Computing and Software Engineering

GET211

September 23, 2025

Components of a Program

We provide a conceptual walkthrough of various aspects specifically on MATLAB, and guidance on how MATLAB program components and workflows fit together. This structure can help you understand MATLAB programming principles from foundational elements to advanced concepts.

MATLAB Programming Fundamentals

We present the core components that form the foundation of MATLAB programming, organized by their role in program development and execution.

Program Structure: The building blocks for organizing MATLAB code

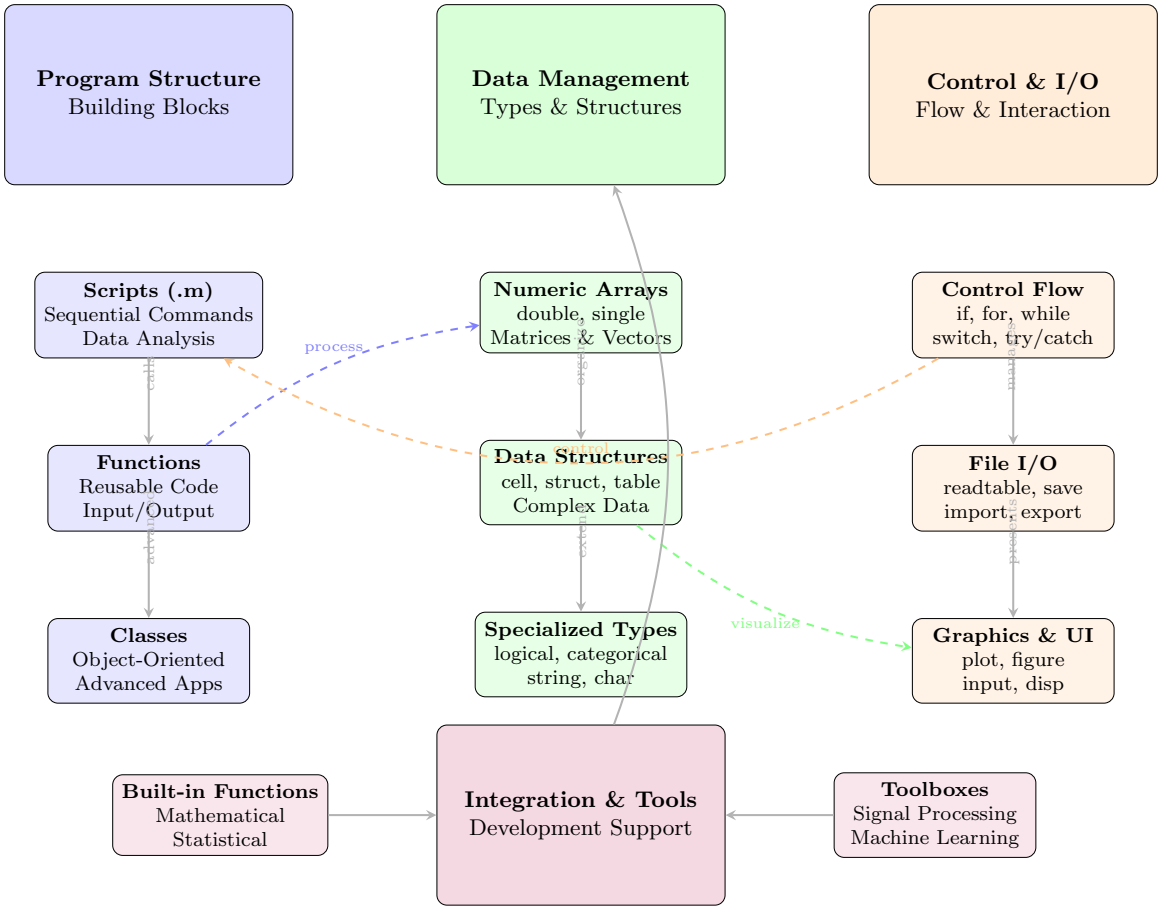
- **Scripts**: Sequence of commands saved in `.m` files for reproducible analysis
- **Functions**: Reusable code blocks with inputs/outputs (`function [y] = myFunc(x)`)
- **Classes**: Object-oriented programming structures for complex applications

