



MATLAB Programming

Complete study notes — the MATLAB environment, variables and matrices, plotting, control flow, functions, data files, numerical methods, curve fitting and Simulink, with runnable examples throughout.

10

Modules

150+

Code examples

200+

Questions & MCQs



Contents

01	Introduction to MATLAB and the Development Environment
02	Variables, Data Types, and Basic Operations
03	Vectors, Matrices, and Array Operations
04	Plotting and Data Visualization
05	Control Flow: Loops and Conditionals
06	Writing Functions and Scripts
07	Working with Data Files (Import/Export)
08	Numerical Methods and Curve Fitting
09	Simulink Basics (Intro to Simulation)
10	Practical Project and Final Revision
11	Bonus: Command Reference, Lab Workbook and Careers
12	Multiple-Choice Questions, Answers and Glossary

How to use these notes

Type every example into MATLAB (or MATLAB Online / GNU Octave, which runs most of this code) and run it — programming is learned by doing. Code appears in dark boxes; `>>` marks commands typed in the Command Window; green text after `%` is a comment.

Blue box — Key idea. The concept to remember.

Yellow box — Syntax. Commands and patterns to learn.

Green box — Try it. Short hands-on tasks.

Orange box — Caution. Common errors and pitfalls.

Worked example. Complete programs with output.

Practice.
Exercises at the end of every module.

Introduction to MATLAB and the Development Environment

What MATLAB is and where it is used, getting access, the desktop — Command Window, Workspace, Current Folder, Editor — and your first commands, scripts and help.

1.1 What is MATLAB?

MATLAB (short for **MAT**rix **LAB**oratory) is a high-level programming language and interactive environment made by MathWorks for numerical computation, data analysis, visualisation and algorithm development. It began in the late 1970s as a simple tool to give students easy access to matrix software, and has become a standard tool in engineering and science.

What makes MATLAB special:

- **Everything is a matrix** — a number is a 1×1 matrix, a list of values is a vector. Operations on whole arrays are written in one line.
- **Interactive** — type a command and see the answer immediately; great for experimenting.
- **Built-in graphics** — professional 2D and 3D plots in a few commands.
- **Thousands of built-in functions**, plus **toolboxes** for signal processing, control systems, image processing, machine learning, optimisation, statistics and more.
- **Simulink** — a graphical environment for modelling and simulating dynamic systems (Module 9).

1.2 Where MATLAB is used

Field	Typical uses
Electrical & electronics	Circuit analysis, signal processing, filter design, power systems, control systems
Mechanical & civil	Vibration analysis, structural calculations, heat transfer, simulations
Communications	Modulation, channel models, 5G algorithms
Automotive & aerospace	Vehicle dynamics, flight control, embedded code generation
Data science & AI	Data cleaning, statistics, machine learning, deep learning
Image & video processing	Medical imaging, computer vision, satellite images

Renewable energy	Solar/wind modelling, battery models, power-converter simulation
Finance & economics	Risk models, time-series analysis
Research & teaching	Numerical methods, labs, theses

1.3 Getting access to MATLAB

- **Campus licence** — many colleges and universities provide MATLAB to students; ask your institution.
- **MATLAB Online** — runs in a web browser with a MathWorks account (limited free use may be available; full use needs a licence).
- **Student / Home licences** — low-cost personal licences from MathWorks.
- **GNU Octave** — a free, open-source program that runs most basic MATLAB code (syntax is largely compatible). Good for practising this course; some functions and all Simulink features differ or are unavailable.

1.4 The MATLAB desktop

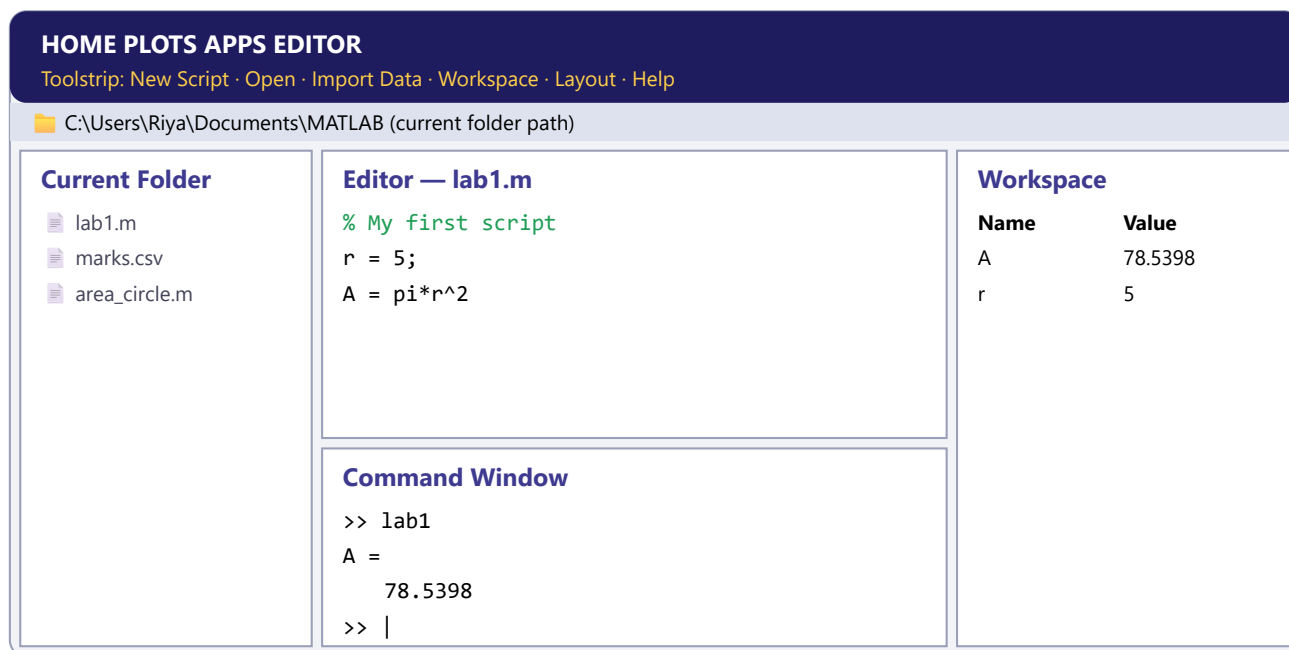


Figure 1.1 — The MATLAB desktop: Toolstrip, Current Folder, Editor, Command Window and Workspace.

Part	Purpose
Command Window	Type commands at the <code>>></code> prompt and see results immediately

Workspace	Lists all variables currently in memory with their size, class and value; double-click to open in the Variables editor
Current Folder	Files in the working directory. MATLAB runs scripts/functions found here (or on the search path)
Editor	Write, save and run scripts (.m files) and functions; shows warnings, supports debugging
Command History	Previous commands (press ↑ in the Command Window to recall them)
Toolstrip	Tabs (Home, Plots, Apps, Editor) with buttons for common tasks
Live Editor	Live scripts (.mlx) mixing code, formatted text, equations and output — great for reports

1.5 Your first commands

```
>> 2 + 3
ans =
     5
>> 10 / 4
ans =
    2.5000
>> 2^10
ans =
    1024
>> sqrt(144)
ans =
    12
>> x = 7
x =
     7
>> y = 3*x + 2;      % semicolon suppresses the output
>> y
y =
    23
```

Key idea — ans and the semicolon

If you don't assign a result to a variable, MATLAB stores it in `ans`. Ending a line with `;` runs it without printing the result — essential inside scripts and loops to avoid flooding the screen.

Arithmetic operators and precedence

Operator	Meaning	Example	Result
<code>+ -</code>	Add, subtract	<code>7 - 2</code>	5
<code>*</code>	Multiply	<code>4 * 2.5</code>	10
<code>/</code>	Divide	<code>9 / 2</code>	4.5000
<code>\</code>	Left divide ($a \backslash b = b/a$)	<code>2 \ 9</code>	4.5000
<code>^</code>	Power	<code>3^4</code>	81

Order: brackets → powers → multiplication/division (left to right) → addition/subtraction. `2 + 3*4^2 = 2 + 3 × 16 = 50`; `(2 + 3)*4^2 = 80`.

Useful built-in functions and constants

Function	Meaning	Function	Meaning
<code>sqrt(x)</code>	Square root	<code>abs(x)</code>	Absolute value
<code>exp(x)</code>	e^x	<code>log(x)</code>	Natural log (ln)
<code>log10(x)</code>	Base-10 log	<code>round, floor, ceil, fix</code>	Rounding
<code>sin, cos, tan</code>	Trig (radians)	<code>sind, cosd, tand</code>	Trig (degrees)
<code>asin, acos, atan</code>	Inverse trig	<code>rem(a,b), mod(a,b)</code>	Remainder
<code>pi</code>	$\pi = 3.14159...$	<code>i</code> or <code>j</code>	$\sqrt{-1}$
<code>Inf</code>	Infinity (e.g. $1/0$)	<code>NaN</code>	Not-a-Number (e.g. $0/0$)

Worked example 1.1 — engineering calculations in one line each

```
>> V = 230; R = 46;
>> I = V/R           % Ohm's law
I =
     5
>> P = V*I           % power in watts
P =
    1150
>> Vm = 230*sqrt(2)   % peak of 230 V RMS
Vm =
    325.2691
>> sind(30) + cosd(60)
ans =
     1
```

1.6 Controlling the display

```
>> format long       % 15 decimal digits
>> pi
ans =
    3.141592653589793
>> format short      % default: 4 decimal digits
>> format compact    % fewer blank lines
>> disp('Hello MATLAB')
Hello MATLAB
>> fprintf('Area = %.2f square metres\n', 78.5398)
Area = 78.54 square metres
```

1.7 Managing the workspace

Command	Action
<code>clc</code>	Clear the Command Window screen
<code>clear</code>	Delete all variables (<code>clear x</code> deletes only x)
<code>who / whos</code>	List variables / with size and class
<code>close all</code>	Close all figure windows
<code>save mydata / load mydata</code>	Save/load workspace variables to/from a .mat file
<code>pwd , cd , ls / dir</code>	Show / change current folder; list files

1.8 Scripts

Typing commands one by one is fine for experiments, but real work goes into **scripts** — plain text files with the extension **.m** containing a sequence of commands. Save once, run any time, edit and re-run.

1. Home → **New Script** (or type `edit myscript`).
2. Type the commands. Start with comments describing the purpose.
3. Save as `area_circle.m` (names must start with a letter; only letters, digits and underscores; no spaces).
4. Click **Run** (F5) or type the script name in the Command Window.

```
% area_circle.m - area and circumference of a circle
% Author: Riya   Date: 2026-09-20
clc; clear;
r = input('Enter radius in metres: ');
A = pi * r^2;
C = 2 * pi * r;
fprintf('Area = %.3f m^2\n', A);
fprintf('Circumference = %.3f m\n', C);
```

Output for $r = 3$:

```
Enter radius in metres: 3
Area = 28.274 m^2
Circumference = 18.850 m
```

Try it — code sections

In the Editor, lines beginning with `%%` create **sections**. Put the cursor in a section and press **Ctrl + Enter** to run only that part — handy for long scripts and lab reports.

1.9 Getting help

Command	What it does
<code>help sqrt</code>	Short help text for a function in the Command Window
<code>doc plot</code>	Full documentation page with examples

<code>lookfor regression</code>	Searches help summaries for a keyword
F1 on a function name	Opens help for that function in the Editor
Tab key	Auto-completes variable and function names

MathWorks' online documentation and MATLAB Answers forum are excellent. Reading error messages carefully (they give the line number) is the fastest way to fix most problems.

1.10 Common beginner errors

Error message	Cause	Fix
Unrecognized function or variable 'x'	Misspelt name or variable not created yet	Check spelling and case (<code>X</code> $\neq$ <code>x</code>); run earlier lines
Invalid expression / parse error	Missing bracket, operator or quote	Check the line the Editor underlines in red
Arrays have incompatible sizes	Operating on arrays of different sizes (Module 3)	Check sizes with <code>size()</code> ; use element-wise operators
Script won't run	File not in Current Folder, or name clashes with a function	Change folder; rename file (e.g. don't call it <code>sin.m</code>)

Module 1 — Summary

- MATLAB = matrix laboratory: interactive computing, built-in plotting, toolboxes, Simulink.
- Desktop: Command Window, Workspace, Current Folder, Editor, Toolstrip.
- Results go to `ans` unless assigned; `;` suppresses output.
- Operators `+` `-` `*` `/` `\` `^` with standard precedence; functions like `sqrt`, `exp`, `log`, `sind`.
- `clc`, `clear`, `who/whos`, `format`, `disp`, `fprintf`.
- Scripts are `.m` files; use comments and `%%` sections; `help`, `doc`, `lookfor` for help.

Practice questions — Module 1

1. What does MATLAB stand for, and what is special about how it treats data?
2. Name the five main parts of the MATLAB desktop and their use.
3. Evaluate by hand, then check: $5 + 2 \cdot 3^2 - 8/4$. (Ans: 21)
4. Compute the hypotenuse of a right triangle with sides 7 and 24. (Ans: 25)
5. What is the difference between `clc` and `clear`?
6. Write a script that asks for temperature in °C and prints it in °F with two decimals.
7. What do `Inf` and `NaN` represent? Give an expression producing each.
8. Find $\sin 45^\circ$ and $\cos(\pi/3)$ in MATLAB. (Ans: 0.7071; 0.5000)

MODULE 02

Variables, Data Types, and Basic Operations

Naming variables, numbers, characters and strings, logicals, complex numbers, cell arrays and structures; operator precedence; relational and logical operators; and formatted input and output.

2.1 Variables

A **variable** is a named storage location in the workspace. You create one simply by assigning a value with `=` — there's no need to declare its type first.

```
speed = 60           % km/h
time_h = 2.5;
distance = speed * time_h % 150
```

Key idea — "=" means "store"

`=` is the **assignment operator**: evaluate the right side, then store the result in the variable on the left. So `count = count + 1` is perfectly valid — it increases count by one.

Naming rules

- Must start with a **letter**; may contain letters, digits and underscores: `vel_1`, `Vout`, `total_marks`.
- **Case-sensitive**: `Area`, `area` and `AREA` are three different variables.
- No spaces or symbols: `my var`, `2x`, `rate%` are invalid.
- Maximum length: `namelengthmax` (63 characters).
- Don't use MATLAB keywords (`if`, `for`, `end`, `function...` — list with `iskeyword`).
- **Avoid** reusing function names (`sum`, `max`, `i`, `pi`, `input`) — it hides the built-in function and causes confusing errors.
- Use meaningful names: `current_A` is better than `c`.

Caution — overwriting built-ins

If you write `sum = 10`, then later `sum([1 2 3])` fails, because `sum` is now your variable. Fix it with `clear sum`.

2.2 Numeric data types

Class	Description	Range / precision
double (default)	64-bit floating point	≈ 15–16 significant digits; up to about 1.8×10^{308}
single	32-bit floating point	≈ 7 digits; less memory
int8, int16, int32, int64	Signed integers	int8: –128 to 127
uint8, uint16, uint32, uint64	Unsigned integers	uint8: 0 to 255 (used for image pixels)

```
a = 5;          class(a)      % 'double'
b = int8(200)    % saturates at 127
c = uint8(250) + uint8(10)    % saturates at 255
d = int16(7) / int16(2)       % 4 (integer division rounds)
realmax, realmin, eps        % limits of double precision
```

Caution — floating-point surprises

Computers store most decimals approximately. `0.1 + 0.2 == 0.3` returns **false** (0), because $0.1 + 0.2$ is actually 0.30000000000000004 . Compare decimals with a tolerance:

```
abs((0.1+0.2) - 0.3) < 1e-10 → true.
```

Scientific notation

```
q = 1.6e-19      % electron charge,  $1.6 \times 10^{-19}$ 
c = 3e8          % speed of light
Na = 6.022e23
```

2.3 Characters and strings

MATLAB has two text types:

	Character array	String (R2016b+)
Created with	Single quotes: <code>'Hello'</code>	Double quotes: <code>"Hello"</code>
Nature	A row of characters (1×5 char)	One text object (1×1 string)
Joining	<code>['Hello ', 'World']</code>	<code>"Hello " + "World"</code>

```

name = 'Riya';
msg = ['Hello, ', name, '!']           % 'Hello, Riya!'
length(name)                          % 4
upper(name)                           % 'RIYA'
s = "Solar"; t = s + " Energy"        % "Solar Energy"
n = num2str(3.14159, 4)                % '3.142' (number to text)
x = str2double('42.5')                 % 42.5 (text to number)
strcmp('abc', 'abc')                   % 1 (true) – compare text
strrep('2026-09-20', '-', '/')        % '2026/09/20'
contains("MATLAB Programming", "Prog") % true
split("a,b,c", ",")                   % ["a";"b";"c"]

```

2.4 Logical values

Logical values are **true (1)** and **false (0)**. They are produced by comparisons and used in decisions and for selecting data.

Relational operators

Operator	Meaning	Example	Result
<code>==</code>	Equal to	<code>5 == 5</code>	1
<code>~=</code>	Not equal to	<code>5 ~= 3</code>	1
<code>< <=</code>	Less than (or equal)	<code>2 < 1</code>	0
<code>> >=</code>	Greater than (or equal)	<code>7 >= 7</code>	1

Caution — = versus ==

`x = 5` **assigns** 5 to x. `x == 5` **tests** whether x equals 5. Mixing them up is a very common beginner bug.

Logical operators

Operator	Meaning	Example	Result
<code>&</code>	AND (element-wise)	<code>[1 0 1] & [1 1 0]</code>	<code>[1 0 0]</code>
<code> </code>	OR (element-wise)	<code>[1 0 1] [0 0 1]</code>	<code>[1 0 1]</code>
<code>~</code>	NOT	<code>~[1 0]</code>	<code>[0 1]</code>
<code>&&, </code>	Short-circuit AND/OR (scalars, in if/while)	<code>(x > 0) && (y > 0)</code>	true/false

<code>xor(a,b)</code>	Exclusive OR	<code>xor(1,1)</code>	0
-----------------------	--------------	-----------------------	---

```
age = 20; marks = 65;
eligible = (age >= 18) && (marks >= 60)    % 1 (true)
```

2.5 Operator precedence (full table)

Priority	Operators
1 (highest)	Parentheses <code>()</code>
2	Transpose <code>' .'</code> , power <code>^ .^</code>
3	Unary plus/minus <code>+ -</code> , NOT <code>~</code>
4	Multiplication/division <code>* / \ .* ./ .\</code>
5	Addition/subtraction <code>+ -</code>
6	Colon <code>:</code>
7	Relational <code>< <= > >= == ~=</code>
8	Element-wise AND <code>&</code>
9	Element-wise OR <code> </code>
10	Short-circuit <code>&&</code> , then <code> </code>

Worked example 2.1 — precedence traps

```
-2^2      % -4 (power first, then minus)
(-2)^2    % 4
2^3^2     % 64 (left to right: (2^3)^2)
10/2*5    % 25 (left to right)
1:3+2     % 1 2 3 4 5 (3+2 first, then colon)
```

2.6 Complex numbers

MATLAB handles complex numbers naturally — very useful for AC circuit analysis (phasors) and signal processing.

```

z = 3 + 4i
abs(z)          % 5      magnitude
angle(z)        % 0.9273 rad
rad2deg(angle(z)) % 53.13 degrees
real(z), imag(z) % 3, 4
conj(z)         % 3 - 4i
z2 = 5*exp(1i*pi/6) % polar form 5∠30°

```

Worked example 2.2 — impedance of an RL circuit

```

R = 30; L = 0.127; f = 50;
XL = 2*pi*f*L;           % 39.9 ohm
Z = R + 1i*XL;           % complex impedance
V = 230;
I = V / Z;               % complex current
fprintf('|Z| = %.1f ohm, |I| = %.2f A, phase = %.1f deg\n', ...
        abs(Z), abs(I), rad2deg(angle(I)));

```

Output: $|Z| = 49.9$ ohm, $|I| = 4.61$ A, phase = -53.1 deg (current lags the voltage). The ... continues a long line.

