Al-Mustaqbal University
College of Technical Engineering
Department of Electrical Engineering Techniques

Computer Application UOMU000032

2025-2026

Lecture 4 – Modular Arithmetic and Looping Techniques in MATLAB

by
Dr. Mohammed Fadhil
PhD in Computer Networks
Email: mohammed.fadhil1@uomus.edu.iq

Learning Objectives

- Understand how to use the **mod** function for modular arithmetic
- Uses of **isempty** to check for empty arrays or variables.
- Understand how to use **for** loops to repeat operations in MATLAB.
- Understand the purpose and structure of **while** loops in MATLAB.

The Structure of if-else in MATLAB

- Basic syntax:

```
if condition
     % Code to execute if condition is true
elseif other_condition
    % Code to execute if other_condition is true
else
    % Code to execute if none of the conditions are true
End
```

- Note: Always close the if-else statement with end.

Example: if-elseif-else Statement

- Example to check if a number is positive, negative, or zero:

```
x = 0;
if x > 0
    disp('x is positive');
elseif x < 0
    disp('x is negative');
else
    disp('x is zero');
end
```

- Explanation:
  - MATLAB evaluates each condition in order until one is true. If none are true, else executes.

Using Logical Operators in Conditions

- Logical Operators allow combining multiple conditions:
  - **&&:** Logical AND (both conditions must be true)
  - **||:** Logical OR (at least one condition must be true)
  - **~:** Logical NOT (inverts true to false and vice versa)
- Example:

```
a = 5;
b = 10;
if a > 0 && b > 0
    disp('Both a and b are positive');
end
```

Nested if-else Statements

- What is Nesting?
  - Placing one if-else statement inside another for complex conditions.
- Example:

```
x = 10;
if x > 0
    if x > 5
        disp('x is positive and greater than 5');
    else
        disp('x is positive but 5 or less');
    end
end
```

Common Mistakes to Avoid

- Forgetting to use **end** to close **if-else** blocks.
- Incorrectly using **=** instead of **==** for equality check.
- Overusing nested **if-else** statements when simpler logic would suffice.
- Mixing up logical operators (e.g., **&&** and **||**).

if with Entire Arrays

- MATLAB evaluates conditions in if statements as true if all elements of an array meet the condition.
- Example:

```
A = [1, 2, 3];
if all(A > 0)
    disp('All elements are positive’);
end
```

- Note: If any element does not satisfy the condition, if will consider the entire condition as false.

Using all and any Functions with Arrays

- all(array): Returns true if all elements of array are true.
- any(array): Returns true if at least one element of array is true.
- Examples:

```
A = [1, -3, 5];
if any(A < 0)
    disp('There are negative elements’);
end

if all(A > 0)
    disp('All elements are positive');
else
    disp('Not all elements are positive');
end
```

Applying Element-wise Condition

- Element-wise conditions allow applying logical tests to each element in an array.
- Syntax: Use element-wise operators with arrays (&, |, ~).
- Example:

```
A = [5, -3, 8];
B = A > 0;  % Element-wise comparison
disp(B);  % Output: [1 0 1]
```

Conditional Indexing with Arrays

- You can use logical conditions to select elements from an array.
- Example:

```
A = [1, -2, 3, -4, 5];
posElements = A(A > 0);  % Select positive elements
disp(posElements);  % Output: [1 3 5]
```

Combining Multiple Conditions on Arrays

- Use logical operators to combine conditions for element-wise evaluations.
- Example:

```
A = [10, 15, 20, 25, 30];
selectedElements = A(A > 10 & A < 25);  % Elements between 10 and 25
disp(selectedElements);  % Output: [15 20]
```

Using Nested if Statements with Arrays

- Use nested if statements for multi-step checks on arrays.
- Example:

```
A = [4, 9, 16, 25];
if all(A > 0)
    if any(sqrt(A) == 5)
        disp('Array contains an element whose square root is 5');
    else
        disp('No element has a square root of 5');
    end
end
```

The mod Function

- Purpose: Calculates the remainder of a division operation.
- Syntax:

```
result = mod(a, b);
```

- Parameters:
  - a: Dividend
  - b: Divisor
- Example:

```
mod(10, 3)  % Returns 1 (remainder of 10/3)
```

Common Uses of the mod Function

- Even or Odd Check:
- To check if a number is even:

```
mod(n, 2) == 0
```

- Example:

```
n = 5;
if mod(n, 2) == 0
    disp('n is even');
else
    disp('n is odd');
end
```

Practical Examples of mod

- Example 1: Find numbers divisible by 3 in a vector.

```
A = [1, 2, 3, 4, 5, 6];
divisibleBy3 = A(mod(A, 3) == 0);  % Returns [3 6]
```

- Example 2: Display every third element in a vector:

```
A = [10, 20, 30, 40, 50, 60];
for i = 1:length(A)
    if mod(i, 3) == 0
        disp(A(i));
    end
end
```

The isempty Function

- Purpose: Checks if a variable or array is empty.
- Syntax:

```
result = isempty(variable);
```

- Returns: true if variable is empty, false otherwise.
- Example:

```
B = [];
isempty(B)  % Returns true
```

Common Uses of isempty

- Check if Arrays Are Empty:
  - Use in conditional statements to avoid errors in code execution.
- Validation Before Operations:
  - Ensures variables have data before performing calculations.
- Example:

```
values = [];
if isempty(values)
    disp('No values to process');
else
    disp( values(0) );
end
```

Understanding for Loops

- Definition: A for loop repeats a block of code a specified number of times.
- Usage: Ideal for iterating over arrays, performing calculations repeatedly, and automating repetitive tasks.
- Basic Structure:

```
for index = start:step:end
    % Code to execute
end
```

Basic Syntax of a for Loop

- Structure:

```
for i = 1:5
    disp(i);  % Displays values from 1 to 5
end
```

- Explanation:
  - i = 1:5 sets the loop to run from 1 to 5, incrementing by 1 each time.
  - Inside the loop, disp(i) displays the current value of i.

