University of Windsor
Bob Little

Course Notes

February 26, 2026

Table of Contents

Tutorial 01: 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:

```
16^-1/2
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:

```
(3*y)/(4*x-8)    % Correct
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:

```
x1 = a*b*d;     % may overflow
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:

```
u = 0:0.1:10;
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:

```
plot(x,y)
xlabel('x')
ylabel('y')
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.

a. $(1 - \frac{1}{x^5})^{-1}$

b. $3\pi x^2$

c. $\frac{3y}{4x-8}$

d. $\frac{4(y-5)}{3x-6}$

```
clear; clc;
x = 5;
y = 2;

% a. (1 - 1/x^5)^-1
result_a = (1 - 1/x^5)^-1;

% b. 3 * pi * x^2
result_b = 3 * pi * x^2;

% c. (3*y) / (4*x - 8)
result_c = (3*y) / (4*x - 8);

% d. (4*(y - 5)) / (3*x - 6)
result_d = (4*(y - 5)) / (3*x - 6);

% Display results
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 $f = 19.83$.

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

```
clear; clc;
a = 1.12; b = 2.34; c = 0.72; d = 0.81; f = 19.83;

x = 1 + a/b + c/f^2;
r = 1 / (1/a + 1/b + 1/c + 1/d);
s = (b - a) / (d - c);
y = a * b * (1/c) * (f^2/2);

disp(['x = ', num2str(x)]);
disp(['r = ', num2str(r)]);
disp(['s = ', num2str(s)]);
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 $a = 3 \times 10^{150}$, $b = 5 \times 10^{200}$.

a. Use MATLAB to calculate $c = ab$.

b. Supposed $d = 5 \times 10^{-200}$ use MATLAB to calculate $f = d/a$.

c. Use MATLAB to calculate the product $x = abd$ two ways.

```
% Check limits
realmax
realmin

a = 3e150;
b = 5e200;

% a. Calculate c = a*b (Expect Overflow)
c = a * b

% b. d = 5e-200, calculate f = d/a (Expect Underflow)
d = 5e-200;
f = d / a

% c. Calculate x = abd in two ways
x1 = a * b * d; % Risk of intermediate overflow
y = b * d;
x2 = a * y;      % Safer calculation

disp(['Method 1: ', num2str(x1)]);
disp(['Method 2: ', num2str(x2)]);
```

Problem 22

Use MATLAB to calculate:

a. $e^{(-2.1)^3} + 3.47\log(14) + \sqrt[4]{287}$

b. $(3.4)^7\log(14) + \sqrt[4]{287}$

c. $\cos^2(\frac{4.12\pi}{6})$

d. $\cos(\frac{4.12\pi}{6})^2$

```
% Note: Source likely implies log base 10 for "log(14)" in standard notation,
% but MATLAB's log() is natural log. Using log10() for base 10.
ans_a = exp((-2.1)^3) + 3.47 * log10(14) + nthroot(287, 4);
ans_b = (3.4)^7 * log10(14) + nthroot(287, 4);
ans_c = cos((4.12 * pi) / 6)^2;
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 $1 \leq t \leq 3$.

```
t = 1:0.01:3;
T = 7 .* log(t) - 8 .* exp(0.3 .* t);

plot(t, T);
title('Temperature vs Time');
xlabel('Time (min)');
ylabel('Temperature (C)');
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 $0 \leq \phi \leq 4\pi$.

```
r = 10;
phi = 0 : 0.01 : 4*pi;
x = r .* (phi - sin(phi));
y = r .* (1 - cos(phi));

plot(x, y);
title('Cycloid Plot (r=10)');
xlabel('x'); ylabel('y');
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 $b_1 = 200$, $b_2 = 180$, $c_1 = 120$, $A_1 = 120°$, $A_2 = 100°$.

