Week 2 Homework Assignments: MATLAB Fundamentals

Global Requirements

- Add, commit, and push all deliverables to your `Week02` folder in your repository.
- At the top of each `.m` file, include:

```
% Student: Firstname Lastname
% Assisted by: First Person, Second Person, etc.
clc; clear;
```

- Some tasks intentionally use commands not covered in lecture. Use MATLAB `help`/`doc` or online documentation to learn them.

---

1. Matrix Arithmetic

Task

Write a MATLAB script that performs basic matrix and vector arithmetic.

Instructions

Script Name: `matrixOperations.m`

Objectives:

- Practice element-wise operations, matrix addition, subtraction, and scalar multiplication.
- Use `disp` or `fprintf` to show results clearly.

Steps:

1. **Initialize a Matrix**:
   - Create a 3x3 matrix `A` where each element is a number between 1 and 9:
     `A = [1, 2, 3; 4, 5, 6; 7, 8, 9];`
   - Create another 3x3 matrix `B` where each element is a number between 9 and 1:
     `B = [9, 8, 7; 6, 5, 4; 3, 2, 1];`
2. **Matrix Addition**:
   - Add `A` and `B` and store the result in `C`.
   - Display `C` with a clear label (e.g., `disp('Matrix C:'); disp(C)` or use `fprintf`).
3. **Matrix Subtraction**:
   - Subtract `B` from `A` and store the result in a new matrix `D`.
   - Display `D`
4. **Element-wise Multiplication**:
   - Multiply `A` and `B` element by element (use `.*`) and store the result in `E`.
   - Display `E`
5. **Matrix Scalar Multiplication**:
   - Multiply matrix `A` by a scalar value (for example, 2) and store the result in `F`.
   - Display `F`
6. **Indexing and Reassignment**:
   - Extract the **second row** of `A` and the **third column** of `B` and display them.
   - Replace the center element of `A` with `99` and display the updated `A`.

Deliverables

1. `matrixOperations.m` containing all of the above steps.
2. Include comments explaining what each part of the code does.
3. Use `disp` or `fprintf` to clearly present each result (do not rely on unsuppressed MATLAB output).

---

2. Vector Arithmetic

Task

Write a MATLAB script that demonstrates basic vector operations.

Instructions

Script Name: `vectorOperations.m`

Steps:

1. **Create a Row Vector**:
   - Define a row vector `x` with elements `[1, 3, 5, 7, 9]`.
2. **Create a Column Vector**:
   - Define a column vector `y` with elements `[2; 4; 6; 8; 10]`.
3. **Element-wise Multiplication**:
   - Multiply `x` by `y` element by element (use `.*`).
4. **Vector Transposition**:
   - Transpose the row vector `x` to a column vector and store it as `xT`.
5. **Vector Summation**:
   - Find the sum of all elements in vector `x` using the `sum` function.
   - Display the sum of x in a clear format (e.g., "The sum of x is… ")

Deliverables

1. `vectorOperations.m` containing all vector operations.
2. Add **comments** explaining each step.

---

3. Simple Data Visualization

Task

Write a MATLAB script that creates a basic line plot.

Instructions

Script Name: `simplePlot.m`

Steps:

1. **Create a Time Vector**:
   - `time = [0, 1, 2, 3, 4, 5];`.
2. **Create a Distance Vector**:
   - `distance = [0, 10, 20, 30, 40, 50];`.
3. **Plot the Data**:
   - Use the `plot` function to plot `time` on the x-axis and `distance` on the y-axis.
4. **Add Labels and Title**:
   - Label the x-axis as "Time (s)" and the y-axis as "Distance (m)".
   - Define a **string** variable for the title (e.g., `plotTitle = "Time vs Distance";`) and use it with `title(plotTitle)`.

Deliverables

1. `simplePlot.m` that creates the plot.
2. **Comments** explaining each part of the code.

---

4. Material Properties Calculation

Task

Create a MATLAB script that performs calculations for stress and strain and then visualizes the results.

Instructions

Script Name: `materialProperties.m`

Objectives:

- Calculate stress and strain using **vector** operations.
- Visualize the stress-strain relationship with a plot.

