Engineering 13300
MATLAB 4

This assignment has three sections.

I) The first section has Team Exercises that are intended to be done with your team. Work the problems together and answer the questions. Type the solutions into MATLAB to check them AFTER you have worked them out by hand. After you find the correct answers, you can move to the second section

II) The second section has five team tasks (Tasks 1-3) that you may work together and develop code together as long as you document who contributed.

III) The third section has two individual tasks (Tasks 4-5). You can work together to conceptually help each other but each person should work on their own code.

Section I

These are to be worked together as a team. There are no files to be uploaded to Brightspace or Gradescope. Rather, work the problems and come up with the answers and be prepared to present them to a member of the instructional team. After you get the right answers, move on to the next section. The intent is to understand the concepts before we begin to write code ourselves.

Track a, b, avector for every iteration. What is displayed at the end when this script is run in MATLAB

```
avector = [];
a = 1;
b = 1;
while a < 10
    if a == 5
        a = a + 2;
        continue;
    end
    avector = [avector a];
    for b = a:4:10
        if b == (a + 4)
            b = b + 4;
            break;
        end
        avector = [avector b];
    end
    a = a + 2;
end
fprintf('avector = ');
disp(avector);
```

Recall the guidelines for team activities:

1. You should work as a team; **all** team members will be held responsible for all material. You may work together and contribute to one program and submit similar codes as long as the contributors to the development of the solution are documented.
2. Each student is responsible for submitting their own assignment.

Team Task 1 (of 5) Community Pool requirements

Engineers work on municipal projects, such as community pool construction. You must write a user-defined function that will display information about the requirements for a new community pool project. A data file named **Data_pool_info.csv** contains information for several pool sizes. Each pool has the same geometry: a simple rectangular surface with a deeper end at one end of the pool. Your function will use programming to select and display information about the smallest allowable swimming pool that will meet a community's needs.
The community sanitation code has the following requirements:

- Surface area per swimmer in the pool: 25 ft^2
- Minimum diving depth, only for pools where diving boards will be installed: 10 ft
- Number of times that all the water must be recirculated through the filter: 3 times per day
- Flow rate through inlets that push the water back into the pool from the recirculation pump: 15 gal/min

You must write a user-defined function that will meet the following requirements: (Use ***ENGR133_MATLAB_UDF_Template.m***)

1. Create a function such that it accepts the following input arguments:
   - The maximum number of swimmers allowed in the pool at one time (scalar)
     - Return an error message if the input is not a scalar whole number greater than 0. (hint: MATLAB has a function "isscalar")
     - Return an error message if the input is greater than allowed by the largest pool in the data set.
   - An indicator for whether or not diving will be allowed that is a scalar value of 1 if diving is allowed or scalar value of 0 if diving is not allowed
     - Return an error message if the dive indicator value is not a scalar 1 or 0
2. Has zero output arguments.
3. Displays the following information to the MATLAB Command Window:
   - The volume of the pool selected, with appropriate units.
   - The maximum number of swimmers the selected pool can allow at one time (note that this may be different than the number of swimmers in the input argument)
   - The minimum total pumping capacity needed to recirculate the water, with units in gallons per minute.
   - The number of inlets needed to circulate the water back into the pool.

Run your function with the following test cases and then copy and paste the function call **and** all Command Window displays to the RESULTS section of your code file as comments. Suppress your function call for publishing by commenting out the function line and assigning the variables to the values of one of the text cases.

Test Case Description	Input 1: Number of swimmers	Input 2: Diving indicator
Valid number of swimmers and dive indicator	195	1
Valid number of swimmers and dive indicator	80	0
Invalid value for number of swimmers	[100 200]	1
Invalid value for number of swimmers	-50	1
Invalid value for number of swimmers	45.5	1
Invalid value for number of swimmers	600	1
Invalid value for diving indicator	100	2
Invalid value for diving indicator	100	[0 1]

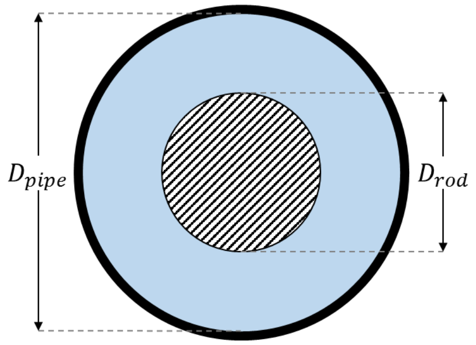
Publish the *Ma4_Task1_username.m* function as a PDF and name it *Ma4_Task1_username*.pdf. Upload *Ma4_Task1_username.m* also.