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

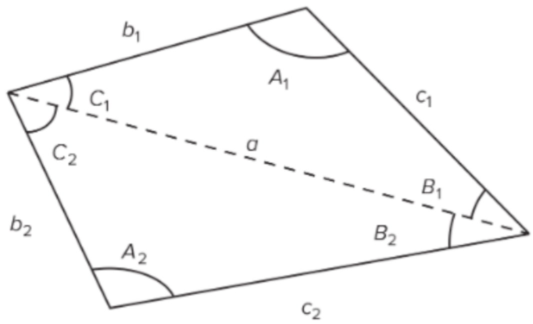

Figure P34

```
% Inputs
b1 = 200; b2 = 180; c1 = 120;
A1_deg = 120; A2_deg = 100;
A1 = deg2rad(A1_deg); A2 = deg2rad(A2_deg);

% 1. Find common side 'a' (Top Triangle Law of Cosines)
a_sq = b1^2 + c1^2 - 2*b1*c1*cos(A1);
a = sqrt(a_sq);

% 2. Find c2 (Bottom Triangle) solving quadratic:
% c2^2 - (2*b2*cos(A2))*c2 + (b2^2 - a^2) = 0
coeff_A = 1;
coeff_B = -2 * b2 * cos(A2);
coeff_C = b2^2 - a_sq;

possible_c2 = roots([coeff_A, coeff_B, coeff_C]);
c2 = possible_c2(possible_c2 > 0); % Filter positive

disp(['Side c2: ', num2str(c2')]);
```

Problem 35

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

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

Tutorial 02: 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.

```
x = 0:10;            % Integers 0 to 10 (step is 1)
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.

```
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:

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

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

4. Matrix Multiplication vs. Array Multiplication

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

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

5. Solving Linear Systems

To solve systems like $Ax = B$:

- 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.

- $P(x) = 2x^2 + 14x + 20 \rightarrow$ `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 $\sqrt{x_1^2 + x_2^2 + ...}$. It is calculated using `norm(x)` or `sqrt(x'*x)`.

Specific Example:

```
x = [2, -4, 5];

L = length(x);      % Result: 3 (elements)
A = abs(x);         % Result: [2, 4, 5] (vector)
M = norm(x);        % Result: 6.7082 (scalar)
% 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):

```
A = [10, 50, 30];
[val, idx] = max(A);
% val = 50
% 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:

```
Z = zeros(3, 4);   % 3x4 matrix of zeros
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:

a. Select just the second row of B.

b. Evaluate the sum of the second row of B.

c. Multiply the second column of B and the first column of A element by element.

d. 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.

e. 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.

```
% Define Matrix A
A = [1, 4, 2;
     2, 4, 100;
     7, 9, 7;
     3, pi, 42];

% Define Matrix B (Natural log is log() in MATLAB)
B = log(A);

% a. Select second row of B
part_a = B(2, :);

% b. Sum of second row of B
part_b = sum(B(2, :));

```

```
% c. Multiply 2nd col of B and 1st col of A element-wise
part_c = B(:, 2) .* A(:, 1);

% d. Max value of result from c
part_d = max(part_c);

% e. Divide 1st row of A by first 3 elements of 3rd col of B
% Note: A(1,:) is 1x3. B(1:3, 3) is 3x1.
% We must transpose B's slice to match dimensions.
vec_e = A(1, :) ./ B(1:3, 3)';
part_e = sum(vec_e);

disp(['Sum (Part b): ', num2str(part_b)]);
disp(['Max (Part d): ', num2str(part_d)]);
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.

```
A = [3, -2, 1; 6, 8, -5; 7, 9, 10];
B = [6, 9, -4; 7, 5, 3; -8, 2, 1];
C = [-7, -5, 2; 10, 6, 1; 3, -9, 8];

% Create 3D array D
D(:, :, 1) = A;
D(:, :, 2) = B;
D(:, :, 3) = C;

% Largest element in each layer
max_layer_1 = max(max(D(:, :, 1)));
max_layer_2 = max(max(D(:, :, 2)));
max_layer_3 = max(max(D(:, :, 3)));

% Largest element in D
max_total = max(D(:));

disp(['Max Total: ', num2str(max_total)]);
```

Problem 15

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

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

```
C = [-3, -9; 7, 8];

% a. A + B + C
res_a = A + B + C;

% b. A - B + C
res_b = A - B + C;

% c. Verify Associative Law: (A+B)+C = A+(B+C)
check_assoc = isequal((A+B)+C, A+(B+C));

% d. Verify Commutative Law: A+B+C = B+C+A = A+C+B
term1 = A + B + C;
term2 = B + C + A;
term3 = A + C + B;
check_comm = isequal(term1, term2) && isequal(term2, term3);

if check_assoc && check_comm
    disp('Laws Verified');
else
    disp('Verification Failed');
end
```