2.7 Cell arrays and structures

Ordinary arrays hold one type of data (all numbers or all characters). To group different kinds of data, use:

Cell arrays — { }

```

info = {'Riya', 19, [85 78 90]}; % text, number, vector together
info{1} % 'Riya'
info{3}(2) % 78
names = {'Asha', 'Ravi', 'Sara'};
numel(names) % 3

```

Structures — named fields

```
student.name = 'Riya';
student.age  = 19;
student.marks = [85 78 90];
student.avg  = mean(student.marks)    % 84.3333
fieldnames(student)
% array of structures
s(1).name = 'Asha'; s(1).cgpa = 8.2;
s(2).name = 'Ravi'; s(2).cgpa = 7.6;
[s.cgpa]                                     % [8.2 7.6]
```

MATLAB also has **tables** (column-oriented data with named variables, like a spreadsheet — Module 7), **datetime** for dates and times, and **categorical** arrays for labels.

2.8 Input and formatted output

input

```
n = input('Enter number of students: ');    % number
city = input('Enter your city: ', 's');      % text (note 's')
```

disp and fprintf

```
disp('Result:')
disp(pi)
fprintf('Voltage = %d V, current = %.2f A\n', 230, 4.6)
fprintf('%-10s %6.1f\n', 'Asha', 82.5)      % left-aligned name, number width 6
```

Format spec	Meaning	Example output
%d	Integer	230
%f, %.2f	Fixed decimal (2 places)	4.60
%e	Scientific	1.600000e-19
%g	Compact of %f/%e	1.6e-19
%s	Text	Asha
%8.3f	Width 8, 3 decimals	78.540
\n, \t	New line, tab	—

Also useful: `sprintf` (same as `fprintf` but returns text), and `num2str`/`string()` for building messages.

Worked example 2.3 — electricity bill calculator

```
% bill.m
units = input('Units consumed: ');
rate = 6.5;           % Rs per unit
fixed = 100;          % fixed charge
energy = units * rate;
tax = 0.05 * (energy + fixed);
total = energy + fixed + tax;
fprintf('Energy charge : Rs %8.2f\n', energy);
fprintf('Fixed charge  : Rs %8.2f\n', fixed);
fprintf('Tax (5%%)      : Rs %8.2f\n', tax);
fprintf('TOTAL         : Rs %8.2f\n', total);
```

For 250 units: energy 1625.00, fixed 100.00, tax 86.25, **total 1811.25**. (Note %% prints a literal percent sign.)

Module 2 — Summary

- Create variables by assignment; names are case-sensitive, start with a letter; don't reuse function names.
- Default number type is double; integer types saturate; compare decimals with a tolerance.
- Text: char arrays ('...') and strings ("..."); many text functions.
- Relational (`=` `~=` `<` `>`) and logical (`&` `|` `~` `&&` `||`) operators give true/false.
- Know operator precedence (power before unary minus!).
- Complex numbers for AC and signals; cell arrays and structures for mixed data.
- `input`, `disp`, `fprintf` with format specifiers for clean I/O.

Practice questions — Module 2

1. Which names are valid? `2ndValue`, `value_2`, `my-value`, `Value`, `end`, `total marks`
2. What does `uint8(300)` return and why? (Ans: 255 — saturates)
3. Evaluate: `-3^2`, `(-3)^2`, `2+3>4`. (Ans: -9, 9, 1)
4. Why does `0.1*3 == 0.3` give 0? How should you compare?
5. Write a line that checks whether x is between 10 and 20 inclusive.
6. For $z = -1 + 1i$, find magnitude and angle in degrees. (Ans: 1.4142; 135°)
7. Create a structure for a resistor with fields value, tolerance and power, and print it.
8. Write `fprintf` commands to print a neat 3-row table of names and marks.

MODULE 03

Vectors, Matrices, and Array Operations

Creating and indexing vectors and matrices, the colon operator, built-in array functions, the crucial difference between matrix and element-wise operators, logical indexing, and solving linear equations.

3.1 Arrays are the heart of MATLAB

In MATLAB, all data are arrays: a scalar is 1×1 , a **row vector** is $1 \times n$, a **column vector** is $n \times 1$, and a **matrix** is $m \times n$ (m rows, n columns). Working with whole arrays at once — instead of one number at a time — is what makes MATLAB code short and fast.

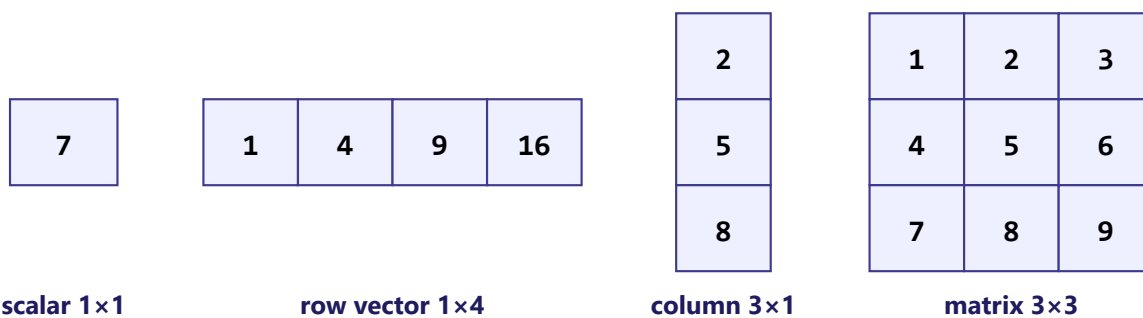


Figure 3.1 — Scalars, vectors and matrices are all arrays.

3.2 Creating vectors

```
r = [1 4 9 16]           % row vector (spaces or commas)
r = [1, 4, 9, 16]
c = [2; 5; 8]            % column vector (semicolons)
c = [2 5 8]';           % transpose a row into a column
x = 0:5                  % colon: 0 1 2 3 4 5
x = 0:0.25:1             % start:step:end → 0 0.25 0.5 0.75 1
x = 10:-2:0              % 10 8 6 4 2 0
t = linspace(0, 1, 5)    % 5 equally spaced points from 0 to 1
e = []                  % empty array
```

Syntax — the colon operator

start : step : stop (step defaults to 1)

`linspace(a, b, n)` gives exactly n points including both ends — preferred for plotting.

3.3 Creating matrices

```
A = [1 2 3; 4 5 6; 7 8 9]    % rows separated by semicolons
Z = zeros(2, 3)              % 2x3 of zeros
O = ones(3)                  % 3x3 of ones
I = eye(3)                   % 3x3 identity matrix
R = rand(2)                  % 2x2 random numbers in (0,1)
N = randi(100, 1, 5)         % 5 random integers 1..100
M = magic(4)                 % magic square
D = diag([2 5 7])            % diagonal matrix
B = [A; 10 11 12]            % append a row → 4x3
C = [A, [0; 0; 0]]           % append a column → 3x4
```

3.4 Size and shape

Function	Returns	For A (3×3 above)
<code>size(A)</code>	[rows cols]	[3 3]
<code>size(A,1)</code> , <code>size(A,2)</code>	rows, columns	3, 3
<code>length(v)</code>	Largest dimension	3
<code>numel(A)</code>	Number of elements	9
<code>reshape(1:6, 2, 3)</code>	Reshape (column order)	[1 3 5; 2 4 6]
<code>A'</code>	Transpose	rows ↔ columns
<code>A(:)</code>	All elements as one column	9×1

3.5 Indexing — getting and changing elements

MATLAB indices start at **1** (not 0). Use round brackets: `A(row, col)` .

```

v = [10 20 30 40 50];
v(2)           % 20
v(end)         % 50 (end = last index)
v([1 3 5])     % 10 30 50
v(2:4)         % 20 30 40
v(end:-1:1)    % reversed
v(3) = 99      % change an element
v(7) = 70      % grows the vector; v(6) becomes 0

A = [1 2 3; 4 5 6; 7 8 9];
A(2,3)         % 6 (row 2, column 3)
A(3,:)         % 7 8 9 (whole row 3)
A(:,2)         % [2;5;8] (whole column 2)
A(1:2, 2:3)    % [2 3; 5 6] sub-matrix
A(:, 1) = []   % delete column 1
A(5)           % linear index: 5th element in column order

```

	col 1	col 2	col 3	
row 1	1	2	3	
row 2	4	5	6	$A(2,3) = 6$
row 3	7	8	9	$A(:,2) = [2;5;8]$

Figure 3.2 — Indexing a matrix by (row, column).

3.6 Matrix operations versus element-wise operations

This is the single most important idea in MATLAB. Operators without a dot follow the rules of **linear algebra**; operators with a dot work **element by element**.

Operation	Matrix (linear algebra)	Element-wise
Multiply	$A * B$ — rows $\times$ columns; inner dimensions must match	$A .* B$ — multiply matching elements; same size
Divide	A / B , $A \setminus B$ — solve linear systems	$A ./ B$ — divide matching elements
Power	$A ^ 2 = A*A$ (square matrices only)	$A .^ 2$ — square each element
Add / subtract	$+$ $-$ are always element-wise (same size, or one is a scalar)	

Worked example 3.1 — the difference in action

```
A = [1 2; 3 4]; B = [5 6; 7 8];
A * B      % [1*5+2*7  1*6+2*8; 3*5+4*7  3*6+4*8] = [19 22; 43 50]
A .* B      % [1*5 2*6; 3*7 4*8] = [5 12; 21 32]
A ^ 2      % A*A = [7 10; 15 22]
A .^ 2      % [1 4; 9 16]
x = 0:4;
y = x.^2    % [0 1 4 9 16] - x^2 would give an ERROR (1x5 not square)
```

Key idea — when plotting a formula, use dots

To evaluate $y = x^2 + 3x$ for a whole vector x , write `y = x.^2 + 3*x`. Scalar $\times$ vector ($3*x$) is fine without a dot, but vector $\times$ vector, vector $\div$ vector and vector powers need `.*`, `./` and `.^`.

Matrix multiplication rule

$$(m \times n) \times (n \times p) = (m \times p)$$

Inner dimensions (n) must be equal; the result takes the outer dimensions.

Figure 3.3 — Size rule for matrix multiplication.

Implicit expansion

Since R2016b, element-wise operations between a matrix and a compatible vector expand automatically: `A - mean(A)` subtracts each column's mean from that column.

3.7 Built-in array functions

Function	For <code>v = [4 9 1 7]</code>	On a matrix
<code>sum(v)</code>	21	Column sums; <code>sum(A,2)</code> row sums; <code>sum(A,'all')</code>
<code>prod(v)</code>	252	Column products
<code>mean(v)</code> , <code>median(v)</code>	5.25, 5.5	Per column

<code>max(v), min(v)</code>	9, 1	<code>[m, idx] = max(v) → m = 9, idx = 2</code>
<code>sort(v)</code>	[1 4 7 9]	<code>sort(v, 'descend')</code>
<code>cumsum(v)</code>	[4 13 14 21]	Running totals
<code>std(v), var(v)</code>	Standard deviation, variance	Per column
<code>find(v > 5)</code>	[2 4] (indices)	Linear indices
<code>any, all</code>	<code>any(v>8) → 1; all(v>0) → 1</code>	Per column
<code>round, abs, sqrt, exp, sin...</code>	Apply to every element automatically	
<code>unique, fliplr, flipud, circshift</code>	Unique values; flip left-right, up-down; rotate elements	

3.8 Logical indexing — selecting data by condition

A comparison on a vector returns a logical vector of 1s and 0s. Using it as an index picks out only the elements where it is true. This replaces many loops.

```
marks = [45 82 37 91 68 29];
pass = marks >= 40           % [1 1 0 1 1 0]
marks(pass)                  % [45 82 91 68]
numPass = sum(pass)          % 4
marks(marks < 40) = 40       % raise failing marks to 40 (grace)
highIdx = find(marks > 80)   % [2 4]
```

Worked example 3.2 — class result analysis

```
names = {'Asha', 'Ravi', 'Sara', 'Tom', 'Uma'};
M = [78 65 90; 45 38 52; 92 88 95; 60 72 58; 35 41 30]; % 3 subjects
total = sum(M, 2); % row totals
avg = mean(M, 2); % row averages
 [~, topIdx] = max(total);
fprintf('Topper: %s (%d marks)\n', names{topIdx}, total(topIdx));
failAny = any(M < 40, 2); % failed at least one subject
disp('Need support:'); disp(names(failAny));
subjAvg = mean(M) % subject-wise averages
```

Output: Topper: Sara (275 marks). Need support: Ravi, Uma. Subject averages $\approx$ [62.0 60.8 65.0].

3.9 Linear algebra essentials

Function	Meaning
<code>det(A)</code>	Determinant
<code>inv(A)</code>	Inverse (square, non-singular)
<code>rank(A)</code>	Rank
<code>trace(A)</code>	Sum of diagonal
<code>[V, D] = eig(A)</code>	Eigenvectors and eigenvalues
<code>dot(a,b)</code> , <code>cross(a,b)</code>	Dot and cross products
<code>norm(v)</code>	Length (magnitude) of a vector

Solving simultaneous linear equations

A system of equations can be written as $\mathbf{A} \mathbf{x} = \mathbf{b}$. MATLAB solves it with the backslash operator: `x = A \ b` — faster and more accurate than `inv(A)*b`.

Worked example 3.3 — circuit mesh currents

From Kirchhoff's laws, a circuit gives: $8I_1 - 6I_2 = 10$ and $-6I_1 + 10I_2 = -5$.

```
A = [8 -6; -6 10];  
b = [10; -5];  
I = A \ b           % I = [1.5909; 0.4545]
```

Three equations work the same way: $2x + y - z = 8$, $-3x - y + 2z = -11$, $-2x + y + 2z = -3$:

```
A = [2 1 -1; -3 -1 2; -2 1 2]; b = [8; -11; -3];  
x = A \ b           % x = 2, y = 3, z = -1
```

Module 3 — Summary

- Row [1 2 3], column [1;2;3], matrix [1 2; 3 4]; colon start:step:stop; linspace(a,b,n).
- zeros, ones, eye, rand, randi, magic, diag; size, numel, length, reshape, transpose '.
- Indexing starts at 1: A(r,c), A(r,:), A(:,c), end, deleting with [].
- *** / ^ are matrix operations; .* ./ .^ are element-wise.**
- sum, mean, max/min with index, sort, cumsum, find, any/all.
- Logical indexing selects data by condition without loops.
- Solve $Ax = b$ with $x = A \setminus b$; det, inv, eig for linear algebra.

Practice questions — Module 3

1. Create the vector 5, 10, 15, ..., 50 in two different ways.
2. For $A = \text{magic}(4)$, extract the last row, the second column and the central 2×2 block.
3. Given $A = [1 \ 2; 3 \ 4]$ and $B = [2 \ 0; 1 \ 3]$, compute $A*B$ and $A.*B$ by hand. (Ans: $[4 \ 6; 10 \ 12]$; $[2 \ 0; 3 \ 12]$)
4. Why does `x = 1:5; y = x^2` give an error? Fix it.
5. For $v = [12 \ 7 \ 25 \ 3 \ 18]$, find the maximum and its position, and the sum of elements greater than 10. (Ans: 25 at 3; 55)
6. Using logical indexing, replace all negative values in a vector with zero.
7. Solve: $x + y + z = 6$, $2y + 5z = -4$, $2x + 5y - z = 27$. (Ans: $x = 5$, $y = 3$, $z = -2$)
8. Compute row and column averages of a 4×3 marks matrix.

MODULE 04

Plotting and Data Visualization

2D line plots, labelling axes properly, multiple lines and legends, subplots, bar/scatter/histogram/pie charts, 3D plots, and exporting figures that belong in a report.

4.1 Your first plot

Plotting in MATLAB takes three steps: create the x values, compute the y values (element-wise!), and call `plot`.

```
x = linspace(0, 2*pi, 200); % 200 points from 0 to 2π
y = sin(x);
plot(x, y)
xlabel('Angle (rad)')
ylabel('sin(x)')
title('Sine wave')
grid on
```

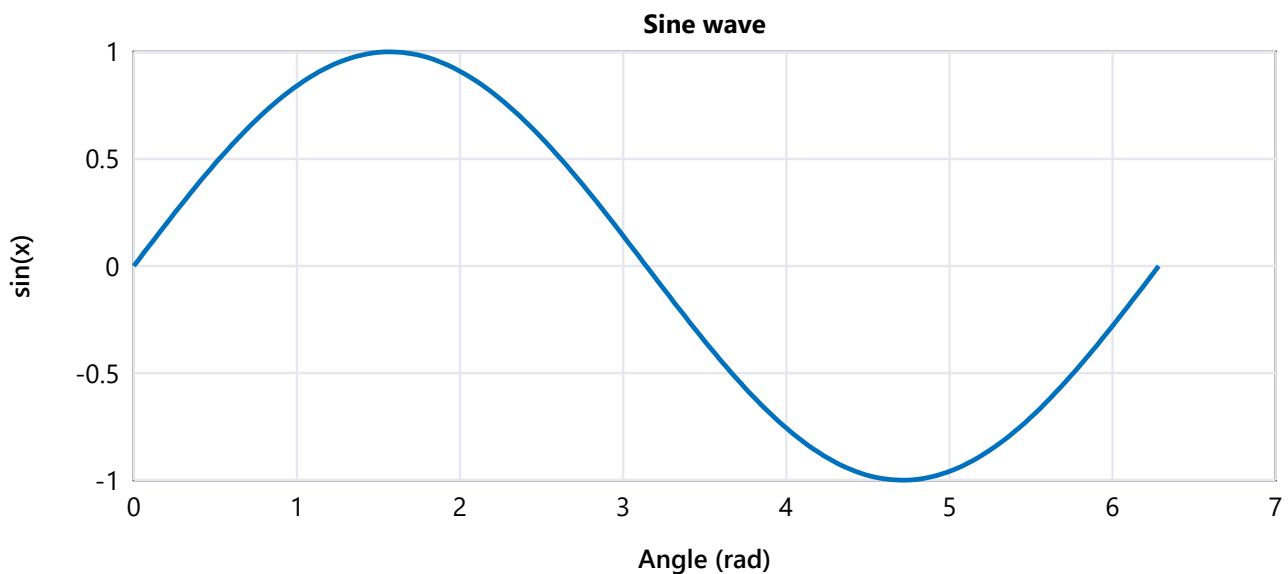


Figure 4.1 — Output of the first plot (MATLAB draws this in a figure window).

Key idea — a figure without labels is incomplete

Every plot for a report needs a **title**, **axis labels with units**, a **legend** if there is more than one line, and usually a **grid**.

