General Instructions

Due Date: Sunday, November 13th, 2022 by 11:59pm (submit via Zybooks)

Assignment Summary Instructions:
This assignment has two problems, summarized below. You will use MATLAB as a tool to solve the problem for the given test cases, ensuring that your code is flexible for any additional test cases that might be used to evaluate it.

1. Magic Square (Application to Data Analysis)
2. Parallel Computing (Application to Computer Engineering/Embedded Systems)

zyBooks Submission Instructions:
After completing this assignment in MATLAB, to receive credit, you must submit your code in Zybooks. The following components must be submitted under the specified chapter of the course Zybooks before the deadline to receive credit.

1. Chapter 36.1 MA7 Problem 1: Main Script
2. Chapter 36.2 MA7 Problem 2: SortProcesses Function
3. Chapter 36.3 MA7 Problem 2: Main Script

To submit your script, copy and paste your code into the submission window, making sure to remove any housekeeping commands. You may submit to Zybooks as many times as you want before the deadline, without any penalty. The highest score attained before the deadline will be graded. All components are due before the due date. No credit will be given if it is not submitted through the Zybooks platform before the deadline. Credit for each component will be awarded based upon the percentage of successfully completed assessments.

Explanation of P-Code:
Under the Additional Resources folder accompanying this prompt, you will find a file named **SortProcesses_Solution.p**. This is a working solution to the SortProcesses function required to complete Problem 2. This file is unopenable and the contents can't be read in a text editor, but it can be called like a typical .m function file in MATLAB. If you get stuck and are unable to successfully develop your own SortProcesses function, you are encouraged to instead copy SortProcesses_Solution.p to your working folder and rename it **SortProcesses.p**. Now, when your main script calls the SortProcesses function, it will automatically call the .p file and you will now have an opportunity to successfully finish developing your main script.

Proficiency Time: Times are included with the Background and Task sections. These times are the estimated amount of time it should take you to **redo** an assignment once you are fully proficient in material that it covers. To practice, reread the background in the given Comprehension Time and attempt to complete the problem in the given Proficiency Time.

Academic Honesty Reminder

The work you submit for this assignment should be your work alone. You are encouraged to support one another through collaboration in brainstorming approaches to the problem and troubleshooting. In this capacity, you are permitted to view other students' solutions, however, copying of another student's work is strongly discouraged.

This assignment will be checked for similarity using a MATLAB code. The similarity code will check each submission for likeness between other student submissions, past student submissions, the solution manual, and online resources and postings. If your submission is flagged for an unreasonably high level of similarity, it will be reviewed by the ENGI 1331 faculty, and action will be taken by faculty if deemed appropriate.

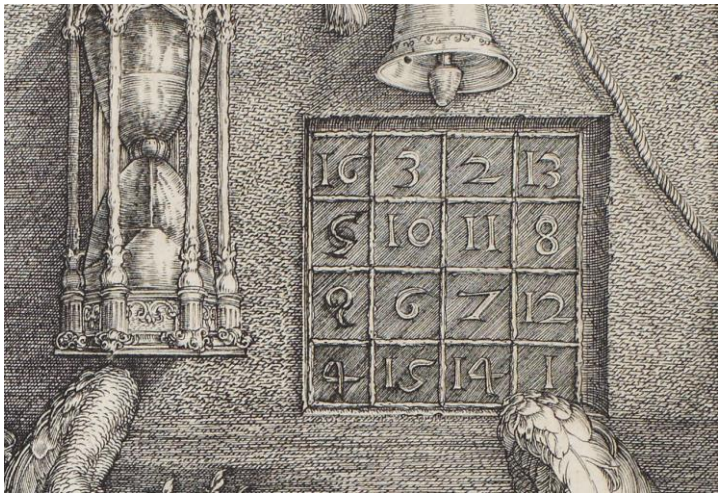
NOTE: Since this is an automated system for all sections, if any of your work is not your own, you will be caught. Changing variable names, adding comments, or spacing will not trick the similarity algorithm.

Problem 1: Magic Square

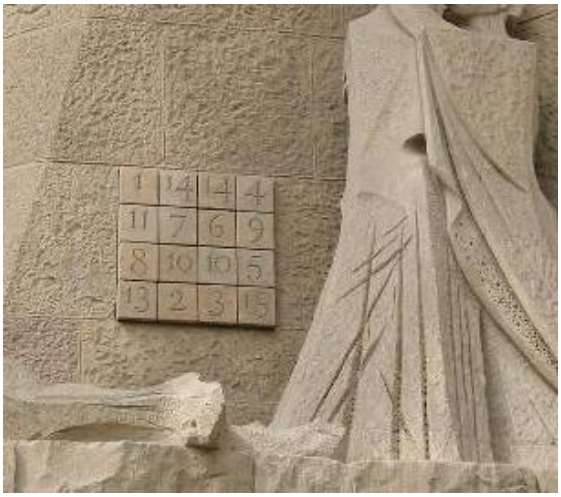
Background — Comprehension Time: 5 – 10 min

As early as 650 BCE, mathematicians had been composing magic squares, a sequence of numbers arranged in a square such that all rows, columns, and diagonals sum to the same constant.

