

Course Notes

February 14, 2026

Table of Contents

- [Tutorial 1: An Overview of MATLAB](#)
- [Tutorial 2: Numeric, Cell and Structure Arrays](#)
- [Tutorial 3: Functions](#)
- [Tutorial 4: Programming with MATLAB](#)
- [Tutorial 9: Numerical Methods](#)

Tutorial 1: An Overview of MATLAB

Key Concepts and Common Pitfalls (Tutorial 1 Summary)

1. MATLAB Arithmetic and Precedence Rules

MATLAB follows a strict order of precedence:

- Parentheses
- Exponentiation
- Multiplication and division
- Addition and subtraction

Incorrect placement of parentheses can completely change results. For example:

$$27^{1/3} \neq 27^1/3$$

Pitfall: Students frequently misinterpret expressions such as:

```
1 16^-1/2
2 16^(-1/2)
```

which produce different answers due to operator precedence.

2. Scalar Operations vs Mathematical Notation

MATLAB syntax must be explicit:

- Multiplication requires `*`
- Division requires clear parentheses

Example:

```
1 (3*y)/(4*x-8) % Correct
2 3*y/4*x-8    % Often misinterpreted
```

Pitfall: Missing parentheses leads to unintended evaluation order.

3. Numerical Limits: Overflow and Underflow

MATLAB floating-point limits can produce:

- `Inf` when numbers exceed `realmax`
- 0 or precision loss near `realmin`

Example concept:

```
1 x1 = a*b*d; % may overflow
2 x2 = a*(b*d); % safer evaluation
```

Pitfall: Intermediate calculations may overflow even if final results are valid.

4. Built-in Functions and Units

Key MATLAB functions:

- `log()` = natural logarithm
- `log10()` = base-10 logarithm
- Trigonometric functions use radians

Pitfall: Confusing `log()` with base-10 logarithm is a very common mistake.

5. Arrays and Vectorization

MATLAB operates efficiently on arrays:

```
1 u = 0:0.1:10;  
2 w = 5*sin(u);
```

Vectorized operations compute many values at once.

Pitfall: Using matrix operators instead of element-wise operators:

- Use element-wise operators for arrays: `.*`, `./`, `.^`.

6. Plotting Basics

Core plotting workflow:

```
1 plot(x,y)  
2 xlabel('x')  
3 ylabel('y')  
4 grid on
```

Important steps:

- Define domain first
- Use consistent units
- Label axes clearly

Pitfall: Forgetting element-wise operators when computing functions for plotting.

7. Script Files and Execution Order

When MATLAB executes a name:

1. Checks variables
2. Checks built-in commands
3. Searches current folder
4. Searches path

Pitfall: Naming scripts the same as MATLAB functions causes execution errors.

8. Engineering Problem-Solving Workflow

Recommended steps:

- Define inputs and outputs clearly
- Verify with simple hand calculations
- Perform a reality check on results

Common Mistake: Trusting MATLAB output without verifying physical meaning or units.

9. Debugging Strategy

Typical error types:

- Syntax errors (missing brackets, commas)
- Runtime errors (division by zero)

Recommended debugging methods:

- Remove semicolons to inspect values
 - Test simplified cases
 - Check intermediate variables
-

Tutorial Problems

Problem 3

Suppose that $x = 5$ and $y = 2$. Use MATLAB to compute the following, and check the results with a calculator.

- $(1 - \frac{1}{x^5})^{-1}$
- $3\pi x^2$
- $\frac{3y}{4x-8}$
- $\frac{4(y-5)}{3x-6}$

```

1 clear; clc;
2 x = 5;
3 y = 2;
4
5 % a. (1 - 1/x^5)^-1
6 result_a = (1 - 1/x^5)^-1;
7
8 % b. 3 * pi * x^2
9 result_b = 3 * pi * x^2;
10
11 % c. (3*y) / (4*x - 8)
12 result_c = (3*y) / (4*x - 8);
13
14 % d. (4*(y - 5)) / (3*x - 6)
15 result_d = (4*(y - 5)) / (3*x - 6);
16
17 % Display results
18 disp(table(result_a, result_b, result_c, result_d));

```

Problem 5

Assuming that the variables a , b , c , d , and f are scalars, write MATLAB statements to compute and display the following expressions. Test your statements for the values $a = 1.12$, $b = 2.34$, $c = 0.72$, $d = 0.81$ and $f = 19.83$.

- $x = 1 + \frac{a}{b} + \frac{c}{f^2}$
- $r = \frac{1}{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}}$
- $s = \frac{b-a}{d-c}$
- $y = ab\frac{1}{c}\frac{f^2}{2}$

```

1 clear; clc;
2 a = 1.12; b = 2.34; c = 0.72; d = 0.81; f = 19.83;
3
4 x = 1 + a/b + c/f^2;
5 r = 1 / (1/a + 1/b + 1/c + 1/d);
6 s = (b - a) / (d - c);
7 y = a * b * (1/c) * (f^2/2);
8
9 disp(['x = ', num2str(x)]);
10 disp(['r = ', num2str(r)]);
11 disp(['s = ', num2str(s)]);
12 disp(['y = ', num2str(y)]);

```

Problem 9

The functions `realmax` and `realmin` give the largest and smallest possible numbers that can be handled by MATLAB. Suppose you have variables $a = 3 \times 10^{150}$, $b = 5 \times 10^{200}$.

- Use MATLAB to calculate $c = ab$.
- Supposed $d = 5 \times 10^{-200}$ use MATLAB to calculate $f = d/a$.
- Use MATLAB to calculate the product $x = abd$ two ways.

```
1 % Check limits
2 realmax
3 realmin
4
5 a = 3e150;
6 b = 5e200;
7
8 % a. Calculate c = a*b (Expect Overflow)
9 c = a * b
10
11 % b. d = 5e-200, calculate f = d/a (Expect Underflow)
12 d = 5e-200;
13 f = d / a
14
15 % c. Calculate x = abd in two ways
16 x1 = a * b * d; % Risk of intermediate overflow
17 y = b * d;
18 x2 = a * y;      % Safer calculation
19
20 disp(['Method 1: ', num2str(x1)]);
21 disp(['Method 2: ', num2str(x2)]);
```

Problem 22

Use MATLAB to calculate:

- $e^{(-2.1)^3} + 3.47 \log(14) + \sqrt[4]{287}$
- $(3.4)^7 \log(14) + \sqrt[4]{287}$
- $\cos^2\left(\frac{4.12\pi}{6}\right)$
- $\cos\left(\frac{4.12\pi}{6}\right)^2$

```
1 % Note: Source likely implies log base 10 for "log(14)" in standard notation,
2 % but MATLAB's log() is natural log. Using log10() for base 10.
3 ans_a = exp((-2.1)^3) + 3.47 * log10(14) + nthroot(287, 4);
4 ans_b = (3.4)^7 * log10(14) + nthroot(287, 4);
```

```

5 ans_c = cos((4.12 * pi) / 6)^2;
6 ans_d = cos(((4.12 * pi) / 6)^2);

```

Problem 27

Use MATLAB to plot the function $T = 7 \ln t - 8e^{0.3t}$ over the interval $1 \leq t \leq 3$.

```

1 t = 1:0.01:3;
2 T = 7 .* log(t) - 8 .* exp(0.3 .* t);
3
4 plot(t, T);
5 title('Temperature vs Time');
6 xlabel('Time (min)');
7 ylabel('Temperature (C)');
8 grid on;

```

Problem 30

A cycloid is described by $x = r(\phi - \sin \phi)$ and $y = r(1 - \cos \phi)$. Plot for $r = 10$ and $0 \leq \phi \leq 4\pi$.

```

1 r = 10;
2 phi = 0 : 0.01 : 4*pi;
3 x = r .* (phi - sin(phi));
4 y = r .* (1 - cos(phi));
5
6 plot(x, y);
7 title('Cycloid Plot (r=10)');
8 xlabel('x'); ylabel('y');
9 axis equal;

```

Problem 34

Develop a procedure for computing the length of side c_2 of the two-triangle figure given sides b_1, b_2, c_1 and angles A_1, A_2 . Test with $b_1 = 200, b_2 = 180, c_1 = 120, A_1 = 120^\circ, A_2 = 100^\circ$.

$$a^2 = b_1^2 + c_1^2 - 2b_1c_1 \cos A_1$$

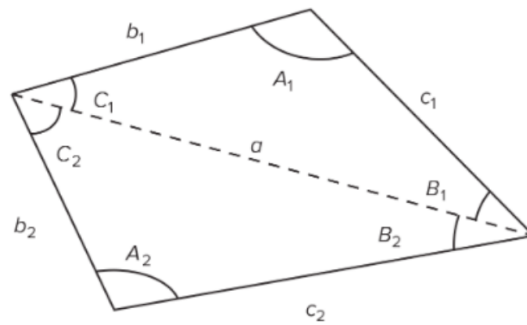


Figure P34

```

1 % Inputs
2 b1 = 200; b2 = 180; c1 = 120;
3 A1_deg = 120; A2_deg = 100;
4 A1 = deg2rad(A1_deg); A2 = deg2rad(A2_deg);
5
6 % 1. Find common side 'a' (Top Triangle Law of Cosines)
7 a_sq = b1^2 + c1^2 - 2*b1*c1*cos(A1);
8 a = sqrt(a_sq);
9
10 % 2. Find c2 (Bottom Triangle) solving quadratic:
11 % c2^2 - (2*b2*cos(A2))*c2 + (b2^2 - a^2) = 0
12 coeff_A = 1;
13 coeff_B = -2 * b2 * cos(A2);
14 coeff_C = b2^2 - a_sq;
15
16 possible_c2 = roots([coeff_A, coeff_B, coeff_C]);
17 c2 = possible_c2(possible_c2 > 0); % Filter positive
18
19 disp(['Side c2: ', num2str(c2)]);

```

Problem 35

Write a script to compute the three roots of $x^3 + ax^2 + bx + c = 0$.

```

1 a = input('Enter a: ');
2 b = input('Enter b: ');
3 c = input('Enter c: ');
4 disp(roots([1, a, b, c]));

```

Tutorial 2: Numeric, Cell and Structure Arrays

Key Concepts and Common Pitfalls (Tutorial 2 Summary)

1. Creating Vectors and Matrices

MATLAB offers multiple ways to create arrays:

- **Row Vector:** `v = [1, 2, 3]` (comma or space separated)
- **Column Vector:** `v = [1; 2; 3]` (semicolon separated)
- **Colon Operator:** `start:step:end` (e.g., `0:0.1:10`)
- **Linspace:** `linspace(x1, x2, n)` for specific number of points

Pitfall: Confusing the syntax for steps versus number of points.

```
1 x = 0:10;           % Integers 0 to 10 (step is 1)
2 x = linspace(0,10); % 100 points between 0 and 10
```

2. Array Addressing and Slicing

MATLAB uses **1-based indexing** (indices start at 1, not 0).