4.2 Line styles, colours and markers

Add a format string as the third argument to `plot`, combining colour, marker and line style:

Colour	Code	Marker	Code	Line style	Code
Blue	b	Circle	o	Solid	-
Red	r	Plus	+	Dashed	--
Green	g	Star	*	Dotted	:
Black	k	Square	s	Dash-dot	-.
Magenta / cyan / yellow	m c y	Diamond / triangle	d ^	None (markers only)	omit

```
plot(x, y, 'r--', 'LineWidth', 2)           % red dashed, thick
plot(t, v, 'bo-', 'MarkerSize', 6, 'MarkerFaceColor', 'b')
plot(x, y, 'Color', [0.1 0.1 0.5])         % custom RGB colour
```

4.3 Several curves on one plot

```
x = linspace(0, 2*pi, 200);
plot(x, sin(x), 'b-', 'LineWidth', 1.5)
hold on                               % keep the current plot
plot(x, cos(x), 'r--', 'LineWidth', 1.5)
hold off
legend('sin(x)', 'cos(x)', 'Location', 'southwest')
xlabel('x (rad)'); ylabel('Amplitude'); title('Sine and cosine')
grid on; axis([0 2*pi -1.2 1.2])
```

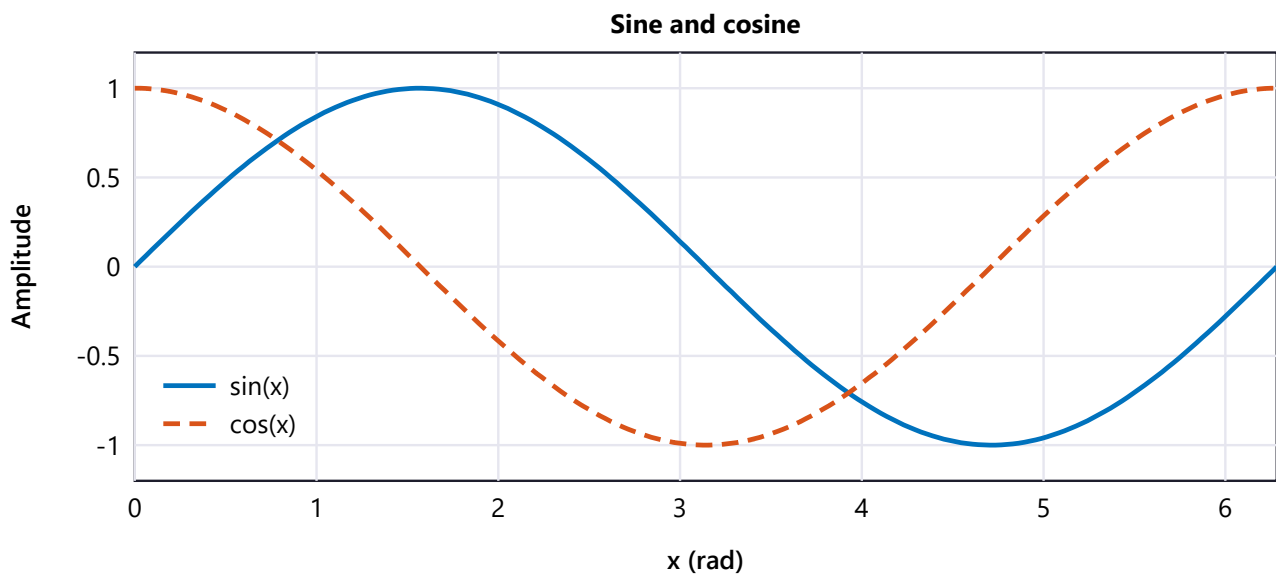


Figure 4.2 — Two curves with hold on, a legend, grid and axis limits.

You can also plot several pairs in one call: `plot(x, sin(x), x, cos(x))`.

Axis control

Command	Effect
<code>axis([xmin xmax ymin ymax])</code>	Set limits
<code>xlim([a b]), ylim([c d])</code>	Set one axis
<code>axis equal / axis tight / axis square</code>	Equal units / fit data / square box
<code>grid on, grid minor, box on</code>	Grid lines, frame
<code>xticks([...]), xticklabels({...})</code>	Custom tick marks and labels
<code>text(x, y, 'label'), annotation</code>	Add text or arrows
<code>semilogx, semilogy, loglog</code>	Logarithmic axes (e.g. frequency response)
<code>yyaxis left/right</code>	Two different y-axes

4.4 Subplots — several plots in one figure

```
t = linspace(0, 0.04, 400);           % two cycles of 50 Hz
v = 325*sin(2*pi*50*t);
i = 10*sin(2*pi*50*t - pi/4);         % current lags by 45 degrees
p = v .* i;                           % instantaneous power
subplot(3,1,1); plot(t*1000, v); ylabel('v (V)'); title('Voltage'); grid on
subplot(3,1,2); plot(t*1000, i, 'r'); ylabel('i (A)'); title('Current'); grid on
subplot(3,1,3); plot(t*1000, p, 'k'); ylabel('p (W)'); xlabel('Time (ms)');
title('Instantaneous power'); grid on
```

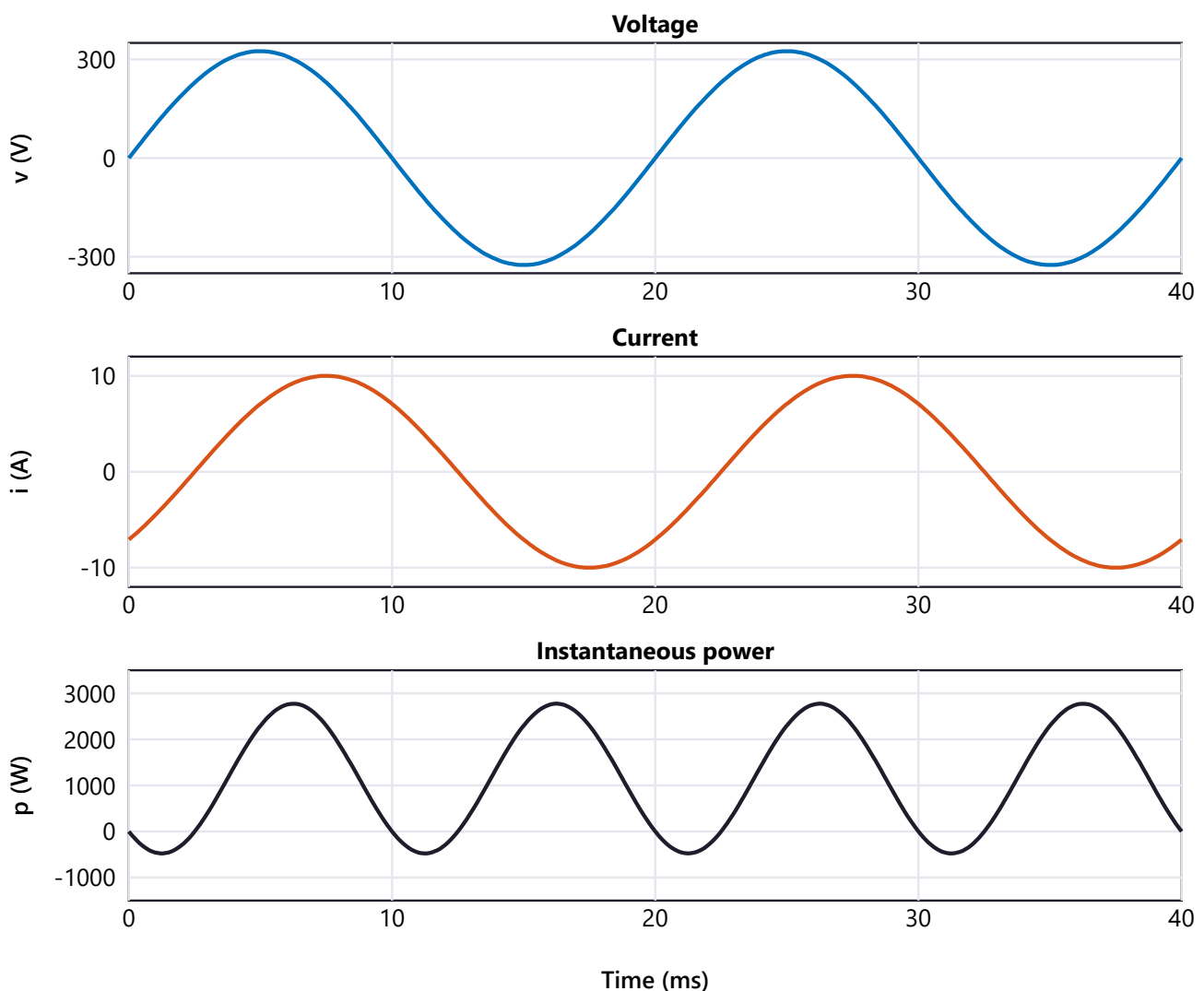


Figure 4.3 — `subplot(3,1,k)`: voltage, lagging current and instantaneous power. Power is mostly positive but dips below zero, showing reactive energy flowing back.

`subplot(m, n, k)` divides the figure into an $m \times n$ grid and draws in the k -th cell (counted across rows). The newer `tiledlayout` / `nexttile` commands give tighter, more flexible layouts.

4.5 Other 2D plot types

Function	Use for	Example
<code>bar</code> , <code>barh</code>	Comparing categories	Marks per subject
<code>scatter</code>	Relationship between two variables	Study hours vs marks
<code>histogram</code>	Distribution of values	Spread of exam scores
<code>pie</code>	Parts of a whole	Monthly budget
<code>stairs</code>	Step changes	Digital signals, tariffs
<code>stem</code>	Discrete samples	Sampled signals (DSP)
<code>errorbar</code>	Measurements with uncertainty	Lab readings
<code>area</code> , <code>fill</code>	Shaded areas	Energy under a curve
<code>polarplot</code>	Angle–radius data	Antenna patterns
<code>fplot</code>	Plot a function directly	<code>fplot(@(x) x.^2, [-2 2])</code>

Worked example 4.1 — bar chart of monthly solar generation

```
months = {'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'};
kWh = [360 380 450 470 480 400 330 320 350 390 370 350];
bar(kWh, 'FaceColor', [0.24 0.23 0.56])
xticks(1:12); xticklabels(months)
ylabel('Energy (kWh)'); title('Monthly generation of a 3 kWp rooftop system')
grid on
fprintf('Annual total = %d kWh, best month = %s\n', sum(kWh), months{kWh ==
max(kWh)});
```

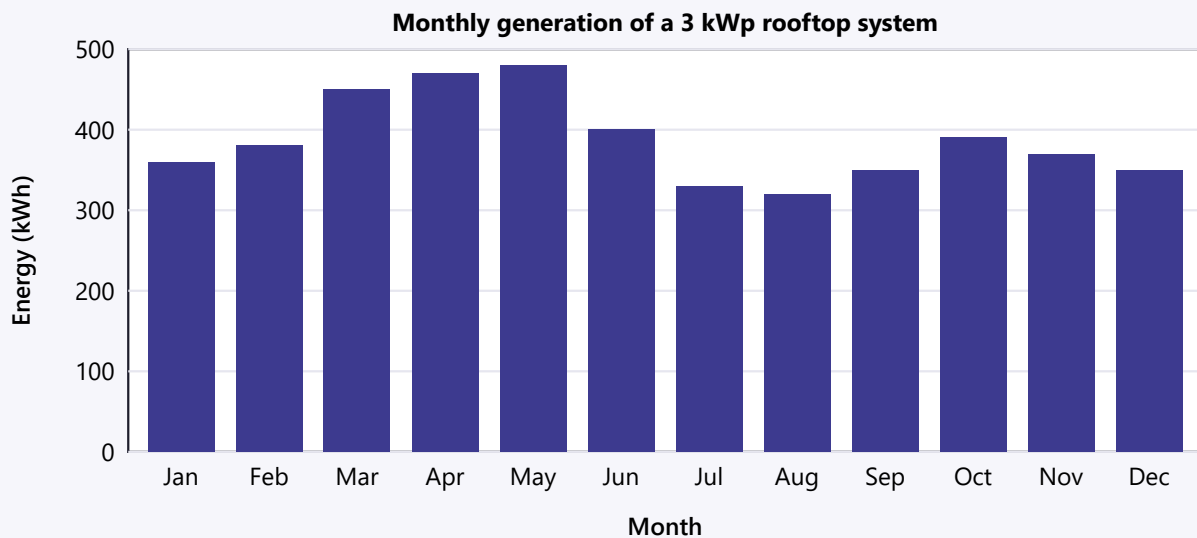


Figure 4.4 — Bar chart produced by Worked example 4.1.

Output: Annual total = 4650 kWh, best month = May .

Worked example 4.2 — scatter plot with a trend line

```
hours = [1 2 2.5 3 4 4.5 5 6 6.5 7];  
marks = [35 42 48 50 58 62 66 72 75 80];  
scatter(hours, marks, 50, 'filled'); hold on  
pcoef = polyfit(hours, marks, 1); % straight-line fit (Module 8)  
plot(hours, polyval(pcoef, hours), 'r-', 'LineWidth', 1.5); hold off  
xlabel('Study hours per day'); ylabel('Marks (%)'); grid on  
legend('Students', 'Trend line', 'Location', 'northwest')
```

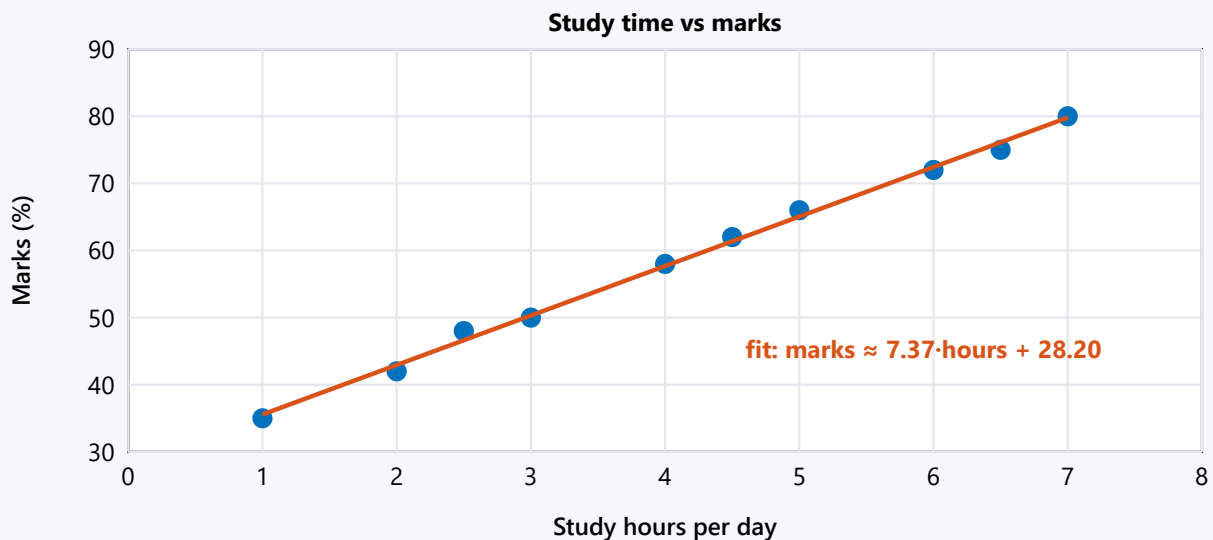


Figure 4.5 — Scatter plot with a least-squares trend line (Worked example 4.2).

4.6 3D plots

Function	Draws
<code>plot3(x, y, z)</code>	A 3D curve (e.g. a helix or trajectory)
<code>[X, Y] = meshgrid(x, y)</code>	Grid of points for surfaces
<code>mesh(X, Y, Z)</code>	Wire-frame surface
<code>surf(X, Y, Z)</code>	Coloured surface
<code>contour, contourf</code>	Contour lines (like a map's height lines)
<code>colorbar, colormap parula/jet/hot</code>	Colour scale
<code>view(az, el), rotate3d on</code>	Viewing angle

```
[X, Y] = meshgrid(-2:0.1:2, -2:0.1:2);
Z = X .* exp(-X.^2 - Y.^2);
surf(X, Y, Z); shading interp; colorbar
xlabel('x'); ylabel('y'); zlabel('z'); title('z = x e^{-x^2-y^2}')

t = linspace(0, 6*pi, 500);
plot3(cos(t), sin(t), t, 'LineWidth', 1.5); grid on    % helix
```

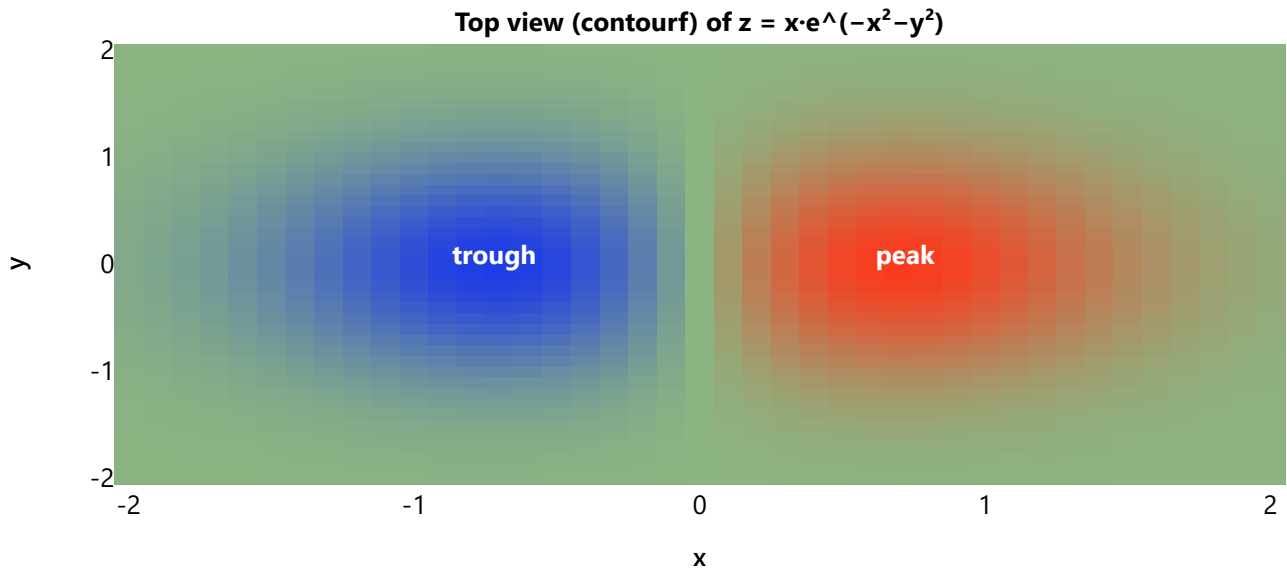


Figure 4.6 — The surface seen from above as a filled contour/heat map: a peak at $x \approx +0.71$ and a trough at $x \approx -0.71$.

4.7 Figures, saving and exporting

```
figure                                % open a new figure window
figure('Name', 'Results', 'Color', 'w')
clf                                  % clear current figure
close all                            % close all figures
saveas(gcf, 'result.png')            % save current figure
exportgraphics(gcf, 'result.png', 'Resolution', 300) % high-quality (R2020a+)
exportgraphics(gcf, 'result.pdf', 'ContentType', 'vector')
```

Report-quality figure checklist

- Title, axis labels with units, legend, grid.
- Readable font size: `set(gca, 'FontSize', 12)`.
- Thicker lines: `'LineWidth', 1.5`.
- Distinguish lines by style and marker, not colour alone (prints in black & white; colour-blind readers).
- Export at 300 dpi PNG or as vector PDF/EPS for documents.

4.8 Handles and properties (introduction)

Every graphic object — figure, axes, line, text — has a **handle** whose properties can be read and changed:

```
h = plot(x, sin(x));
h.LineWidth = 2; h.Color = 'r';
ax = gca;           % current axes
ax.FontSize = 12; ax.XGrid = 'on';
fig = gcf; fig.Position = [100 100 800 400]; % size in pixels
```

4.9 Animations (bonus)

```
x = linspace(0, 2*pi, 200);
h = plot(x, sin(x)); ylim([-1.2 1.2]); grid on
for k = 1:100
    h.YData = sin(x - 0.1*k); % shift the wave
    drawnow                  % update the screen
end
```

Module 4 — Summary

- `plot(x, y, 'format', Name, Value)`; always add title, xlabel, ylabel, legend, grid.
- hold on/off for multiple curves; axis, xlim, ylim, log axes, yyaxis.
- subplot/tiledlayout for several plots in one figure.
- bar, scatter, histogram, pie, stem, stairs, errorbar, fplot for other data.
- 3D: plot3, meshgrid + mesh/surf/contour, colorbar, view.
- Export with exportgraphics at 300 dpi or as vector PDF.

Practice questions — Module 4

1. Plot $y = e^{-0.5t} \sin(2\pi t)$ for $0 \leq t \leq 5$ with labels and a grid.
2. Plot the RC charging curve $v = 10(1 - e^{-t/2})$ for $0 \leq t \leq 10$ s and mark the point at $t = 2$ s.
3. Plot $\sin$, $\cos$ and $\tan$ (limit y to $-2..2$) on one axes with different line styles and a legend.
4. Create a 2×2 subplot showing x , x^2 , x^3 and $\sqrt{x}$ for $0 \leq x \leq 2$.
5. Draw a bar chart of five subjects' marks with subject names on the x-axis.
6. Draw a histogram of 1,000 random normal values (`randn`) with 30 bins.
7. Plot the surface $z = \sin(\sqrt{x^2 + y^2})$ over $-8..8$.
8. Export one of your figures as a 300-dpi PNG.

MODULE 05

Control Flow: Loops and Conditionals

Making decisions with if/elseif/else and switch, repeating work with for and while loops, break and continue, preallocation, and when a vectorised operation should replace a loop entirely.