Through history magic squares have been given magical, mythical, and religious significance, and have appeared in works of art around the world. Albrecht Dürer's engraving *Melencolia I* (1514 C.E.) is considered the first time a magic square appears in European art. The magic constant of Dürer's square is 34; that is, each row, column, and diagonal of the magic square all sum to 34. In Barcelona, the Sagrada Família's Façade of Passion contains a magic square embedded in its side whose magic constant is 33. These are both pictured below.



***Melencolia I* (1514 C.E.)**



Sagrada Família's Façade of Passion

You will develop a program to prove a series of numbers is indeed an *n* x *n* magic square. The program will also classify the magic square based on its numerical properties.

Tasks — Proficiency Time: 30 – 50 min

TASK 1: Matrix size (3 – 5 min)
Prompt the user to enter their proposed magic square in a single input statement (e.g. [1 2 3; 4 5 6; 7 8 9] – note that this example is a square matrix but is not a magic square). Determine if the matrix entered is a square matrix. If it is not, inform the user of their mistake and prompt the user to re-enter another matrix. Repeat this check until the user successfully enters a square matrix.

TASK 2: Whole numbers (10 – 15 min)
Check that each value in the matrix is a whole number. If it is not, prompt the user to round the number up or down (see example menu statements). Based on their choice, replace the decimal number with the rounded whole value. Once the entire matrix has been checked, save the matrix to **MA7_Task2.csv**.

TASK 3: Positive numbers (7 – 10 min)
Check that each value in the matrix is a positive number. Create a formatted output for the number of nonpositive (≤ 0) values found. For each nonpositive value, prompt the user to replace it with a positive whole number. If the user does not enter a valid number, continue prompting the user until a positive whole number is entered. Once the entire matrix has been checked, save the matrix to **MA7_Task3.csv**.

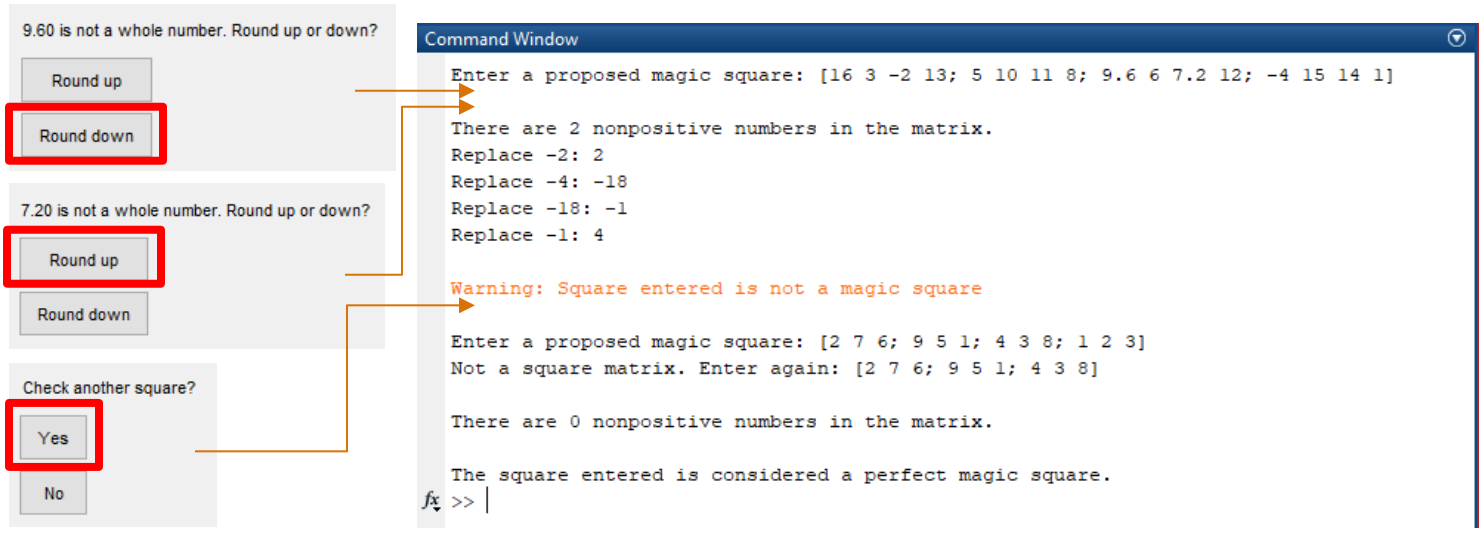
TASK 4: Determine magic square classification (10 – 20 min)
Determine if the matrix is a form of magic square. Each criterion below defines a classification of magic square. If none of the criteria are met, produce a warning informing the user that they have not entered a magic square, and ask the user if they wish to try entering a new matrix. Rerun the entire program if the user chooses to enter a new matrix.

