Computing and Software Engineering

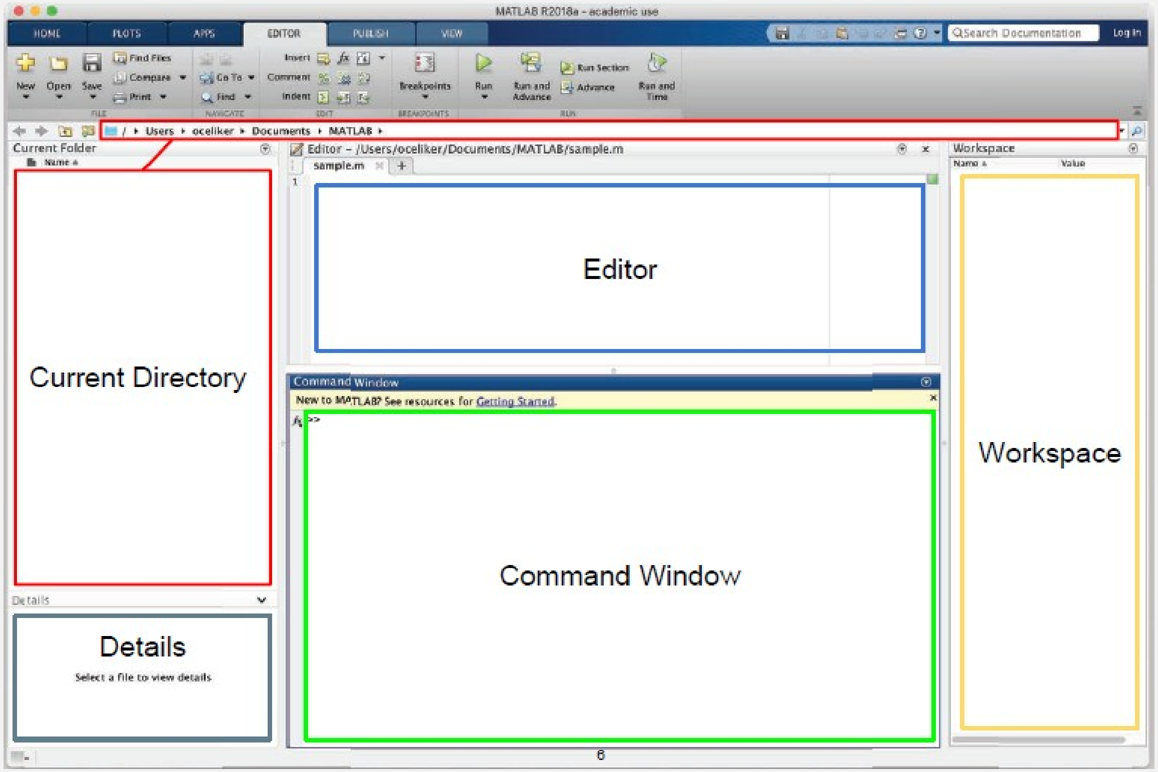
GET211

September 23, 2025

Outline

MATLAB

MATLAB (MATrix LABoratory) is a high-level programming language and numerical computing environment developed by MathWorks. It's widely used in various fields including engineering, scientific computing, signal processing, image processing, control systems, and more.


MATLAB R2018a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Log In
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
/ ▸ Users ▸ oceliker ▸ Documents ▸ MATLAB ▸
Current Folder
Name
Editor - /Users/oceliker/Documents/MATLAB/sample.m
sample.m
Workspace
Name
Value
Current Directory
Editor
Command Window
New to MATLAB? See resources for Getting Started.
>>
Command Window
Workspace
Details
Details
Select a file to view details

MATLAB environment and interface

Desktop Layout

- The MATLAB desktop is highly customizable, allowing users to arrange various tools and windows according to their preferences.
- It typically includes the Command Window, Workspace, Current Folder, and Editor, among other tools.

Command Window

- This is the primary interface for entering commands and executing MATLAB code interactively.
- It displays the MATLAB prompt (>>) where users can type commands directly.
- Results of computations are immediately displayed here.
- It also shows error messages and warnings.

MATLAB environment and interface

Workspace Browser

- Displays all variables currently in memory, along with their size, type, and other properties.
- Allows users to view and manipulate variable contents.
- Provides a quick way to import, export, or clear variables.

Current Folder Browser

- Shows the contents of the current working directory.
- Allows navigation through the file system.
- Provides quick access to MATLAB scripts, functions, and data files.