5.1 Why control flow?

So far, scripts have run line by line from top to bottom. Real programs must **decide** (do this only if a condition is true) and **repeat** (do this for every student, every time step, until an error is small enough). These structures are called **control flow**. In MATLAB, every block ends with the keyword **end**.

5.2 if, elseif, else

Syntax

```
if condition1
    statements                % run when condition1 is true
elseif condition2
    statements                % run when condition1 false, condition2 true
else
    statements                % run when none are true
end
```

Worked example 5.1 — grade calculator

```
marks = input('Enter marks (0-100): ');
if marks < 0 || marks > 100
    disp('Invalid marks')
elseif marks >= 90
    grade = 'O';
elseif marks >= 80
    grade = 'A+';
elseif marks >= 70
    grade = 'A';
elseif marks >= 60
    grade = 'B';
elseif marks >= 40
    grade = 'C';
else
    grade = 'F';
end
if exist('grade', 'var')
    fprintf('Grade: %s\n', grade);
end
```

Order matters: conditions are tested from top to bottom, and the **first** true branch runs. Testing `marks >= 40` first would give every student above 40 a "C".

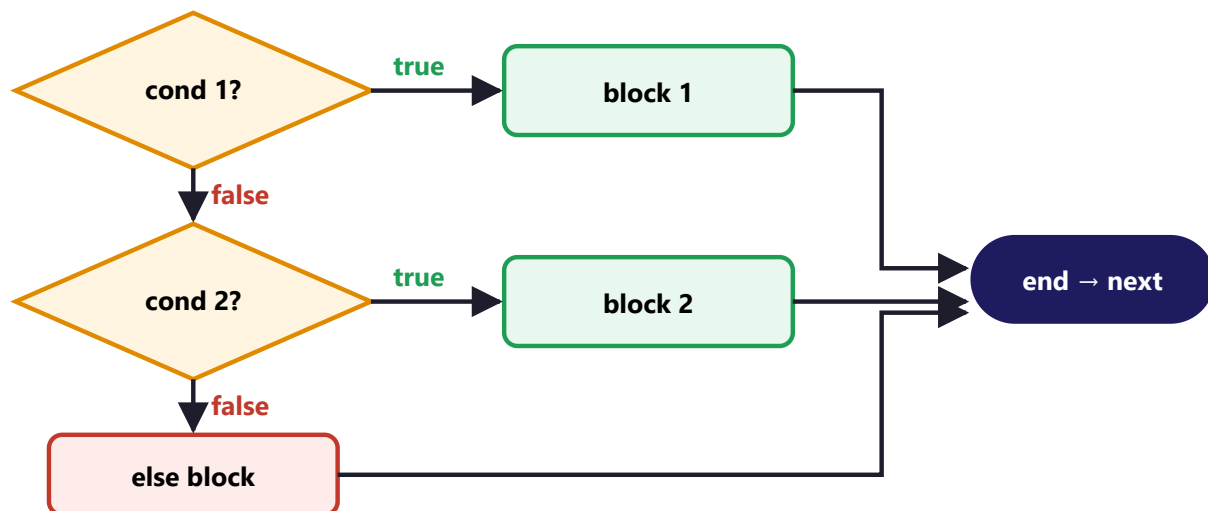


Figure 5.1 — Flow of an if / elseif / else statement.

Caution — conditions on vectors

`if v > 0` on a vector is true only if **all** elements are non-zero/true — often not what you mean. Use `any()` or `all()` explicitly, or logical indexing.

5.3 switch

`switch` is cleaner than a long if-elseif chain when you compare one variable against several specific values (numbers or text).

```
unit = input('Convert to (km/m/cm): ', 's');
d = 2.5;                                     % distance in km
switch lower(unit)
    case 'km'
        fprintf('%.2f km\n', d);
    case 'm'
        fprintf('%.0f m\n', d*1000);
    case {'cm', 'centimetre'}               % several matches
        fprintf('%.0f cm\n', d*1e5);
    otherwise
        disp('Unknown unit');
end
```

5.4 for loops

A **for** loop repeats a block a known number of times, taking each value of a vector in turn.

Syntax

```
for k = startValue:step:endValue
    statements
end
```

```
for k = 1:5
    fprintf('%d squared is %d\n', k, k^2);
end

% loop over any vector
for R = [100 220 470 1000]
    fprintf('R = %4d ohm, I = %.2f mA\n', R, 5/R*1000);
end
```

Worked example 5.2 — multiplication table and a running total

```
n = 7;
for k = 1:10
    fprintf('%2d x %2d = %3d\n', n, k, n*k);
end

% sum of the series 1 + 1/2^2 + 1/3^2 + ... (approaches pi^2/6)
total = 0;
for k = 1:1000
    total = total + 1/k^2;
end
fprintf('Sum = %.6f, pi^2/6 = %.6f\n', total, pi^2/6);
```

Output: Sum = 1.643935, $\pi^2/6$ = 1.644934 .

Nested loops

```
% build a 5x5 multiplication table matrix
T = zeros(5);
for r = 1:5
    for c = 1:5
        T(r, c) = r * c;
    end
end
disp(T)
```

5.5 while loops

A **while** loop repeats as long as a condition stays true — used when you don't know in advance how many repetitions are needed.

Syntax

```
while condition
    statements           % must eventually make the condition false!
end
```

Worked example 5.3 — how many years to double an investment?

```
amount = 10000; rate = 0.075; years = 0;
while amount < 20000
    amount = amount * (1 + rate);
    years = years + 1;
end
fprintf('Doubles after %d years (Rs %.0f)\n', years, amount);
```

Output: Doubles after 10 years (Rs 20610) .

Worked example 5.4 — input validation

```
age = -1;
while age < 0 || age > 120
    age = input('Enter a valid age (0-120): ');
end
```

Caution — infinite loops

If nothing inside a while loop changes the condition, it runs forever. Press **Ctrl + C** in the Command Window to stop it. Always ask: "What makes this loop end?"

5.6 break and continue

- `break` — exit the loop immediately.
- `continue` — skip the rest of this pass and go to the next one.

```

% find the first number above 100 whose square ends in 21
for n = 101:1000
    if mod(n^2, 100) == 21
        fprintf('Found %d (square %d)\n', n, n^2);
        break
    end
end

% print only odd numbers
for k = 1:10
    if mod(k, 2) == 0
        continue
    end
    fprintf('%d ', k);
end
fprintf('\n');

```

Output: Found 111 (square 12321) and 1 3 5 7 9.

5.7 Numerical example: an iterative solution

Worked example 5.5 — square root by Newton's method

To find $\sqrt{a}$, start with a guess x and repeat $x \leftarrow (x + a/x)/2$ until the change is tiny.

```

a = 50; x = a; tol = 1e-10; iter = 0;
while true
    xNew = (x + a/x) / 2;
    iter = iter + 1;
    if abs(xNew - x) < tol
        break
    end
    x = xNew;
end
fprintf('sqrt(%g) = %.10f after %d iterations (built-in: %.10f)\n', a, xNew, iter,
sqrt(a));

```

It converges in about 8 iterations to 7.0710678119 — the same as `sqrt(50)`. Module 8 generalises this idea to any equation.

5.8 Preallocation

Growing an array inside a loop (adding one element each time) forces MATLAB to repeatedly find new memory — slow for large loops. **Preallocate** the array at its final size first.

```

% slow: y grows each time
y = [];
for k = 1:100000
    y(k) = k^2;
end
% fast: preallocated
y = zeros(1, 100000);
for k = 1:100000
    y(k) = k^2;
end

```

5.9 Vectorisation — often no loop needed

Because MATLAB operates on whole arrays, many loops can be replaced by a single **vectorised** statement, which is shorter, clearer and much faster.

Task	Loop version	Vectorised version
Squares of 1..N	<pre>for k=1:N, y(k)=k^2; end</pre>	<code>y = (1:N).^2;</code>
Sum of $1/k^2$	loop with running total	<code>s = sum(1./(1:1000).^2);</code>
Count values above 50	loop + if + counter	<code>n = sum(v > 50);</code>
Replace negatives with 0	loop + if	<code>v(v < 0) = 0;</code>
Evaluate a function on a grid	nested loops	<code>[X,Y] = meshgrid(x,y); Z = X.^2 + Y.^2;</code>

Worked example 5.6 — timing the difference

```

N = 1e6;
tic; s1 = 0; for k = 1:N, s1 = s1 + sqrt(k); end; tLoop = toc;
tic; s2 = sum(sqrt(1:N)); tVec = toc;
fprintf('Loop: %.4f s, Vectorised: %.4f s\n', tLoop, tVec);

```

`tic / toc` measure elapsed time. The vectorised version is usually several times faster (exact times depend on your computer and MATLAB version).

Key idea — when to use a loop

Use vectorised code when each element is computed independently. Use a loop when each step depends on the previous one (iterations, simulations over time, "until" conditions) — like Newton's method or the investment example.

5.10 Worked mini-project: electricity tariff with slabs

Worked example 5.7 — slab tariff for many consumers

```
% Slabs: 0-100 units Rs 3; 101-200 Rs 4.5; 201-300 Rs 6; above 300 Rs 7.5
units = [80 150 260 420 305];
bill = zeros(size(units));
for k = 1:numel(units)
    u = units(k);
    if u <= 100
        b = u*3;
    elseif u <= 200
        b = 100*3 + (u-100)*4.5;
    elseif u <= 300
        b = 100*3 + 100*4.5 + (u-200)*6;
    else
        b = 100*3 + 100*4.5 + 100*6 + (u-300)*7.5;
    end
    bill(k) = b;
end
table(units', bill', 'VariableNames', {'Units','Bill_Rs'})
```

Bills: 80 → 240; 150 → 525; 260 → 1110; 420 → 2250; 305 → 1387.5.

Module 5 — Summary

- if / elseif / else / end — first true branch runs; use && and || for scalar conditions.
- switch / case / otherwise for matching one value against options.
- for k = a:b — known number of repetitions; while — until a condition changes.
- break exits; continue skips to the next pass; Ctrl + C stops runaway loops.
- Preallocate arrays with zeros() before loops.
- Vectorise independent element-wise work; loop when steps depend on earlier ones.

Practice questions — Module 5

1. Write an if statement that prints "Hot", "Warm" or "Cold" for temperatures ≥ 35 , 20–34 and below 20 °C.
2. Use switch to print the number of days in a month given its name.
3. Use a for loop to compute 10! and check with `factorial(10)` . (Ans: 3628800)
4. Use a while loop to find the smallest n such that $1 + 2 + \dots + n$ exceeds 1,000. (Ans: 45)
5. Print all prime numbers below 50 using nested loops (then compare with `primes(50)`).
6. Rewrite without a loop: count how many elements of v are between 10 and 20.
7. Why is preallocation important? Demonstrate with tic/toc.
8. Modify Worked example 5.7 to add a 5% electricity duty and a fixed charge of Rs 50.

MODULE 06

Writing Functions and Scripts

Scripts versus functions, writing function files with inputs and outputs, local functions, anonymous functions, variable scope, input checking, debugging, and organising a project into files.

6.1 Scripts versus functions

	Script	Function
What it is	A sequence of commands in a .m file	A reusable block with its own inputs and outputs
Variables	Share the base workspace (can overwrite your variables)	Have their own private workspace
Inputs/outputs	None (uses whatever is in the workspace)	Explicit arguments in and results out
Best for	Main programs, experiments, lab reports	Calculations you repeat: formulas, conversions, analyses

Key idea — "don't repeat yourself"

When you find yourself copying the same lines of code into several places, turn them into a function. Fix a bug once, and it's fixed everywhere.

6.2 Writing a function file

Syntax

```
function [out1, out2] = functionName(in1, in2)
% FUNCTIONNAME One-line summary (shown by help)
% Longer description of inputs and outputs.
% statements that compute out1 and out2
end
```

Save the file with **exactly the same name** as the function: `functionName.m`.

Worked example 6.1 — a simple function

```
% file: circle_props.m
function [A, C] = circle_props(r)
% CIRCLE_PROPS Area and circumference of a circle.
% [A, C] = circle_props(r) returns area A and circumference C
% for radius r (scalar or vector).
    A = pi * r.^2;
    C = 2 * pi * r;
end
```

Using it from the Command Window or a script:

```
>> [a, c] = circle_props(3)
a =
    28.2743
c =
    18.8496
>> area = circle_props([1 2 3]) % only the first output, for a vector
area =
    3.1416    12.5664    28.2743
>> help circle_props % shows your comment lines
```

Because the function uses `.^`, it works for vectors too — a good habit.

Multiple inputs and outputs

```
% file: ohms_law.m
function [I, P] = ohms_law(V, R)
% OHMS_LAW Current and power in a resistor.
    I = V ./ R;
    P = V .* I;
end

>> [I, P] = ohms_law(230, [46 23 115])
I = 5    10    2
P = 1150  2300  460
```

6.3 Variable scope

Variables created inside a function exist only while it runs, in the function's own workspace. The function sees only its input arguments — not the variables in your Command Window — and the caller receives only the output arguments.

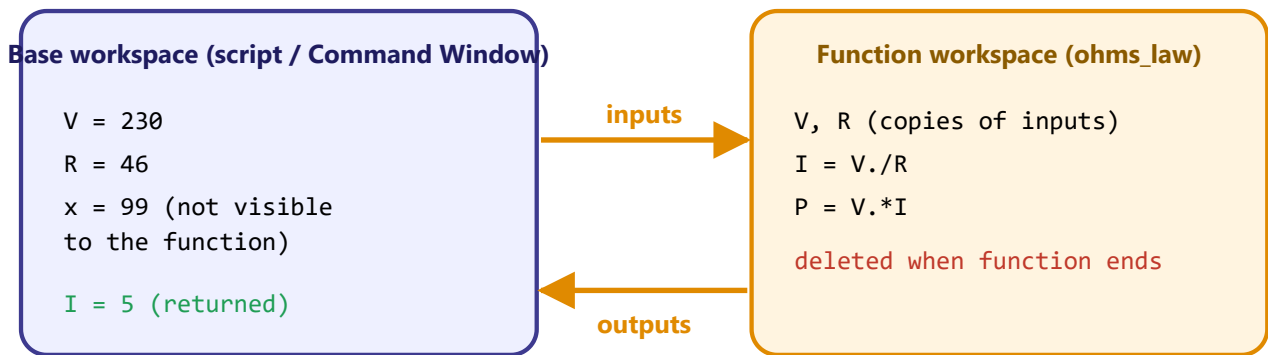


Figure 6.1 — Functions communicate only through input and output arguments.

This isolation is a feature: functions can't accidentally overwrite your variables, and the same variable names can be reused safely in different functions. (Global variables exist but are best avoided.)

6.4 Local functions and scripts with functions

A function file can contain additional **local functions** after the main one; they are visible only inside that file. Since R2016b, **scripts** can also contain local functions at the end of the file.

```
% file: analyse_marks.m (script with a local function)
marks = [72 45 88 39 91 60];
[g, avgMark] = summarise(marks);
fprintf('Average %.1f, grades: %s\n', avgMark, strjoin(g, ', '));

function [grades, avgMark] = summarise(m)
    avgMark = mean(m);
    grades = cell(size(m));
    for k = 1:numel(m)
        grades{k} = letter(m(k));
    end
end

function g = letter(x)
    if x >= 80, g = 'A';
    elseif x >= 60, g = 'B';
    elseif x >= 40, g = 'C';
    else, g = 'F';
    end
end
```

Output: Average 65.8, grades: B, C, A, F, A, B.

6.5 Anonymous functions

For short one-line formulas, an **anonymous function** avoids creating a file. It is stored in a variable called a **function handle** (the `@` symbol).

```
f = @(x) x.^2 - 4*x + 3;      % f(x) = x^2 - 4x + 3
f(0)                        % 3
f([1 2 3])                  % 0 -1 0
g = @(V, R) V.^2 ./ R;      % power from voltage and resistance
g(12, 4)                     % 36
c2f = @(c) c*9/5 + 32;
c2f(37)                      % 98.6
fplot(f, [-1 5]); grid on   % plot directly
r = fzero(f, 0.5)           % root near 0.5 → 1 (Module 8)
area = integral(f, 0, 3)    % definite integral → 0
```

Function handles can also point to named functions: `h = @sin; h(pi/2) → 1`. Many MATLAB tools (`fzero`, `integral`, `ode45`, `fminsearch`, `arrayfun`) take a function handle as input.

6.6 Checking inputs and giving defaults

```
function P = power_dissipated(I, R)
% POWER DISSIPATED P = I^2 R, with input checks.
    if nargin < 2
        R = 1; % default resistance
    end
    if any(R(:) <= 0)
        error('Resistance must be positive.');
```

- `nargin` / `nargout` — number of inputs/outputs actually passed.
- `error('message')` stops with a clear message; `warning('message')` continues.
- Newer MATLAB versions offer an `arguments` block for declaring input types, sizes and defaults.

6.7 Debugging

Bugs are normal; finding them systematically is a skill.

Tool	How to use
Error messages	Read the message and click the line link — it jumps to the problem

Code Analyzer	Orange/red underlines and the message bar in the Editor warn you before running
Breakpoints	Click the line number (a red dot appears). Run: execution pauses there (<code>⌘>></code> prompt)
Step, Step In, Continue	Execute line by line; enter a called function; run to next breakpoint
Inspect variables	Hover over a variable or look at the Workspace while paused
<code>dbstop if error</code>	Automatically pause where an error occurs
Print checks	Temporarily remove semicolons or add <code>disp / fprintf</code> to see intermediate values

Worked example 6.2 — find the bug

```
function avg = mean_positive(v)
    total = 0; count = 0;
    for k = 1:length(v)
        if v(k) > 0
            total = total + v(k);
        end
        count = count + 1;      % BUG
    end
    avg = total / count;
end
```

`mean_positive([4 -2 6])` should return 5 but returns 3.33. Stepping through shows `count` increases even for negative values — the line belongs **inside** the `if` block. Better still, vectorise: `avg = mean(v(v > 0));`

6.8 Organising a project

- One folder per project with sub-folders: `data/`, `functions/`, `results/`.
- A **main script** (e.g. `main.m`) that loads data, calls functions and saves results — anyone can re-run the whole analysis.
- Add folders to the search path: `addpath('functions')` (or right-click → Add to Path).
- Start scripts with `clc; clear; close all;` so results don't depend on leftover variables.
- Comment generously; include author, date, purpose and units.
- Use version control (Git, supported in MATLAB's Source Control) for larger projects.

- **MATLAB Projects** and **Live Scripts** help package code and reports.

Worked example 6.3 — a small engineering function library

```
% file: rc_charge.m
function v = rc_charge(t, V, R, C)
% RC_CHARGE Capacitor voltage v(t) = V(1 - exp(-t/(RC))).
    tau = R*C;
    v = V * (1 - exp(-t ./ tau));
end

% file: main.m
clc; clear; close all;
t = linspace(0, 5, 200);           % seconds
V = 10; C = 100e-6;
for R = [5e3 10e3 20e3]
    plot(t, rc_charge(t, V, R, C), 'LineWidth', 1.5); hold on
end
hold off; grid on
xlabel('Time (s)'); ylabel('v_C (V)'); title('RC charging for different R')
legend('R = 5 k\Omega', 'R = 10 k\Omega', 'R = 20 k\Omega', 'Location', 'southeast')
```

The time constants are 0.5 s, 1 s and 2 s; larger R charges more slowly. Note the TeX codes (`_C`, `\Omega`) that MATLAB renders in labels.

Module 6 — Summary

- Scripts share the base workspace; functions have private workspaces and explicit inputs/outputs.
- `function [outs] = name(ins) ... end`, saved as `name.m`; comment lines become help text.
- Local functions live at the end of a file; anonymous functions: `f = @(x) ...` for one-liners.
- Check inputs with `nargin`, `error`, `warning` (or arguments blocks).
- Debug with error links, Code Analyzer, breakpoints, stepping and `dbstop` if error.
- Organise projects: folders, a main script, `addpath`, comments, version control.

Practice questions — Module 6

1. Give three differences between scripts and functions.
2. Write a function `[F] = c2f(C)` that converts Celsius to Fahrenheit for vectors.
3. Write a function that returns the mean, median and standard deviation of a vector.
4. Write a function `series_R(R)` and `parallel_R(R)` that return the equivalent resistance of a vector of resistors.
5. Create an anonymous function for the quadratic $y = 2x^2 - 3x - 5$ and find its roots with `fzero`. (Ans: -1 and 2.5)
6. Explain variable scope with an example.
7. How do you set a breakpoint and step through a function?
8. Add input checking to `rc_charge` so that negative R or C gives an error.