Steps:

1. **Variable Initialization**:
   - Define a vector `forces = [100, 200, 300, 400, 500];` (in Newtons).
   - Define a scalar `crossSectionArea = 50;` (in mmˆ2).
   - Define a vector `displacements = [0.1, 0.2, 0.3, 0.4, 0.5];` (in mm).
   - Define a scalar `originalLength = 100;` (in mm).
2. **Stress Calculation**:
   - Calculate stress using:
   $$\text{stress} = \frac{\text{force}}{\text{crossSectionArea}}$$
   for each force value.
   - Store the results in `stress`. Use element-wise calculations where needed.
3. **Strain Calculation**:
   - Calculate strain using:
   $$\text{strain} = \frac{\text{displacement}}{\text{originalLength}}$$
   for each displacement value.
   - Store the results in `strain`. Use element-wise calculations where needed.
4. **Logical Check**:
   - Create a logical vector that identifies which stress values are above 5: `highStress = stress > 5;`
   - Display the logical vector.
5. **Plot the Stress-Strain Curve**:
   - Plot the stress-strain curve using the calculated values.
   - Label the x-axis as "Strain" and the y-axis as "Stress (N/mmˆ2)".
   - Add a title "Stress-Strain Curve".

Deliverables

1. `materialProperties.m` containing all calculations and the plot.
2. Comments explaining each calculation and the final purpose of the plot.

---

5. Projectile Motion Simulation

Task

Simulate the vertical motion of a projectile under gravity, and visualize the results.

Instructions

Script Name: `projectileMotion.m`

Objectives:

- Compute the projectile's height at different times using vectorized operations.
- Visualize the motion using time vs. height.

Steps:

1. **Variable Initialization**:
   - Define constants: `gravity = 9.81;` (m/s^2) and `initialVelocity = 50;` (m/s).
   - Create a time vector `time = 0:0.1:10;` (representing 0 to 10 seconds).
2. **Height Calculation**:
   - Calculate the height at each time point using:

$$y = \text{initialVelocity} \cdot \text{time} - \frac{1}{2} \cdot \text{gravity} \cdot \text{time}^2$$

   - Store the results in a vector `height` using **dot operators** (e.g., `.*`, `.^`) because `time` is a vector.
3. **Plotting the Trajectory**:
   - Use `plot(time, height);` to create the plot.
   - Label the x-axis as "Time (s)" and the y-axis as "Height (m)".
   - Add a title "Projectile Motion Under Gravity".

Deliverables

1. `projectileMotion.m` with calculations and the plot.
2. Comments explaining each calculation and the purpose of the final plot.

---

6. Bug Hunt Challenge

Task

Identify and fix errors in a MATLAB script that calculates the total cost of items in a grocery list and displays the result.

Instructions

1. Copy the provided buggy MATLAB code to a script file named `buggyScript2.m`.
2. Open the script in the MATLAB Editor and try running it. Observe the errors or unexpected behaviors.
3. Identify and fix the bugs in the script. The bugs could include syntax errors, incorrect operations, or function misuse.
   - Pay attention to how arrays and variables are used and manipulated.
   - At least one error involves **operator precedence** (missing parentheses).
4. Use comments to explain each fix you make and describe the original error.

Example Buggy Script (`buggyScript2.m`)

```
% Task: Calculate the total cost of items in a grocery list
```

```
itemPrices = [2.5, 3.0, 4.5, 5]; % Prices of 4 items
itemQuantities = [2, 1, 3, 4]; % Quantities of each item bought
taxRate = 8.5; % percent

% Calculate the total cost for each item
totalCost = itemPrices * itemQuantities;

% Apply tax and display the total cost
grandTotal = sum(totalCost) * 1 + taxRate/100;
fprintf('The total cost of the grocery items is: %.2f\n', grandTotal);
```

Deliverables

1. Submit the corrected script file `fixedScript2.m`.
    - Ensure that all calculations and logic work as intended.
    - Include comments explaining each error you found and how you fixed it.
2. Write a short report (`debuggingReport2.txt`) containing the following:
    - Summarize the errors you encountered.
    - Explain how you found the solution to fix them.
    - Explain what you learned from the debugging process.

Definition of Done

- You have a GitHub repository with `gberl001` invited as a collaborator.
- Your `Week02` folder contains at minimum:
    - `matrixOperations.m`
    - `vectorOperations.m`
    - `simplePlot.m`
    - `materialProperties.m`
    - `projectileMotion.m`
    - `buggyScript2.m`
    - `fixedScript2.m`
    - `debuggingReport2.txt`
    - fixedScript2.m
    - debuggingReport2.txt