Editor

- A full-featured text editor for creating and modifying MATLAB scripts (.m files) and functions.
- Includes syntax highlighting, code folding, and auto-completion features.
- Integrated debugging tools for finding and fixing errors in code.

MATLAB environment and interface

Figure Window

- Displays graphical output from plotting commands.
- Includes tools for manipulating and annotating plots.

Help Browser

- Provides comprehensive documentation for MATLAB functions and features.
- Includes examples, syntax information, and related functions.

Toolstrip

- A ribbon-like interface at the top of the MATLAB desktop.
- Provides quick access to commonly used functions and tools.

Command History

- Keeps a record of previously executed commands.
- Allows easy reuse of past commands.

Variables

A variable is a storage location identified by a name (or identifier) that holds a value, which can be of various types (e.g., numbers, strings, arrays). In essence, variables provide a way to label and store data that can be retrieved, modified, and used throughout a program.
Unlike some programming languages, MATLAB is dynamically typed, meaning you don't need to declare the type of a variable explicitly. The type is determined automatically when you assign a value to the variable.

Characteristics of Variables:
- Named Storage Location: Each variable has a unique name (identifier), which is used to access the stored value.
- Data Type: Variables can store different types of data (integers, floating-point numbers, characters, etc.).
- Mutable: Variables can change value over time during the execution of a program.
- Scope: Variables have a defined region (scope) where they can be accessed. This can be local (within a function) or global (across the entire program).
- Lifetime: A variable's lifetime refers to the period during which the variable exists in memory (from its creation to its destruction).

Variable Declaration and Assignment

In MATLAB, you create a variable simply by assigning a value to it:

```
x = 5;
name = 'Alice';
```

MATLAB uses the = operator for assignment. The variable name is on the left, and the value is on the right.

Variable Naming Conventions

Variable naming conventions are a set of rules and guidelines followed by programmers to name variables in a readable, maintainable, and error-free manner. Consistent naming conventions make the code more understandable, especially when collaborating with others or when revisiting code after some time.
The following are the general rules for variable naming:
- **Start with a Letter:** Variable names must begin with a letter (upper or lower case) or an underscore (_), followed by letters, numbers, or underscores.

```
myVariable, _variable1    % Correct naming
1variable, #var           % Incorrect naming
```

- **No Spaces:** Spaces are not allowed in variable names. Instead, programmers use underscores (‘‘) or camelCase (more on this later).

```
my_variable, myVariable % Correct naming
my variable             %% Incorrect naming
```

- **Case Sensitivity:** Variable names are often case-sensitive, meaning $myVar$ and $myvar$ are considered two different variables.

```
  x = 10;  % Assigns 10 to variable x
  X = 20;  % Assigns 20 to a different variable X
```

- **No Reserved Words:** Variable names should not use keywords or reserved words in the programming language, such as 'if', 'for', or 'while'.

```
    % Reserved words such as 'while' cannot be used
        as variable names
    whileVar = 10; % Correct
```

- **Meaningful Names:** Variables should have descriptive names that indicate the role or value they represent. Avoid single-letter names (except in loop counters or short, local variables).

```
    radius = 5;                    % Good variable name
    r = 5;                         % Not descriptive
```

Use Descriptive Names

- Variables should have names that clearly describe what they represent. This makes the code more readable and maintainable.

Good	**Bad**

`principalAmount = 1000;`
`interestRate = 0.05;`
	`a = 1000;`
	`b = 0.05;`

Avoid Single-letter Variable Names

Single-letter names like ‘x‘, ‘y‘, or ‘i‘ should be avoided unless used in very simple loops or for common mathematical concepts.

Bad	Good
`i = 10;`	`index = 10;`

Short but Descriptive for Loop Variables

In loops or short scopes, it’s common to use short variable names like ‘i‘, ‘j‘, ‘k‘ for counters or indices.

```
for i = 1:10
    disp(i);
end
```

Use Consistent Naming Conventions

Stick to one naming convention throughout the program for consistency. Avoid mixing camelCase and snake_case in the same program.