MODULE 07

Working with Data Files (Import/Export)

Reading and writing CSV, Excel, text and MAT files, working with tables, cleaning messy real-world data — missing values, outliers, duplicates, dates — and summarising data before you analyse it.

7.1 Why data files?

Real data rarely comes typed into a script. It arrives as CSV files from data loggers, Excel sheets from colleagues, text files from instruments, or exports from websites and databases. MATLAB can read almost all of these, clean them, analyse them and write the results back out.

Format	Extension	Typical source	Read with	Write with
Comma-separated values	.csv	Data loggers, spreadsheets, websites	<code>readtable</code> , <code>readmatrix</code>	<code>writetable</code> , <code>writematrix</code>
Excel workbook	.xlsx, .xls	Office work, reports	<code>readtable</code> (with <code>Sheet</code>)	<code>writetable</code>
Plain text / tab-delimited	.txt, .dat	Instruments, old software	<code>readtable</code> , <code>readmatrix</code> , <code>fgetl</code>	<code>writematrix</code> , <code>fprintf</code>
MATLAB data	.mat	Saving variables between sessions	<code>load</code>	<code>save</code>
JSON	.json	Web APIs	<code>jsondecode(fileread(f))</code>	<code>jsonencode</code>
Images, audio	.png/.jpg, .wav/.mp3	Cameras, recorders	<code>imread</code> , <code>audioread</code>	<code>imwrite</code> , <code>audiowrite</code>

Try it — the Import Tool

On the Home tab, click **Import Data** (or double-click a file in the Current Folder). The Import Tool previews the file, lets you choose columns, data types and how to handle missing cells, and can **generate the MATLAB code** for you — a great way to learn the commands.

7.2 Tables — the best container for real data

A **table** stores column-oriented data where each column (variable) has a name and can have its own type — numbers, text, dates, categories — exactly like a spreadsheet.

```
T = readtable('students.csv')    % column names come from the header row
```

Suppose `students.csv` contains:

```
Name,Gender,Maths,Physics,Chemistry
Asha,F,78,65,90
Ravi,M,45,38,52
Sara,F,92,88,95
Tom,M,60,72,58
Uma,F,35,41,30
```

```
height(T), width(T)           % 5 rows, 5 columns
head(T, 3)                     % first 3 rows
T.Properties.VariableNames     % column names
T.Maths                        % one column as a vector
T(2, :)                        % row 2 as a table
T{2, 'Maths'}                  % the value 45 (curly braces extract data)
T.Total = T.Maths + T.Physics + T.Chemistry; % add a new column
T.Avg = round(T.Total / 3, 1);
T = sortrows(T, 'Total', 'descend') % sort by total
girls = T(T.Gender == "F", :) % filter rows
summary(T)                    % min/median/max of each column
```

Key idea — () versus {} versus . on tables

`T(rows, cols)` returns a smaller **table**. `T{rows, cols}` returns the **data inside** (a plain array). `T.Name` returns one whole column's data.

7.3 Reading files — details that matter

```
% numbers only (no header) into a matrix
M = readmatrix('readings.csv');

% a particular Excel sheet and range
T = readtable('results.xlsx', 'Sheet', 'Semester3', 'Range', 'A1:F61');

% control how the file is read
opts = detectImportOptions('log.csv');
opts.SelectedVariableNames = {'Time', 'Voltage', 'Current'};
opts = setvartype(opts, 'Time', 'datetime');
T = readtable('log.csv', opts);

% text file with a different delimiter and header lines
T = readtable('meter.txt', 'Delimiter', '\t', 'NumHeaderLines', 2);

% plain text lines
txt = fileread('notes.txt');           % whole file as one char array
lines = readlines('notes.txt');        % string array, one per line
```

Reading line by line (low-level file I/O)

```
fid = fopen('sensor.txt', 'r');          % open for reading
if fid == -1, error('Cannot open file'); end
while ~feof(fid)                        % until end of file
    line = fgetl(fid);
    if startsWith(line, '#'), continue; end % skip comments
    vals = sscanf(line, '%f,%f');
    % ... process vals ...
end
fclose(fid);                            % always close
```

7.4 Writing files

```
writetable(T, 'results.csv') % table to CSV
writetable(T, 'results.xlsx', 'Sheet', 'Summary') % to an Excel sheet
writematrix(M, 'matrix.txt', 'Delimiter', 'tab')
save('analysis.mat', 'T', 'M') % chosen variables
load('analysis.mat') % restores T and M

% formatted report with fprintf
fid = fopen('report.txt', 'w');
fprintf(fid, '%-10s %6s\n', 'Name', 'Total');
for k = 1:height(T)
    fprintf(fid, '%-10s %6d\n', T.Name{k}, T.Total(k));
end
fclose(fid);
```

Caution — overwriting files

`fopen(file, 'w')`, `writetable` and `save` replace an existing file of the same name without asking. Use a new name (e.g. with the date) for important outputs, or `'a'` (append) mode with `fopen`.

7.5 Cleaning messy real-world data

Real data is rarely clean. Before analysis, always inspect and clean it — otherwise results are wrong ("garbage in, garbage out").

Problem	Detect	Fix
Missing values (blank cells, "NA")	<code>ismissing(T)</code> , <code>sum(ismissing(T))</code>	<code>rmmissing(T)</code> (drop rows) or <code>fillmissing(T, 'linear')</code> / <code>'previous'</code> / constant
Outliers / impossible values	<code>isoutlier(x)</code> , plots, range checks	Remove, cap, or replace with NaN and fill
Duplicate rows	<code>[~, idx] = unique(T, 'rows')</code>	<code>T = unique(T, 'rows', 'stable')</code>
Wrong types (numbers read as text)	<code>class(T.col)</code> , <code>summary(T)</code>	<code>str2double</code> , <code>setvartype</code> in import options
Inconsistent text ("Male", "M", "male")	<code>unique(T.Gender)</code>	<code>strtrim</code> , <code>lower</code> , <code>replace</code> , then <code>categorical</code>

Dates in text form	<code>class(T.Date)</code>	<code>datetime(T.Date, 'InputFormat', 'dd-MM-yyyy')</code>
Units mixed (mA and A)	Check ranges	Convert to one unit
Noisy signals	Plot	<code>smoothdata(x, 'movmean', 5)</code>

Worked example 7.1 — cleaning a solar data-logger file

A CSV `solar_log.csv` has columns Time, Irradiance (W/m²), Power (W). Some rows are blank, the logger sometimes writes -999 for errors, and a few spikes are impossible.

```
T = readtable('solar_log.csv');
T.Time = datetime(T.Time, 'InputFormat', 'dd-MM-yyyy HH:mm');

% 1. error codes to missing
T.Power(T.Power == -999) = NaN;
T.Irradiance(T.Irradiance < 0 | T.Irradiance > 1400) = NaN; % physically impossible

% 2. report missing values
fprintf('Missing: %d irradiance, %d power\n', sum(isnan(T.Irradiance)), sum(isnan(T.Power)));

% 3. fill short gaps by linear interpolation
T.Power = fillmissing(T.Power, 'linear');
T.Irradiance = fillmissing(T.Irradiance, 'linear');

% 4. remove duplicate time stamps
[~, keep] = unique(T.Time, 'stable');
T = T(keep, :);

% 5. smooth and plot
T.PowerSmooth = smoothdata(T.Power, 'movmean', 5);
plot(T.Time, T.Power, ':', T.Time, T.PowerSmooth, 'LineWidth', 1.5)
ylabel('Power (W)'); legend('Raw', 'Smoothed'); grid on

% 6. daily energy (Wh) with 15-minute samples
energy_Wh = sum(T.Power) * 0.25;
fprintf('Energy today = %.2f kWh\n', energy_Wh/1000);
```

7.6 Dates and times

```
t = datetime('now')
d = datetime(2026, 9, 20, 14, 30, 0)
d + days(10) % arithmetic with durations
between(datetime(2026,1,1), d) % calendar difference
dt = minutes(15); t2 = d + dt
month(d), day(d), weekday(d)
d.Format = 'dd-MMM-yyyy' % display format
TT = table2timetable(T, 'RowTimes', 'Time');
daily = retime(TT, 'daily', 'sum'); % resample: daily totals
hourly = retime(TT, 'hourly', 'mean');
```

Timetables are tables indexed by time — ideal for logged data. `retime` aggregates or resamples them in one line.

7.7 Summarising and grouping

```
mean(T.Maths), std(T.Maths), max(T.Maths)
groupsummary(T, 'Gender', 'mean', {'Maths','Physics','Chemistry'})
% counts per category
T.Result = categorical(repmat("Pass", height(T), 1));
T.Result(T.Avg < 40) = "Fail";
countcats(T.Result)
histogram(T.Avg, 5) % distribution of averages
```

Worked example 7.2 — class report from an Excel file

```
T = readtable('students.csv');
T.Total = sum(T{:, {'Maths','Physics','Chemistry'}}, 2);
T.Percent = round(T.Total / 300 * 100, 1);
grades = strings(height(T), 1);
grades(T.Percent >= 80) = "A";
grades(T.Percent >= 60 & T.Percent < 80) = "B";
grades(T.Percent >= 40 & T.Percent < 60) = "C";
grades(T.Percent < 40) = "F";
T.Grade = grades;
T = sortrows(T, 'Percent', 'descend');
writetable(T, 'class_report.xlsx');
disp(T(:, {'Name','Total','Percent','Grade'}))
```

Result (sorted): Sara 275 (91.7, A); Asha 233 (77.7, B); Tom 190 (63.3, B); Ravi 135 (45.0, C); Uma 106 (35.3, F).

7.8 Good data habits

- **Never edit the raw data file.** Keep it untouched in `data/raw`; write cleaned versions to `data/clean`. Your script documents every cleaning step and can be re-run.
- Record units in column names: `Power_W`, `Temp_C`.
- Plot the data first — most problems are obvious in a picture.
- Check row counts before and after each cleaning step.
- Use relative paths (`fullfile('data', 'raw', 'log.csv')`) so the project works on any computer.

Module 7 — Summary

- `readtable/readmatrix` and `writetable/writematrix` for CSV, Excel and text; `save/load` for `.mat`.
- Tables: named columns of mixed types; `T.col`, `T(rows,:)`, `T{r,c}`; `sortrows`, filtering, summary.
- Import options and the Import Tool control types, columns and missing values.
- Clean data: `ismissing/fillmissing/rmmissing`, `isoutlier`, `unique`, type fixes, text standardising, `datetime`.
- Timetables + `retime` for time-series; `groupsummary` for group statistics.
- Keep raw data untouched; script every cleaning step.

Practice questions — Module 7

1. Name the functions to read and write CSV and Excel files as tables.
2. What is the difference between `T(2,:)`, `T{2,3}` and `T.Maths`?
3. Write code to add a column "Total" to a table with columns Q1, Q2, Q3.
4. How would you find and count missing values in every column of a table?
5. A temperature log has occasional readings of 999. Write code to replace them with interpolated values.
6. Convert text dates like "20-09-2026" to `datetime`.
7. Use `groupsummary` to find average marks by gender.
8. Why should raw data files never be edited by hand?

Numerical Methods and Curve Fitting

Finding roots of equations, numerical integration and differentiation, polynomials, interpolation, fitting curves to noisy data with polyfit and the Curve Fitter, optimisation, and solving differential equations with ode45.

8.1 Why numerical methods?

Many engineering equations cannot be solved exactly with algebra — for example $x = \cos x$, or the current in a diode circuit. Numerical methods find answers to **any accuracy you want** by systematic approximation. MATLAB provides fast, reliable built-in functions for all the classic methods; understanding how they work helps you use them correctly and spot when results are wrong.

8.2 Roots of equations

A **root** of $f(x)$ is a value where $f(x) = 0$ (where the graph crosses the x-axis). Always **plot the function first** to see roughly where the roots are.

The bisection method (the idea)

If $f(a)$ and $f(b)$ have opposite signs, a root lies between them. Halve the interval, keep the half where the sign changes, and repeat. Each step halves the error — slow but guaranteed.

```
% bisection for f(x) = x^3 - x - 2 on [1, 2]
f = @(x) x.^3 - x - 2;
a = 1; b = 2; tol = 1e-6; n = 0;
while (b - a)/2 > tol
    c = (a + b)/2; n = n + 1;
    if f(a)*f(c) < 0
        b = c;
    else
        a = c;
    end
end
fprintf('Root = %.6f after %d steps\n', (a+b)/2, n);
```

Output: Root = 1.521380 after 19 steps .

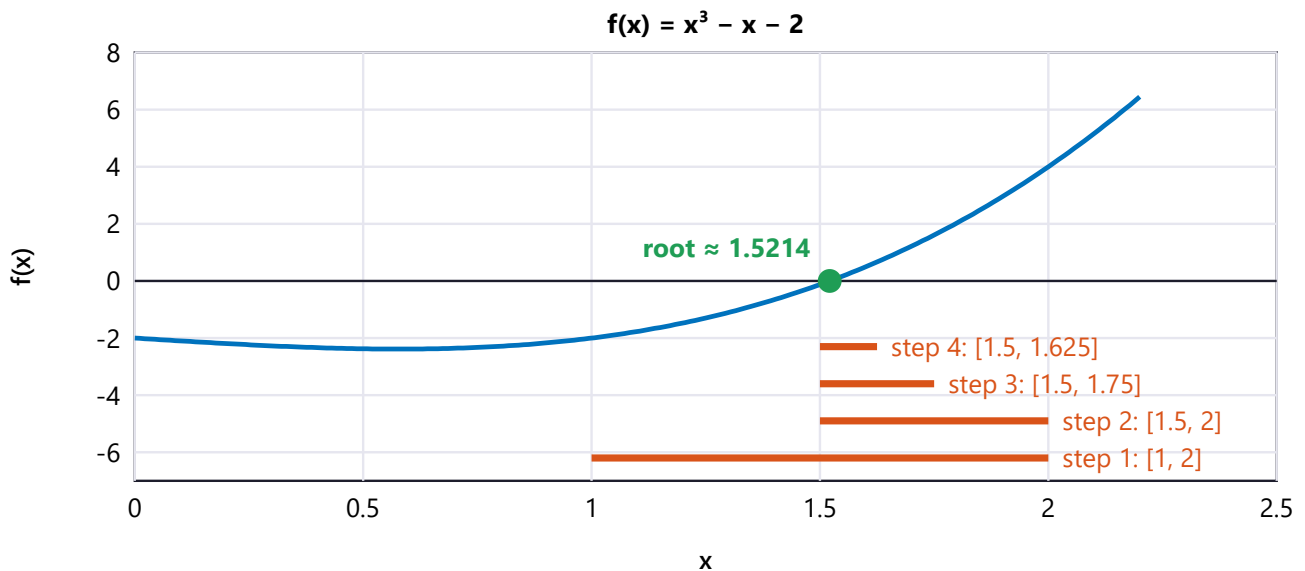


Figure 8.1 — Bisection keeps the half-interval where f changes sign; each step halves it.

Newton–Raphson method

Uses the slope to jump towards the root: $x_{\text{new}} = x - f(x)/f'(x)$. Very fast (the number of correct digits roughly doubles each step) but needs the derivative and a reasonable starting guess.

```
f = @(x) x.^3 - x - 2;
df = @(x) 3*x.^2 - 1;
x = 1.5;
for k = 1:6
    x = x - f(x)/df(x);
    fprintf('iter %d: x = %.10f\n', k, x);
end
```

It reaches 1.5213797068 in about 3–4 iterations.

Built-in root finders

```
r = fzero(f, 1.5)           % one root near a guess → 1.5214
r = fzero(f, [1 2])        % root inside an interval
g = @(x) x - cos(x);
fzero(g, 0.5)              % 0.7391 (solves x = cos x)
p = [1 0 -1 -2];          % polynomial x^3 - x - 2
roots(p)                   % all 3 roots (1 real, 2 complex)
% systems of nonlinear equations: fsolve (Optimization Toolbox)
```

Worked example 8.1 — diode circuit operating point

A 5 V source, 1 k Ω resistor and a diode with $I = I_S(e^{V/V_T} - 1)$, $I_S = 10^{-12}$ A, $V_T = 0.02585$ V.
KVL: $5 - 1000 \cdot I(V) - V = 0$.

```
Is = 1e-12; VT = 0.02585; R = 1000; Vs = 5;
h = @(V) Vs - R*Is*(exp(V/VT) - 1) - V;
Vd = fzero(h, [0.3 0.9]);
Id = (Vs - Vd)/R;
fprintf('Vd = %.4f V, Id = %.3f mA\n', Vd, Id*1000);
```

Output: about $v_d = 0.5741$ V, $I_d = 4.426$ mA — close to the "0.7 V rule of thumb" but more exact for this diode model.

8.3 Polynomials

MATLAB represents a polynomial by its coefficient vector, highest power first: $3x^2 - 2x + 5 \rightarrow [3 \ -2 \ 5]$.

Function	Does	Example
<code>polyval(p, x)</code>	Evaluate	<code>polyval([3 -2 5], 2) → 13</code>
<code>roots(p)</code>	All roots	<code>roots([1 -5 6]) → 3, 2</code>
<code>poly(r)</code>	Polynomial from roots	<code>poly([2 3]) → [1 -5 6]</code>
<code>conv(p, q)</code>	Multiply	<code>conv([1 1], [1 -1]) → [1 0 -1]</code>
<code>[q, r] = deconv(p, d)</code>	Divide	quotient and remainder
<code>polyder(p)</code> , <code>polyint(p)</code>	Derivative, integral	<code>polyder([1 0 -1 -2]) → [3 0 -1]</code>

8.4 Numerical integration

The definite integral $\int_a^b f(x) dx$ is the area under the curve. Numerically, we split the area into thin strips.

- **Trapezoidal rule:** strips are trapezoids. `trapz(x, y)` works on data points (e.g. measured values).
- **Simpson's rule:** fits parabolas through triples of points — more accurate for smooth functions.

- **Adaptive quadrature:** `integral(f, a, b)` automatically refines strips until the answer is accurate — use this when you have a formula.

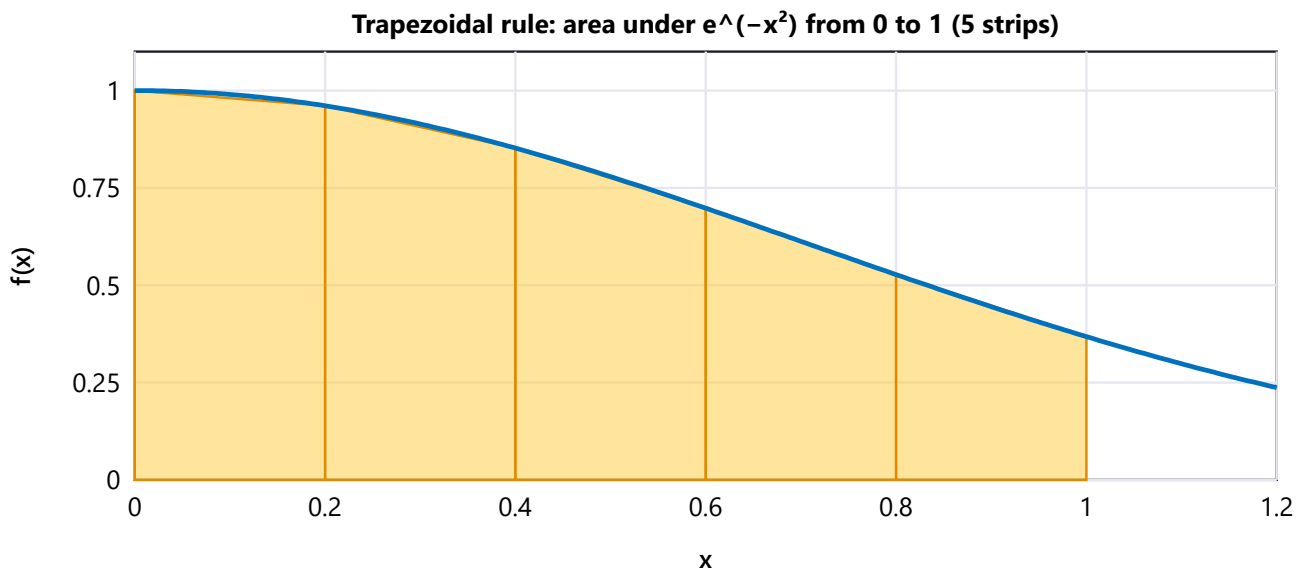


Figure 8.2 — Each strip is a trapezoid; more strips give a more accurate area.

```
f = @(x) exp(-x.^2);
integral(f, 0, 1)           % 0.746824 (accurate)
x = linspace(0, 1, 11);    % 10 strips
trapz(x, f(x))             % 0.746211 (close)
% double integral: integral2(@(x,y) x.*y, 0, 1, 0, 2) → 1
```

Worked example 8.2 — energy from a measured power curve

A solar inverter logged power (kW) every hour from 6 am to 6 pm:

```
t = 6:18;                                % hours
P = [0 0.3 0.9 1.6 2.2 2.6 2.8 2.7 2.3 1.7 1.0 0.4 0]; % kW
E = trapz(t, P);                          % kWh
fprintf('Energy generated = %.2f kWh\n', E);
```

Output: Energy generated = 18.50 kWh. (Energy is the area under the power–time curve.)