- `A(row, col)` selects a specific element.
- `A(:, n)` selects the entire n^{th} column.
- `A(m, :)` selects the entire m^{th} row.

Pitfall: Attempting to access index 0 or an index outside the array dimensions triggers an error.

```
1 val = A(0);          % Error: Indices must be positive integers
```

3. Element-by-Element Operations (The "Dot" Operators)

When performing arithmetic between two arrays of the same size, you MUST distinguish between matrix math and element-wise math.

- **Multiplication:** `.*`
- **Division:** `./`
- **Exponentiation:** `.^`

Example:

```
1 y = x.^2 + 3*x;    % Correct for vector x
2 y = x^2 + 3*x;     % Error (Matrix power requires square matrix)
```

Pitfall: Omitting the dot (.) when plotting functions. If \mathbf{x} is a vector, $\mathbf{y} = \mathbf{x}*\mathbf{x}$ fails because inner dimensions do not agree. You must use $\mathbf{y} = \mathbf{x}.*\mathbf{x}$.

4. Matrix Multiplication vs. Array Multiplication

- $\mathbf{A}*\mathbf{B}$ performs standard linear algebra matrix multiplication (Row \times Column). Inner dimensions must match.
- $\mathbf{A}.*\mathbf{B}$ multiplies corresponding elements. Dimensions must be identical.

Pitfall: Assuming matrix multiplication is commutative. In MATLAB (and math), $\mathbf{A} * \mathbf{B} \neq \mathbf{B} * \mathbf{A}$.

5. Solving Linear Systems

To solve systems like $\mathbf{Ax} = \mathbf{B}$:

- Use the **Left Division** operator (`\`).
- Syntax: `x = A \B`.

Pitfall: Using right division (`/`) or inverse (`inv(A)*B`). Left division is numerically more stable and faster for linear equations.

6. Polynomials in MATLAB

Polynomials are represented as row vectors of coefficients in descending order.

- $P(x) = 2x^2 + 14x + 20 \rightarrow \mathbf{p} = [2, 14, 20]$
- **Find Roots:** `roots(p)`
- **Evaluate:** `polyval(p, x)`

Pitfall: Forgetting to include zeros for missing powers. For $x^3 + 5$, the vector is `[1, 0, 0, 5]`, not `[1, 5]`.

7. Vector Properties: Magnitude, Length, and Absolute Value

It is crucial to distinguish between these three terms in MATLAB:

- **Length:** `length(x)` returns the number of elements in the vector.
- **Absolute Value:** `abs(x)` returns a vector where every element is positive.
- **Magnitude (Geometric Length):** This is a scalar value representing the geometric length $\sqrt{x_1^2 + x_2^2 + \dots}$. It is calculated using `norm(x)` or `sqrt(x'*x)`.

Specific Example:

```
1 x = [2, -4, 5];
2
3 L = length(x);      % Result: 3 (elements)
4 A = abs(x);         % Result: [2, 4, 5] (vector)
5 M = norm(x);        % Result: 6.7082 (scalar)
6 % Magnitude Calculation: sqrt(2^2 + (-4)^2 + 5^2) = 6.7082
```

Pitfall: Confusing `length(x)` (count of items) with `norm(x)` (geometric size/magnitude).

8. Essential Data Analysis Functions

MATLAB provides built-in functions to analyze and locate data within arrays.

- **Finding Indices:** `find(A)`
 - `k = find(A)`: Returns linear indices of nonzero elements.
 - `[row, col] = find(A)`: Returns row and column indices separately.
 - `[row, col, val] = find(A)`: Returns row, column, AND the nonzero values themselves.

- **Min/Max Values:** `min(A)` and `max(A)`
 - `val = max(A)`: Returns the largest value.
 - `[val, k] = max(A)`: Returns the largest value **and** its index `k`.
- **Sorting and Summing:**
 - `sort(A)`: Sorts each column in ascending order.
 - `sum(A)`: Computes the sum of elements (column-wise for matrices).

Specific Example (Min/Max Indices):

```
1 A = [10, 50, 30];
2 [val, idx] = max(A);
3 % val = 50
4 % idx = 2
```

Pitfall: If `A` contains complex numbers, `max(A)` returns the element with the largest **magnitude**, not the largest real component.

9. Array Dimensions

- `size(A)`: Returns a vector `[rows, cols]`.
- `length(A)`: Returns the size of the **largest** dimension.

Pitfall: Using `length()` on a matrix when you specifically need the number of rows. Always use `size(A, 1)` for rows.

10. Special Matrix Initialization

MATLAB has dedicated functions to create specific matrices efficiently.

- **Zeros:** `zeros(m, n)` creates an $m \times n$ matrix of zeros.
- **Ones:** `ones(m, n)` creates an $m \times n$ matrix of ones.
- **Identity Matrix:** `eye(n)` creates an $n \times n$ identity matrix (1s on diagonal, 0s elsewhere).

Example Usage:

```
1 Z = zeros(3, 4); % 3x4 matrix of zeros
2 I = eye(5);      % 5x5 identity matrix
```

Pitfall: Confusing the empty matrix `[]` with the zero matrix.

- `A = []` deletes data or creates an empty container.
 - `A = 0` creates a scalar zero.
 - `A = zeros(2)` creates a 2×2 matrix of zeros.
-

Tutorial Problems

Problem 10

Consider the array $A = \begin{bmatrix} 1 & 4 & 2 \\ 2 & 4 & 100 \\ 7 & 9 & 7 \\ 3 & \pi & 42 \end{bmatrix}$ and $B = \ln(A)$.

Write MATLAB expressions to do the following:

- Select just the second row of B.
- Evaluate the sum of the second row of B.
- Multiply the second column of B and the first column of A element by element.
- Evaluate the maximum value in the vector resulting from element-by-element multiplication of the second column of B with the first column of A.
- Use element-by-element division to divide the first row of A by the first three elements of the third column of B. Evaluate the sum of the elements of the resulting vector.

```
1 % Define Matrix A
2 A = [1, 4, 2;
3      2, 4, 100;
4      7, 9, 7;
5      3, pi, 42];
6
7 % Define Matrix B (Natural log is log() in MATLAB)
8 B = log(A);
9
10 % a. Select second row of B
11 part_a = B(2, :);
12
13 % b. Sum of second row of B
14 part_b = sum(B(2, :));
15
```

```

16 % c. Multiply 2nd col of B and 1st col of A element-wise
17 part_c = B(:, 2) .* A(:, 1);
18
19 % d. Max value of result from c
20 part_d = max(part_c);
21
22 % e. Divide 1st row of A by first 3 elements of 3rd col of B
23 % Note: A(1,:) is 1x3. B(1:3, 3) is 3x1.
24 % We must transpose B's slice to match dimensions.
25 vec_e = A(1, :) ./ B(1:3, 3)';
26 part_e = sum(vec_e);
27
28 disp(['Sum (Part b): ', num2str(part_b)]);
29 disp(['Max (Part d): ', num2str(part_d)]);
30 disp(['Sum (Part e): ', num2str(part_e)]);

```

Problem 11

Create a three-dimensional array D whose three "layers" are matrices A, B, and C. Use MATLAB to find the largest element in each layer of D and the largest element in D.

```

1 A = [3, -2, 1; 6, 8, -5; 7, 9, 10];
2 B = [6, 9, -4; 7, 5, 3; -8, 2, 1];
3 C = [-7, -5, 2; 10, 6, 1; 3, -9, 8];
4
5 % Create 3D array D
6 D(:, :, 1) = A;
7 D(:, :, 2) = B;
8 D(:, :, 3) = C;
9
10 % Largest element in each layer
11 max_layer_1 = max(max(D(:, :, 1)));
12 max_layer_2 = max(max(D(:, :, 2)));
13 max_layer_3 = max(max(D(:, :, 3)));
14
15 % Largest element in D
16 max_total = max(D(:));
17
18 disp(['Max Total: ', num2str(max_total)]);

```

Problem 15

Given matrices A, B, and C, verify the associative and commutative laws for addition.

```

1 A = [-7, 11; 4, 9];
2 B = [4, -5; 12, -2];