Data Management: Core data types and structures for scientific computing

- **Arrays**: Numeric matrices (`double`, `single`), the foundation of MATLAB
- **Data Structures**: Cell arrays (`cell`), tables (`table`), and structures (`struct`)
- **Specialized Types**: Logical arrays, categorical data, and strings/character arrays

Program Control & I/O: Tools for program flow and external interaction

MATLAB Programming Fundamentals



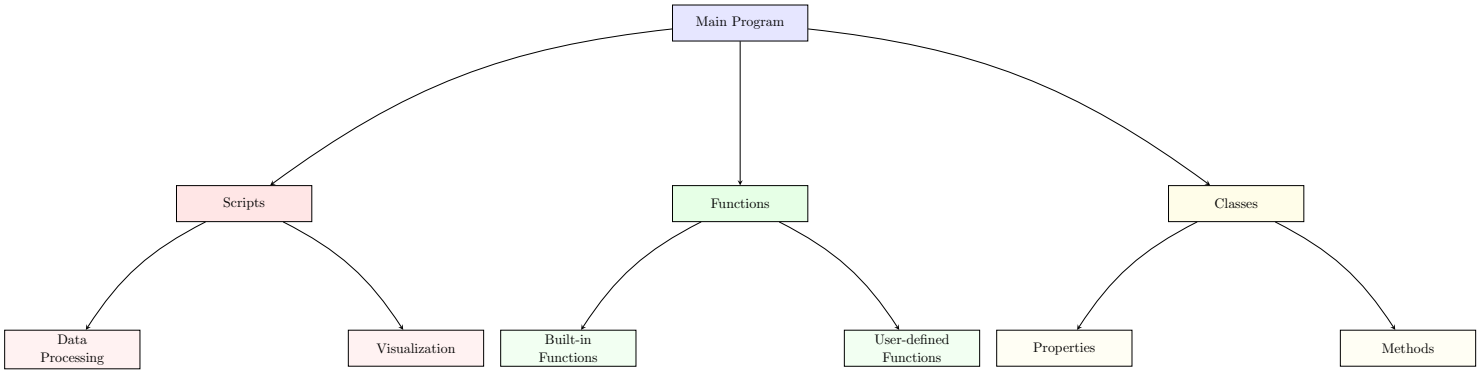
Program Hierarchy and Data Flow

We illustrate the hierarchical structure of MATLAB programs and how data flows between components. MATLAB programs consist of three main types: Scripts, Functions, and Classes, each serving specific purposes:

- Scripts execute sequences of commands and are commonly used for data processing and visualization tasks
- Functions can be either Built-in or User-defined, providing reusable code blocks.
- Classes are organized into Properties and Methods, encapsulating data and behaviors.

The data flow arrows indicate how these components interact, showing a coordinated structure where higher-level programs can call and utilize lower-level components to perform specific operations.

Program Hierarchy and Data Flow



MATLAB Data Flow and System Architecture

We illustrate how data flows through MATLAB programs and how core system components interact during program execution.

