Date: 02/09/2024

Experiment 4

Selection Statements: Experiments on if statements, with else and elseif clauses and switch statements

Aim

i) **Grade Classification**

Write a MATLAB script that:

- Prompts the user to enter a score (0-100).
- Uses if-else and elseif statements to classify the score into grades (A, B, C, D, F) and displays the corresponding grade.

ii) **Quadratic equation**

- Prompts the user to enter the coefficients of the quadratic equation and calculate the roots

iii) **Switch Statement**

Write a MATLAB script that:

- Prompts the user to enter the choice (to calculate the area or perimeter)and the radius of the circle
- Uses a switch statement to perform the corresponding operation and displays the result.

Theoretical Background

Selection statements are fundamental constructs in programming languages that allow for decision-making. They enable a program to execute different sections of code based on specific conditions. MATLAB, a high-level programming language used extensively in engineering and scientific computing, includes several types of selection statements for controlling the flow of execution in scripts and functions.

Types of Selection Statements in MATLAB

MATLAB supports the following primary types of selection statements:

1. if Statements
2. switch Statements

if Statements

- The if statement is a fundamental control structure that executes a block of code if a specified condition is true. MATLAB provides a versatile if statement syntax that can handle multiple conditions through elseif clauses and an optional else clause.
- Syntax:

```
if condition
   % Code to execute if the condition is true
elseif another_condition
   % Code to execute if the another_condition is true
else
   % Code to execute if none of the above conditions are true
end
```

- Example:

```
x = 10;
if x > 0
   disp('x is positive');
elseif x < 0
   disp('x is negative');
else
   disp('x is zero');
end
```

switch Statements

- The switch statement provides a way to handle multiple discrete cases based on the value of a single expression. This is particularly useful when a variable can take one of several possible values, and you want to execute different code for each possible value.
- Syntax:

```
switch expression
   case value1
      % Code to execute if expression equals value1
   case value2
      % Code to execute if expression equals value2
   otherwise
      % Code to execute if expression does not match any case
end
```

- Example:

```
day = 'Monday';
switch day
   case 'Monday'
      disp('Start of the work week');
   case 'Friday'
      disp('End of the work week');
   otherwise
      disp('Midweek day');
end
```

Usage and Applications

- Conditional Execution: Selection statements are used to execute code conditionally based on the value of variables, user input, or other factors.
- Branching Logic: They allow for the implementation of complex branching logic where different paths in the code are executed based on varying conditions.
- Error Handling: if statements are often used to check for errors or invalid inputs and to handle exceptions gracefully.

Code

i)

```
canContinue = true;
while canContinue
   score = input("Enter Score: ");
   if(score>=90 && score<=100)
      disp("Grade: A");
   elseif(score>=80 && score<90)
      disp("Grade: B");
   elseif(score>=70 && score<80)
      disp("Grade: C");
   elseif(score>=60 && score<70)
      disp("Grade: D");
   elseif(score>=50 && score<60)
      disp("Grade: E");
   else
      disp("Grade: F");
   end
   loop = input("Do you want to add more scores?(Y/N) ","s");
   if(loop=="Y")
      canContinue = true;
   else
      canContinue = false;
   end
end
```

```
Enter Score: 95
Grade: A
Do you want to add more scores?(Y/N)
Y
Enter Score: 85
Grade: B
Do you want to add more scores?(Y/N)
Y
Enter Score: 75
Grade: C
Do you want to add more scores?(Y/N)
Y
Enter Score: 65
Grade: D
Do you want to add more scores?(Y/N)
Y
Enter Score: 55
Grade: E
```

```
Do you want to add more scores?(Y/N)
Y
Enter Score: 45
Grade: F
Do you want to add more scores?(Y/N)
Y
Enter Score: 25
Grade: F
Do you want to add more scores?(Y/N)
N
```