Task 1 Files:

- Ma4_Task1_username.m
- Ma4_Task1_username.pdf (published file)

Team Task 2 (of 5) Pipe Fluid Velocities

A pipe manufacturer is trying to understand the effects of changing the inner and outer diameters of a pipe system on the velocity of fluid through the pipe. The figure shows the schematic of pipe configuration consisting of a solid rod in the center of a hollow outer pipe. Fluid flows through the hollow part of the outer pipe. The manufacturer can vary both the internal diameter of the outer pipe and the diameter of the solid inner rod. They want you to create a function that will determine the velocity of a fluid moving through the pipe for different configurations. You can predict the velocity using the mass flow rate equation:



$$\dot{m} = \rho U A$$

Where $\dot{m}$ is a fixed mass flow rate of 2 kg/s, ρ is the fluid density at 1000 kg/m^3, U is the fluid velocity in m/s, and A is the cross-sectional area of the hollow region of the pipe system. The function should return only pipe systems that maintain velocities below 300 m/s.

Create a function (Use ***ENGR133_MATLAB_UDF_Template.m***) such that it:

- Accepts 2 input arguments: one vector of outer pipe diameters, D_{pipe}, (m) and one vector of corresponding inner pipe diameters, D_{rod}, (m). Each vector can have N-many elements, where N is a scalar whole number.
    - **Note**: these vectors can have different dimensions from each other. Your script should perform the calculations for each combination of diameters.
- Gives meaningful error message(s) if any outer pipe or rod diameter is 0 or smaller.
    - If an error message is produced, then the function returns an output argument of [-1 -1] and stops trying to find valid configurations.
- Return 1 output argument: one Nx2 array that contains valid combinations of outer pipe diameters in Column 1 and inner rod diameters in Column 2 that maintain fluid velocity greater than 0 m/s but slower than 300 m/s.

Run your function with the following test cases and then copy and paste the function call **and** all Command Window displays to the RESULTS section of your code file as comments. Suppress your function call for publishing.

Outer Diameter vector, D_{pipe}, (m)	**Inner Rod Diameter vector, D_{rod}, (m)**
0.01:0.01:0.05	0.03:0.02:0.09
[5 10 3]	[0 1 2 3 4]
[5 8 10]	[2 4]

Publish the *Ma4_Task2_username.m* function as a PDF and name it *Ma4_Task2_username*.pdf. Upload *Ma4_Task2_username.m* also.

Task 2 Files:

- Ma4_Task2_username.m
- Ma4_Task2_username.pdf (published file)

Team Task 3 (of 5) Generating a Matrix

1. Open the template file **ENGR133_MATLAB_Template.m** and save it as **Ma4_Task3_username.m**
2. Modify the file to create a script file that will create a matrix of order $M \times N$, where M denotes the number of rows and N denotes the number of columns. Your script should prompt the user for the values of M (number of rowns) and N (number of columns). Each element of the matrix will be assigned a value according to the following rules:
   a) Given an element in row i and column j:
      - If the value of i x j is odd, the element has a value of 0;
      - If the value of i x j is even, then the element has a value of i x j.
      - If the value of i is equal to j, then the element has a value of -1.
3. Run your script with the following test cases and then copy and paste the user inputs **and** all Command Window displays to the RESULTS section of your code file as comments. Comment the input commands for publishing and replace with hard coded values of M = 1 and N = 5.
   i. $M = 1, N = 5$
   ii. $M = 4, N = 3$

Publish the Ma4_Task3_*username.m* function as a PDF and name it Ma4_Task3_*username*.pdf. Upload Ma4_Task3_*username.m* also.