Problem 19

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

```
x = -2 : 0.05 : 16; % Smooth interval
f = (4 .* cos(x)) ./ (x + exp(-0.75 .* x));

plot(x, f);
title('Plot of f(x)');
xlabel('x');
ylabel('f(x)');
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.

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

```

```
% Distance formula d = sqrt(x^2 + y^2)
distances = sqrt(x.^2 + y.^2);

% Find minimum distance
min_dist = min(distances);

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.

```
% Data vectors
Force = [400, 550, 700, 500, 600]; % Newtons
Distance = [3, 0.5, 0.75, 1.5, 5]; % Meters

% a. Work per segment (Element-wise multiplication)
Work_segments = Force .* Distance;

% b. Total work
Work_total = sum(Work_segments);

disp('Work per segment (J):');
disp(Work_segments);
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.

```
% Data
F = [11, 7, 8, 10, 9];          % Force (N)
k = [1000, 600, 900, 1300, 700]; % Constant (N/m)

% a. Compression x = F / k
x = F ./ k;

% b. Potential Energy PE = 0.5 * k * x^2
PE = 0.5 .* k .* (x.^2);

% Display results table
disp(table(F', k', x', PE', 'VariableNames', {'Force','k','Compression','PE'}));
```

Problem 41

Solve the following system using the left-division method.

$$6x - 3y + 4z = 41$$
$$12x + 5y - 7z = -26$$
$$-5x + 2y - 6z = 16$$

```
% Coefficient Matrix A
A = [ 6, -3,  4;
     12,  5, -7;
     -5,  2, -6];

% Constant Vector B
B = [41; -26; 16];

% Solve for X = [x; y; z] using left division
Solution = A \ B;

disp('Solution [x; y; z]:');
disp(Solution);
```

Tutorial 03: 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:

```
function [out1, out2] = my_func_name(in1, in2)
    % Comments explaining the function (H1 line)

    out1 = in1 + in2;    % Perform calculations
    out2 = in1 .* in2;   % Assign values to output variables
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:

```
F = @(x) 3*x.^2 + 2*x + 5;
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.

```
fun = @(x) x.^2 + 4*sin(x);
[x_min, val_min] = fminbnd(fun, -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 $x = 0$.

```
fun = @(x) cos(x) - x;
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 $[1, 1]$.

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

```
A = 5; % Defined in Workspace
% 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 $h = v_0t - \frac{1}{2}gt^2$. 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, $g = 9.81m/s^2$.

```
% --- Main Script ---
h = 100; v0 = 50; g = 9.81;

% Call the function
t_solutions = compute_time(h, v0, g);

disp('Times to reach 100m (seconds):');
disp(t_solutions);
% Interpretation: The object reaches 100m twice.
% Once on the way up, and once on the way down.

% --- Function Definition ---
function t = compute_time(h, v0, g)
    % Solves 0.5*g*t^2 - v0*t + h = 0
    % Using quadratic formula: ax^2 + bx + c = 0
    % a = 0.5*g, b = -v0, c = h

    roots_vec = roots([0.5*g, -v0, h]);
    t = roots_vec;
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 $A = \pi r\sqrt{r^2 + h^2}$.

a. Eliminate h to obtain A as a function of r and V.

b. Create a function for A and use **fminbnd** to find r that minimizes A for $V = 10$ in^3.

```
% --- Main Script ---
global V
V = 10; % Volume constraint

% Minimize Area function between r=0.1 and r=10
[r_min, A_min] = fminbnd(@cone_area, 0.1, 10);

% Calculate corresponding h
h_min = 3 * V / (pi * r_min^2);

disp(['Optimal r: ', num2str(r_min)]);
disp(['Optimal h: ', num2str(h_min)]);
disp(['Minimum Area: ', num2str(A_min)]);

% --- Function Definition ---
function A = cone_area(r)
    global V
    % Eliminate h: h = 3V / (pi*r^2)
    h = 3 * V ./ (pi .* r.^2);
    % Substitute into A
    A = pi .* r .* sqrt(r.^2 + h.^2);
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 $A = \pi^2(b^2 - a^2)$.

a. Create a function for V and A.

b. Plot A vs a for $0.25 \leq a \leq 4$ given $b = a + 2$.