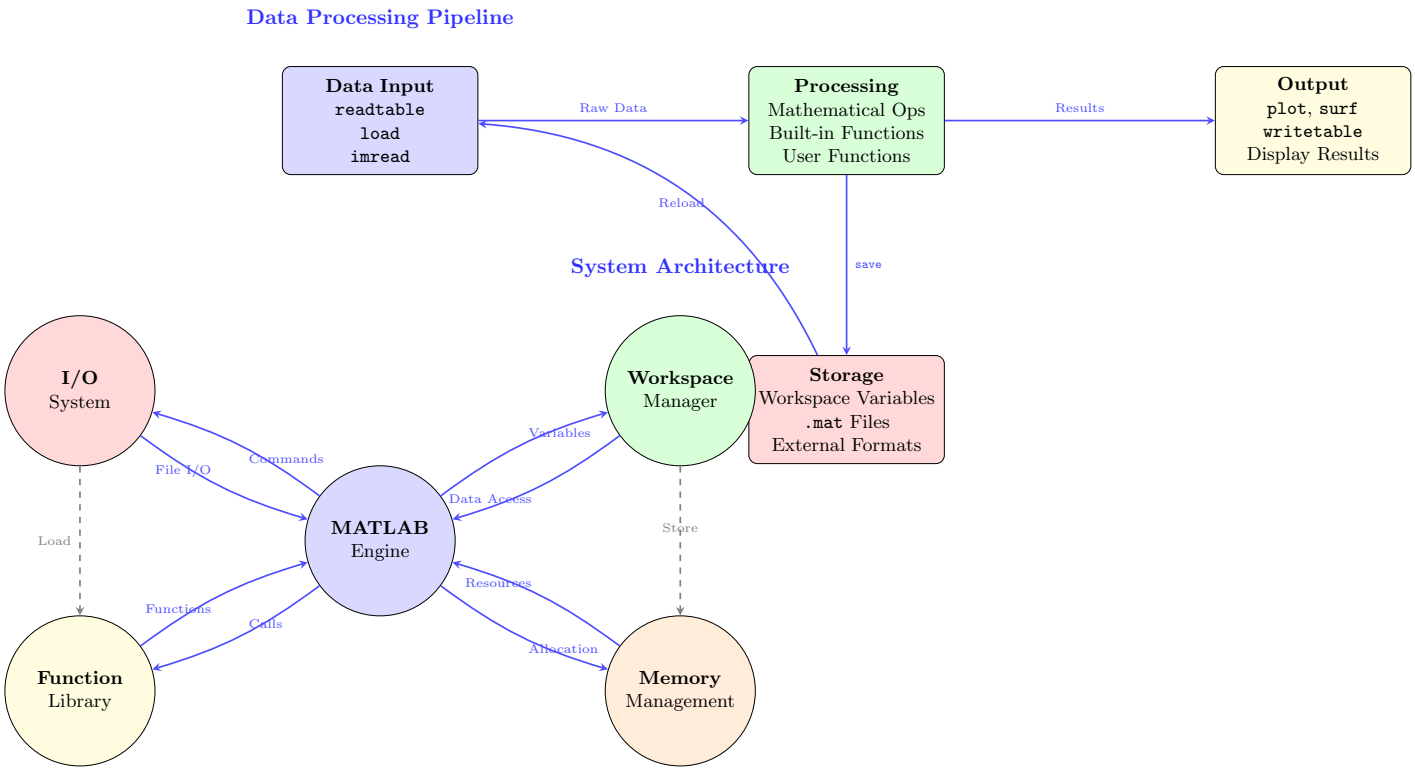
Data Processing Pipeline:

- **Data Input**: Import functions (`readtable`, `load`, `imread`) bring data into the workspace
- **Processing**: Mathematical operations, built-in functions, and user-defined functions transform the data
- **Output**: Visualization (`plot`, `surf`) and export functions (`writetable`, `save`) present results
- **Storage**: Variables and data persist in workspace or files for reuse in subsequent operations

System Architecture:

- **MATLAB Engine**: Coordinates execution and manages the computational workflow
- **Workspace Manager**: Handles variable storage and memory management
- **Function Library**: Provides access to built-in functions and loaded toolboxes
- **I/O System**: Manages file operations and external data connections

MATLAB Data Flow and System Architecture



MATLAB Development and Execution Environment

We present the complete MATLAB development workflow, from initial coding to program execution, emphasizing the iterative nature of MATLAB programming and the integration between development tools and runtime environment.

Development Environment:

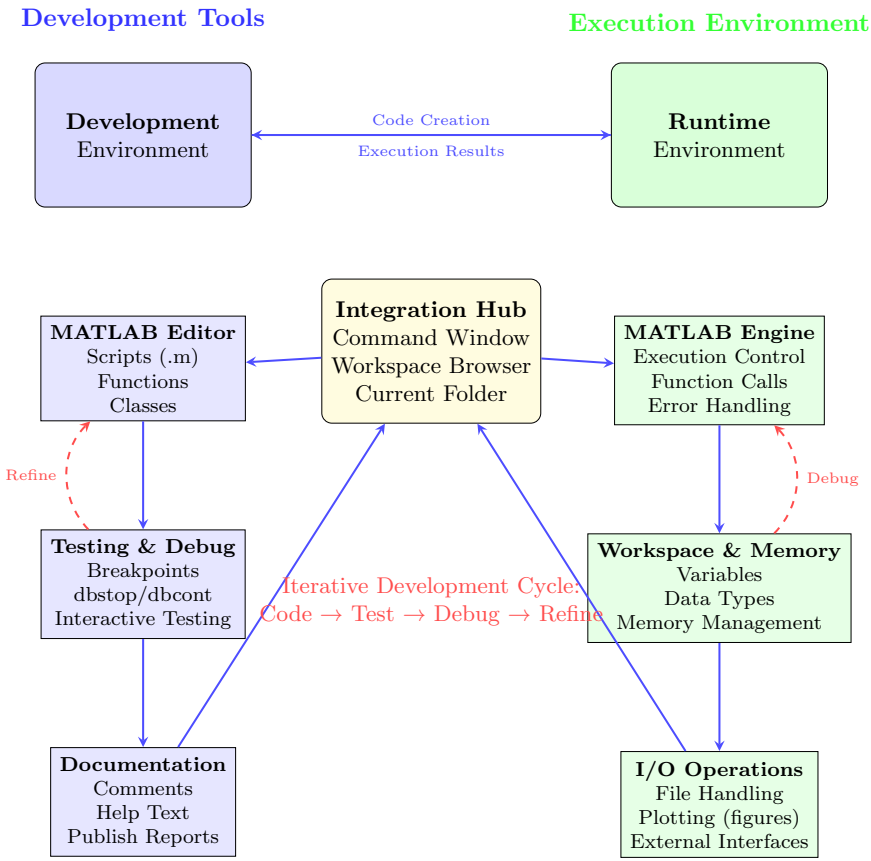
- **Code Creation**: Writing scripts (`.m`), functions, and classes using the MATLAB Editor
- **Testing & Debugging**: Using breakpoints, `dbstop`, and the Command Window for interactive testing
- **Documentation**: Adding comments, function help text, and publishing reports with `publish`

Runtime Environment:

- **Execution Control**: MATLAB Engine manages program flow and function calls
- **Memory & Data**: Workspace variables, memory allocation, and data type management
- **I/O Operations**: File handling, plotting, and external interface connections

Development Cycle: MATLAB development is iterative—code, test, debug, refine—with seamless transitions between the Editor, Command Window, and Workspace.

MATLAB Development and Execution Environment



Command Window Input

Inputs are essential for interactive programming, enabling users to provide data that influences the execution of scripts and functions.

User Input

User input can be captured interactively from the command line using the `input` function. This allows users to provide numerical or string values while the program is running.

Numerical Input To prompt the user for a number, you can use the `input` function without any additional parameters.

```
num = input('Enter a number: ');  % User enters a
    number
```

String Input To prompt for a string, you can specify the second argument as `'s'`.

```
name = input('Enter your name: ', 's');  % User
    enters a string
```

Command Window Output

Outputs in MATLAB refer to the results produced by scripts and functions after executing code. This section provides an overview of various methods for displaying outputs, including command window outputs, returning values from functions, and exporting data to files.

Displaying Outputs

The most common way to display outputs in MATLAB is through the command window. You can use several functions to show results to the user.

Using `disp`

The `disp` function displays a message or variable value without printing the variable name.

```
result = 10 * 5;   % Perform a calculation
disp(result);      % Display the result
```

Command Window Output

Using `fprintf`

The `fprintf` function provides formatted output, allowing you to control the appearance of the output text.

```
name = 'Alice';
age = 30;
fprintf('Name: %s, Age: %d\n', name, age);  % Formatted output
```

Using `sprintf`

The `sprintf` function formats data into a string without displaying it immediately. This can be useful for constructing messages that will be displayed later.

```
outputString = sprintf('The result is: %.2f', result);  % Format as string
disp(outputString);
    % Display the formatted string
```

Escape Characters: Special characters such as $\backslash n$ (newline) and $\backslash t$ (tab) can be used to format text output.

Exporting Data

MATLAB allows you to export data to various file formats, making it easier to share results or save them for later use.

Writing to Text Files

You can write data to text files using the `writetable` or `writematrix` functions.

```
data = [1, 2, 3; 4, 5, 6];
writematrix(data, 'outputData.csv');  % Write matrix to CSV file
```

Writing to Excel Files

MATLAB can also write data directly to Excel files using the `writetable` function.

```
T = table([1; 2], [3; 4], 'VariableNames', {'Column1', 'Column2'});
writetable(T, 'outputData.xlsx');  % Write table to Excel file
```

Importing Data

MATLAB can read data from external files, which is useful for processing datasets or configurations stored outside the workspace.

