MAE 3403 **Fall 2022** **Homework 2**

Due at 11:59 pm, Friday, September 16th, in the Canvas Dropbox

You will write four Python programs named ***hw2a.py, hw2b.py, hw2c.py and hw2d.py.*** Those programs will fulfill all of the requirements given in parts a, b, c, and d below. You will place those four python files into a single compressed file named ***firstname_lastname_hw2.zip***. You will upload the ***.zip*** file to the Homework 2 dropbox.

You will test your functions and using the particular numerical values and function calls given. When we grade your assignment, we will run your program with those given numerical values, looking for correct answers. Then we will change the numerical values (including changing the SIZES of the arrays) and look for correct answers for those modified values as well. We will only use numerical values, array sizes and functions that make sense. We will not be testing your program to see how it handles bad data.

In this assignment, you must use variables, loops, if statements, your own function definitions and function calls to write the required functions. For now, you ***may not*** use any of the powerful functions available in python modules, with two exceptions: you may import functions from the ***math*** module and the ***copy*** module.

You must include reasonable comments in your code.

See your MAE 3013 textbook and the MAE 3013 Equation Sheet for a reminder of:

The Secant Method for finding the solution (root or zero) or a nonlinear equation
The Simpson's 1/3 rule for numerical integration
The Gauss-Jacobi and Gauss-Seidel methods for solving a set of linear equations

a) Write a function defined as: def Simpson(func, a, b, npoints = 35):

Purpose: use Simpson's 1/3 rule to estimate the integral of func(x), between the limits of a and b.

func: the function we want to integrate
a and b: the lower and upper limits of integration
npoints: The number of integration points used in the range a to b (inclusive). Npoints must be an ODD number. If npoints is not ODD, then add 1 to make it odd!
return value: the estimate of the integral

Write and call a main() function that uses your Simpson function to estimate the value of the integral. The main function will print the value of integral. Use the following 3 test cases:

x – 3 * sin(x) with a = 1, b = 3 and npoints = 10

$\cos(2x)\cdot x^3$ with a = 2, b = 3 and npoints = 23

$\cos(2x)\cdot x^3$ with a = 2, b = 3 with npoints unspecified

b) Write a function defined as: def Secant(func, x0, x1, maxiter=15, xtol=0.0001):

Purpose: use the Secant Method to find the root of func(x), in the neighborhood of x0 and x1.

func: the function for which we want to find the root
x0 and x1: two x values in the neighborhood of the root
xtol: exit if the $|x_{newest} - x_{previous}| < xtol$
maxiter: exit if the number of iterations performed (***new x values***) equals this number
return value: the final estimate of the root (most recent new x value)

Write and call a main() function that uses your Secant function to estimate and print the solution of:

x – 3 * cos(x) = 0 with x0=1, x1= 2, maxiter = 5 and xtol = 1e-4

$\cos(2x)\cdot x^3 = 0$ with x0=1, x1= 2, maxiter = 15 and xtol = 1e-8

$\cos(2x)\cdot x^3 = 0$ with x0=1, x1= 2, maxiter = 3 and xtol = 1e-8

c) Write a function defined as: def GaussJacobi (A, b, xguess, maxiter = 10):

Purpose: use the GaussJacobi method to estimate the solution to a set of N linear equations expressed in matrix form as A x = b.

A: a square, full matrix of numbers having N rows and N columns, where N is the number of equations in the set.
b: an array of numbers having N terms, where N is the number of equations in the set.
xguess: a vector (array) contain the values of the initial guess
maxiter: the number of iterations (new x vectors) to compute
return value: the final new x vector.

Write and call a main() function that uses your GaussJacobi function to estimate and print the solution to the sets of linear equations contained in the following:

```
A1 = [[ 4, -1, -1],
      [-2, -3,  1],
      [-1,  1,  7]]

b1 = [3, 9, -6]
```

using initial guesses of all zeros. Perform 22 iterations.

```
A2 = [[ 4,  3,  1, -1],
      [ 2, -5,  0, -2],
      [-3,  3, -6,  1],
      [ 0,  1,  4,  8]]

b2 = [2, -3, 5, -2]
```

using initial guesses of all ONES. Perform 3 iterations.