Consistent	Inconsistent

`numStudents = 100;`
`totalScore = 85;`
	`num_students = 100;`
	`TotalScore = 85;`

Common Naming Conventions

Different programming languages and communities have specific conventions for naming variables. While the specific rules vary, there are some widely used naming styles:
Camel Case (camelCase): In camelCase, the first word is lowercase, and subsequent words are capitalized with no spaces between them. Often used in variable names.

```
myVariableName
totalSum
```

Pascal Case (PascalCase): Similar to camelCase, but the first letter of every word (including the first word) is capitalized. Often used for naming classes

```
MyVariableName
TotalSum
```

Snake Case (snake_case): In snake_case, all letters are lowercase, and words are separated by underscores. Often used in variable names.

```
my_variable_name
total_sum
```

Uppercase with Underscores (UPPER_CASE:) All letters are uppercase, and words are separated by underscores. This convention is often used for constants.

```
MAX_VALUE
PI_CONSTANT
```

Special Variables in MATLAB

Variable	**Description**	**Example**
`ans`	The default variable where the result of an operation is stored if no other variable is specified. Stores the result of the most recent calculation.	`2 + 2;` `disp(ans);` % Displays 4
`pi`	A predefined constant representing the value of (3.1416).	`area = pi * r^2;`
`i` / `j`	Represent the imaginary unit in complex numbers.	`z = 3 + 4i;` % Complex number 3 + 4i
`inf`	Represents infinity. Used for values too large to represent.	`x = 1 / 0;` % Results in inf
`NaN`	Represents "Not-a-Number", used for undefined or unrepresentable numeric results.	`result = 0/0;` % This will return NaN

Table 1: Special Variables in MATLAB

Variable Scope

Workspace Variables
Variables created in the MATLAB command window or in scripts are stored in the base workspace.
Local Variables
Variables created inside a function are local to that function.
Global Variables
Declared with the 'global' keyword, accessible across different functions and scripts:

```
global shared_var;
shared_var = 100;
```

Persistent Variables
Retain their values between function calls but are only accessible within the function:

```
function count = counter()
    persistent n
    if isempty(n)
        n = 0;
    end
    n = n + 1;
    count = n;
end
```

Variable Information and Management

Checking Variables
- who: List variables in the workspace.
- whos: Provides a detailed list of variables, including type and size information.
- exist('varname', 'var'): Checks if a variable with the name 'varname' exists in the workspace.

Clearing Variables
- clear x: Clears the specific variable ‘x‘ from the workspace.
- clear: Clears all variables from the workspace.
- clearvars: Provides more flexibility in clearing variables by name or pattern.
- clc: Clears the Command Window but does not affect workspace variables.

Exercise

Create descriptive variable names for the following concepts. Consider clarity and maintainability.

- The total number of students in a class
- The average score of students
- The current temperature in Celsius
- The maximum value in a dataset
- A flag indicating whether the data is valid (true/false)

Exercise

Create three variables in MATLAB:
- Assign the value ‘10‘ to a variable named ‘a‘.
- Assign the value ‘5.5‘ to a variable named ‘b‘.
- Assign the string ‘”MATLAB”‘ to a variable named ‘c‘.
- Then, use the ‘disp‘ function to display each variable’s value.

Exercise

- Create two variables:
- x with a value of 15
- y with a value of 4
- Using these variables, perform the following operations and store the results in separate variables:
- Sum of x and y
- Difference between x and y
- Product of x and y
- Quotient of x divided by y
- Display each result using the ‘disp‘ function.

Exercise

Create two variables:
- a with a value of 5
- b with a value of 10
- Swap the values of a and b so that a becomes 10 and b becomes 5.
- Display the new values of a and b after swapping.

Exercise

Create a variable called ‘total‘ with an initial value of ‘50‘.
- Update ‘total‘ by adding ‘20‘ to its current value.
- Then, update ‘total‘ again by multiplying it by ‘2‘.
- Display the final value of ‘total‘.

Exercise

Create two variables:
- ‘length‘ with a value of ‘10‘
- ‘width‘ with a value of ‘5‘

Use these variables to:
- Calculate the area of a rectangle and store it in a variable ‘area‘ (use the formula ‘area = length * width‘).
- Calculate the perimeter of the rectangle and store it in a variable ‘perimeter‘ (use the formula ‘perimeter = 2 * (length + width)‘).
- Display the values of ‘area‘ and ‘perimeter‘.

Exercise

- Define a constant value ‘piValue = 3.1416‘ to represent the value of π.
- Define a variable ‘radius‘ with a value of ‘7‘.
- Calculate the circumference of a circle using the formula ‘circumference = 2 * piValue * radius‘ and store the result in a variable ‘circumference‘.
- Display the value of ‘circumference‘.