ii)

```
canContinue = true;
while canContinue
   a = input("Enter coefficient of x^2: ")
   b = input("Enter coefficient of x: ")
   c = input("Enter constant value: ")
   if(a==0)
      disp("The entered values do not
form a quadratic equation!")
   else
      fprintf('The given quadratic
equation is %dx^2 + %dx + %d\n', a, b,
c);
      d = b^2 - 4*a*c;
      if d > 0
         root1 = (-b + sqrt(d)) / (2*a);
         root2 = (-b - sqrt(d)) / (2*a);
         fprintf('The roots are real and
distinct: %.2f and %.2f\n', root1, root2);
      elseif d == 0
         root1 = -b / (2*a);
         fprintf('The root is real and
equal: %.2f\n', root1);
      else
         realPart = -b / (2*a);
         imagPart = sqrt(-d) / (2*a);
         fprintf('The roots are imaginary:
%.2f + %.2fi and %.2f - %.2fi\n',
realPart, imagPart, realPart, imagPart);
      end
   end
   loop = input("Do you want to
continue?(Y/N) ","s");
   if(loop=="Y")
      canContinue = true;
   else
   canContinue = false;
      end
```

```
Enter coefficient of x^2: 1
a =
    1
Enter coefficient of x: -5
b =
   -5
Enter constant value: 6
c =
    6
The given quadratic equation is 1x^2 +
-5x + 6
The roots are real and distinct: 3.00 and
2.00
Do you want to continue?(Y/N) Y
Enter coefficient of x^2: 1
a =
    1
Enter coefficient of x: -4
b =
   -4
Enter constant value: 4
c =
    4
The given quadratic equation is 1x^2 +
-4x + 4
The root is real and equal: 2.00
Do you want to continue?(Y/N) Y
Enter coefficient of x^2: 1
a =
    1
Enter coefficient of x: 1
b =
    1
Enter constant value: 1
c =
    1
The given quadratic equation is 1x^2 +
1x + 1
The roots are imaginary: -0.50 + 0.87i
and -0.50 - 0.87i
Do you want to continue?(Y/N) N
```

iii)

```
canContinue = true;
while canContinue
   r = input('Enter radius of circle: ');
   disp('Operation:');
```

```
   disp('1. Perimeter of circle');
   disp('2. Area of circle');
   choice = input('Enter your choice of
operation: ');
   switch choice
      case 1
         p = 2 * pi * r;
         fprintf('Perimeter of the circle is:
%.2f\n', p);
      case 2
         A = pi * r^2;

fprintf('Area of the circle is: %.2f\n', A);
      otherwise
         fprintf('Invalid choice. Please
enter 1 or 2.\n');
   end

loop = input('Do you want to continue?
(Y/N): ', 's');
   if strcmpi(loop, 'Y')
      canContinue = true;
 else
      canContinue = false;
   end
end
```

```
Enter radius of circle: 3
Operation:
1. Perimeter of circle
2. Area of circle
Enter your choice of operation: 1
Perimeter of the circle is: 18.85
Do you want to continue? (Y/N): Y
Enter radius of circle: 3
Operation:
1. Perimeter of circle
2. Area of circle
Enter your choice of operation: 2
Area of the circle is: 28.27
Do you want to continue? (Y/N): N
```

Conclusion

Selection statements in MATLAB provide essential control over program flow, enabling developers to write flexible and adaptive code. The if and switch statements offer powerful mechanisms for decision-making, making it possible to execute different code blocks based on variable values or conditions. Understanding and utilizing these constructs effectively is crucial for writing robust MATLAB programs.

Date: 02/09/2024

Experiment 5
Programs based on counted (for) and conditional (while) loops

Aim

i) **Summation using For Loop**

Write a MATLAB script that:

- Computes the sum of the first ‘n’ natural numbers using a for loop, where ‘n’ is provided by the user.
- Displays the result

ii) **Factorial Calculation using While Loop**

Write a MATLAB script that:

- Computes the factorial of number ‘n’ using a while loop, where ‘n’ is provided by the user.
- Displays the result.

iii) **Switch Statement**

Write a MATLAB script that:

- Prompts the user to enter the choice (to calculate the area or perimeter)and the radius of the circle
- Uses a switch statement to perform the corresponding operation and displays the result.

