CaYPT
Canadian Young Physicists' Tournament

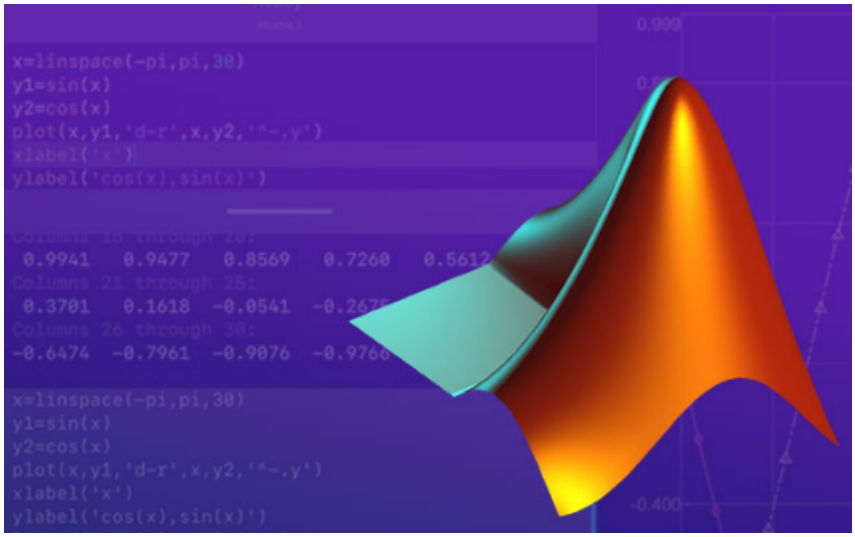
MATLAB PROGRAMMING BASICS

Author: Javier Huang

Date Created: 06/10/23
Version 1.0

Abstract

In this SOP, the principle of basic MATLAB programming is documented.



Contents

List of Figures

1 MATLAB Basics[1]

1.1 Data Types and Variables

1. Data Types:
   - Numeric Types: Integers (`int`), floating-point numbers (`float`)

     Numeric types in MATLAB do not require explicit declaration of the type beforehand. You can assign values directly to variables, and the MATLAB interpreter will automatically recognize the appropriate numeric type based on the assigned value. Numeric types include integers (whole numbers) and floating-point numbers (decimal numbers).

     Example:

     ```
     intNumber = 10;         % Integer
     floatNumber = 3.14;     % Floating-point number
     ```

   - Characters: Single characters (`char`)

     Characters in MATLAB are enclosed in single quotes (`'`). You can assign a single character to a variable. Characters are often used to represent individual symbols, letters, or digits.

     Example:

     ```
     charValue = 'A';        % Character
     ```

   - Strings: Sequence of characters (`string`)

     Strings in MATLAB are enclosed in double quotes (`"`). You can assign a sequence of characters to a variable. Strings are commonly used to represent text or a collection of characters.

     Example:

     ```
     stringValue = "Hello";  % String
     ```

   - Logical: Boolean values (`true`, `false`)

     Logical types in MATLAB represent boolean values, where `true` and `false` indicate the logical states. Logical values are used in conditional statements and logical operations.

     Example:

     ```
     logicalValue = true;    % Logical value
     ```

2. Variables:
   - Variable Declaration and Naming: Assigning values to variables

     In MATLAB, variables are assigned values using the assignment operator (=). The variable name is chosen by the programmer and can contain letters, numbers, and underscores. MATLAB is case-sensitive, so variable names in different cases are treated as different variables.

     Example:

     ```
     variableName = value;
     ```

1.2 Control Flow and Loops

1. Control Flow:
   - Conditional Statements: `if`, `else`, `elseif`

     Conditional statements allow you to control the flow of your program based on certain conditions. The `if` statement is used to execute a block of code if a condition is true. The `else` statement is optional and allows you to specify a block of code to execute if the condition is false. The `elseif` statement allows you to check additional conditions.

     Example:

     ```
     if condition
         statement;
     elseif condition
         statement;
     else
         statement;
     end
     ```

2. Loops:
   - For Loop: Iterate a fixed number of times

     The `for` loop allows you to iterate a fixed number of times. You can specify the loop variable, the initial value, the increment, and the final value. The loop variable takes on each value in the specified range, and the statements inside the loop are executed for each iteration.
     The `start:increment:end` operator is called the colon operator and it generates values from the start to the end with increments specified.

     Example:

     ```
     for variable = start:increment:end
         statement;
     end
     ```

   - While Loop: Execute as long as a condition is true

     The `while` loop allows you to execute a block of code as long as a condition is true. The condition is checked at the beginning of each iteration. If the condition is true, the statements inside the loop are executed. The loop continues until the condition becomes false.

     Example:

     ```
     while condition
     statement;
     end
     ```

1.3 Functions and Modularity

1. Functions:
   - Function Definition: Create reusable blocks of code

Functions in MATLAB allow you to create modular and reusable blocks of code. A function is defined using the `function` keyword, followed by the function name, input arguments, and output arguments. The function body contains the statements to be executed, and the output values are assigned to the output arguments.