```

```

3 C = [-3, -9; 7, 8];
4
5 % a. A + B + C
6 res_a = A + B + C;
7
8 % b. A - B + C
9 res_b = A - B + C;
10
11 % c. Verify Associative Law: (A+B)+C = A+(B+C)
12 check_assoc = isequal((A+B)+C, A+(B+C));
13
14 % d. Verify Commutative Law: A+B+C = B+C+A = A+C+B
15 term1 = A + B + C;
16 term2 = B + C + A;
17 term3 = A + C + B;
18 check_comm = isequal(term1, term2) && isequal(term2, term3);
19
20 if check_assoc && check_comm
21     disp('Laws Verified');
22 else
23     disp('Verification Failed');
24 end

```

Problem 19

Plot the function $f(x) = \frac{4\cos x}{x+e^{-0.75x}}$ over the interval $-2 \leq x \leq 16$.

```

1 x = -2 : 0.05 : 16; % Smooth interval
2 f = (4 .* cos(x)) ./ (x + exp(-0.75 .* x));
3
4 plot(x, f);
5 title('Plot of f(x)');
6 xlabel('x');
7 ylabel('f(x)');
8 grid on;

```

Problem 22

A ship travels on a straight line course described by $y = (200 - 5x)/6$. The ship starts when $x = -20$ and ends when $x = 40$. Calculate the distance at closest approach to a lighthouse located at the origin (0,0) without using a plot.

```

1 % Define path range
2 x = -20 : 0.01 : 40;
3 y = (200 - 5 .* x) ./ 6;
4

```

```

5 % Distance formula d = sqrt(x^2 + y^2)
6 distances = sqrt(x.^2 + y.^2);
7
8 % Find minimum distance
9 min_dist = min(distances);
10
11 disp(['Closest approach distance: ', num2str(min_dist), ' km']);

```

Problem 23

Calculate work done $W = FD$ for five segments of a path given force and distance data. Find (a) work for each segment and (b) total work.

```

1 % Data vectors
2 Force = [400, 550, 700, 500, 600]; % Newtons
3 Distance = [3, 0.5, 0.75, 1.5, 5]; % Meters
4
5 % a. Work per segment (Element-wise multiplication)
6 Work_segments = Force .* Distance;
7
8 % b. Total work
9 Work_total = sum(Work_segments);
10
11 disp('Work per segment (J):');
12 disp(Work_segments);
13 disp(['Total Work (J): ', num2str(Work_total)]);

```

Problem 27

Calculate compression x and potential energy $PE = \frac{1}{2}kx^2$ for five springs given Force $F = kx$ and spring constant k .

```

1 % Data
2 F = [11, 7, 8, 10, 9]; % Force (N)
3 k = [1000, 600, 900, 1300, 700]; % Constant (N/m)
4
5 % a. Compression x = F / k
6 x = F ./ k;
7
8 % b. Potential Energy PE = 0.5 * k * x^2
9 PE = 0.5 .* k .* (x.^2);
10
11 % Display results table
12 disp(table(F', k', x', PE', 'VariableNames', {'Force', 'k', 'Compression', 'PE'}));

```


Problem 41

Solve the following system using the left-division method.

$$\begin{aligned}6x - 3y + 4z &= 41 \\12x + 5y - 7z &= -26 \\-5x + 2y - 6z &= 16\end{aligned}$$

```
1 % Coefficient Matrix A
2 A = [ 6, -3, 4;
3       12, 5, -7;
4       -5, 2, -6];
5
6 % Constant Vector B
7 B = [41; -26; 16];
8
9 % Solve for X = [x; y; z] using left division
10 Solution = A \ B;
11
12 disp('Solution [x; y; z]:');
13 disp(Solution);
```

Tutorial 3: Functions

Key Concepts and Common Pitfalls (Tutorial 3 Summary)

1. Anatomy of a User-Defined Function

A function must be defined in a separate file (usually) with the following syntax:

```
1 function [out1, out2] = my_func_name(in1, in2)
2     % Comments explaining the function (H1 line)
3
4     out1 = in1 + in2;    % Perform calculations
5     out2 = in1 .* in2;  % Assign values to output variables
6 end
```

Key Rules:

- **First Line:** Must start with the keyword `function`.
- **File Name:** The text file must be named exactly as the function name (e.g., `my_func_name.m`).
- **Inputs/Outputs:** Inputs are passed by value; outputs must be assigned within the function body before the function terminates.

Pitfall: Naming the file differently than the function name. MATLAB uses the **filename** to execute the function, not the name inside the file.

- File: `calc.m`
- Code: `function y = compute(x)`
- Result: You must call `calc(x)`, not `compute(x)`.

2. Anonymous Functions

Simple, one-line functions created without a separate file.

Syntax: `handle = @(arguments) expression`

Example:

```
1 F = @(x) 3*x.^2 + 2*x + 5;  
2 result = F(2); % Returns 21
```

Pitfall: Forgetting element-wise operators (`.*`, `./`, `.^`) in the definition.

- **Wrong:** `g = @(x) x^2;` (Fails if `x` is a vector)
- **Right:** `g = @(x) x.^2;`

3. Function Functions (Optimization & Zero Finding)

These are functions that accept *other* functions (as handles) as input arguments.

A. Finding a Minimum of a Single Variable: `fminbnd`

Used to find the minimum of a function $f(x)$ on a fixed interval $x_1 < x < x_2$.

Syntax: `[x, fval] = fminbnd(fun, x1, x2)`

Example: Find the minimum of $y = x^2 + 4\sin(x)$ between -3 and 3 .

```
1 fun = @(x) x.^2 + 4*sin(x);  
2 [x_min, val_min] = fminbnd(fun, -3, 3);  
3 % Returns x_min (location) and val_min (function value)
```

B. Finding a Zero (Root) of a Function: `fzero`

Used to find *where* a function crosses zero ($f(x) = 0$) near a guess x_0 .

Syntax: `x = fzero(fun, x0)`

Example: Find the zero of $y = \cos(x) - x$ near $x = 0$.

```
1 fun = @(x) cos(x) - x;  
2 x_zero = fzero(fun, 0);
```

C. Multivariable Minimization: `fminsearch`

Used to find the minimum of a function of *multiple variables* (unconstrained), starting at an initial guess vector x_0 .

Syntax: `[x, fval] = fminsearch(fun, x0)`

Example: Find the minimum of $z = x^2 + y^2$ starting at $[1, 1]$.

```
1 % Define function accepting a vector v where v(1)=x, v(2)=y
2 fun = @(v) v(1)^2 + v(2)^2;
3 start_point = [1, 1];
4 [v_min, val_min] = fminsearch(fun, start_point);
```

Pitfall: Confusing `fzero` (finds roots of non-polynomials) with `roots` (finds roots of polynomials only).

- Use `roots([1, 0, -5])` for $x^2 - 5$.
- Use `fzero(@(x) exp(x) - 5, 0)` for $e^x - 5$.

4. Variable Scope: Local vs. Global

- **Local Variables:** Variables defined inside a function are *local*. They are invisible to the MATLAB workspace and other functions. They are erased from memory when the function finishes.
- **Global Variables:** Variables declared as `global` (e.g., `global G`) are shared between the workspace and functions. Both must declare the variable as `global`.

Pitfall: Assuming a variable in your Workspace is available inside your function.

```
1 A = 5; % Defined in Workspace
2 % Inside function: y = A * x; -> Error! 'A' is unknown.
```

You must pass `A` as an input argument or declare it `global` (less recommended).

5. Subfunctions

You can define multiple functions in a single file.

- The **Primary Function** is the first one; it is callable from outside.
- **Subfunctions** follow the primary function; they are only callable by the primary function (or other subfunctions in the same file).

Pitfall: Trying to call a subfunction from the Command Window. It will not be found.

6. Comparison: Script vs. Function

Script	Function
No input/output arguments	Accepts inputs / returns outputs
Operates on Workspace variables	Uses local variables (mostly)
Useful for drivers/main logic	Useful for reusable modules

Tutorial Problems

Problem 10

An object thrown vertically with a speed v_0 reaches a height h at time t , where $h = v_0 t - \frac{1}{2}gt^2$. Write and test a function that computes the time t required to reach a specified height h , for a given value of v_0 . The function's inputs should be h, v_0, g . Test for $h = 100$ m, $v_0 = 50$ m/s, $g = 9.81$ m/s².

```
1 % --- Main Script ---
2 h = 100; v0 = 50; g = 9.81;
3
4 % Call the function
5 t_solutions = compute_time(h, v0, g);
6
7 disp('Times to reach 100m (seconds):');
8 disp(t_solutions);
9 % Interpretation: The object reaches 100m twice.
10 % Once on the way up, and once on the way down.
11
12 % --- Function Definition ---
13 function t = compute_time(h, v0, g)
14     % Solves 0.5*g*t^2 - v0*t + h = 0
15     % Using quadratic formula: ax^2 + bx + c = 0
16     % a = 0.5*g, b = -v0, c = h
17
18     roots_vec = roots([0.5*g, -v0, h]);
19     t = roots_vec;
20 end
```

Problem 17

The volume and paper surface area A of a conical paper cup are given by $V = \frac{1}{3}\pi r^2 h$ and $A = \pi r \sqrt{r^2 + h^2}$.