Theoretical Background

Loops are essential programming constructs that allow for the repeated execution of code blocks. They are fundamental for performing repetitive tasks, iterating over arrays or matrices, and automating processes. MATLAB, a high-level programming language primarily used for numerical computing, supports several types of loops to handle different iterative tasks.

Types of Loops in MATLAB

MATLAB supports the following primary loop constructs:

1. for Loop
    - The for loop in MATLAB is used to iterate over a range of values or elements in an array. It executes a block of code a specific number of times, with each iteration using a different value from the specified range.
    - Syntax:

```
for index = startValue:endValue
    % Code to execute in each iteration
end
```

index: The loop variable that takes on each value in the specified range.
startValue: The initial value of the loop variable.
endValue: The final value of the loop variable.

- Example:

```
for i = 1:5
    disp(['Iteration number: ', num2str(i)]);
end
```

This loop will display the iteration number from 1 to 5.

2. while Loop
    - The while loop in MATLAB executes a block of code as long as a specified condition remains true. It is more flexible than the for loop as it allows for conditions that are not necessarily based on a fixed range.
    - Syntax:

```
while condition
    % Code to execute as long as the condition is true
end
```

condition: A logical expression that determines whether the loop body should execute.

- Example:

```
count = 1;
while count <= 5
    disp(['Iteration number: ', num2str(count)]);
    count = count + 1; % Update the loop variable
end
```

This loop will also display the iteration number from 1 to 5, but it uses a condition to control the loop.

Practical Applications of Loops

- Repetitive Calculations: Loops are used to perform repetitive calculations or operations, such as summing elements or computing factorials.
- Iterating Over Arrays: Loops facilitate processing each element in arrays or matrices, allowing for operations like filtering or transformation.
- Simulations and Modeling: In simulations, loops can run through multiple iterations or time steps to model dynamic systems.

Performance Considerations

- Vectorization: In MATLAB, operations that can be vectorized (i.e., performed on entire arrays at once) are usually preferred over loops for performance reasons. MATLAB is optimized for matrix operations, and vectorized code is often more efficient.
- Loop Efficiency: When using loops, especially with large datasets, ensure that the code inside the loop is optimized, and consider preallocating arrays to improve performance.

Code

i)

```
n = input("Enter number: ");
sum = 0;
for i=1:n
sum = sum + i;
end
fprintf('Sum of the first %d numbers is
%d\n', n, sum);
```

```
Enter number:
8
Sum of the first 8 numbers is 36
```

ii)

```
n = input("Enter number: ");
factorial = 1;
if(n == 1 || n == 0)
   factorial = 1;
else
   for i=1:n
      factorial = factorial * i;
   end
end
fprintf('Factorial of the %d is %d\n', n,
factorial);
```

```
Enter number:
5
Factorial of the 5 is 120
```

iii)

```
v = [1 2 3 4 5 6 7 8 9 10]
squares = zeros(1, length(v));
for i = 1:length(v)
   squares(i) = v(i)^2;
end
disp("Square of each element in the
vector is: ");
disp(squares);
```

```
v =
    1    2    3    4    5    6    7    8    9
10

Square of each element in the vector is:
    1    4    9   16   25   36   49   64
81  100
```

Conclusion

Loops are a crucial component of MATLAB programming, enabling the execution of repetitive tasks and complex iterative processes. Understanding how to effectively use for and while loops, and knowing when to leverage vectorized operations, is essential for efficient and effective programming in MATLAB.

Date: 22/09/2024

Experiment 6
Graphs

Aim

Write a MATLAB script that:

- Creates a figure with two subplots.
- In the first subplot, plots a sine wave.
- In the second subplot, plots a cosine wave.
- Adds titles, labels, and legends to the plots.

Theoretical Background

Graphing is a vital technique in data analysis and visualization, allowing for the representation of mathematical functions, relationships between variables, and experimental results. In MATLAB, effective graphing enhances comprehension and facilitates decision-making by presenting complex data visually.

Fundamental Concepts

- Axes: Every graph consists of horizontal and vertical axes (x-axis and y-axis) that define the coordinate system. The intersection of these axes represents the origin (0,0).
- Plot Types: Various types of plots serve different purposes:
    - Line Plots: Used to depict continuous data over a range.
    - Scatter Plots: Useful for displaying the relationship between two variables.
    - Bar Graphs: Effective for comparing categorical data.
    - Histograms: Used to represent the distribution of numerical data.

