Problem Set #5

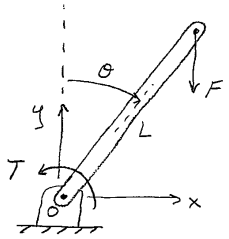
MMAE 350 – Computational Mechanics

Unless stated otherwise, complete all problems using
Python, Matlab, or Mathematica.

Submit Problem Set via Blackboard as a single pdf file.

Problem #1

Consider the torsional bar shown below.



The rigid bar is massless and of length L with a vertical force F at its end. A torsional spring at the bar's base has torsion given by $T = k\theta$, where θ is the angle of the bar from the vertical, and k is the torsional stiffness of the spring. The angle θ is sought where the system is in static equilibrium under the action of the force F and torsion T.

a) Apply the principles of static equilibrium to a free-body diagram to show that the governing equation is given by

$$f(\theta) = k\theta - FL\sin\theta = 0.$$

Observe that this is a nonlinear equation for θ given k, F, and L.

b) Evaluate the first two iterations of Newton's method by hand to obtain an estimate of θ for $k = 8$, $L = 10$, and $F = 1$ with an initial guess of $\theta^{(0)} = 0.9$ radians.

c) Use a built-in root-finding function in Python, Matlab, or Mathematica to obtain the fully converged root for θ and compare with your solution for part (b).

Problem #2

Consider an object of mass m free falling under the action of its own weight $W = mg$ and resisted by drag $F_D = -cv$, where $v(t)$ is the velocity and $c > 0$.

a) Apply Newton's second law to a free-body diagram to show that the governing equation is given by

$$\frac{dv}{dt} + \frac{c}{m}v = -g, \quad v = 0 \text{ at } t = 0,$$

where g is the acceleration due to gravity.

b) Show that the exact solution to this differential equation is given by

$$v(t) = \frac{mg}{c}\left(e^{-\frac{c}{m}t} - 1\right).$$

Therefore, given the mass m, acceleration due to gravity g, and the drag constant c, one may determine the velocity v "explicitly" as a function of time.

c) Suppose instead that one would like to determine the drag constant c given m, g, and the time t required for the object to reach a specified velocity v. In this case, it is not possible to solve explicitly for the drag constant c in terms of the remaining parameters. Instead, the *nonlinear equation*

$$f(c) = \frac{mg}{c}\left(e^{-\frac{c}{m}t} - 1\right) - v = 0$$

must be solved to determine the root c. Use a built-in root-finding function in Python, Matlab, or Mathematica to obtain the fully converged root for c with $m = 1$, $g = 9.81$, $t = 2$, $v = -7.2$, and an appropriate initial guess.

Problem #3

Recall from class that the Colebrook equation

$$\frac{1}{\sqrt{f}} = -2.0\log_{10}\left[\frac{\epsilon/D}{3.7} + \frac{2.51}{Re\sqrt{f}}\right]$$

can be used to approximate the Moody diagram that relates the friction factor f in turbulent fluid flow through pipes to the Reynolds number Re and the surface roughness ϵ/D.

For $Re = 2 \times 10^5$ and $\epsilon/D = 0.004$, solve the Colebrook equation for the corresponding friction factor f using built-in functions in Python, Matlab, or Mathematica with an appropriate initial guess.

Problem #4

Write a user-defined function in Python, Matlab, or Mathematica that approximates the root of a nonlinear equation $f(x) = 0$ using the *bisection method*. The input arguments to the user-defined function are the function $f(x)$, the two starting values x_0 and x_1 that bracket the sought after root, and the maximum tolerance to determine when a root is adequately approximated ($x_1 - x_0 \leq$ tolerance). The outputs of the function should be the estimate of the root and the number of iterations required to determine it. The function should check to be sure that the starting values bracket a single root; if not, output an appropriate message.

Use your user-defined bisection function to solve Problems #1 and #2 and compare your solutions to those obtained using built-in functions.

Problem #5

Consider the two nonlinear algebraic equations

$$4x_1^2 - x_2^3 = -28,$$
$$3x_1^3 + 4x_2^2 = 145.$$

a) Determine the general form of the system of linear algebraic equations (5.11) in the lecture notes that is to be solved for $\Delta x_1^{(k)}$ and $\Delta x_2^{(k)}$ at each iteration of Newton's method.

b) Using built-in functions in Python, Matlab, or Mathematica, calculate the point where the nonlinear equations intersect. Use the initial guesses $x_1^{(0)} = 1$ and $x_2^{(0)} = 1$.