Example:

```
function output = functionName(input)
    statements;
    output = value;
end
```

- Function Call: Use the defined function

  To use a function, you can call it by its name and provide the necessary input arguments. The function will execute its code block and return the output values, which can be stored in variables for further use.

  Example:

  ```
  result = functionName(input);
  ```

2. Built-in Math Functions:

   Math functions in MATLAB performs various mathematical operations on numeric data. These functions provide a wide range of capabilities for calculations involving absolute values, square roots, trigonometric operations, logarithms, and more[1].

   Example:

   ```
   x = -5;
   abs_x = abs(x);             % Absolute value of x: 5

   y = 9;
   sqrt_y = sqrt(y);           % Square root of y: 3

   angle_deg = 45;
   angle_rad = deg2rad(angle_deg);     % Convert angle from degrees to radians

   sin_angle = sin(angle_rad);     % Sine of angle: 0.7071

   value = 1000;
   log_value = log10(value);   % Base-10 logarithm of value: 3
   ```

   The `abs()` function calculates the absolute value of `x`, the `sqrt()` function computes the square root of `y`, and the `log10()` function calculates the base-10 logarithm of `value`. The `deg2rad()` function is used to convert the `angle_deg` variable from degrees to radians before computing the sine using the `sin()` function.

1.4 Arrays and Matrices

1. Arrays:

   - Creating Arrays: Assigning values to arrays

     In MATLAB, arrays are used to store multiple values of the same type. You can create arrays using square brackets (`[ ]`) or by using built-in functions like `zeros`, `ones`, or `rand`.

     Example:

```
array = [1, 2, 3, 4, 5];          % Array with explicit values

zerosArray = zeros(1, 5);         % Array of zeros
onesArray = ones(1, 5);           % Array of ones
randomArray = rand(1, 5);         % Array of random numbers
```

- Accessing Array Elements: Retrieving values from arrays

  You can access individual elements of an array using indexing. MATLAB uses one-based indexing, where the first element is at index 1.

  Example:

  ```
  element = array(index);            % Accessing a specific element
  ```

2. Matrices:
   - Creating Matrices: Assigning values to matrices

     Matrices are two-dimensional arrays that store multiple values in rows and columns. You can create matrices using square brackets (`[ ]`) or by using built-in functions like `zeros`, `ones`, or `rand`.

     Example:

     ```
     matrix = [1, 2, 3; 4, 5, 6];            % Matrix with explicit values

     zerosMatrix = zeros(2, 3);              % Matrix of zeros
     onesMatrix = ones(2, 3);                % Matrix of ones
     randomMatrix = rand(2, 3);              % Matrix of random numbers
     ```

   - Accessing Matrix Elements: Retrieving values from matrices

     You can access individual elements of a matrix using indexing. The indexing format is `matrix(row, column)`. MATLAB uses one-based indexing.

     Example:

     ```
     element = matrix(row, column);          % Accessing a specific element
     ```

1.5 Console Input and Output

1. Input:
   - Reading User Input: Prompting for user input in the console

     You can use the `input` function to prompt the user for input during program execution. The user's input is captured as a string, and you can convert it to the appropriate data type if needed.

     Example:

     ```
     userInput = input('Enter a value: ');      % Prompt for user input
     ```

2. Output:
   - Displaying Output: Printing values to the console

     You can use the `disp` function to display output in the console. The function accepts multiple arguments and displays them as separate lines of output.

     Example:

     ```
     disp('Hello, world!');                     % Display a message
     disp(variable);                            % Display a variable's value
     ```

1.6 File Input and Output

1. Reading from Files:
   - Reading Text Files: Loading text data from files

     You can use the `fread`, `fscanf`, or `readtable` functions to read data from text files. These functions allow you to read data in various formats, such as numeric values, strings, or tables.

     Example:

     ```
     data = fread(fileID, size, precision);      % Read numeric data from a file
     data = fscanf(fileID, formatSpec);          % Read formatted data from a file
     data = readtable('data.txt');               % Read data into a table
     ```

2. Writing to Files:
   - Writing Text Files: Saving data to text files

     You can use the `fwrite`, `fprintf`, or `writetable` functions to write data to text files. These functions allow you to write data in various formats, such as numeric values, strings, or tables.

     Example:

     ```
     fwrite(fileID, data, precision);            % Write numeric data to a file
     fprintf(fileID, formatSpec, data);          % Write formatted data to a file
     writetable(data, 'output.txt');             % Write data from a table to a file
     ```

1.7 Plotting and Visualization

1. Basic Plots:
   - Line Plot: Plotting data as a line graph

     You can use the `plot` function to create line plots. The function accepts x and y values as input and generates a line graph based on the data.