8.5 Numerical differentiation

The derivative is the slope. From data, approximate it with differences:

```
t = 0:0.5:5; s = 4.9*t.^2;           % distance of a falling object
v = diff(s) ./ diff(t);             % forward differences (one shorter)
v2 = gradient(s, 0.5);              % central differences, same length
% true velocity is 9.8 t; gradient() is exact here for a quadratic (interior points)
```

Caution — differentiation amplifies noise

Small measurement noise becomes large noise in the derivative. Smooth the data first (`smoothdata`) or fit a curve and differentiate the fit.

8.6 Interpolation

Interpolation estimates values **between** known data points — for example, reading a value from a table of properties.

```
T = [0 20 40 60 80 100];           % temperature (°C)
rho = [999.8 998.2 992.2 983.2 971.8 958.4]; % water density (kg/m^3)
interp1(T, rho, 35)                 % linear → 993.7
interp1(T, rho, 35, 'spline')       % smooth cubic spline
interp1(T, rho, 35, 'pchip')        % shape-preserving
% 2D tables: interp2; scattered data: griddata / scatteredInterpolant
```

Interpolation versus curve fitting

Interpolation passes **exactly through** every data point — use it for accurate tabulated data. **Curve fitting** finds a smooth model that passes **close to** noisy data — use it for measurements with errors. Neither should be trusted far outside the data range (**extrapolation**).

8.7 Curve fitting with polyfit

`p = polyfit(x, y, n)` finds the polynomial of degree n that best fits the data in the **least-squares** sense — it minimises the sum of squared vertical distances between the data and the curve.

Worked example 8.3 — fitting noisy lab data

Voltage–current readings from a resistor (true $R \approx 47\ \Omega$) with measurement noise:

```
I = [0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16]; % A
V = [0.98 1.92 2.79 3.80 4.67 5.70 6.55 7.58]; % V
p = polyfit(I, V, 1); % straight line V = p(1)*I + p(2)
fprintf('R = %.2f ohm, offset = %.3f V\n', p(1), p(2));
Vfit = polyval(p, I);
R2 = 1 - sum((V - Vfit).^2) / sum((V - mean(V)).^2); % goodness of fit
fprintf('R^2 = %.4f\n', R2);
plot(I, V, 'o', I, Vfit, '-'); grid on
xlabel('Current (A)'); ylabel('Voltage (V)'); legend('Measured', 'Linear fit')
```

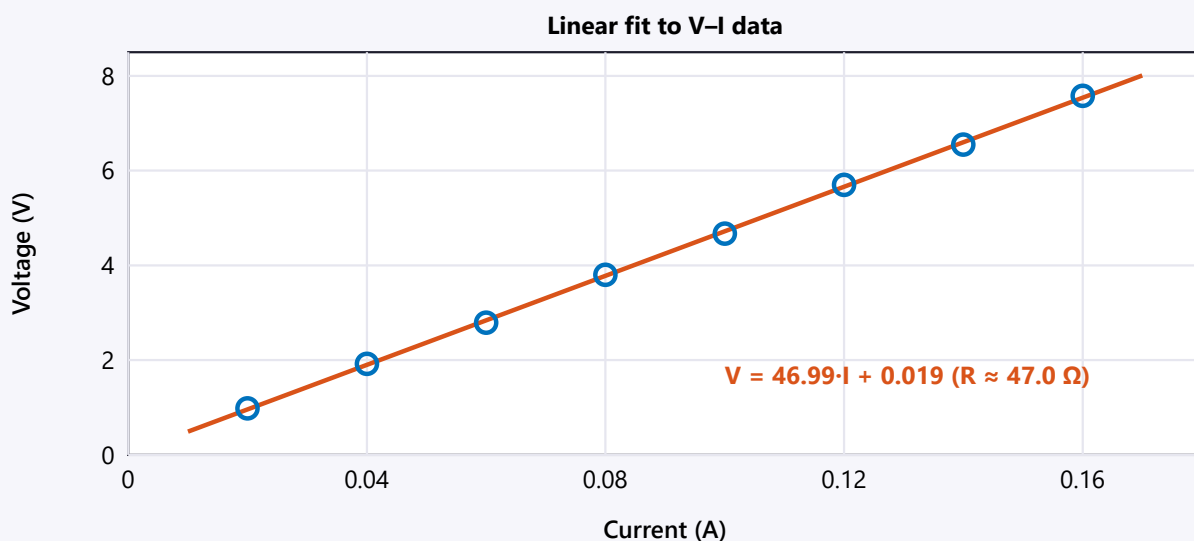


Figure 8.3 — Measured points (circles) and the least-squares straight line.

The slope gives the resistance $\approx 47\ \Omega$, and R^2 very close to 1 shows an excellent linear fit.

Choosing the degree

- Use the lowest degree that matches the physics: linear for Ohm's law, quadratic for projectile height, etc.
- High degrees (> 4 – 5) can wiggle wildly between points — **overfitting**.
- Look at the **residuals** (data – fit): they should look random, with no pattern.

Fitting non-polynomial models by transforming

Exponential models $y = a \cdot e^{bx}$ become straight lines if you take logs: $\ln y = \ln a + bx$. Fit a line to $(x, \ln y)$:

```
t = 0:1:6; v = [10.0 6.1 3.7 2.2 1.4 0.8 0.5]; % capacitor discharge
p = polyfit(t, log(v), 1);
tau = -1/p(1); V0 = exp(p(2));
fprintf('V0 = %.2f V, time constant = %.2f s\n', V0, tau); % ≈ 10 V, ≈ 2 s
```

Curve Fitter app and fit()

The **Curve Fitter** app (Apps tab, Curve Fitting Toolbox) lets you choose models interactively — polynomial, exponential, power, Fourier, Gaussian, custom equations — see goodness-of-fit statistics and residual plots, and generate code. From the command line: `f = fit(x', y', 'exp1')`. Without the toolbox, `fminsearch` or `lsqcurvefit` (Optimization Toolbox) can fit any custom model.

8.8 Optimisation (finding minimum or maximum)

```
% maximum power transfer: P = V^2 RL/(Rs+RL)^2 with V = 12, Rs = 8
P = @(RL) 12^2*RL ./ (8 + RL).^2;
[RLbest, negP] = fminbnd(@(RL) -P(RL), 0.1, 50);
fprintf('Best RL = %.2f ohm, Pmax = %.2f W\n', RLbest, -negP); % 8 ohm, 4.5 W
% several variables: fminsearch(@(v) ..., v0)
```

8.9 Ordinary differential equations (ODEs)

Many physical systems are described by how things **change**: $dy/dt = f(t, y)$. MATLAB's workhorse solver is **ode45** (Runge–Kutta 4/5 with automatic step size).

Worked example 8.4 — RC charging from its differential equation

The capacitor voltage obeys $dv/dt = (V_s - v)/(RC)$.

```
Vs = 10; R = 10e3; C = 100e-6; % tau = 1 s
dvdt = @(t, v) (Vs - v)/(R*C);
[t, v] = ode45(dvdt, [0 5], 0); % from t = 0 to 5 s, v(0) = 0
plot(t, v, 'LineWidth', 1.5); hold on
plot(t, Vs*(1 - exp(-t/(R*C))), 'r--'); hold off % exact solution
legend('ode45', 'exact'); xlabel('t (s)'); ylabel('v (V)'); grid on
```

The numerical and exact curves overlap — the solver is accurate. The same approach handles equations with no neat formula, such as a pendulum with large swings or a nonlinear circuit.

Worked example 8.5 — a second-order system (mass–spring–damper)

$m x'' + c x' + k x = 0$ is rewritten as two first-order equations with $y = [x; x']$:

```
m = 1; c = 0.5; k = 4;
f = @(t, y) [y(2); -(c*y(2) + k*y(1))/m];
[t, y] = ode45(f, [0 20], [1; 0]); % start at x = 1, at rest
plot(t, y(:,1)); xlabel('t (s)'); ylabel('x (m)'); title('Damped oscillation'); grid
on
```

Module 8 — Summary

- Roots: plot first; bisection (safe), Newton (fast); fzero for functions, roots for polynomials.
- Polynomials as coefficient vectors: polyval, roots, poly, conv, polyder, polyint.
- Integration: integral(f,a,b) for formulas, trapz(x,y) for data.
- Differentiation: diff, gradient — smooth noisy data first.
- interp1 passes through points; polyfit fits noisy data (least squares); check R^2 and residuals; avoid overfitting.
- Optimisation with fminbnd/fminsearch; ODEs with ode45.

Practice questions — Module 8

1. Find the root of $e^x = 3x$ between 0 and 1 using fzero. (Ans: ≈ 0.6191)
2. Write a bisection loop for $x^2 - 7 = 0$ on $[2, 3]$ to 6 decimal places. (Ans: 2.645751)
3. Find all roots of $x^3 - 6x^2 + 11x - 6$. (Ans: 1, 2, 3)
4. Compute $\int_0^\pi \sin x \, dx$ with integral and with trapz using 20 strips. (Ans: 2; ≈ 1.9959)
5. Interpolate the water density at 55 °C from the table in 8.6 (linear). (Ans: 985.45 kg/m³)
6. Fit a quadratic to $x = 0:5$, $y = [1.1 \ 2.9 \ 9.2 \ 19.1 \ 32.8 \ 51.0]$ and predict y at 6.
7. Explain the difference between interpolation and curve fitting.
8. Solve $dy/dt = -2y + 1$, $y(0) = 0$ with ode45 and compare with $y = (1 - e^{-2t})/2$.

MODULE 09

Simulink Basics (Intro to Simulation)

What Simulink is, the library browser and common blocks, building and running your first model, solver settings, reading scopes, subsystems, and modelling an RC circuit, a motor speed control loop and a simple solar–battery system.

9.1 What is Simulink?

Simulink is a graphical environment inside MATLAB for modelling, simulating and analysing **dynamic systems** — systems whose behaviour changes over time. Instead of writing equations as code, you connect **blocks** (integrators, gains, sums, sources, scopes) with **signal lines**, like drawing a block diagram in a textbook. Simulink then solves the underlying differential equations for you.

Used for	Examples
Control systems	Motor speed control, cruise control, temperature control, drones
Electrical & power	Circuits, converters, inverters, motor drives, solar/battery systems (Simscape Electrical)
Mechanical	Suspensions, robots, vehicle dynamics
Signal processing & communications	Filters, modulators
Embedded systems	Automatic C code generation for microcontrollers (model-based design)

Note on access

Simulink is a separate MathWorks product included in most campus licences and MATLAB Online. GNU Octave does **not** include Simulink (the free Xcos in Scilab is a similar tool). Everything else in this course runs without Simulink.

9.2 Starting Simulink

1. Type `simulink` in the Command Window (or click **Simulink** on the Home tab).
2. Choose **Blank Model**. A model window opens (saved as a **.slx** file).
3. Open the **Library Browser** (toolbar button) — or simply click on the canvas and start typing a block name (quick insert).

4. Drag blocks onto the canvas; connect an output port (► on the right of a block) to an input port by dragging a line.
5. Double-click a block to set its **parameters**.
6. Set the **Stop Time** in the toolbar and press **Run ►**.
7. Double-click the **Scope** to see the results.

9.3 Commonly used blocks

Library	Block	What it does
Sources	Constant	Fixed value
	Step	Jumps from one value to another at a set time (switching on)
	Sine Wave	Sinusoid with amplitude, frequency, phase
	Pulse Generator, Ramp	Square pulses; steadily rising signal
	Clock, From Workspace	Simulation time; data from MATLAB variables
Math Operations	Gain	Multiplies by a constant
	Sum / Add	Adds or subtracts signals (set signs like +-)
	Product	Multiplies or divides signals
	Math Function, Abs, Saturation	Squares, exponentials; limits
Continuous	Integrator (1/s)	Integrates its input over time — the heart of differential equations
	Transfer Fcn	A linear system given as a ratio of polynomials in s
	PID Controller	Proportional–integral–derivative controller
Sinks	Scope	Plots signals against time
	To Workspace	Sends data to MATLAB for further analysis
	Display	Shows a numerical value
Signal Routing	Mux, Demux, Switch, Goto/From	Combine, split and route signals

9.4 Your first model: sine wave × gain

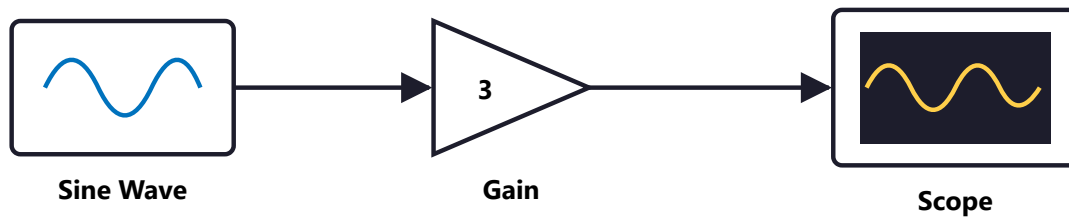


Figure 9.1 — The simplest model: a sine wave amplified 3 times and displayed on a scope.

1. Add **Sine Wave** (Amplitude 1, Frequency 2π rad/s $\rightarrow$ 1 Hz), **Gain** (3) and **Scope**.
2. Connect them in order. Set Stop Time = 5 s. Run.
3. The Scope shows a sine wave of amplitude 3 with 5 cycles.

9.5 Solving a differential equation with blocks

The key trick: if you know the derivative of a quantity, an **Integrator** block produces the quantity itself. So you build the right-hand side of the equation, feed it into an integrator, and loop the integrator's output back.

Worked example 9.1 — RC charging circuit

Equation: $dv/dt = (V_s - v)/(RC)$, with $V_s = 10$ V and $RC = 1$ s.

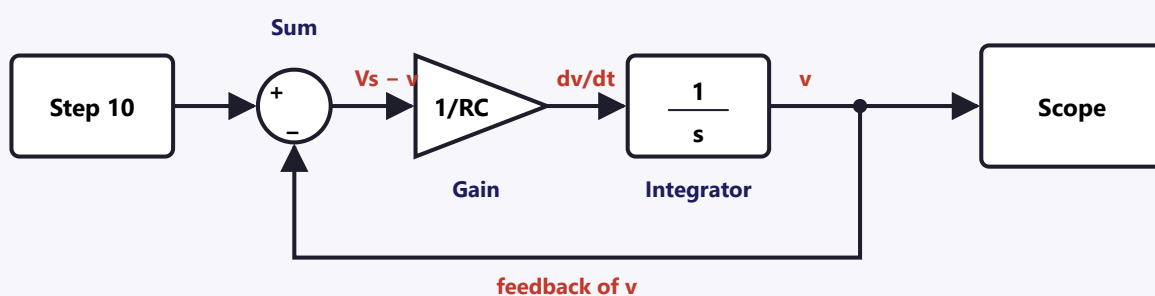


Figure 9.2 — Block diagram for $dv/dt = (V_s - v)/RC$.

Run for 5 s: the Scope shows v rising from 0 towards 10 V, reaching 6.32 V at 1 s — the same curve as the formula and ode45 in Module 8.

9.6 Solver and simulation settings

Open **Model Settings** (Ctrl + E) → **Solver**:

Setting	Meaning / advice
Start / Stop time	Simulation time range (not real time)
Solver type: variable-step	Adjusts step size automatically (default ode45) — best for most continuous models
Solver type: fixed-step	Constant step — needed for code generation and real-time hardware
Stiff solvers (<code>ode15s</code> , <code>ode23t</code>)	For systems with very fast and very slow parts together (e.g. power electronics); if a simulation is extremely slow, try a stiff solver
Max step size / tolerances	Reduce if curves look jagged or miss fast events

9.7 Transfer functions and a control loop

Engineers often describe linear systems by a **transfer function** $G(s)$ — the ratio of output to input in the Laplace domain. A DC motor's speed response to voltage can be approximated as a first-order system $G(s) = K/(\tau s + 1)$.

Worked example 9.2 — motor speed control with a PI controller

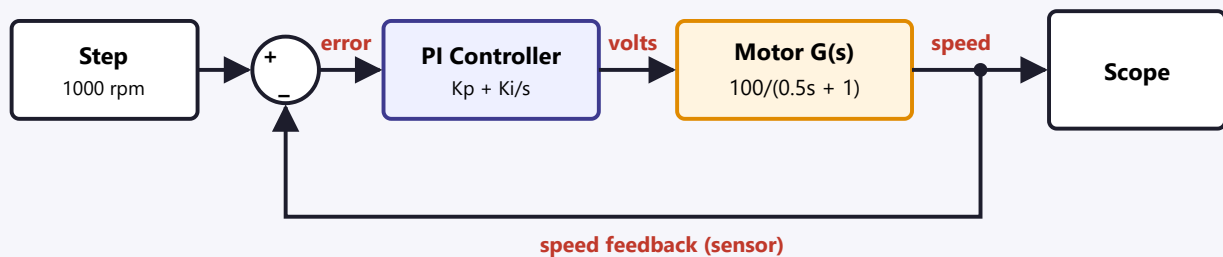


Figure 9.3 — Closed-loop speed control: the controller adjusts voltage to reduce the error between desired and actual speed.

- Reference: Step to 1000 rpm at $t = 0$. Plant: Transfer Fcn with numerator [100], denominator [0.5 1].
- Try P-only control ($K_p = 0.1$, $K_i = 0$): the speed settles **below** 1000 rpm — a steady-state error.
- Add integral action ($K_p = 0.05$, $K_i = 0.2$): the error goes to zero; the Scope shows speed rising smoothly to 1000 rpm.
- The PID block's **Tune** button can choose gains automatically for a desired response time.

Key idea — feedback

Open loop: apply a fixed input and hope. **Closed loop:** measure the output, compare it with the goal, and correct — like a driver watching the speedometer. Feedback makes systems accurate and robust to disturbances (a hill, a load change).

Reading a step response

Term	Meaning
Rise time	Time to go from 10% to 90% of the final value
Overshoot	How far it goes past the final value (%)
Settling time	Time to stay within $\pm 2\%$ of the final value
Steady-state error	Difference between the goal and the final value

9.8 Subsystems and model organisation

- Select several blocks → right-click → **Create Subsystem from Selection** — group them into one block with a clear name (e.g. "Battery model").
- Use **Goto/From** tags to avoid long crossing lines.
- Name signals (double-click a line) and blocks meaningfully; add annotations.
- Use **MATLAB variables** for parameters (type `R`, `C` in blocks and define them in a script or the Model Workspace) — change a value in one place.
- Send results to MATLAB with **To Workspace** or signal logging, then analyse and plot with scripts.

9.9 Physical modelling with Simscape (preview)

For circuits and machines, **Simscape** (and Simscape Electrical) lets you connect physical components — resistors, capacitors, diodes, MOSFETs, motors, batteries, PV arrays — with real electrical connections instead of writing equations. Every Simscape model needs a **Solver Configuration** block and an **Electrical Reference** (ground).