```
% --- Main Script ---
a = 0.25 : 0.01 : 4;
b = a + 2; % Constraint

% Compute A and V using arrays
[V, A] = torus_calc(a, b);

plot(a, A);
title('Torus Surface Area vs Inner Radius a');
xlabel('a (inches)');
ylabel('Surface Area A');
grid on;
```

```
% --- Function Definition ---
function [V, A] = torus_calc(a, b)
    V = 0.25 * pi^2 .* (a + b) .* (b - a).^2;
    A = pi^2 .* (b.^2 - a.^2);
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 $a = 1, b = 2$.

```
% --- Main Script ---
plot_ellipse(1, 2);

% --- Function Definition ---
function plot_ellipse(a, b)
    % Use parametric equations for full ellipse
    t = linspace(0, 2*pi, 100);
    x = a * cos(t);
    y = b * sin(t);

    figure;
    plot(x, y);
    title(['Ellipse: a=', num2str(a), ', b=', num2str(b)]);
    axis equal;
    grid on;
end
```

Problem 25

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

a. Plot to approximate minimum.

b. Use `fminbnd` to determine the precise minimum location.

```
f = @(x) 30*x.^2 - 300*x + 4;

% a. Plotting
x_plot = -5:0.1:15;
plot(x_plot, f(x_plot));
grid on; title('Plot of 30x^2 - 300x + 4');

% b. Finding minimum
[x_min, val_min] = fminbnd(f, 0, 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`.

```
% Data
t = 0:15;
y_data = [13, 16, 20, 25, 31, 39, 45, 49, 55, 63, 69, 77, 82, 86, 89, 92];

% Model Function: y = c / (1 + a*exp(-b*t))
model_fun = @(p, t) p(3) ./ (1 + p(1) * exp(-p(2) * t));

% Error Function (Sum of Squared Errors)
err_fun = @(p) sum((y_data - model_fun(p, t)).^2);

% Initial Guess: c around 100 (max percent), a and b generic guesses
guess = [10, 0.5, 100];

% Optimization
p_opt = fminsearch(err_fun, guess);
a_est = p_opt(1); b_est = p_opt(2); c_est = p_opt(3);

% Plotting results
t_smooth = 0:0.1:15;
y_fit = model_fun(p_opt, t_smooth);

plot(t, y_data, 'ko', t_smooth, y_fit, 'b-');
legend('Data', 'Logistic Fit');
title('Logistic Growth Regression');
disp(['Estimated: a=',num2str(a_est),', b=',num2str(b_est),', c=',num2str(c_est)]);
```

Tutorial 04: 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
`==`	Equal to	`~=`	Not equal to
`<`	Less than	`<=`	Less than or equal to
`>`	Greater than	`>=`	Greater than or equal to

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

```
if x = 5   % Error! Assigns 5 to x inside the condition.
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.

```
if x < 0
    y = -x;
elseif x == 0
    y = 0;
else
    y = x^2;
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.

```
switch units
    case {'inch', 'in'}
        y = x * 2.54;
    case {'meter', 'm'}
        y = x * 100;
    otherwise
        disp('Unknown unit');
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.

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

```
error = 100;
while error > 0.01
    % Update estimate
    % Update error
end
```

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

```
x = 5;
while x > 0
    disp(x);
    % Missing x = x - 1; -> Infinite loop!
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).

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

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

```
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):

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

With Pre-allocation (Fast):

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

```
for k = 1:5
    if k == 2
        continue; % Skips 2, goes to 3
    end
    if k == 4
        break;    % Stops loop completely at 4
    end
    disp(k);      % Displays: 1, 3
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) $4, 24, 100$.