- Eliminate h to obtain A as a function of r and V .
- Create a function for A and use `fminbnd` to find r that minimizes A for $V = 10$ in³.

```

1 % --- Main Script ---
2 global V
3 V = 10; % Volume constraint
4
5 % Minimize Area function between r=0.1 and r=10
6 [r_min, A_min] = fminbnd(@cone_area, 0.1, 10);
7
8 % Calculate corresponding h
9 h_min = 3 * V / (pi * r_min^2);
10
11 disp(['Optimal r: ', num2str(r_min)]);
12 disp(['Optimal h: ', num2str(h_min)]);
13 disp(['Minimum Area: ', num2str(A_min)]);
14
15 % --- Function Definition ---
16 function A = cone_area(r)
17     global V
18     % Eliminate h: h = 3V / (pi*r^2)
19     h = 3 * V ./ (pi .* r.^2);
20     % Substitute into A
21     A = pi .* r .* sqrt(r.^2 + h.^2);
22 end

```

Problem 18

A torus with inner radius a and outer radius b has volume $V = \frac{1}{4}\pi^2(a+b)(b-a)^2$ and surface area $A = \pi^2(b^2 - a^2)$.

- Create a function for V and A .
- Plot A vs a for $0.25 \leq a \leq 4$ given $b = a + 2$.

```

1 % --- Main Script ---
2 a = 0.25 : 0.01 : 4;
3 b = a + 2; % Constraint
4
5 % Compute A and V using arrays
6 [V, A] = torus_calc(a, b);
7
8 plot(a, A);
9 title('Torus Surface Area vs Inner Radius a');
10 xlabel('a (inches)');
11 ylabel('Surface Area A');
12 grid on;

```

```

13
14 % --- Function Definition ---
15 function [V, A] = torus_calc(a, b)
16     V = 0.25 * pi^2 .* (a + b) .* (b - a).^2;
17     A = pi^2 .* (b.^2 - a.^2);
18 end

```

Problem 21

Create a function that will plot the entire ellipse $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$, given inputs a and b . Test for $a = 1, b = 2$.

```

1 % --- Main Script ---
2 plot_ellipse(1, 2);
3
4 % --- Function Definition ---
5 function plot_ellipse(a, b)
6     % Use parametric equations for full ellipse
7     t = linspace(0, 2*pi, 100);
8     x = a * cos(t);
9     y = b * sin(t);
10
11     figure;
12     plot(x, y);
13     title(['Ellipse: a=', num2str(a), ', b=', num2str(b)]);
14     axis equal;
15     grid on;
16 end

```

Problem 25

Create an anonymous function for $30x^2 - 300x + 4$.

- Plot to approximate minimum.
- Use `fminbnd` to determine the precise minimum location.

```

1 f = @(x) 30*x.^2 - 300*x + 4;
2
3 % a. Plotting
4 x_plot = -5:0.1:15;
5 plot(x_plot, f(x_plot));
6 grid on; title('Plot of 30x^2 - 300x + 4');
7
8 % b. Finding minimum
9 [x_min, val_min] = fminbnd(f, 0, 10);
10 disp(['Minimum occurs at x = ', num2str(x_min)]);

```

Problem 31

Estimate the three coefficients a, b, c of the logistic growth model $y(t) = \frac{c}{1+ae^{-bt}}$ using the provided data and `fminsearch`.

```
1 % Data
2 t = 0:15;
3 y_data = [13, 16, 20, 25, 31, 39, 45, 49, 55, 63, 69, 77, 82, 86, 89, 92];
4
5 % Model Function: y = c / (1 + a*exp(-b*t))
6 model_fun = @(p, t) p(3) ./ (1 + p(1) * exp(-p(2) * t));
7
8 % Error Function (Sum of Squared Errors)
9 err_fun = @(p) sum((y_data - model_fun(p, t)).^2);
10
11 % Initial Guess: c around 100 (max percent), a and b generic guesses
12 guess = [10, 0.5, 100];
13
14 % Optimization
15 p_opt = fminsearch(err_fun, guess);
16 a_est = p_opt(1); b_est = p_opt(2); c_est = p_opt(3);
17
18 % Plotting results
19 t_smooth = 0:0.1:15;
20 y_fit = model_fun(p_opt, t_smooth);
21
22 plot(t, y_data, 'ko', t_smooth, y_fit, 'b-');
23 legend('Data', 'Logistic Fit');
24 title('Logistic Growth Regression');
25 disp(['Estimated: a=', num2str(a_est), ', b=', num2str(b_est), ', c=', num2str(c_est)]);
```

Tutorial 4: Programming with MATLAB

Key Concepts and Common Pitfalls (Tutorial 4 Summary)

1. Relational and Logical Operators

MATLAB uses specific symbols for comparisons. A common source of bugs is confusing assignment with equality.

Operator	Description	Operator	Description
<code>==</code>	Equal to	<code>~=</code>	Not equal to
<code><</code>	Less than	<code><=</code>	Less than or equal to
<code>></code>	Greater than	<code>>=</code>	Greater than or equal to

Pitfall: Confusing `=` (assignment) with `==` (comparison).

```
1 if x = 5    % Error! Assigns 5 to x inside the condition.
2 if x == 5   % Correct. Checks if x is equal to 5.
```

Logical Operators & Short-Circuiting

MATLAB distinguishes between element-wise and short-circuit operators.

- **Element-wise** (`&`, `|`, `~`): Operates on arrays. Returns an array of logicals.
- **Short-circuit** (`&&`, `||`): Operates on **scalars** only. Used primarily in `if` and `while` statements.
 - `A && B`: Evaluates A. If A is false, it stops (B is never evaluated).
 - `A || B`: Evaluates A. If A is true, it stops (B is never evaluated).

Order of Precedence:

1. Arithmetic operations (`+`, `*`, `^`)
2. Relational operations (`>`, `<`, `==`)
3. Logical operations (`~`, `&`, `|`)

2. Conditional Branching

The if-elseif-else Structure

Evaluates expressions sequentially. The first true expression executes its block, and the structure terminates.

```
1 if x < 0
2     y = -x;
3 elseif x == 0
4     y = 0;
5 else
6     y = x^2;
7 end
```

The switch Structure

An alternative to `if` when comparing a single variable against specific distinct values (cases). It is often more readable for discrete logic.

```
1 switch units
2     case {'inch', 'in'}
3         y = x * 2.54;
4     case {'meter', 'm'}
5         y = x * 100;
6     otherwise
7         disp('Unknown unit');
8 end
```

Pitfall: Using `switch` for range comparisons (e.g., $x < 5$). `switch` checks for **equality** only. Use `if` for ranges.

3. Iterative Structures (Loops)

The for Loop

Used when the number of iterations is known **before** the loop starts.

```
1 for k = 1:2:10
2     x(k) = k^2;
3 end
```

Note: If you iterate over a matrix **A** (for **k = A**), MATLAB iterates over the **columns** of **A**.

The while Loop

Used when the number of iterations is unknown and depends on a condition (e.g., convergence errors).

```
1 error = 100;
2 while error > 0.01
3     % Update estimate
4     % Update error
5 end
```

Pitfall: Creating an **Infinite Loop**. You must ensure the variables inside the **while** condition change; otherwise, the loop never ends.

```
1 x = 5;
2 while x > 0
3     disp(x);
4     % Missing x = x - 1; -> Infinite loop!
5 end
```

4. Logical Indexing vs. The find Command

Extracting data based on conditions is a core MATLAB skill.

Method A: Logical Masking (Preferred for simple replacement) Returns a logical array (1s and 0s).

```
1 A = [5, -2, 3];
2 mask = A < 0;    % mask = [0, 1, 0]
3 A(mask) = 0;     % A becomes [5, 0, 3]
```

Method B: The find Command Returns the **indices** where the condition is true.

```
1 indices = find(A < 0); % indices = 2
```

Pitfall: Using **find** when a logical mask suffices.

- **Bad:** `A(find(A>5)) = 0;` (Slower, unnecessary function call)
- **Good:** `A(A>5) = 0;` (Faster, cleaner)

5. Performance: Pre-allocation

MATLAB arrays are dynamic, but resizing them inside a loop is computationally expensive (slow). Always "pre-allocate" memory (reserve space) before the loop.

Without Pre-allocation (Slow):

```
1 for k = 1:10000
2     y(k) = k^2; % MATLAB must resize 'y' 10,000 times!
3 end
```

With Pre-allocation (Fast):

```
1 y = zeros(1, 10000); % Create full array first
2 for k = 1:10000
3     y(k) = k^2; % Fills existing slots
4 end
```

6. Loop Control: break vs continue

- **break**: Terminates the loop entirely. Execution jumps to the statement **after** the **end**.
- **continue**: Skips the rest of the **current iteration** and jumps to the next iteration.

Example:

```
1 for k = 1:5
2     if k == 2
3         continue; % Skips 2, goes to 3
4     end
5     if k == 4
6         break; % Stops loop completely at 4
7     end
8     disp(k); % Displays: 1, 3
9 end
```

Tutorial Problems

Problem 2

The roots of $ax^2 + bx + c = 0$ are $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Write a program to compute both roots, identifying real and imaginary parts. Test for cases: (1) 2, 10, 12 (2) 3, 24, 48 (3) 4, 24, 100.

```
1 % Define test cases
2 cases = [2, 10, 12;
3          3, 24, 48;
4          4, 24, 100];
5
6 for i = 1:size(cases, 1)
7     a = cases(i, 1); b = cases(i, 2); c = cases(i, 3);
8
9     disc = b^2 - 4*a*c;
10
11     if disc > 0
12         x1 = (-b + sqrt(disc))/(2*a);
13         x2 = (-b - sqrt(disc))/(2*a);
14         type = 'Real and distinct';
15     elseif disc == 0
16         x1 = -b/(2*a);
17         x2 = x1;
18         type = 'Real and repeated';
19     else
20         real_part = -b/(2*a);
21         imag_part = sqrt(abs(disc))/(2*a);
22         x1 = complex(real_part, imag_part);
23         x2 = complex(real_part, -imag_part);
24         type = 'Complex conjugates';
25     end
26
27     disp(['Case ', num2str(i), ': ', type]);
28     disp(['Roots: ', num2str(x1), ' and ', num2str(x2)]);
29 end
```

Problem 9

Determine how many days the price of stock A was below the price of stock B given arrays.

```
1 price_A = [19, 18, 22, 21, 25, 19, 17, 21, 27, 29];
2 price_B = [22, 17, 20, 23, 24, 18, 16, 25, 28, 27];
3
4 % Logical comparison
```

```

5 days_below = price_A < price_B;
6
7 % Count true values
8 num_days = sum(days_below);
9
10 disp(['Days A was below B: ', num2str(num_days)]);

```

Problem 16

In this problem, we write a MATLAB script using conditional statements to evaluate the piecewise-defined function

$$y(x) = \begin{cases} e^x + 1, & x < -1, \\ 2 + \cos(\pi x), & -1 \leq x < 5, \\ 10(x - 5) + 1, & x \geq 5. \end{cases}$$

Using the script, we evaluate y at $x = -5$, $x = 3$, and $x = 15$, and then verify the results by hand.