Worked example 9.3 — conceptual solar + battery energy model (signal-based)

A simple energy-balance model you can build with basic blocks:

- **PV power**: a Sine Wave (or From Workspace irradiance profile) → Gain (array kWp × PR) → Saturation (no negative power at night).
- **Load**: a Constant 0.5 kW plus a Pulse Generator for evening peaks.
- **Sum**: PV – Load = net power into the battery.
- **Integrator** with limits (0 to battery capacity, e.g. 5 kWh) → battery energy (state of charge).
- **Scope**: plot PV, load and state of charge over 48 hours.

Change array size or battery capacity and re-run to see whether the battery ever empties — a quick way to test the sizing rules from the Solar Energy course.

9.10 Running Simulink from MATLAB scripts

```
open_system('rc_model')           % open a saved model
R = 10e3; C = 100e-6;             % parameters used by blocks
out = sim('rc_model', 'StopTime', '5'); % run it
v = out.simout;                   % data from a To Workspace block
% parameter sweep
for Rk = [5e3 10e3 20e3]
    R = Rk;
    out = sim('rc_model');
    plot(out.tout, out.simout); hold on
end
hold off; grid on; legend('5k', '10k', '20k')
```

9.11 Common Simulink problems

Problem	Fix
Red dashed line / unconnected port warning	Connect or terminate every port (Terminator / Ground blocks)
"Algebraic loop" warning	A signal depends on itself instantly — add a delay/integrator or restructure
Simulation very slow	Try a stiff solver; increase tolerances slightly; reduce scope logging
Jagged curves	Reduce max step size or use Refine output
Undefined variable in block	Define it in the MATLAB workspace or model workspace before running

Module 9 — Summary

- Simulink models dynamic systems with blocks and signal lines; files are .slx.
- Key blocks: sources (Step, Sine), math (Gain, Sum), Integrator, Transfer Fcn, PID, Scope, To Workspace.
- Build differential equations by feeding the derivative into an Integrator and feeding back.
- Choose solvers (ode45 default; stiff solvers for hard problems); set stop time.
- Closed-loop control with a PI/PID controller removes steady-state error; read rise time, overshoot, settling.
- Organise with subsystems, variables and To Workspace; Simscape for physical components; `sim()` from scripts.

Practice questions — Module 9

1. What is Simulink, and how does it differ from writing MATLAB scripts?
2. Name the blocks needed to display $5\sin(2\pi t)$ and add 2 V DC offset.
3. Draw the Simulink diagram for $dy/dt = -3y + 6$, $y(0) = 0$. What is the final value?
(Ans: 2)
4. Explain why an Integrator block is central to modelling differential equations.
5. What is the difference between open-loop and closed-loop control?
6. Define rise time, overshoot and settling time.
7. When would you choose a fixed-step solver?
8. How can you run a Simulink model several times with different parameter values?

MODULE 10

Practical Project and Final Revision

A complete data-analysis project — from a raw data file to a cleaned dataset, statistics, curve fits, plots and a written report — followed by your capstone brief and a one-page revision of the whole course.

10.1 The project: analysing a rooftop solar system's performance

A school installed a 5-kWp rooftop solar system. Its inverter logs data every 15 minutes to a CSV file for one month. The principal asks: *"How much energy did we generate? Is the system performing as expected? When is generation highest? Does temperature reduce output?"* We will answer with MATLAB.

The data file (first lines of `solar_sept.csv`)

```
Timestamp,Irradiance_Wm2,ModuleTemp_C,ACPower_W
01-09-2026 06:00,12,24.1,0
01-09-2026 06:15,35,24.6,110
01-09-2026 06:30,70,25.3,265
...
01-09-2026 12:00,905,56.8,3710
...
```



Figure 10.1 — The data-analysis workflow used in the project.

10.2 Step 1 — Import

```
%% main_solar_analysis.m – Solar system performance, September 2026
clc; clear; close all;
raw = readtable(fullfile('data', 'solar_sept.csv'));
raw.Timestamp = datetime(raw.Timestamp, 'InputFormat', 'dd-MM-yyyy HH:mm');
fprintf('Rows read: %d\n', height(raw));
summary(raw)
```

10.3 Step 2 — Clean

```
T = raw;
% impossible values → missing
T.Irradiance_Wm2(T.Irradiance_Wm2 < 0 | T.Irradiance_Wm2 > 1400) = NaN;
T.ACPower_W(T.ACPower_W < 0 | T.ACPower_W > 5500) = NaN;
T.ModuleTemp_C(T.ModuleTemp_C < -5 | T.ModuleTemp_C > 90) = NaN;
fprintf('Missing after checks: %d\n', sum(ismissing(T), 'all'));
% fill short gaps; remove duplicate timestamps
T = fillmissing(T, 'linear', 'DataVariables',
{'Irradiance_Wm2', 'ModuleTemp_C', 'ACPower_W'});
[~, keep] = unique(T.Timestamp, 'stable'); T = T(keep, :);
TT = table2timetable(T, 'RowTimes', 'Timestamp');
```

10.4 Step 3 — Compute energy and performance ratio

```
dt_h = 0.25; % 15-minute samples
Pkwp = 5; % array size
TT.Energy_kWh = TT.ACPower_W * dt_h / 1000;
TT.Insol_kWhm2 = TT.Irradiance_Wm2 * dt_h / 1000;
daily = retime(TT(:, {'Energy_kWh', 'Insol_kWhm2'}), 'daily', 'sum');
daily.PR = daily.Energy_kWh ./ (Pkwp * daily.Insol_kWhm2);
totalE = sum(daily.Energy_kWh);
fprintf('Total energy: %.1f kWh (%.2f kWh/day average)\n', totalE,
mean(daily.Energy_kWh));
fprintf('Specific yield: %.1f kWh/kWp for the month\n', totalE / Pkwp);
fprintf('Average PR: %.2f\n', mean(daily.PR, 'omitnan'));
[~, best] = max(daily.Energy_kWh);
fprintf('Best day: %s (%.1f kWh)\n', string(daily.Timestamp(best), 'dd-MMM'),
daily.Energy_kWh(best));
lowPR = daily(daily.PR < 0.65, :) % days that need checking
```

Engineering check

For a healthy system, PR is typically 0.75–0.80 (Solar course, Module 6/9). Days with low PR despite good sunshine may indicate soiling, shading or an inverter fault — the analysis tells the maintenance team exactly which days to investigate.

10.5 Step 4 — Analyse: temperature effect and power model

```
% only bright periods, to isolate temperature
bright = T(T.Irradiance_Wm2 > 700, :);
norm_P = bright.ACPower_W ./ bright.Irradiance_Wm2;           % W per W/m^2
p = polyfit(bright.ModuleTemp_C, norm_P, 1);
coeff = p(1) / polyval(p, 25) * 100;                         % % per °C relative to 25 °C
fprintf('Estimated temperature coefficient: %.2f %%/degC\n', coeff);

% power vs irradiance: linear model
q = polyfit(T.Irradiance_Wm2, T.ACPower_W, 1);
Ppred = polyval(q, T.Irradiance_Wm2);
R2 = 1 - sum((T.ACPower_W - Ppred).^2) / sum((T.ACPower_W - mean(T.ACPower_W)).^2);
fprintf('P = %.2f * G + %.1f (R^2 = %.3f)\n', q(1), q(2), R2);

% average daily profile (hour of day)
hr = hour(T.Timestamp) + minute(T.Timestamp)/60;
[g, hours] = findgroups(hr);
profile = splitapply(@mean, T.ACPower_W, g);
```

A negative temperature coefficient of about -0.3 to -0.4 $\%/^{\circ}\text{C}$ would match module datasheets — confirming that hot afternoons reduce output.

10.6 Step 5 — Visualise

```
figure('Color', 'w', 'Position', [100 100 900 700]);
tiledlayout(2, 2, 'TileSpacing', 'compact');

nexttile; bar(daily.Timestamp, daily.Energy_kWh, 'FaceColor', [0.24 0.23 0.56]);
ylabel('kWh'); title('Daily energy'); grid on

nexttile; plot(hours, profile/1000, 'LineWidth', 1.5);
xlabel('Hour of day'); ylabel('kW'); title('Average daily power profile'); grid on;
xlim([6 18])

nexttile; scatter(T.Irradiance_Wm2, T.ACPower_W/1000, 6, T.ModuleTemp_C, 'filled');
c = colorbar; c.Label.String = 'Module temp (°C)';
xlabel('Irradiance (W/m^2)'); ylabel('AC power (kW)'); title('Power vs irradiance'); grid on

nexttile; plot(daily.Timestamp, daily.PR, 'o-'); yline(0.75, '--', 'Target 0.75');
ylabel('PR'); title('Daily performance ratio'); grid on; ylim([0.4 1])

exportgraphics(gcf, fullfile('results', 'solar_dashboard.png'), 'Resolution', 300);
```

10.7 Step 6 — Report

```
writetable(timetable2table(daily), fullfile('results', 'daily_summary.xlsx'));
fid = fopen(fullfile('results', 'summary.txt'), 'w');
fprintf(fid, 'Solar performance report – September 2026\n');
fprintf(fid, 'Total energy: %.1f kWh\n', totalE);
fprintf(fid, 'Average PR: %.2f\n', mean(daily.PR, 'omitnan'));
fprintf(fid, 'Days needing inspection (PR < 0.65): %d\n', height(lowPR));
fclose(fid);
```

Alternatively, write the whole project as a **Live Script** and use **Export** → **PDF/Word** to produce a formatted report with text, equations, code and figures together.

10.8 Report structure

1. **Title and summary** — key findings in 3–4 sentences.
2. **Introduction** — the question and why it matters.
3. **Data** — source, period, variables, units, sampling interval.
4. **Method** — cleaning steps, formulas (energy, PR), fits used.
5. **Results** — tables and figures with captions, explained in words.
6. **Discussion** — what the results mean; limitations (sensor accuracy, missing data).
7. **Conclusions and recommendations** — e.g. "clean panels fortnightly in dry months".
8. **Appendix** — code listing.

10.9 Your capstone brief

Choose one project (or propose your own)

1. **Weather data analysis** — download a year of your city's daily temperature/rainfall (e.g. from a public weather service as CSV); clean it; find monthly averages, extremes and trends; fit a sinusoidal seasonal model; plot a dashboard.
2. **Electricity bill analyser** — enter 24 months of household units; compute costs with slab tariffs (functions!), seasonal patterns, forecast next year with a fitted trend, and simulate savings from a rooftop solar system.
3. **Lab data analyser** — take any experiment from your course (Ohm's law, RC charging, pendulum); fit the physical model; extract the constant (R , τ , g) with uncertainty; compare with theory.
4. **Simulink project** — model an RC/RL circuit or a DC motor speed loop; tune a PI controller; report rise time, overshoot and settling time.

Deliverables: organised project folder (data/raw, data/clean, functions, results); a main script; at least two of your own functions; cleaning steps; one curve fit with R^2 ; a four-panel figure exported at 300 dpi; a 4–6 page report.

10.10 Course revision — one page

Module	Key points
1 Environment	Command Window, Workspace, Editor; <code>;</code> suppresses output; <code>ans</code> ; <code>clc/clear</code> ; scripts (.m); <code>help/doc</code>
2 Variables	Case-sensitive names; double default; char vs string; <code>==</code> vs <code>=</code> ; <code>&&</code> <code> </code> ; precedence ($-2^2 = -4$); complex numbers; cell/struct; <code>fprintf</code> formats
3 Arrays	<code>[1 2 3]</code> , <code>[1;2;3]</code> , <code>a:s:b</code> , <code>linspace</code> ; <code>A(r,c)</code> , <code>end</code> ; * vs .* ; <code>sum/mean/max/find</code> ; logical indexing; <code>x = A\b</code>
4 Plotting	<code>plot</code> + labels + legend + grid; <code>hold on</code> ; <code>subplot/tiledlayout</code> ; <code>bar/scatter/histogram</code> ; <code>surf/contour</code> ; <code>exportgraphics</code>
5 Control flow	<code>if/elseif/else</code> , <code>switch</code> ; <code>for</code> , <code>while</code> ; <code>break/continue</code> ; <code>preallocate</code> ; vectorise when possible
6 Functions	<code>function [out] = name(in) in name.m</code> ; private workspace; local & anonymous functions (<code>@</code>); <code>nargin/error</code> ; breakpoints

7 Data files	readtable/writetable; tables T.col, T{r,c}; fillmissing, isoutlier, unique; datetime; timetables + retime; groupsummary
8 Numerical	fzero, roots; integral, trapz; diff/gradient; interp1; polyfit/polyval + R^2 ; fminbnd; ode45
9 Simulink	Blocks + lines; Integrator for ODEs; Scope; solvers; transfer functions; PI/PID feedback; subsystems; sim()
10 Project	Import → clean → compute → analyse → visualise → report; organised folders; reproducible main script

Module 10 — Summary

- Real projects follow import → clean → compute → analyse → visualise → report.
- Keep raw data untouched; script every step so results are reproducible.
- Combine tables, timetables, functions, fits and multi-panel figures.
- Communicate clearly: figures with captions, numbers with units, recommendations.

Practice questions — Module 10

1. List the six stages of a data-analysis project.
2. Write code to compute daily energy from 15-minute power readings in watts.
3. Explain how the performance ratio identifies faulty days.
4. Why is only bright-sun data used to estimate the temperature coefficient?
5. Write the tiledlayout code for a 1×3 figure of three plots.
6. Describe the structure of a technical report.

REFERENCE

MATLAB Command Reference

The commands used in this course, grouped by task — keep this beside you while you code.

Environment and workspace

Command	Purpose	Command	Purpose
<code>clc</code>	Clear Command Window	<code>clear</code> , <code>clear x</code>	Delete variables
<code>close all</code>	Close figures	<code>who</code> , <code>whos</code>	List variables
<code>help f</code> , <code>doc f</code>	Help on f	<code>lookfor word</code>	Search help
<code>format long/short</code>	Display digits	<code>pwd</code> , <code>cd</code> , <code>dir</code>	Folders
<code>save</code> , <code>load</code>	.mat files	<code>tic</code> , <code>toc</code>	Timing
<code>addpath</code>	Search path	<code>exist</code> , <code>isfile</code>	Check existence

Creating arrays

Command	Result	Command	Result
<code>a:s:b</code>	Range with step	<code>linspace(a,b,n)</code>	n points
<code>zeros</code> , <code>ones</code>	Filled arrays	<code>eye(n)</code>	Identity
<code>rand</code> , <code>randn</code> , <code>randi</code>	Random numbers	<code>magic(n)</code>	Magic square
<code>repmat(A,m,n)</code>	Tile an array	<code>meshgrid(x,y)</code>	2D grids
<code>size</code> , <code>numel</code> , <code>length</code>	Dimensions	<code>reshape</code> , <code>'</code>	Reshape, transpose

Maths and statistics

Command	Purpose	Command	Purpose
<code>sum</code> , <code>prod</code> , <code>cumsum</code>	Totals	<code>mean</code> , <code>median</code> , <code>mode</code>	Averages
<code>max</code> , <code>min</code>	Extremes (+ index)	<code>std</code> , <code>var</code>	Spread

<code>sort, sortrows</code>	Order	<code>find, any, all</code>	Search, tests
<code>abs, sqrt, exp, log</code>	Elementary	<code>sind, cosd, tand</code>	Trig in degrees
<code>round, floor, ceil, fix</code>	Rounding	<code>mod, rem</code>	Remainders
<code>det, inv, rank, eig</code>	Linear algebra	<code>A\b</code>	Solve $Ax = b$
<code>unique, histcounts</code>	Distinct values, counts	<code>corrcoef</code>	Correlation

Text

Command	Purpose	Command	Purpose
<code>num2str, string</code>	Number → text	<code>str2double</code>	Text → number
<code>strcmp, contains</code>	Compare, search	<code>upper, lower, strtrim</code>	Tidy text
<code>strrep, replace, split, join</code>	Edit text	<code>sprintf, fprintf</code>	Format text

Plotting

Command	Purpose	Command	Purpose
<code>plot, plot3</code>	Lines	<code>bar, barh, pie</code>	Charts
<code>scatter, histogram</code>	Data plots	<code>stem, stairs, area</code>	Signals
<code>surf, mesh, contour</code>	3D/contours	<code>fplot</code>	Plot a function
<code>title, xlabel, ylabel</code>	Labels	<code>legend, grid, axis</code>	Legend, grid, limits
<code>hold on/off</code>	Overlay	<code>subplot, tiledlayout</code>	Layouts
<code>figure, gcf, gca</code>	Figures/axes	<code>exportgraphics, saveas</code>	Save images

Programming

Construct	Purpose	Construct	Purpose
<code>if/elseif/else/end</code>	Decisions	<code>switch/case/otherwise</code>	Multi-way choice
<code>for, while</code>	Loops	<code>break, continue</code>	Loop control
<code>function ... end</code>	Functions	<code>@(x) ...</code>	Anonymous function

<code>nargin, error, warning</code>	Checks	<code>try ... catch ... end</code>	Handle errors
<code>input, disp</code>	Simple I/O	<code>dbstop if error</code>	Debugging

Data files and numerical methods

Command	Purpose	Command	Purpose
<code>readtable, writetable</code>	Tables ↔ files	<code>readmatrix, writematrix</code>	Matrices ↔ files
<code>fopen, fgetl, fclose</code>	Low-level I/O	<code>ismissing, fillmissing, rmmissing</code>	Missing data
<code>isoutlier, smoothdata</code>	Outliers, smoothing	<code>datetime, retime</code>	Time data
<code>fzero, roots</code>	Roots	<code>integral, trapz</code>	Integration
<code>diff, gradient</code>	Derivatives	<code>interp1, interp2</code>	Interpolation
<code>polyfit, polyval</code>	Curve fitting	<code>fminbnd, fminsearch</code>	Optimisation
<code>ode45, ode15s</code>	Differential equations	<code>sim</code>	Run Simulink model

15 Hands-On Exercises

Work through these in order; each builds on the modules before it.

Lab 1 — Calculator

Compute: the area of a triangle with sides 7, 8, 9 (Heron's formula); the compound interest on ₹50,000 at 8% for 5 years; the peak of 415 V RMS. (Ans: 26.83; ₹23,466.40; 586.9 V)

Lab 2 — Unit converter script

Ask for a value and a unit (km, mi, m) using `input(..., 's')` and print it in the other two units with `fprintf`.

Lab 3 — Matrix practice

Create `A = magic(5)`. Find the sums of rows, columns and both diagonals (they are all equal — why?). Replace all even numbers with 0 using logical indexing.

Lab 4 — Linear equations

Write the mesh equations for a three-loop resistor circuit of your choice and solve them with `A\b`. Verify one KVL equation with the result.

Lab 5 — Function plots

Plot $y = x^3 - 6x^2 + 11x - 6$ for $0 \leq x \leq 4$, mark its roots with red circles, and label everything.

Lab 6 — AC waveforms

Plot $v = 325 \sin(2\pi \cdot 50t)$ and $i = 14 \sin(2\pi \cdot 50t - 30^\circ)$ for 40 ms on two y-axes (`yyaxis`); compute the average power `mean(v.*i)` and compare with $V_{\text{rms}}I_{\text{rms}}\cos \varphi$.

Lab 7 — Loops

Print the Fibonacci numbers below 1,000 with a while loop, then compute the ratio of successive terms (it approaches 1.618...).

Lab 8 — Slab tariff function

Write `bill = elec_bill(units)` implementing the slab tariff of Worked example 5.7 so that it works for a vector of units.

Lab 9 — Grade report

Create a CSV of 10 students $\times$ 4 subjects. Read it with `readtable`, add Total, Percent and Grade columns, sort, and write an Excel report.

Lab 10 — Cleaning

Insert a few blank cells and impossible values (e.g. -5, 180) into your CSV. Detect, count and fix them with `fillmissing`, and document each step.

Lab 11 — Roots

Solve $\cos x = x/3$ for all roots between -5 and 5: plot first, then call `fzero` from several starting points.

Lab 12 — Integration

Find the energy (kWh) from a day's power readings of your own design using `trapz`, and the area under e^{-x^2} from $-\infty$ to ∞ with `integral` (compare with $\sqrt{\pi}$).