Using Custom Step Sizes

- Syntax: Define step sizes by specifying start:step:end.
- Example:

```
for j = 1:2:10
    disp(j);  % Displays odd numbers from 1 to 9
end
```

- Explanation: The loop starts at 1, increments by 2 each time, and stops at 10.

Iterating Over Arrays

- Purpose: for loops are commonly used to access each element in an array.
- Example:

```
A = [3, 6, 9, 12];
for k = 1:length(A)
    disp(A(k));  % Displays each element in A
end
```

- Explanation: The loop runs from 1 to length(A), displaying each element in A sequentially.

Using Nested for Loops

- Definition: A for loop inside another for loop.
- Common Use: Useful for iterating over matrices and multidimensional arrays.
- Example:

```
for i = 1:3
    for j = 1:3
        disp([i, j]);  % Displays all combinations of i and j
    end
end
```

- Explanation: The outer loop runs for each row, while the inner loop iterates through each column.

Example: Sum Array

- Problem: Write a for loop to calculate the sum of all elements in an array.
- Solution:

```
A = [1, 2, 3, 4];
total = 0;
for i = 1:length(A)
    total = total + A(i);
end
disp(total);  % Displays 10
```

Using break in a for Loop

- Purpose: break stops the loop when a condition is met.
- Example:

```
A = [3, 5, 8, 2];
for i = 1:length(A)
    if A(i) == 8
        disp('Found 8');
        break;  % Exit loop once 8 is found
    end
end
```

- Explanation: The loop stops immediately when A(i) == 8.

Using continue to Skip Iterations

- Purpose: continue skips to the next iteration without executing the remaining code in the loop.
- Example:

```
for i = 1:5
    if mod(i, 2) == 0
        continue;  % Skip even numbers
    end
    disp(i);  % Displays only odd numbers
end
```

- Explanation: The loop displays only odd numbers, as it skips even iterations.

Understanding while Loops

- Definition: A while loop repeats a block of code as long as a specified condition remains true.
- When to Use: Ideal when the number of iterations is not known in advance but depends on a condition.
- Basic Structure:

```
while condition
    % Code to execute repeatedly
end
```

Basic while Loop Syntax

- Structure:

```
x = 0;
while x < 5
    disp(x);
    x = x + 1;
End
```

- Explanation:
  - The loop will continue as long as **x < 5**.
  - Each iteration increments **x** by **1** and displays its value.

Avoiding Infinite Loops

- Explanation: If the loop condition is always true, the loop will run indefinitely.
- Solution: Ensure that a variable inside the loop changes so the condition can eventually become false.
- Example of Infinite Loop:

```
x = 1;
while x > 0
    disp(x);  % This will run indefinitely
end
```

- Fix: Increment or modify x within the loop to avoid infinite execution.

Using while Loops with Arrays

- Example: Finding the first negative element in an array.
- Solution:

```
A = [3, 5, -2, 8, -7];
i = 1;
while i <= length(A) && A(i) >= 0
    i = i + 1;
end
if i <= length(A)
    disp(['First negative element is ', num2str(A(i))]);
else
    disp('No negative elements found');
end
```

Try yourself! - Using Nested while Loops

- Definition: A while loop inside another while loop, useful for multi-level conditions.
- Example: Filling a 3x3 matrix with increasing numbers until a limit.

```
limit = 9;
matrix = zeros(3);
i = 1;
j = 1;
count = 1;
while count <= limit
    while j <= 3
        matrix(i, j) = count;
        count = count + 1;
        j = j + 1;
    end
    j = 1;  % Reset column
    i = i + 1;  % Move to next row
end
disp(matrix);
```

Using break in a while Loop

- Purpose: break stops the loop immediately when a condition is met.
- Example:

```
A = [3, 5, 7, -2, 4];
i = 1;
while i <= length(A)
    if A(i) < 0
        disp(['Negative number found: ', num2str(A(i))]);
        break;  % Exit loop when a negative number is found
    end
    i = i + 1;
end
```

Review of Key Concepts

- mod Function:Useful for finding remainders, checking divisibility, and periodic checks.
- isempty Function: Helps check if arrays or variables are empty, which is useful for validation and preventing errors.
- Loop Structure: Use for to repeat a block of code.
- Step Sizes: Customize increments with start:step:end.
- Loop Control: Use break to exit early and continue to skip iterations.
- Loop Structure: Use while to repeat code while a condition is true.

Practice Exercise

- Task 1: Create a for loop to display the square of numbers from 1 to 10.
- Task 2: Create a nested while loop to print the multiplication table up to 5x5.
- Task 3: Using break, stop a loop once it finds a value greater than 50 in an array.

Let’s try MATLAB

Launch MATLAB and work towards the exercises