By-hand check:

$$\begin{aligned}
y(-5) &= e^{-5} + 1 \approx 1.0067379, \\
y(3) &= 2 + \cos(3\pi) = 2 - 1 = 1, \\
y(15) &= 10(15 - 5) + 1 = 101.
\end{aligned}$$

```

1 % --- Main Script (Problem 16) ---
2 xs = [-5, 3, 15];
3
4 for k = 1:length(xs)
5     x = xs(k);
6
7     if x < -1
8         y = exp(x) + 1;
9     elseif x ≥ -1 && x < 5
10        y = 2 + cos(pi*x);
11    else % x ≥ 5
12        y = 10*(x - 5) + 1;
13    end
14
15    disp(['x = ', num2str(x), ' -> y = ', num2str(y)]);
16 end

```

Problem 21

In this problem, we create a MATLAB function `fxxy(x,y)` to evaluate a piecewise-defined function $f(x,y)$ based on the signs of x and y . The function is defined as:

$$f(x,y) = \begin{cases} x+y, & x \geq 0, y \geq 0, \\ x-y, & x \geq 0, y < 0, \\ -x^2y, & x < 0, y \geq 0, \\ -x^2y^2, & x < 0, y < 0. \end{cases}$$

To verify correctness, we evaluate the function at four test points: $(1,1)$, $(1,-1)$, $(-1,1)$, and $(-1,-1)$, which cover all four regions.

```
1 % --- Main Script ---
2 disp(['f(1,1) = ', num2str(fxy(1,1))]);
3 disp(['f(1,-1) = ', num2str(fxy(1,-1))]);
4 disp(['f(-1,1) = ', num2str(fxy(-1,1))]);
5 disp(['f(-1,-1) = ', num2str(fxy(-1,-1))]);
6
7 % --- Function Definition ---
8 function val = fxy(x, y)
9     if x ≥ 0 && y ≥ 0
10         val = x + y;
11     elseif x ≥ 0 && y < 0
12         val = x - y;
13     elseif x < 0 && y ≥ 0
14         val = -x^2 * y;
15     else % x < 0 and y < 0
16         val = -x^2 * y^2;
17     end
18 end
```

Problem 28

Consider the matrix

$$A = \begin{bmatrix} 3 & 5 & -4 \\ -8 & -1 & 33 \\ -17 & 6 & -9 \end{bmatrix}.$$

We compute an array B by applying the following rule to each element of A :

$$B_{ij} = \begin{cases} \ln(A_{ij}) + 20, & A_{ij} \geq 1, \\ A_{ij}, & A_{ij} < 1. \end{cases}$$

This is done in two ways: (a) using a `for` loop with conditional statements, and (b) using a logical mask.

Expected result (approx.):

$$B \approx \begin{bmatrix} 21.0986 & 21.6094 & -4 \\ -8 & -1 & 23.4965 \\ -17 & 21.7918 & -9 \end{bmatrix}.$$

```
1 % --- Main Script (Problem 28) ---
2 A = [ 3 5 -4;
3       -8 -1 33;
4       -17 6 -9];
5
6 %% (a) Using a for loop + conditionals
7 B1 = A; % start by copying A
8 [m,n] = size(A);
9
10 for i = 1:m
11     for j = 1:n
12         if A(i,j) >= 1
13             B1(i,j) = log(A(i,j)) + 20; % natural log + 20
14         end
15     end
16 end
17
18 %% (b) Using a logical mask
19 B2 = A; % start by copying A
20 mask = (A >= 1); % logical array (true where condition holds)
21 B2(mask) = log(A(mask)) + 20;
22
23 %% Display results
24 disp('A ='); disp(A);
25 disp('B1 (loop) ='); disp(B1);
26 disp('B2 (mask) ='); disp(B2);
27
28 % Check they match (should be all zeros)
29 disp('Max difference between B1 and B2:');
30 disp(max(abs(B1(:) - B2(:))));
```

Problem 40

A weight W is supported by two cables anchored a distance D apart. The left cable length L_{AB} is known, while the right cable length L_{AC} must be selected. For static equilibrium, the horizontal and vertical force sums at point B must be zero, giving

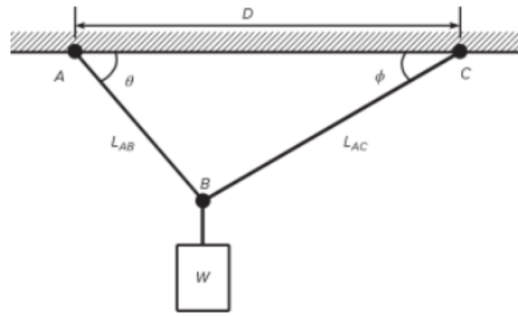


Figure P40

$$\begin{aligned} -T_{AB} \cos \theta + T_{AC} \cos \phi &= 0 \\ T_{AB} \sin \theta + T_{AC} \sin \phi &= W \end{aligned}$$

Figure 1: Cable system showing weight W supported by two cables with lengths L_{AB} and L_{AC} , anchored at distance D apart.

$$-T_{AB} \cos \theta + T_{AC} \cos \phi = 0, \quad T_{AB} \sin \theta + T_{AC} \sin \phi = W.$$

The angles θ and ϕ depend on the cable lengths. Using triangle geometry:

$$\theta = \cos^{-1} \left(\frac{D^2 + L_{AB}^2 - L_{AC}^2}{2DL_{AB}} \right), \quad \phi = \sin^{-1} \left(\frac{L_{AB} \sin \theta}{L_{AC}} \right).$$

Given $D = 6$ ft, $L_{AB} = 3$ ft, and $W = 2000$ lb, we use a **while** loop in MATLAB to find $L_{AC,\min}$ (the shortest L_{AC} such that neither T_{AB} nor T_{AC} exceeds 2000 lb). Then we plot T_{AB} and T_{AC} versus L_{AC} for $L_{AC,\min} \leq L_{AC} \leq 6.7$.

```

1 % --- Main Script (Problem 40) ---
2 clear; clc; close all;
3
4 D = 6;           % ft
5 LAB = 3;         % ft
6 W = 2000;        % lb
7 LAC_max = 6.7;   % ft (given)
8
9 % Step size for searching LAC_min
10 dL = 1e-3;
11
12 % Start from just above the triangle lower bound |D-LAB| = 3
13 LAC = abs(D - LAB) + 1e-4;
14
15 TAB = inf;  TAC = inf;
16
17 % ---- WHILE LOOP to find LAC_min ----
18 while (TAB > W) || (TAC > W)

```

```

19     % Compute angles (radians)
20     theta = acos((D^2 + LAB^2 - LAC^2)/(2*D*LAB));
21     phi    = asin((LAB*sin(theta))/LAC);
22
23     % Solve equilibrium for [TAB; TAC]
24     A = [-cos(theta), cos(phi);
25          sin(theta), sin(phi)];
26     b = [0; W];
27
28     T = A\b;          % T(1)=TAB, T(2)=TAC
29     TAB = T(1);
30     TAC = T(2);
31
32     % Increase LAC until both tensions are ≤ W
33     if (TAB > W) || (TAC > W)
34         LAC = LAC + dL;
35     end
36
37     % Safety stop (should not happen for this problem)
38     if LAC > LAC_max
39         error('No feasible LAC found up to LAC_max.');
```

```

40     end
41 end
42
43 LAC_min = LAC;
44 fprintf('LAC_min = %.4f ft\n', LAC_min);
45 fprintf('TAB = %.2f lb, TAC = %.2f lb\n', TAB, TAC);
46
47 % ---- Compute tensions for plotting from LAC_min to 6.7 ----
48 Lvec = LAC_min:dL:LAC_max;
49 TABv = zeros(size(Lvec));
50 TACv = zeros(size(Lvec));
51
52 for k = 1:length(Lvec)
53     LAC = Lvec(k);
54
55     theta = acos((D^2 + LAB^2 - LAC^2)/(2*D*LAB));
56     phi    = asin((LAB*sin(theta))/LAC);
57
58     A = [-cos(theta), cos(phi);
59          sin(theta), sin(phi)];
60     b = [0; W];
61
62     T = A\b;
63     TABv(k) = T(1);
64     TACv(k) = T(2);
65 end
66
67 % ---- Plot ----
68 figure;
69 plot(Lvec, TABv, 'LineWidth', 1.5); hold on;
70 plot(Lvec, TACv, 'LineWidth', 1.5);
71 yline(W, '--', 'LineWidth', 1.2); % limit line at 2000 lb
72 grid on;

```

```

73 xlabel('L_{AC} (ft)');
74 ylabel('Tension (lb)');
75 title('T_{AB} and T_{AC} vs. L_{AC}');
76 legend('T_{AB}', 'T_{AC}', 'W = 2000 lb', 'Location', 'best');

```

Problem 42

The circuit in Fig. P42 is governed by the five equations

$$-v_1 + R_1 i_1 + R_4 i_4 = 0, \quad -R_4 i_4 + R_2 i_2 + R_5 i_5 = 0, \quad -R_5 i_5 + R_3 i_3 + v_2 = 0,$$

$$i_1 = i_2 + i_4, \quad i_2 = i_3 + i_5.$$

Given $R_1 = 5 \text{ k}\Omega$, $R_2 = 100 \text{ k}\Omega$, $R_3 = 200 \text{ k}\Omega$, $R_4 = 150 \text{ k}\Omega$, $R_5 = 250 \text{ k}\Omega$ and $v_1 = 100 \text{ V}$, each resistor is rated for a current magnitude no larger than $I_{\max} = 1 \text{ mA}$.