Determine the classification of the magic square using the following requirements:

1. If each row and column sum to the same value (the magic constant), the magic square is classified as "semi-magic"
2. If, in addition to criterion 1, both diagonals sum to the same value as the rows and columns, the magic square is classified as "normal"
3. If, in addition to criteria 1 and 2, the values in the square are only the sequence of numbers from 1 to n^2 (for example, the values in a 3x3 square would be 1:9, and for a 4x4 square would be 1:16), the magic square is classified as "perfect"

Sample Output:

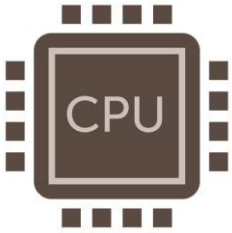
Additional test case scenarios are described in the Additional Resources folder accompanying this prompt.



Problem 2: Parallel Computing

Background Comprehension Time: 5 – 10 min

Computers execute processes, or commands from a program, using a central processing unit (CPU). When a computer must accomplish a given task, the CPU performs all the basic calculations, serves as a communication device between the other components in your computer, and manages how the memory of your computer is divided between programs.



In the first generations of modern CPUs (circa 1960s and 70s), a CPU had a single processing unit contained within it called a core. This means that the CPU ran one process at a time. CPUs today almost always have multiple cores, which allow them to perform several tasks at once.

As a computer engineer you are given an unorganized list of processes that must be completed on a dual-core CPU, as well as their desired start times (military time). This information is included in **Processes.mat**. You must write a program to organize these processes such that they can be performed in order on the CPU.

NOTE:
To ensure that your program is compatible with the CPU, **you can only use** the following functions:

length(), size(), isstring(), isvector(), isnumeric(), error(), load(), save()

You also **cannot use** implicit loops in your program. An implicit loop is defined as using MATLAB's built-in functions to index or manipulate data in an array. For example, adding 7 to the first five values of A by saying `A(1:5) = A(1:5) + 7;` instead of writing a for loop such as `for i = 1:5, A(i) = A(i) + 7; end` The first example, which uses the colon **:** as an operator, is considered an implicit loop because MATLAB is executing the second example, an explicit loop, in the background.

Tasks — Proficiency Time: 35 min – 1 hr

TASK 1: Data validation (5 – 10 min)
Load in **Processes.mat**, which contains the string array **Names** and the numerical vector **Times**. Verify that **Times** is numerical and that **Names** is a string. If either of these variables are scalars, matrices, or the wrong data type, produce an error that states the invalid variable(s) and terminate the program.

TASK 2: Organize processes (20 – 35 min)
Create a user-defined function named **SortProcesses** that sorts the CPU processes by start time in ascending order. The function will take two inputs: the vector of CPU process names and the associated vector of CPU process start times. The function should output the vector of CPU process names sorted by start time. After sorting the CPU processes, in your main script, save the sorted processes names to **MA7_sorted.mat**. Remember that your entire program (including SortProcesses) must only use the most basic functions, listed on Pg. 4 of this assignment.

TASK 3: Allocate processes (10 – 15 min)
Your dual-core CPU will function optimally if processes are distributed evenly between the two cores, with every other process occurring on Core 2. Using the organized list of processes from Task 2, split the list of processes between the two cores. Save the list of processes allocated to Core 1 as **Core1.mat**. Save the list of processes allocated to Core 2 as **Core2.mat**. Assume no processes have the same start times and assume there will be an even number of processes.

Sample Output:

Additional test case scenarios are described in the Additional Resources folder accompanying this prompt.

MA7_sorted.mat

	1
1	A
2	B
3	C
4	D
5	E
6	P
7	Q
8	R
9	S
10	T
11	K
12	L
13	M
14	N
15	O
16	U
17	V
18	W
19	X
20	Y
21	Z
22	F
23	G
24	H
25	I
26	J
27	

Core1.mat

	1
1	A
2	C
3	E
4	Q
5	S
6	K
7	M
8	O
9	V
10	X
11	Z
12	G
13	I
14	

Core2.mat

	1
1	B
2	D
3	P
4	R
5	T
6	L
7	N
8	U
9	W
10	Y
11	F
12	H
13	J
14	

Example of error prompts (all three cases shown):

```
Command Window
  Error using MA8_cougarnetID (line 20)
  Provided Names variable is invalid.

  Error using MA8_cougarnetID (line 22)
  Provided Times variable is invalid.

  Error using MA8_cougarnetID (line 24)
  Both provided variables are invalid.
```