Task 3 Files:

- Ma4_Task3_*username*.m
- Ma4_Task3_*username*.pdf (published file)

Individual Tasks

Guidelines for Tasks 4-5:
Tasks 4-5 are individual tasks. You may seek help from classmates, the instructional team or others but the work you submit should be your own. If you collaborate with others and use information developed together or by someone else, ALWAYS document and reference that material.

Individual Task 4 (of 5) Image Rotation

Background
Image rotation is an increasingly commonly used feature in many smart-phone image apps. In this problem you will create an implementation of image rotation based on user input in MATLAB.

Problem Steps

1. Open the **ENGR133_MATLAB_UDF_Template.m** file. Complete the header information. Save your script as Ma4_Task4_image_flip_username.m
2. Create a MATLAB program that reads in the attached PNG image file (*imread*) and outputs it to the screen using the *imshow* function. Properly label this image as **Original Image**
3. Next, use *menu* function in MATLAB to generate a menu ask the user to select a rotation. Give the user the following options:
    a. 90 degrees clockwise
    b. 90 degrees counter-clockwise
    c. 180 degrees (flipped)
4. Write each of the above-mentioned operations in separate .m files as functions**
    a. Ma4_Task4_90_clockwise_username.m
    b. Ma4_Task4_90_counterclockwise_username.m
    c. Ma4_Task4_180_flipped_username.m
5. In the main program, ask the user which operation they would like to perform. Use the user selection to manipulate the included PNG image file.
6. Output the modified image to the screen and label the output **Image Rotated XX degrees** where XX indicates the rotation selected by the user (i.e. 90, -90, or 180 degrees)

** do **<u>not</u>** use any built in MATLAB functions such as *imrotate* to complete this task. Use double -nested for loops to implement.

Publish the Ma4_Task4_image_flip_*username*.m function as a PDF and name it Ma4_Task4_image_flip_*username*.pdf. The menu function will work when publishing

Task 4 files to submit:

- Ma4_Task4_90_clockwise_username.m
- Ma4_Task4_90_counterclockwise_username.m
- Ma4_Task4_180_flipped_username.m
- Ma4_Task4_image_flip_username.m
- Ma4_Task4_image_flip_username.pdf(published file)

Individual Task 5 (of 5) Image Conversion to Grayscale and Inverted

Background:
Image filters are an increasingly common feature of many image manipulation smart-phone apps. One commonly used image filter is to invert the colors of an image. In this task you will create MATLAB code to first convert an image to greyscale image, then invert it.

Problem Steps:
1. Open the **ENGR133_MATLAB_Template.m** file. Complete the header information. Save your script as Ma4_Task5_image_grey_invert.m
2. Create a MATLAB program that reads in the attached PNG image file and outputs it to the screen using the imshow function. Properly label this image as **Original Image**
3. Then convert the color image to a grayscale image using the ITU-R Recommendation BT.709 standard, where R is the red, G is the green, and B is the blue component of the image:
$$Y = 0.2126R + 0.7152G + 0.0722B$$
4. Display the grayscale image to the screen using the imshow function. Properly label this image as **Grayscale Image**
5. Next invert the grayscale image. If you'll recall from the Python image project, each pixel in a greyscale image ranges from 0-255 or 0 - 1. Inverting an image is changing the proportion of black and white present in the greyscale image. To invert an image, use one of the following equation, depending on your image values:

   Inverted_image = 255 - original_image_pixel_value

   or Inverted_image = 1 - original_image_pixel_value

6. Use the above equation to create a new array that is the newly inverted image. To do this create a new matrix that same size as the original image. Then, apply the equation to each pixel in the image and copy over the result to the newly created array.
7. Once you have inverted the image in the new array, display it to the screen with the label **Inverted Image**

Publish the Ma4_Task5_*username.m* function as a PDF and name it Ma4_Task5_*username*.pdf. Upload Ma4_Task5_*username.m* also.

Task 5 files to submit:
- Ma4_Task5_username.m
- Ma4_Task5_username.pdf (published file)