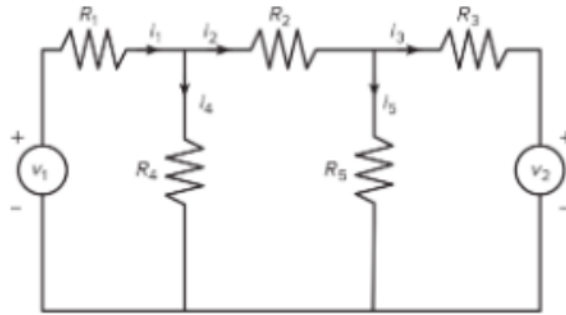


Figure P42

Figure 2: Circuit diagram for Problem 42.

(a) We determine the allowable range of *positive* values of v_3 such that

$$|i_1|, |i_2|, |i_3|, |i_4|, |i_5| \leq I_{\max}.$$

Result for the given numbers (approx.):

$$31.67 \text{ V} \leq v_2 \leq 742.31 \text{ V}$$

(the lower bound is set by $|i_1| \leq 1 \text{ mA}$ and the upper bound is set by $|i_5| \leq 1 \text{ mA}$).

(b) To study the effect of R_3 , we vary R_3 from 150 to 250 $\text{k}\Omega$ and compute the corresponding upper allowable limit on v_2 , then plot $v_{2,\max}$ versus R_3 .

```

1 % --- Main Script (Problem 42) ---
2 clear; clc; close all;
3
4 % Given values (kOhm -> Ohm)
5 R1 = 5e3;    R2 = 100e3;  R3 = 200e3;  R4 = 150e3;  R5 = 250e3;
6 v1 = 100;           % V
7 Imax = 1e-3;        % A
8
9 % Helper function: compute allowable v2 interval for a given R3
10 % Returns [v2min, v2max] for v2 > 0, based on |currents| ≤ Imax
11 allowable_v2 = @(R3val) local_allowable_v2(R1,R2,R3val,R4,R5,v1,Imax);
12
13 %% (a) Allowable range for the given R3
14 [v2min, v2max] = allowable_v2(R3);
15 fprintf('(a) Allowable v2 range: %.4f V ≤ v2 ≤ %.4f V\n', v2min, v2max);
16
17 %% (b) Vary R3 from 150 to 250 kOhm, plot the upper allowable limit v2max
18 R3vec = linspace(150e3, 250e3, 200);
19 v2max_vec = zeros(size(R3vec));
20
21 for k = 1:length(R3vec)
22     [~, v2max_vec(k)] = allowable_v2(R3vec(k));
23 end
24
25 figure;
26 plot(R3vec/1e3, v2max_vec, 'LineWidth', 1.5);
27 grid on;
28 xlabel('R_3 (k\Omega)');
29 ylabel('Upper allowable v_2 (V)');
30 title('Allowable upper limit of v_2 vs. R_3');
31
32 %% ----- Local function (kept at end of script) -----
33 function [v2min, v2max] = local_allowable_v2(R1,R2,R3,R4,R5,v1,Imax)
34     % Unknowns: i1,i2,i3,i4,i5
35     % Build linear system A*i = b, where b depends on v2.
36     %
37     % Equations:
38     % -v1 + R1 i1 + R4 i4 = 0
39     % -R4 i4 + R2 i2 + R5 i5 = 0
40     % -R5 i5 + R3 i3 + v2 = 0
41     % i1 - i2 - i4 = 0
42     % i2 - i3 - i5 = 0
43
44     A = [ R1,    0,    0,    R4,    0;
45           0,    R2,    0,   -R4,    R5;
46           0,    0,    R3,    0,   -R5;
47           1,   -1,    0,   -1,    0;
48           0,    1,   -1,    0,   -1];
49
50     % Solve i(v2) = a + b*v2 by two solves: v2=0 and v2=1
51     b0 = [v1; 0; 0; 0; 0]; % v2 = 0 -> third equation RHS is 0
52     b1 = [v1; 0; -1; 0; 0]; % v2 = 1 -> third equation becomes R3*i3 - R5*i5 = -1
53

```

```

54     i0 = A\b0;                % currents when v2 = 0
55     i1 = A\b1;                % currents when v2 = 1
56     slope = i1 - i0;          % di/dv2
57     offset = i0;              % i(v2)=offset + slope*v2
58
59     % For each current: |offset + slope*v2| ≤ Imax gives an interval in v2
60     v_low = -inf;
61     v_high = inf;
62
63     for k = 1:5
64         a = offset(k);
65         m = slope(k);
66
67         if abs(m) < 1e-15
68             % current independent of v2
69             if abs(a) > Imax
70                 v_low = 1; v_high = 0; % empty interval
71                 break;
72             end
73         else
74             % Solve  $-I_{\max} \leq a + m \cdot v_2 \leq I_{\max}$ 
75             v1k = (-Imax - a)/m;
76             v2k = ( Imax - a)/m;
77             lo = min(v1k, v2k);
78             hi = max(v1k, v2k);
79
80             v_low = max(v_low, lo);
81             v_high = min(v_high, hi);
82         end
83     end
84
85     % Also require v2 > 0
86     v_low = max(v_low, 0);
87
88     v2min = v_low;
89     v2max = v_high;
90 end

```

Problem 44

We are given the MATLAB script:

```

1  k = 1; b = -2; x = -1; y = -2;
2  while k ≤ 3
3      k, b, x, y
4      y = x^2 - 3;
5      if y < b
6          b = y;
7      end
8      x = x + 1;

```

```

9      k = k + 1;
10 end

```

The line `k, b, x, y` displays the values *immediately after entering the while-loop body*, i.e., before updating y , possibly updating b , and incrementing x and k . Since the loop condition is $k \leq 3$, the loop executes exactly three times.

Pass	k	b	x	y
First	1	-2	-1	-2
Second	2	-2	0	-2
Third	3	-3	1	-3

Quick check (updates each pass):

Pass 1: $y = (-1)^2 - 3 = -2$, b stays -2 , $x \rightarrow 0$, $k \rightarrow 2$,
 Pass 2: $y = (0)^2 - 3 = -3$, $b \rightarrow -3$, $x \rightarrow 1$, $k \rightarrow 3$,
 Pass 3: $y = (1)^2 - 3 = -2$, b stays -3 , $x \rightarrow 2$, $k \rightarrow 4$ (stop).

Pass	k	b	x	y
First				
Second				
Third				
Fourth				
Fifth				

Figure 3: Figure for Problem 44.

Tutorial 9: Numerical Methods

Key Concepts and Common Pitfalls (Tutorial 9 Summary)

1. Numerical Integration (Quadrature)

MATLAB provides two primary approaches for integration: using function handles (for mathematical formulas) or data points (for experimental data).

A. Integrating a Function Handle: `integral`

Uses adaptive Simpson's rule. High accuracy.

- **Syntax:** `q = integral(fun, a, b)`
- **Example:** $\int_0^\pi \sin(x) dx$

```
1 fun = @(x) sin(x);  
2 area = integral(fun, 0, pi); % Returns 2.0
```

B. Integrating Data Points: `trapz`

Uses the Trapezoidal Rule. Used when you have vectors of data x and y , not a formula.

- **Syntax:** `area = trapz(x, y)`

```
1 x = 0:0.1:pi;  
2 y = sin(x);  
3 area = trapz(x, y); % Approx 2.0 (depends on spacing)
```

Pitfall: Confusing the two methods.

- You cannot pass a vector to `integral`.
- You cannot pass a function handle to `trapz` (unless you evaluate it first).

2. Numerical Differentiation

Differentiation is sensitive to "noise" in data. MATLAB uses the `diff` function to calculate differences between adjacent elements.

Syntax: `d = diff(x)`

- Result vector is 1 element shorter than the input vector ($N - 1$ elements).
- **Approximate Derivative:** $\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x}$

```
1 x = [0, 1, 2, 3];
2 y = x.^2;           % [0, 1, 4, 9]
3 dy = diff(y);       % [1, 3, 5] (Length is 3)
4 dx = diff(x);       % [1, 1, 1]
5 deriv = dy ./ dx;
```

Pitfall: Plotting the derivative against the original x vector.

```
1 plot(x, deriv) % Error! Vectors must be same length.
```

Fix: Use `x(1:end-1)` or calculate a midpoint vector for plotting.

3. Solving ODEs (ode45)

The workhorse for solving Ordinary Differential Equations in MATLAB is `ode45`. It solves systems of the form $\frac{dy}{dt} = f(t, y)$.