```
% Define test cases
cases = [2, 10, 12;
         3, 24, 48;
         4, 24, 100];

for i = 1:size(cases, 1)
    a = cases(i, 1); b = cases(i, 2); c = cases(i, 3);

    disc = b^2 - 4*a*c;

    if disc > 0
        x1 = (-b + sqrt(disc))/(2*a);
        x2 = (-b - sqrt(disc))/(2*a);
        type = 'Real and distinct';
    elseif disc == 0
        x1 = -b/(2*a);
        x2 = x1;
        type = 'Real and repeated';
    else
        real_part = -b/(2*a);
        imag_part = sqrt(abs(disc))/(2*a);
        x1 = complex(real_part, imag_part);
        x2 = complex(real_part, -imag_part);
        type = 'Complex conjugates';
    end

    disp(['Case ', num2str(i), ': ', type]);
    disp(['Roots: ', num2str(x1), ' and ', num2str(x2)]);
end
```

Problem 9

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

```
price_A = [19, 18, 22, 21, 25, 19, 17, 21, 27, 29];
price_B = [22, 17, 20, 23, 24, 18, 16, 25, 28, 27];

% Logical comparison
```

```
days_below = price_A < price_B;

% Count true values
num_days = sum(days_below);

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

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

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

By-hand check:

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

```
% --- Main Script (Problem 16) ---
xs = [-5, 3, 15];

for k = 1:length(xs)
    x = xs(k);

    if x < -1
        y = exp(x) + 1;
    elseif x ≥ -1 && x < 5
        y = 2 + cos(pi*x);
    else % x ≥ 5
        y = 10*(x - 5) + 1;
    end

    disp(['x = ', num2str(x), ' -> y = ', num2str(y)]);
end
```

Problem 21

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

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

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.

```
% --- Main Script ---
disp(['f(1,1) = ', num2str(fxy(1,1))]);
disp(['f(1,-1) = ', num2str(fxy(1,-1))]);
disp(['f(-1,1) = ', num2str(fxy(-1,1))]);
disp(['f(-1,-1) = ', num2str(fxy(-1,-1))]);

% --- Function Definition ---
function val = fxy(x, y)
    if x ≥ 0 && y ≥ 0
        val = x + y;
    elseif x ≥ 0 && y < 0
        val = x - y;
    elseif x < 0 && y ≥ 0
        val = -x^2 * y;
    else % x < 0 and y < 0
        val = -x^2 * y^2;
    end
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.):

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

```
% --- Main Script (Problem 28) ---
A = [  3   5  -4;
      -8  -1  33;
     -17   6  -9];

%% (a) Using a for loop + conditionals
B1 = A;  % start by copying A
[m,n] = size(A);

for i = 1:m
    for j = 1:n
        if A(i,j) ≥ 1
            B1(i,j) = log(A(i,j)) + 20;  % natural log + 20
        end
    end
end

%% (b) Using a logical mask
B2 = A;                    % start by copying A
mask = (A ≥ 1);           % logical array (true where condition holds)
B2(mask) = log(A(mask)) + 20;

%% Display results
disp('A =');  disp(A);
disp('B1 (loop) ='); disp(B1);
disp('B2 (mask) ='); disp(B2);

% Check they match (should be all zeros)
disp('Max difference between B1 and B2:');
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