Line Specifications

MATLAB allows customization of plot appearance through line specifications, which can include:

- Color: Different colors help distinguish between multiple datasets.
- Line Style: Options include solid lines, dashed lines, dotted lines, etc.
- Markers: Shapes like circles, squares, or stars can highlight specific data points, making the graph more informative.

Customization Features

Customization is crucial for enhancing clarity and aesthetic appeal in graphs:

- Axis Limits: Setting limits on the x-axis and y-axis focuses attention on relevant portions of the data, improving interpretability.
- Grid Lines: Adding grid lines aids in reading and estimating values from the graph, providing reference points.
- Legends: Legends identify different datasets or components within the graph, facilitating quick understanding of the presented information.
- Labels: Proper labeling of axes and providing titles enrich the context, ensuring the viewer comprehends what the graph represents.

Subplots

Subplots allow multiple graphs to be displayed within a single figure, which is particularly useful for comparing different datasets or functions. Each subplot can have its own axes, titles, labels, and legends, providing a clear, organized view of multiple related graphs in one visual space.

Practical Applications

Graphing in MATLAB is widely applicable in various fields:

- Engineering: To visualize signals, waveforms, and system responses.
- Physics: For analyzing motion, forces, and other phenomena.
- Data Science: To explore datasets, trends, and correlations visually.
- Education: As a teaching tool to demonstrate mathematical concepts and relationships.

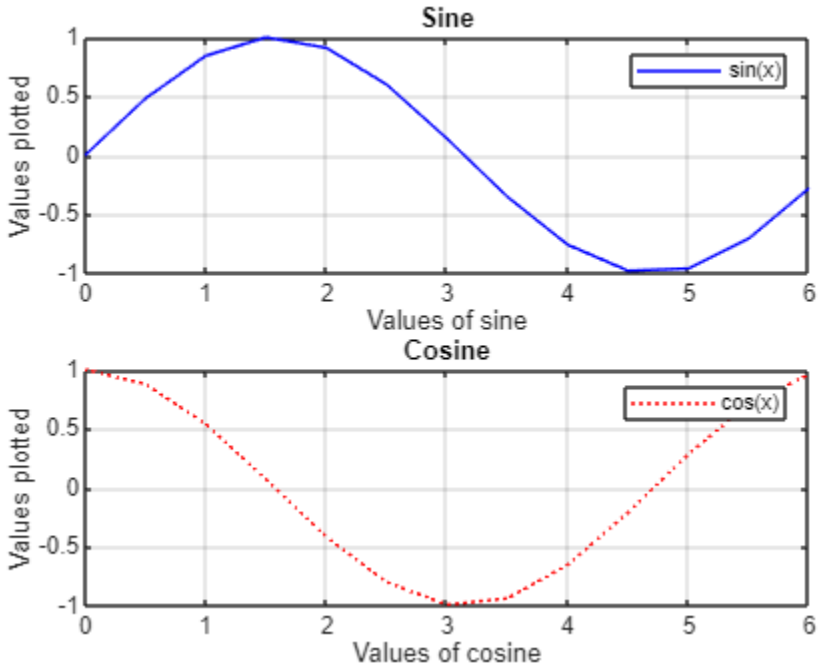
Code

```
x = 0:0.5:2*pi
y = sin(x)
z = cos(x)

subplot(2,1,1)
plot(x, y, 'b-')
title("Sine")
xlabel("Values of sine")
legend('sin(x)');
ylabel("Values plotted")
grid on

subplot(2,1,2)
plot(x, z, 'r:')
title("Cosine")
xlabel("Values of cosine")
legend('cos(x)');
ylabel("Values plotted")
grid on
```

Output



Conclusion

Understanding the theoretical aspects of graphing in MATLAB equips users with the tools needed to create effective visual representations of data. The ability to customize and manipulate graphical output enhances the interpretability and presentation of information, making it an essential skill in research, analysis, and education.