A. The Basic Syntax

`[t, y] = ode45(ode_fun, t_span, initial_conditions)`

- **ode_fun:** A handle `@(t, y) ...` that returns the column vector of derivatives.
- **t_span:** `[t_start, t_end]`
- **initial_conditions:** Vector of starting values for y (and y' if higher order).

B. Solving Higher-Order ODEs

You must convert higher-order ODEs into a system of first-order ODEs using ****State Variables****.

Example: Mass-Spring-Damper $\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$ 1. Let $x_1 = x$ (Position) 2. Let $x_2 = \dot{x}$ (Velocity) 3. Derivatives:

- $\dot{x}_1 = x_2$
- $\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$

```
1 % Parameters: m=1, c=2, k=5, F=0
2 ode_sys = @(t, x) [x(2); (1/1)*(0 - 2*x(2) - 5*x(1))];
3 [t, sol] = ode45(ode_sys, [0, 10], [1; 0]); % Init: pos=1, vel=0
```

Pitfall: The derivative function MUST return a **column vector**.

- **Wrong:** $[x(2), -x(1)]$ (Row vector)
- **Right:** $[x(2); -x(1)]$ (Column vector)

4. ODE Events (Stopping Early)

Sometimes you need to stop integration based on a condition (e.g., "stop when the rocket hits the ground, $h = 0$ "), not just time.

Steps: 1. Define an event function. 2. Set options using `odeset`. 3. Pass options to `ode45`.

```
1 function [value, isterminal, direction] = my_event(t, y)
2     value = y(1);      % Detect when y(1) (height) = 0
3     isterminal = 1;    % 1 = Stop integration
4     direction = -1;    % -1 = Only detect falling (neg slope)
5 end
6
7 % Usage
8 opts = odeset('Events', @my_event);
9 [t, y] = ode45(fun, [0, 100], [10; 0], opts);
```

5. Summary of Functions

Function	Purpose
<code>integral(fun, a, b)</code>	Numerical integration of a formula
<code>trapz(x, y)</code>	Numerical integration of data arrays
<code>diff(x)</code>	Difference between adjacent elements
<code>gradient(M)</code>	Numerical gradient of a matrix
<code>ode45</code>	Standard ODE solver (Runge-Kutta)
<code>odeset</code>	Create options structure for ODE solvers

6. Control Systems: Transfer Functions and State Variable Form

MATLAB's Control System Toolbox provides specialized tools for modeling and analyzing Linear Time-Invariant (LTI) systems. You can define these systems in two primary ways: Transfer Functions and State-Space models.

A. Transfer Functions (`tf`)

A transfer function represents the relationship between the output signal of a control system and the input signal, for all possible input values.

Syntax: `sys = tf(right, left)`

- **right:** A vector containing the coefficients on the right side of the equation, arranged in descending derivative order.
- **left:** A vector containing the coefficients on the left side of the equation, also arranged in descending derivative order.

Example: Consider the differential equation: $5\ddot{y} + 7\dot{y} + 5y = 5\dot{f} + f(t)$.

```
1 % Create the transfer function model form named sys
2 sys = tf([5, 1], [5, 7, 5]);
3
4 % Plot the unit step response for zero initial conditions
5 step(sys);
```

B. State-Space Form (ss)

For linear differential equations, you can organize your state variables into a standardized matrix format known as the state variable form. This is especially useful for systems with multiple interacting variables.

The standard state-space model relies on four matrices (A, B, C, and D):

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

Syntax: `sys = ss(A, B, C, D)`

Specific Example: Mass-Spring-Damper System

Let's model a mechanical system with mass $m = 2$, damping surface friction $c = 5$, and spring stiffness $k = 3$. The output we want to track is the position, so $y = x_1$.

[Image of a mass-spring-damper free body diagram]

Step 1. The Starting Point: Newton's Second Law

For a Mass-Spring-Damper system, the fundamental equation of motion is based on the sum of forces ($\Sigma F = ma$). The basic second-order differential equation is:

$$m\ddot{x} + c\dot{x} + kx = F$$

Where:

- m = mass
- c = damping coefficient
- k = spring stiffness
- x = position
- \dot{x} = velocity (first derivative of position)
- \ddot{x} = acceleration (second derivative of position)
- F = external force (often written as $u(t)$ in control systems)

If we rearrange this equation to solve for acceleration (\ddot{x}), we divide everything by m :

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$$

Step 2. Defining the State Variables (Why $\dot{x}_1 = x_2$)

To reduce this second-order equation into first-order equations, we invent new variables called "state variables" to represent the distinct physical states of the system (position and velocity).

- Let $x_1 = \text{Position } (x)$
- Let $x_2 = \text{Velocity } (\dot{x})$

Now, let's take the first derivative of x_1 : If x_1 is position, then taking its derivative with respect to time (\dot{x}_1) gives us velocity (\dot{x}). Since we already defined velocity as x_2 , it mathematically follows that:

$$\dot{x}_1 = x_2$$

Step 3. Substituting into the Original Equation (Why $\dot{x}_2 = \dots$)

Now we need an equation for the derivative of our second state variable, \dot{x}_2 . Since x_2 is velocity (\dot{x}), its derivative \dot{x}_2 is acceleration (\ddot{x}).

We go back to our rearranged equation of motion from Step 1:

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$$

Substitute our new state variables into this equation:

- Replace acceleration (\ddot{x}) with \dot{x}_2 .
- Replace velocity (\dot{x}) with x_2 .
- Replace position (x) with x_1 .
- Replace the input force F with $u(t)$ (standard notation for inputs).

This gives us the final translated equation:

$$\dot{x}_2 = \frac{1}{m}u(t) - \frac{k}{m}x_1 - \frac{c}{m}x_2$$

Mathematical Explanation of Matrices A, B, C, and D:

These equations can be put into matrix form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t)$$

- **Matrix A (System Matrix):** Defines the internal system dynamics based on the coefficients of x_1 and x_2 . Here, $A = \begin{bmatrix} 0 & 1 \\ -k/m & -c/m \end{bmatrix}$.
- **Matrix B (Input Matrix):** Defines how the external input $u(t)$ enters the system. Since the force only directly affects acceleration (\dot{x}_2), the first row is 0 and the second is $1/m$, making $B = \begin{bmatrix} 0 \\ 1/m \end{bmatrix}$.
- **Matrix C (Output Matrix):** Maps the state variables to the desired output. Since we want position ($y = x_1$), we take $1 \cdot x_1$ and $0 \cdot x_2$, so $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$.

- **Matrix D (Feedthrough Matrix):** Represents any direct routing from input to output. For most mechanical systems without direct feedthrough, $D = [0]$.

To build this in MATLAB using the `ss` function, we define the exact values for the A , B , C , and D matrices:

```

1 % Define parameters
2 m = 2; c = 5; k = 3;
3
4 % Define Matrices
5 A = [0, 1; -k/m, -c/m];
6 B = [0; 1/m];
7 C = [1, 0];
8 D = 0;
9
10 % Create the LTI state-space object
11 sys = ss(A, B, C, D);

```

C. Solving with ode45 vs. State-Space

While the State-Space (`ss`) form is incredibly powerful for Linear Time-Invariant (LTI) systems, MATLAB's `ode45` is a general-purpose numerical solver that can handle both linear and nonlinear systems.

To use `ode45` for equations higher than order 1, you must also write the equation as a set of first-order equations (often called the Cauchy form or the state-variable form). However, instead of strictly passing defined matrices, you pass a function handle that computes the column vector of derivatives $f(t, y)$.

Using `ode45` for the same Mass-Spring-Damper System:

```

1 function xdot = msd(t, x)
2     % Define parameters and a sample constant input force
3     m = 2; c = 5; k = 3; u = 10;
4
5     % Define matrices for clean calculation
6     A = [0, 1; -k/m, -c/m];
7     B = [0; 1/m];
8
9     % Return column vector of derivatives
10    xdot = A*x + B*u;
11 end
12
13 % Call ode45 in the main script
14 % For 0 ≤ t ≤ 5, with initial conditions x1(0)=0, x2(0)=0
15 [t, x] = ode45(@msd, [0, 5], [0; 0]);

```

Key Differences for Engineering Applications:

- **Linearity Restrictions:** The `ss` command requires a strictly linear system. The `ode45` function can seamlessly simulate nonlinearities, such as a pendulum where the state equation relies on $-\frac{g}{L} \sin(x_1)$ instead of a linear coefficient.
- **Toolbox Integration:** Creating a `sys` object unlocks the Control System Toolbox, allowing you to use high-level, single-command analysis functions like `step(sys)`, `impz(sys)`, and `lsim(sys)`. Conversely, `ode45` returns raw time and state data that you must plot and analyze manually using standard plotting commands like `plot(t, x(:,1))`.

C. Simulating Linear Time-Invariant (LTI) Objects

Once your model is created (using either `tf` or `ss`), MATLAB has several functions to evaluate how it behaves over time.

- `step(sys)`: Computes and plots the unit-step response of the LTI object `sys`.
- `impz(sys)`: Computes and plots the unit-impulse response of the LTI object `sys`.
- `initial(sys, x0)`: Computes and plots the free response of the LTI object `sys` given in state-model form, for the initial conditions specified in the vector `x0`.
- `lsim(sys, u, t)`: Simulated time response. Computes and plots the response of the LTI object `sys` to the input specified by the vector `u`, at the times specified by the vector `t`.

Creating Custom Inputs: The `gensig` function makes it easy to construct periodic input functions.

- **Syntax:** `[u, t] = gensig(type, period)`
- The `type` can be defined as `'sin'`, `'square'`, or `'pulse'`.