Reading Text Files

You can use various functions to read data from text files, such as `readtable`, `readmatrix`, and `readcell`.

Using `readtable` This function reads data from a text file into a table.

```
data = readtable('data.csv');  % Read data from a CSV file
```

Using `readmatrix` This function reads numerical data into a matrix.

```
matrixData = readmatrix('data.csv');  % Read numerical data from a CSV file
```

Text Files

Writing Text Files

- Use `fprintf` for writing.

```
fid = fopen('output.txt', 'w');
fprintf(fid, 'This is a test.\n');
fclose(fid);
```

‘fopen‘ opens a file, ‘fgets‘ reads a line at a time, and ‘fscanf‘ reads formatted data.

- Use `fopen`, `fgets`, `fscanf` for reading.

```
fid = fopen('output.txt', 'r');
line = fgets(fid);
disp(line);
fclose(fid);
```

Binary File I/O

MATLAB supports reading and writing binary files using ‘fopen‘, ‘fread‘, ‘fwrite‘, and ‘fclose‘.

Writing Binary Files

Binary files store data in compact binary format, which can be efficient for large datasets.

```
fid = fopen('data.bin', 'w');
fwrite(fid, magic(3), 'double');
fclose(fid);
```

Reading Binary Files

You can read binary data from files using `fopen` and `fread`.

```
fileID = fopen('data.bin', 'r');    % Open binary file
    for reading
binaryData = fread(fileID);          % Read binary data
fclose(fileID);                      % Close the file
```

CSV Files

CSV (Comma-Separated Values) files are a common format for storing tabular data.

- The `csvwrite()` function writes numeric data to a CSV file.

```
% Create a 3x3 magic square matrix
M = magic(3);

% Write the matrix to a CSV file
csvwrite('magic.csv', M);
```

- The `csvread()` function reads numeric data from a CSV file.

```
% Read data from the CSV file
N = csvread('magic.csv');

% Display the data
disp(N);
```

Image File

MATLAB provides functions for importing and exporting image files in various formats (e.g., JPEG, PNG, BMP).

- The `imread()` function reads image data from a file.

```
% Read an image file
img = imread('myimage.jpg');

% Display the image
imshow(img);
```

- The `imwrite()` function writes image data to a file.

```
% Save the image to a new file in PNG format
imwrite(img, 'newimage.png');
```

MAT-File I/O

MATLAB's native file format ('.mat') stores variables in binary format and is efficient for saving and loading MATLAB variables.

Saving Workspace Variables

You can save all workspace variables to a MAT-file for later use using the `save` function.

```
save('myWorkspace.mat');  % Save all variables to a .mat file
```

You can also save particular variables

```
x = 1:10;
y = x.^2;
save('mydata.mat', 'x', 'y');
```

- Loading Data:

```
    load('myData.mat');
```

Exercise

Write a MATLAB script that prompts the user to enter their name and age.
- Use the ‘input‘ function to get this information.
- Display a greeting message that includes the user’s name and age using the ‘disp‘ function.

Exercise

Create a script that asks the user to input a temperature in Celsius and converts it to Fahrenheit.
- Use the formula: $F = \frac{9}{5} \times C + 32$.
- Display both the Celsius and Fahrenheit values in a formatted message.

Exercise

Write a script that prompts the user for the radius of a circle and calculates its area.
- Use the formula: Area $= \pi \times r^2$.
- Output the area in a formatted string.

Exercise

Ask the user to enter three numbers.
- Calculate their average and display the result using the ‘fprintf‘ function to format the output.

Exercise

Create a script that prompts the user to enter five test scores.

- Store these scores in a numeric array and write the array to a CSV file named test_scores.csv.
- Display a message indicating the file has been created.

Exercise

Write a simple quiz script that asks the user a question (e.g., "What is the capital of France?") and provides multiple-choice answers.
Capture the user's response using the 'input' function and display it.

Exercise

Create a script that asks the user for their scores in three subjects. Calculate the average score and display the score.