #ft
    return (w0/(120*E*I*L)) * (-x**5 + (2 * L**2 * x**3) - (L**4 * x))

#analytically computed derivative of beam
def dydx(x):
    #Beam properties:
    E = 29 * 10**4  #psi
    I = 723         #in^4
    w0 = 3000       #lbs/ft
    L = 15          #ft
    return (w0/(120*E*I*L)) * (-5 * x**4 + (6 * L**2 * x**2) - (L**4))

#analytically computed second derivative of beam
def d2ydx2(x):
    #Beam properties:
    E = 29 * 10**4  #psi
    I = 723         #in^4
    w0 = 3000       #lbs/ft
    L = 15          #ft
    return (w0/(120*E*I*L)) * (-20 * x**3 + (12 * L**2 * x))

#One iteration of Newton-Raphson method
def newtIter( func, dfunc, x1):
    return x1 - (func(x1)/dfunc(x1))

def newtonRaphson( func, dfunc, x1, tolerance= 1 * 10**-5, maxIterations=10):
    nIterations = 0
    xn = x1
    while (np.abs(func(xn)) > tolerance) and (nIterations < maxIterations):
         xn = newtIter( func, dfunc, xn)
         nIterations += 1
    return [xn, nIterations]

x = np.linspace(start=0, stop=15, num=100)
plt.figure()
plt.plot(x, y(x),label="Profile" )
#plt.plot(x, dydx(x),label="First derivative" )
plt.legend()
plt.xlabel("x")
plt.ylabel("y")
plt.title("Profile of rod")
plt.grid(True)
#plt.show()

# Newton-Raphson result
tolerance = 1 * 10**-8
root, iterations = newtonRaphson(dydx, d2ydx2, 7.5, tolerance)
print(f'Zero estimated at: {root}\" in {iterations} iterations with tolerance of {tolerance}')

# Zero using scipy
zero_scipy = scipy.optimize.fsolve(dydx, 7.5)
print(f'Zero found using scipy: {zero_scipy[0]}\"')

#The Newton-Raphson result matches the result using scipy
```

```
Zero estimated at: 6.708203968759237" in 2 iterations with tolerance of 1e-08
Zero found using scipy: 6.708203932499369"
```

$$E = P_1 V \left[\ln\left(\frac{P_1}{P_0}\right) + \frac{P_0}{P_1} - 1 \right]$$

$$\Delta P_0 \frac{\partial E}{\partial P_0} = \left(V - \frac{P_1 V}{P_0} \right) \Delta P_0$$

$$\Delta P_1 \frac{\partial E}{\partial P_1} = V \ln\left(\frac{P_1}{P_0}\right) \Delta P_1$$

$$\Delta V \frac{\partial E}{\partial V} = \left(P_1 \ln\left(\frac{P_1}{P_0}\right) + P_0 - P_1 \right) \Delta V$$

$$\Delta E = \left| \left(V - \frac{P_1 V}{P_0} \right) \Delta P_0 \right| + \left| V \ln\left(\frac{P_1}{P_0}\right) \Delta P_1 \right| + \left| \left(P_1 \ln\left(\frac{P_1}{P_0}\right) + P_0 - P_1 \right) \Delta V \right|$$

$$\frac{F}{L^2}\left(L^3\right) \qquad F(L) = E$$

$$\frac{\partial E}{\partial P_0} = \left(P_1 V \ln\left(\frac{P_1}{P_0}\right) + P_1 V \frac{P_0}{P_1} - P_1 V \right) \frac{\partial}{\partial P}$$

$$= \frac{P_1 V}{\frac{P_0}{P_0}} \left(\frac{-P_1}{P_0^2}\right) + V - 0 = \left(\frac{-P_1 V}{P_0} + V \right) \Delta P_0$$

$$\frac{\partial E}{\partial P_1} = V \ln\left(\frac{P_1}{P_0}\right) + 0 + 0 - V = \left(V \ln\left(\frac{P_1}{P_0}\right) - V \right) \Delta P_1$$

$$\frac{\partial E}{\partial V} = P_1 \ln\left(\frac{P_1}{P_0}\right) + P_0 - P_1 = \left(P_1 \ln\left(\frac{P_1}{P_0}\right) + P_0 - P_1 \right) \Delta V$$

```
import math
import numpy as np


P0 = 88113.8        #pascals
P1 = P0 + 344737.9  #pascals
r = .1143           #meters
h = .55             #meters
#Errors
dr = .0015875       #meters -> 1/16 in
dh = .0015875       #meters -> 1/16 in
dP0 = 16.93194      #pascal -> .005 inHg
dP1 = 6894.76       #pascal -> 1psi

#Error in Volume (partial derivatives of r and h)
def errorV(r, h, dr, dh):
    #returns partial derivates in terms of r and h
    return [np.abs(math.pi*2*r*h*dr) ,
            np.abs(math.pi * r**2 * dh) ]

#exergy formula
def exergy(P0, P1, r, h):
    V = np.pi * r**2 * h
    return P1 * V * (math.log(P1/P0) + (P0/P1) - 1)

def error(P0, P1, r, h, dP0, dP1, dr, dh):
    """
    Arguments:
        P0 (Pascals) : Pressure outside tank
        P1 (Pascals) : Pressure inside tank
        r (meters): Radius of tank
        h (meters): Height of tank
        dP0 (Pascals) : Pressure uncertainty outside tank
        dP1 (Pascals): Pressure uncertainty inside tank
        dr (meters): Radius uncertainty
        dh (meters): Height uncertainty
    """
    V = math.pi * r**2 * h
    #partial derivatives multiplied by errors
    gradP0 = np.abs((V - (P1*V)/P0) * dP0)
    gradP1 = np.abs((V * math.log(P1/P0)) * dP1)
    dVr , dVh = errorV(r,h,dr, dh)
    gradr = np.abs((P1 * math.log(P1/P0) + P0 - P1) * (dVr))
    gradh = np.abs((P1 * math.log(P1/P0) + P0 - P1) * (dVh))
    #printing errors
    print(f'Error from p0: {gradP0:.5} Joules \nError from P1: {gradP1:.5} Joules')
    print(f'Error from r: {gradr:.5} Joules')
    print(f'Error from h: {gradh:.5} Joules')
    print(f'Volume Errors:\n\tError from r: {dVr:.5} m^3\n\tError from h: {dVh:.5} m^3')
    return gradP0 + gradP1 + gradh + gradr

#outputting results
error = error(P0, P1, r, h, dP0, dP1, dr, dh)
print(f'Total error: {error:.5} Joules' )
print(f'Exergy = {exergy(P0, P1, r, h):.5} ± {error:.5} Joules')


# Since the largest error is from the error in P1 (the pressure inside the tank) it would make sense to invest
# in better pressure sensors inside the tank.
```

```
Error from p0: 1.4954 Joules
Error from P1: 247.74 Joules
Error from r: 215.87 Joules
Error from h: 22.431 Joules
Volume Errors:
        Error from r: 0.00062705 m^3
        Error from h: 6.5156e-05 m^3
Total error: 487.54 Joules
Exergy = 7771.3 ± 487.54 Joules
```