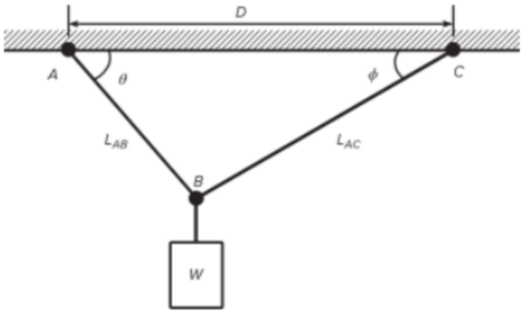

Figure P40

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

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

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

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

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

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 $L_{AC,\min} \le L_{AC} \le 6.7$.

```
% --- Main Script (Problem 40) ---
clear; clc; close all;

D   = 6;          % ft
LAB = 3;          % ft
W   = 2000;       % lb
LAC_max = 6.7;    % ft (given)

% Step size for searching LAC_min
dL = 1e-3;

% Start from just above the triangle lower bound |D-LAB| = 3
LAC = abs(D - LAB) + 1e-4;

TAB = inf;  TAC = inf;

% ---- WHILE LOOP to find LAC_min ----
while (TAB > W) || (TAC > W)
```

```
    % Compute angles (radians)
    theta = acos((D^2 + LAB^2 - LAC^2)/(2*D*LAB));
    phi   = asin((LAB*sin(theta))/LAC);

    % Solve equilibrium for [TAB; TAC]
    A = [-cos(theta),  cos(phi);
          sin(theta),  sin(phi)];
    b = [0; W];

    T = A\b;         % T(1)=TAB, T(2)=TAC
    TAB = T(1);
    TAC = T(2);

    % Increase LAC until both tensions are ≤ W
    if (TAB > W) || (TAC > W)
        LAC = LAC + dL;
    end

    % Safety stop (should not happen for this problem)
    if LAC > LAC_max
        error('No feasible LAC found up to LAC_max.');
    end
end

LAC_min = LAC;
fprintf('LAC_min = %.4f ft\n', LAC_min);
fprintf('TAB = %.2f lb, TAC = %.2f lb\n', TAB, TAC);

% ---- Compute tensions for plotting from LAC_min to 6.7 ----
Lvec = LAC_min:dL:LAC_max;
TABv = zeros(size(Lvec));
TACv = zeros(size(Lvec));

for k = 1:length(Lvec)
    LAC = Lvec(k);

    theta = acos((D^2 + LAB^2 - LAC^2)/(2*D*LAB));
    phi   = asin((LAB*sin(theta))/LAC);

    A = [-cos(theta),  cos(phi);
          sin(theta),  sin(phi)];
    b = [0; W];

    T = A\b;
    TABv(k) = T(1);
    TACv(k) = T(2);
end

% ---- Plot ----
figure;
plot(Lvec, TABv, 'LineWidth', 1.5); hold on;
plot(Lvec, TACv, 'LineWidth', 1.5);
yline(W, '--', 'LineWidth', 1.2);  % limit line at 2000 lb
grid on;
```

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

Problem 42

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

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

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

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

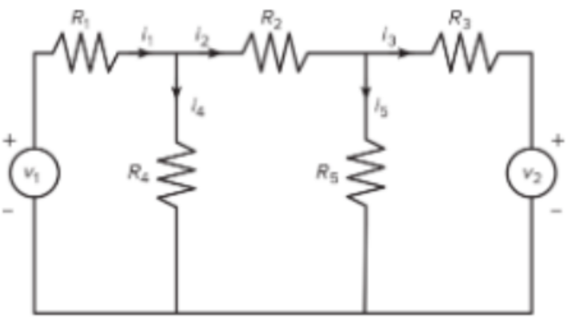

Figure P42

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.):

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

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

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

```
% --- Main Script (Problem 42) ---
clear; clc; close all;

% Given values (kOhm -> Ohm)
R1 = 5e3;    R2 = 100e3;  R3 = 200e3;  R4 = 150e3;  R5 = 250e3;
v1 = 100;              % V
Imax = 1e-3;            % A

% Helper function: compute allowable v2 interval for a given R3
% Returns [v2min, v2max] for v2 > 0, based on |currents| ≤ Imax
allowable_v2 = @(R3val) local_allowable_v2(R1,R2,R3val,R4,R5,v1,Imax);

%% (a) Allowable range for the given R3
[v2min, v2max] = allowable_v2(R3);
fprintf('(a) Allowable v2 range: %.4f V ≤ v2 ≤ %.4f V\n', v2min, v2max);

%% (b) Vary R3 from 150 to 250 kOhm, plot the upper allowable limit v2max
R3vec = linspace(150e3, 250e3, 200);
v2max_vec = zeros(size(R3vec));

for k = 1:length(R3vec)
    [¬, v2max_vec(k)] = allowable_v2(R3vec(k));
end

figure;
plot(R3vec/1e3, v2max_vec, 'LineWidth', 1.5);
grid on;
xlabel('R_3 (k\Omega)');
ylabel('Upper allowable v_2 (V)');
title('Allowable upper limit of v_2 vs. R_3');

%% -------- Local function (kept at end of script) --------
function [v2min, v2max] = local_allowable_v2(R1,R2,R3,R4,R5,v1,Imax)
    % Unknowns: i1,i2,i3,i4,i5
    % Build linear system A*i = b, where b depends on v2.
    %
    % Equations:
    % -v1 + R1 i1 + R4 i4 = 0
    % -R4 i4 + R2 i2 + R5 i5 = 0
    % -R5 i5 + R3 i3 + v2 = 0
    % i1 - i2 - i4 = 0
    % i2 - i3 - i5 = 0

    A = [ R1,    0,    0,  R4,    0;
           0,   R2,    0, -R4,   R5;
           0,    0,   R3,    0, -R5;
           1,   -1,    0,   -1,    0;
           0,    1,   -1,    0,   -1];

    % Solve i(v2) = a + b*v2 by two solves: v2=0 and v2=1
    b0 = [v1; 0; 0; 0; 0];      % v2 = 0 -> third equation RHS is 0
    b1 = [v1; 0; -1; 0; 0];     % v2 = 1 -> third equation becomes R3*i3 - R5*i5 = -1