Lab 13 — Curve fitting

Measure (or invent) the cooling of a cup of tea every 2 minutes. Fit $T(t) = T_{\text{room}} + (T_0 - T_{\text{room}})e^{-kt}$ by linearising, and report k .

Lab 14 — ODE

Solve the pendulum equation $\theta'' = -(g/L) \sin \theta$ with ode45 for $L = 1$ m and $\theta_0 = 10^\circ$ and 80° . Compare the periods with $2\pi\sqrt{L/g}$.

Lab 15 — Simulink

Build the RC model of Worked example 9.1, then the PI motor speed loop of 9.2. Record rise time and overshoot for three sets of gains.

BONUS

Coding Style, Common Errors and Careers

Write MATLAB code others can read, avoid the classic mistakes, and see where MATLAB skills can take you.

B.1 Good coding style

- Start every script with a header comment: purpose, author, date, inputs/outputs.
- Use meaningful names with units: `current_A`, `time_s`, `temp_C`.
- One task per function; keep functions short.
- Avoid "magic numbers": define `tariff_Rs = 6.5;` once at the top.
- Indent consistently (Editor: select all → Ctrl + I smart-indent).
- Use sections (%%) to structure long scripts.
- Vectorise where it improves clarity; preallocate when looping.
- Check results against a hand calculation or known case.

B.2 Twelve classic mistakes

Mistake	Symptom	Fix
<code>x^2</code> on a vector	"Incorrect dimensions for raising a matrix to a power"	<code>x.^2</code>
<code>a*b</code> for two vectors	"Incorrect dimensions for matrix multiplication"	<code>a.*b</code>
<code>1/x</code> for a vector	Unexpected result or error	<code>1./x</code>
Degrees in sin/cos	Wrong values (sin(30) = -0.988)	<code>sind(30)</code> or convert with <code>deg2rad</code>
Indexing from 0	"Index must be a positive integer"	MATLAB starts at 1
Overwriting built-ins (<code>sum = 5</code>)	Later calls fail strangely	<code>clear sum</code> ; rename
Missing <code>end</code>	Parse error / wrong block	Smart-indent to see structure
<code>=</code> in a condition	Error or wrong logic	<code>==</code>
Forgetting <code>;</code> in loops	Screen floods, very slow	End statements with <code>;</code>

Comparing decimals exactly	<code>0.1*3 == 0.3</code> is false	Use a tolerance
Script named like a function (<code>plot.m</code>)	Built-in stops working	Rename the file
Running a function file with Run button without inputs	"Not enough input arguments"	Call it with arguments from a script

B.3 Careers using MATLAB

Area	Roles	Add these skills
Control & automotive	Controls engineer, model-based design, calibration	Simulink, control theory, code generation
Power & renewable energy	Power systems analyst, solar/EV/battery modelling	Simscape Electrical, power electronics
Signal & image processing	DSP engineer, computer vision, biomedical imaging	Signal/Image Processing Toolboxes, filters, FFT
Communications	Wireless/5G algorithm engineer	Communications Toolbox, probability
Data & AI	Data analyst, ML engineer, researcher	Statistics & ML Toolbox, Python, SQL
Research & academia	Research assistant, PhD, lecturer	Numerical methods, scientific writing

MathWorks offers free self-paced **Onramp** courses (MATLAB, Simulink, Machine Learning and more) that issue completion certificates — a good complement to this course. Many skills also transfer directly to Python (NumPy, Matplotlib, SciPy).

REVISION

100 Multiple-Choice Questions

Test yourself on all ten modules. The answer key follows the questions.

- MATLAB stands for:
(a) Mathematical Laboratory (b) Matrix Laboratory (c) Machine Tabulator (d) Math Lab Tool
- Commands are typed at the:
(a) Workspace (b) Command Window (c) Current Folder (d) Toolstrip
- Variables in memory are listed in the:
(a) Editor (b) Workspace (c) Command History (d) Help browser
- A semicolon at the end of a statement:
(a) causes an error (b) suppresses the output (c) comments the line (d) ends the program
- If a result is not assigned, it is stored in:
(a) x (b) ans (c) result (d) out
- clc does what?
(a) clears variables (b) clears the Command Window (c) closes figures (d) clears the path
- clear does what?
(a) clears the screen (b) removes variables from the workspace (c) deletes files (d) closes MATLAB
- MATLAB script files have the extension:
(a) .mat (b) .m (c) .fig (d) .slx
- Comments in MATLAB begin with:
(a) // (b) # (c) % (d) --
- $2 + 3 \times 4^2$ evaluates to:
(a) 50 (b) 80 (c) 146 (d) 200
- -2^2 evaluates to:
(a) 4 (b) -4 (c) 0 (d) error
- sqrt(144) returns:
(a) 12 (b) 72 (c) 14 (d) 1
- To compute sine of 30 degrees use:
(a) sin(30) (b) sind(30) (c) sin(pi/30) (d) deg(30)
- The value of pi is displayed with 15 digits after:
(a) format short (b) format long (c) format bank (d) format rat
- Which variable name is valid?
(a) 2value (b) my-value (c) value_2 (d) end
- MATLAB variable names are:
(a) case-insensitive (b) case-sensitive (c) limited to 8 characters (d) numbers only
- The default numeric class is:
(a) int32 (b) single (c) double (d) uint8
- uint8(300) returns:
(a) 300 (b) 44 (c) 255 (d) error
- $0.1 + 0.2 == 0.3$ returns:
(a) 1 (b) 0 (c) error (d) NaN
- The not-equal operator is:
(a) != (b) <> (c) ~= (d) /=

21. Short-circuit AND is written:
(a) & (b) && (c) and (d) +
22. Text in double quotes creates a:
(a) char array (b) string (c) cell (d) struct
23. To convert text '42.5' to a number use:
(a) num2str (b) str2double (c) int2str (d) char
24. `abs(3+4i)` returns:
(a) 7 (b) 5 (c) 1 (d) 25
25. A container holding mixed data with { } is a:
(a) matrix (b) cell array (c) table (d) struct
26. In `fprintf, %.2f` prints:
(a) an integer (b) a number with 2 decimals
(c) text (d) a newline
27. A column vector is written:
(a) [1 2 3] (b) [1;2;3] (c) (1,2,3) (d) {1 2 3}
28. `0:2:8` produces:
(a) 0 2 4 6 8 (b) 0 1 2 ... 8 (c) 2 4 6 8 (d) 0 8
29. `linspace(0,1,5)` produces how many points?
(a) 4 (b) 5 (c) 6 (d) 10
30. MATLAB indexing starts at:
(a) 0 (b) 1 (c) -1 (d) any value
31. The last element of `v` is accessed by:
(a) `v(0)` (b) `v(last)` (c) `v(end)` (d) `v(-1)`
32. `A(2,:)` returns:
(a) column 2 (b) row 2 (c) element 2 (d) the whole matrix
33. `A(:,3)` returns:
(a) row 3 (b) column 3 (c) element 3 (d) nothing
34. `eye(3)` creates:
(a) a 3x3 zero matrix (b) a 3x3 identity matrix
(c) a 3x3 ones matrix (d) a random matrix
35. `size(zeros(2,5))` returns:
(a) [5 2] (b) [2 5] (c) 10 (d) 7
36. Element-wise multiplication uses:
(a) * (b) .* (c) x (d) **
37. For `x = 1:5, y = x^2` gives:
(a) [1 4 9 16 25] (b) an error (c) 55 (d) 25
38. `[1 2; 3 4] * [1 0; 0 1]` equals:
(a) [1 2; 3 4] (b) [1 0; 0 4] (c) [2 2; 3 5] (d) error
39. `[1 2] .* [3 4]` equals:
(a) [3 8] (b) 11 (c) [4 6] (d) error
40. The transpose of a matrix is written:
(a) `A^T` (b) `A'` (c) `trans(A)` (d) `A~`
41. To solve $Ax = b$ the preferred command is:
(a) `inv(A)*b` (b) `A\b` (c) `b/A` (d) `A*b`
42. `sum([4 9 1 7])` returns:
(a) 21 (b) 252 (c) 9 (d) 4
43. `[m, i] = max([4 9 1 7])` gives:
(a) `m = 9, i = 2` (b) `m = 2, i = 9` (c) `m = 7, i = 4` (d) `m = 9, i = 1`
44. `v(v > 5)` returns:
(a) indices of values above 5 (b) values of `v` above 5
(c) a logical array (d) the count
45. `find([0 3 0 5])` returns:
(a) [3 5] (b) [2 4] (c) 2 (d) [0 1 0 1]

46. The plot command that keeps the current graph for new lines is:
(a) keep on (b) hold on (c) grid on (d) axis on
47. Axis labels are added with:
(a) xname/ynname (b) xlabel/ylabel (c) label/axis (d) title/legend
48. subplot(2,3,4) draws in which grid position?
(a) row 2, column 1 (b) row 1, column 4 (c) row 4, column 2 (d) row 2, column 3
49. To compare categories the best plot is:
(a) pie (b) bar (c) surf (d) stem
50. 3D surfaces are drawn with:
(a) plot (b) surf (c) bar (d) histogram
51. meshgrid is used to:
(a) create 2D grids of x and y (b) draw meshes only (c) sort data (d) save figures
52. A high-resolution figure file is saved with:
(a) exportgraphics (b) printfig (c) savefigure (d) imwrite only
53. Every if block ends with:
(a) endif (b) end (c) fi (d) stop
54. switch compares a value with several:
(a) loops (b) cases (c) files (d) functions
55. A for loop is best when:
(a) the number of repetitions is known (b) a condition is unknown (c) never (d) only for strings
56. A while loop repeats:
(a) a fixed number of times (b) while its condition is true (c) once (d) forever always
57. break inside a loop:
(a) skips one pass (b) exits the loop (c) pauses (d) restarts
58. continue inside a loop:
(a) exits the loop (b) skips to the next pass (c) stops MATLAB (d) ends the script
59. To stop a runaway loop press:
(a) Ctrl+Z (b) Ctrl+C (c) Esc (d) F5
60. Preallocating arrays with zeros() makes loops:
(a) slower (b) faster (c) impossible (d) unchanged
61. Replacing loops with whole-array operations is called:
(a) recursion (b) vectorisation (c) compilation (d) indexing
62. A function file must be named:
(a) anything (b) the same as the function (c) main.m (d) fun.m
63. Variables inside a function are:
(a) shared with the workspace (b) local to the function (c) global (d) deleted before running
64. An anonymous function is written:
(a) $f = @(x) x.^2$ (b) function $f = x^2$ (c) $f := x^2$ (d) $\lambda x: x^2$
65. nargin gives:
(a) number of outputs (b) number of inputs passed (c) number of lines (d) the function name
66. error('msg') in a function:
(a) prints and continues (b) stops with a message (c) saves a file (d) is ignored

57. A breakpoint is used for:
(a) plotting (b) debugging (c) saving (d) importing
58. Local functions in a file are placed:
(a) at the start (b) at the end (c) in another file (d) anywhere in the middle
59. readtable reads data into a:
(a) matrix (b) table (c) cell (d) string
70. To write a table to Excel use:
(a) writetable (b) xlswriteonly (c) savexl (d) exporttable
71. T{2,'Maths'} returns:
(a) a table (b) the value in row 2 of column Maths (c) column Maths (d) row 2 as a table
72. Missing numeric values in MATLAB are represented by:
(a) 0 (b) NaN (c) Inf (d) -1
73. To fill missing values by interpolation use:
(a) fillmissing(x,'linear') (b) rmmissing (c) ismissing (d) isnan only
74. To remove duplicate rows use:
(a) unique(T,'rows') (b) sort (c) find (d) diff
75. Text dates are converted with:
(a) datestr only (b) datetime (c) datenumonly (d) time2num
76. retime is used to:
(a) resample or aggregate timetables (b) rename columns (c) reset MATLAB (d) retry imports
77. fzero is used to find:
(a) integrals (b) roots of a function (c) maxima only (d) derivatives
78. roots([1 -5 6]) returns:
(a) 2 and 3 (b) -2 and -3 (c) 1 and 6 (d) 5 and 6
79. The bisection method requires:
(a) the derivative (b) a sign change on the interval (c) a polynomial (d) a matrix
80. Newton's method uses:
(a) only function values (b) the function and its derivative (c) random guesses (d) interpolation
81. integral(f,a,b) computes:
(a) a derivative (b) a definite integral (c) a root (d) a sum of a vector
82. trapz(x,y) integrates:
(a) a formula (b) data points using trapezoids (c) only polynomials (d) only positive data
83. diff([1 4 9 16]) returns:
(a) [3 5 7] (b) [1 3 5 7] (c) 30 (d) [4 9 16]
84. interp1 estimates values:
(a) outside the data only (b) between known data points (c) at random (d) for images only
85. polyfit(x,y,1) fits a:
(a) straight line (b) parabola (c) cubic (d) exponential
86. polyval(p,x) does what?
(a) fits a polynomial (b) evaluates a polynomial (c) finds roots (d) differentiates
87. An R^2 value close to 1 means:
(a) a poor fit (b) a very good fit (c) overfitting always (d) no data

38. Using a very high polynomial degree may cause:
 (a) underfitting (b) overfitting and wiggles
 (c) faster code (d) nothing
39. An exponential model $y = a e^{(bx)}$ can be fitted linearly by fitting:
 (a) x vs y^2 (b) x vs $\log(y)$ (c) $\log(x)$ vs y (d) y vs x^2
30. fminbnd finds:
 (a) a root (b) a minimum of a function on an interval (c) an integral (d) a matrix inverse
31. The general-purpose ODE solver is:
 (a) ode45 (b) solve45 (c) diffeq (d) integral
32. Simulink model files have the extension:
 (a) .m (b) .slx (c) .mat (d) .sim
33. The block that integrates a signal over time is the:
 (a) Gain (b) Integrator (c) Scope (d) Sum
34. Signals are displayed against time using a:
 (a) Display block (b) Scope (c) Constant (d) Mux
95. A Gain block:
 (a) adds signals (b) multiplies by a constant
 (c) integrates (d) stores data
96. A controller that removes steady-state error uses:
 (a) only P action (b) integral (I) action (c) no feedback (d) a Scope
97. In closed-loop control the output is:
 (a) ignored (b) measured and fed back (c) fixed (d) random
98. The time for a response to go from 10% to 90% is the:
 (a) settling time (b) rise time (c) overshoot
 (d) delay
99. Data is sent from Simulink to MATLAB with the:
 (a) To Workspace block (b) Scope only (c) Constant block (d) Gain block
100. A Simulink model can be run from a script with:
 (a) run_model (b) sim (c) simulate (d) start

ANSWERS

MCQ Answer Key

If you got a question wrong, go back to the module it came from and re-read that section.

Q	Ans	Q	Ans	Q	Ans	Q	Ans	Q	Ans
1	b	21	b	41	b	61	b	81	b
2	b	22	b	42	a	62	b	82	b
3	b	23	b	43	a	63	b	83	a
4	b	24	b	44	b	64	a	84	b
5	b	25	b	45	b	65	b	85	a
6	b	26	b	46	b	66	b	86	b
7	b	27	b	47	b	67	b	87	b
8	b	28	a	48	a	68	b	88	b
9	c	29	b	49	b	69	b	89	b
10	a	30	b	50	b	70	a	90	b
11	b	31	c	51	a	71	b	91	a
12	a	32	b	52	a	72	b	92	b
13	b	33	b	53	b	73	a	93	b
14	b	34	b	54	b	74	a	94	b
15	c	35	b	55	a	75	b	95	b
16	b	36	b	56	b	76	a	96	b
17	c	37	b	57	b	77	b	97	b
18	c	38	a	58	b	78	a	98	b
19	b	39	a	59	b	79	b	99	a
20	c	40	b	60	b	80	b	100	b

Short-answer questions (2–3 marks)

1. What is the purpose of the Workspace and the Current Folder?
2. What is the difference between `clc`, `clear` and `close all`?
3. Explain the difference between `*` and `.*` with an example.
4. What does `linspace(0, 10, 5)` return?
5. Write the command to extract row 2 of a matrix A.
6. What is logical indexing? Give an example.
7. Differentiate a script and a function.
8. Write an anonymous function for $f(x) = 3x^2 + 2$.
9. Differentiate for and while loops.
10. What does `polyfit(x, y, 2)` return?
11. Differentiate interpolation and curve fitting.
12. What is the use of `fzero`?
13. How are missing values handled in a table?
14. What is the role of the Integrator block in Simulink?
15. What is a transfer function?

Long-answer questions (5–10 marks)

1. Describe the MATLAB desktop and explain how to write, save and run a script, with an example.
2. Explain vectors and matrices in MATLAB: creation, indexing, matrix versus element-wise operations, and solving linear equations.
3. Write a MATLAB program to plot $\sin(x)$, $\cos(x)$ and their sum on the same axes with full labelling, and explain each command.
4. Explain if-elseif-else, switch, for and while with examples. Write a program to classify marks into grades.
5. Write a function that returns the mean, maximum and standard deviation of a vector, and explain variable scope.
6. Explain how to import a CSV file, clean it (missing values, outliers, duplicates) and export the results.
7. Explain the bisection and Newton–Raphson methods and write MATLAB code for one.
8. Explain least-squares curve fitting with `polyfit`, including R^2 and the choice of polynomial degree, with an example.

9. Describe how to model and simulate an RC circuit in Simulink and interpret the Scope output.
10. Describe a complete data-analysis project workflow in MATLAB from raw data to report.

Glossary of Terms

Key words used throughout these notes, in alphabetical order.

Anonymous function

One-line function stored in a handle, e.g. `@(x) x.^2`.

ans

Default variable for unassigned results.

Array

Ordered collection of values; all MATLAB data are arrays.

Backslash (\)

Operator that solves linear systems $A \setminus b$.

Bisection

Root-finding by repeatedly halving an interval.

Block

Component in a Simulink model.

Breakpoint

Point where execution pauses for debugging.

Cell array

Container of mixed data, indexed with `{ }`.

Closed loop

Control using feedback of the output.

Colon operator

`a:b` creates ranges and selects all elements.

Command Window

Where commands are typed and results shown.

Curve fitting

Finding a model that best matches noisy data.

datetime

Data type for dates and times.

double

Default 64-bit floating-point number type.

Element-wise operation

Operation applied to each element (`.* ./ .^`).

Editor

Tool for writing scripts and functions.

fprintf

Formatted printing to screen or file.

Function

Reusable code with inputs, outputs and a private workspace.

Function handle

Reference to a function, created with `@`.

Integrator

Simulink block that integrates its input (1/s).

Interpolation

Estimating values between known points.

Least squares

Fitting that minimises squared errors.

Linear indexing

Accessing matrix elements by a single index in column order.

Live Script

`.mlx` file mixing code, text and output.

Logical indexing

Selecting elements with a true/false array.

Loop

Repeated execution (for, while).

MAT-file

File storing MATLAB variables (`.mat`).

Matrix

Two-dimensional array of m rows and n columns.

NaN

Not-a-Number; marks missing or undefined values.

Newton-Raphson

Fast root-finding using the derivative.

ODE

Ordinary differential equation.

ode45

MATLAB's general-purpose ODE solver.

Overfitting

A model that follows noise rather than the trend.

PID controller

Proportional–integral–derivative feedback controller.

polyfit / polyval

Fit / evaluate polynomials.

Preallocation

Creating an array at full size before filling it.

R²

Coefficient of determination; goodness of fit.

Scope

Simulink block that plots signals.

Script

File of commands sharing the base workspace.

Simulink

Graphical environment for simulating dynamic systems.

Solver

Numerical method that advances a simulation in time.

String

Text object created with double quotes.

Structure

Data type with named fields.

Subplot / tiledlayout

Several plots in one figure.

Table

Column-oriented data with named variables.

Timetable

Table indexed by time.

Toolbox

Add-on collection of functions for a field.

Transfer function

Output/input ratio of a linear system in s.

Transpose

Swapping rows and columns (').

Vector

One-dimensional array (row or column).

Vectorisation

Replacing loops with whole-array operations.

Workspace

Set of variables currently in memory.

Keep learning

Use the AI tutor on www.aitutorspark.co.in to ask any question from these notes — it explains step by step. Complete the course quiz to earn your certificate. All the best!