     Example:

     ```
     x = [1, 2, 3, 4, 5];
     y = [10, 15, 7, 9, 12];
     plot(x, y);
     ```

   - Scatter Plot: Plotting data as individual points

     The `scatter` function allows you to create scatter plots. It takes x and y values as input and generates a plot with individual points representing the data.

     Example:

     ```
     x = [1, 2, 3, 4, 5];
     y = [10, 15, 7, 9, 12];
     scatter(x, y);
     ```

2. Customizing Plots:
   - Adding Labels and Titles: Providing information for the plot

     You can use functions like `xlabel`, `ylabel`, and `title` to add labels and titles to your plots. These functions allow you to provide additional information about the data being plotted.

     Example:

```
xlabel('X-axis');
ylabel('Y-axis');
title('Plot Title');
```

- Adding Legends: Differentiating multiple data series

  If you have multiple data series in a plot, you can use the `legend` function to add a legend that differentiates the series. The legend provides a visual representation of the data and helps identify each series.

  Example:

  ```
  legend('Series 1', 'Series 2');
  ```

2 MATLAB Code Examples

2.1 Physics Programming Examples

1. **Calculating Projectile Motion**

   This example calculates the maximum height and range of a projectile launched at an angle, considering the effects of gravity. It utilizes variables, loops, conditions, and basic physics equations.

   ```
   % Prompt the user for input in the console
   v0 = input('Enter the initial velocity (m/s): ');
   theta = input('Enter the launch angle (degrees): ');
   g = 9.8;  % acceleration due to gravity (m/s^2)

   % Convert the angle from degrees to radians
   theta_rad = deg2rad(theta);

   % Calculate the time of flight
   t_flight = (2 * v0 * sin(theta_rad)) / g;

   % Calculate the maximum height
   h_max = (v0^2 * sin(theta_rad)^2) / (2 * g);

   % Calculate the horizontal range
   range = v0^2 * sin(2 * theta_rad) / g;

   % Display the results
   disp(['Time of flight: ', num2str(t_flight), ' seconds']);
   disp(['Maximum height: ', num2str(h_max), ' meters']);
   disp(['Horizontal range: ', num2str(range), ' meters']);
   ```

   This program prompts the user to enter the initial velocity (`v0`) and launch angle (`theta`) of a projectile. It then calculates the time of flight, maximum height, and horizontal range using the provided inputs and the equations of projectile motion. The calculated values are displayed using the `disp` function.

2. **Calculating Electric Field Intensity**

   This example calculates the electric field intensity at a point due to a point charge. It uses variables, conditions, and basic physics equations related to Coulomb's law.

```
% Prompt the user for input in the console
k = 9e9;  % Coulomb's constant (Nm^2/C^2)
q = input('Enter the charge (C): ');
r = input('Enter the distance from the charge (m): ');

% Calculate the electric field intensity
E = k * abs(q) / r^2;

% Display the result
disp(['Electric Field Intensity: ', num2str(E), ' N/C']);
```

This program prompts the user to enter the charge (`q`) and distance (`r`) from the charge to the point where the electric field intensity is to be calculated. It then applies Coulomb's law equation to calculate the electric field intensity (`E`). The calculated value is displayed using the `disp` function.

2.2 MATLAB Plotting Example

1. **Plotting a Sine Function**

   This example demonstrates how to plot a sine function using MATLAB's plotting capabilities.

   ```
   % Generate x values from 0 to 2*pi
   x = 0:0.01:2*pi;

   % Calculate y values as the sine of x
   y = sin(x);

   % Plot the sine function
   plot(x, y);

   % Add labels and title
   xlabel('x');
   ylabel('sin(x)');
   title('Plot of the Sine Function');
   ```

   This program generates a set of x-values ranging from 0 to 2π using the colon operator (`0:0.01:2*pi`). It then calculates the corresponding y-values as the sine of the x-values (`y = sin(x)`). Finally, it plots the sine function by calling the `plot` function with the x and y values as arguments. The `xlabel`, `ylabel`, and `title` functions are used to add labels and a title to the plot. The output is shown in Fig. 1

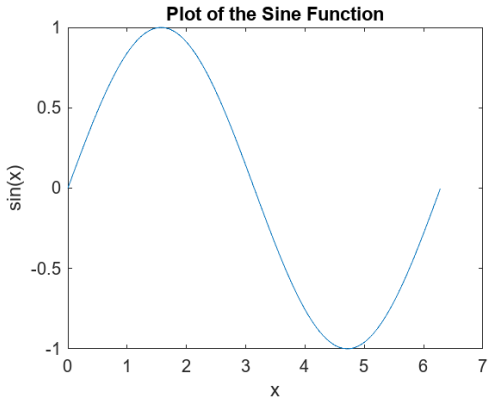


Figure 1: Sine Function Plot

REFERENCES

[1] Programming scripts, functions, and classes. https://www.mathworks.com/help/matlab/programming-and-data-types.html.