Tutorial Problems

Problem 5

Acceleration $a(t) = 5t \sin(8t)$. Compute velocity at $t = 20$ if $v(0) = 0$.

```

1 % v(t) = integral of a(t) from 0 to 20
2 a_fun = @(t) 5 .* t .* sin(8 .* t);
3 v_20 = integral(a_fun, 0, 20);
4
5 disp(['Velocity at t=20: ', num2str(v_20), ' m/s']);

```

Problem 10

Rocket equation: $m(t)\frac{dv}{dt} = T - m(t)g$. Calculate velocity at burnout ($t = 40$). $T = 48000, m_0 = 2200, r = 0.8, g = 9.81$.

```

1 T = 48000; m0 = 2200; r = 0.8; g = 9.81; b = 40;
2
3 % ODE: dv/dt = T/m(t) - g
4 % m(t) = m0 * (1 - r*t/b)
5 dvdt = @(t, v) (T ./ (m0 * (1 - r*t/b))) - g;
6
7 [t_sol, v_sol] = ode45(dvdt, [0, b], 0);
8
9 disp(['Velocity at burnout: ', num2str(v_sol(end)), ' m/s']);
10 plot(t_sol, v_sol); title('Rocket Velocity'); xlabel('t'); ylabel('v');

```

Problem 21

Use the `diff` function to estimate the derivative of $y = e^{-2x} \frac{\sin(4x)}{x^2+3}$ at $x = 0.6$.

```

1 % Define x with fine resolution around 0.6
2 dx = 0.001;
3 x = 0 : dx : 1;
4 y = exp(-2*x) .* sin(4*x) ./ (x.^2 + 3);
5
6 % Calculate approximate derivative dy/dx
7 % diff(y) is difference between adjacent elements
8 % dividing by dx gives the slope
9 dydx = diff(y) ./ dx;
10
11 % Find index corresponding to x = 0.6
12 % Note: diff result is 1 element shorter than x
13 x_diff = x(1:end-1);
14 [~, idx] = min(abs(x_diff - 0.6));
15
16 deriv_val = dydx(idx);
17
18 disp(['Approximate derivative at x=0.6: ', num2str(deriv_val)]);
19
20 % Analytical check (optional, for verification)

```

```

21 % y' via Chain/Quotient Rule
22 x0 = 0.6;
23 % ... (manual calc omitted for brevity)

```

Problem 29

Spherical tank draining. $\pi(2rh - h^2)\frac{dh}{dt} = -C_d A \sqrt{2gh}$. Radius $r = 3$, drain radius 2cm (0.02m), $C_d = 0.5$, $h(0) = 5$. Estimate empty time.

```

1 r_tank = 3;
2 r_drain = 0.02;
3 A_drain = pi * r_drain^2;
4 Cd = 0.5; g = 9.81;
5
6 % ODE: dh/dt = - (Cd * A * sqrt(2gh)) / (pi * (2rh - h^2))
7 dhdt = @(t, h) -(Cd * A_drain * sqrt(2*g*h)) ./ (pi * (2*r_tank*h - h.^2));
8
9 % Integrate until h is near 0 (event function typically used, or guess time)
10 % Using ode45 with events to stop at h=0
11 options = odeset('Events', @stop_event);
12 [t, h] = ode45(dhdt, [0, 50000], 5, options);
13
14 disp(['Time to empty: ', num2str(t(end)/3600), ' hours']);
15 plot(t, h); title('Tank Draining');
16
17 % Event function definition
18 function [value, isterminal, direction] = stop_event(t, h)
19     value = h - 0.01; % Stop when height is 1cm
20     isterminal = 1;
21     direction = 0;
22 end

```

Problem 32

The motion of a mass is described by $3\ddot{y} + 18\dot{y} + 102y = f(t)$ with $f(t) = 0$ for $t < 0$ and $f(t) = 10$ for $t \geq 0$.

- Plot $y(t)$ for $y(0) = \dot{y}(0) = 0$
- Plot $y(t)$ for $y(0) = 0, \dot{y}(0) = 10$. Discuss the effect of nonzero initial velocity.

```

1 % Rewrite as: y_ddot = (1/3)*(f(t) - 18*y_dot - 102*y)
2 % State x1 = y, x2 = y_dot
3 % dx1 = x2
4 % dx2 = (1/3)*(f - 18*x2 - 102*x1)
5

```



```

6 t_span = [0, 5];
7 f_val = 10;
8
9 % a. Zero Initial Conditions
10 IC_a = [0; 0];
11 ode_a = @(t, x) [x(2); (1/3)*(f_val - 18*x(2) - 102*x(1))];
12 [t_a, y_a] = ode45(ode_a, t_span, IC_a);
13
14 % b. Non-zero Initial Velocity (y(0)=0, y_dot(0)=10)
15 IC_b = [0; 10];
16 ode_b = @(t, x) [x(2); (1/3)*(f_val - 18*x(2) - 102*x(1))];
17 [t_b, y_b] = ode45(ode_b, t_span, IC_b);
18
19 % Plotting
20 figure;
21 plot(t_a, y_a(:,1), 'b-', 'LineWidth', 1.5); hold on;
22 plot(t_b, y_b(:,1), 'r--', 'LineWidth', 1.5);
23 legend('Case A: Zero ICs', 'Case B: Init Vel = 10');
24 title('Response of Mass-Spring-Damper');
25 xlabel('Time (s)'); ylabel('Displacement y(t)');
26 grid on;

```

Problem 44

State model with $m = 1, c = 2, k = 5$:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -5 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

- Use the **initial** function to plot position x_1 if $x(0) = [5; 3]$.
- Use the **step** function to plot the response for step input of magnitude 10.

```

1 % Define Matrices
2 A = [0, 1; -5, -2];
3 B = [0; 1];
4 C = [1, 0]; % Output y = x1 (Position)
5 D = 0;
6
7 % Create State Space System (requires Control System Toolbox)
8 sys = ss(A, B, C, D);
9
10 % a. Initial Response (Free response to initial conditions)
11 x0 = [5; 3];
12 figure;
13 subplot(2,1,1);
14 initial(sys, x0);
15 title('a. Response to Initial Conditions x0=[5; 3]');
16 grid on;
17

```

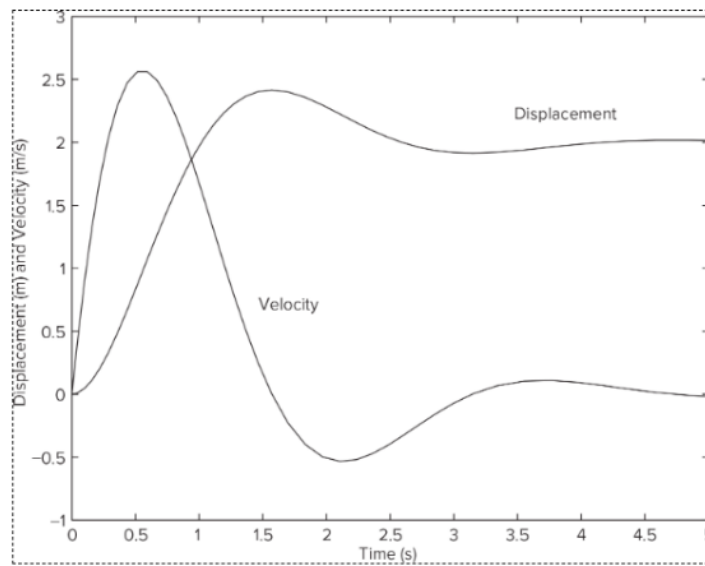


Figure 9.5-1 Displacement and velocity of the mass as a function of time.

```

18 % b. Step Response (Step magnitude 10)
19 % The standard step(sys) assumes input is 1.
20 % For magnitude 10, we scale the system or the input.
21 sys_scaled = sys * 10; % Scale input channel by 10
22
23 subplot(2,1,2);
24 step(sys_scaled);
25 title('b. Step Response (Input Magnitude 10)');
26 grid on;

```

Problem 45

Equation $5\ddot{y} + 2\dot{y} + 10y = f(t)$.

- Free response: $y(0) = 10, \dot{y}(0) = -5$.
- Step response: Zero ICs, unit step input.
- Total response superposition.

```

1 % State Space: x1 = y, x2 = y_dot
2 % y_ddot = (f - 2y_dot - 10y)/5
3 % dx1 = x2
4 % dx2 = 0.2f - 0.4x2 - 2x1

```

```

5
6 % a. Free Response (f=0)
7 ode_free = @(t, x) [x(2); -0.4*x(2) - 2*x(1)];
8 [t_free, x_free] = ode45(ode_free, [0, 15], [10; -5]);
9
10 % b. Step Response (f=1, IC=0)
11 ode_step = @(t, x) [x(2); 0.2*1 - 0.4*x(2) - 2*x(1)];
12 [t_step, x_step] = ode45(ode_step, [0, 15], [0; 0]);
13
14 % c. Total Response (f=1, IC=[10, -5])
15 ode_total = @(t, x) [x(2); 0.2*1 - 0.4*x(2) - 2*x(1)];
16 [t_tot, x_tot] = ode45(ode_total, [0, 15], [10; -5]);
17
18 % Plotting
19 figure;
20 plot(t_free, x_free(:,1), '--', 'DisplayName', 'Free'); hold on;
21 plot(t_step, x_step(:,1), ':', 'DisplayName', 'Step');
22 plot(t_tot, x_tot(:,1), 'k-', 'LineWidth', 1.5, 'DisplayName', 'Total');
23 legend; title('Superposition of Responses');

```