```

```
    i0 = A\b0;                  % currents when v2 = 0
    i1 = A\b1;                  % currents when v2 = 1
    slope = i1 - i0;            % di/dv2
    offset = i0;                % i(v2)=offset + slope*v2

    % For each current: |offset + slope*v2| ≤ Imax gives an interval in v2
    v_low = -inf;
    v_high = inf;

    for k = 1:5
        a = offset(k);
        m = slope(k);

        if abs(m) < 1e-15
            % current independent of v2
            if abs(a) > Imax
                v_low = 1; v_high = 0; % empty interval
                break;
            end
        else
            % Solve -Imax ≤ a + m*v2 ≤ Imax
            v1k = (-Imax - a)/m;
            v2k = ( Imax - a)/m;
            lo = min(v1k, v2k);
            hi = max(v1k, v2k);

            v_low = max(v_low, lo);
            v_high = min(v_high, hi);
        end
    end

    % Also require v2 > 0
    v_low = max(v_low, 0);

    v2min = v_low;
    v2max = v_high;
end
```

Problem 44

We are given the MATLAB script:

```
k = 1; b = -2; x = -1; y = -2;
while k ≤ 3
    k, b, x, y
    y = x^2 - 3;
    if y < b
        b = y;
    end
    x = x + 1;
```

```
    k = k + 1;
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):

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

Pass	k	b	x	y
First				
Second				
Third				
Fourth				
Fifth				

Figure 3: Figure for Problem 44.

Tutorial 09: 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:** $\int_0^\pi \sin(x)dx$

```
fun = @(x) sin(x);
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)`

```
x = 0:0.1:pi;
y = sin(x);
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:** $\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x}$

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

Pitfall: Plotting the derivative against the original x vector.

```
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:

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

```
% Parameters: m=1, c=2, k=5, F=0
ode_sys = @(t, x) [x(2); (1/1)*(0 - 2*x(2) - 5*x(1))];
[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`.

```
function [value, isterminal, direction] = my_event(t, y)
    value = y(1);      % Detect when y(1) (height) = 0
    isterminal = 1;    % 1 = Stop integration
    direction = -1;    % -1 = Only detect falling (neg slope)
end

% Usage
opts = odeset('Events', @my_event);
[t, y] = ode45(fun, [0, 100], [10; 0], opts);
```

5. Summary of Functions

Function	Purpose
`integral(fun, a, b)`	Numerical integration of a formula
`trapz(x, y)`	Numerical integration of data arrays
`diff(x)`	Difference between adjacent elements
`gradient(M)`	Numerical gradient of a matrix
`ode45`	Standard ODE solver (Runge-Kutta)
`odeset`	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: $5\ddot{y} + 7\dot{y} + 5y = 5\dot{f} + f(t)$.

```
% Create the transfer function model form named sys
sys = tf([5, 1], [5, 7, 5]);

% Plot the unit step response for zero initial conditions
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):

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

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 = Position (x)
- Let x_2 = 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:

```
% Define parameters
m = 2; c = 5; k = 3;

% Define Matrices
A = [0, 1; -k/m, -c/m];
B = [0; 1/m];
C = [1, 0];
D = 0;

% Create the LTI state-space object
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:

```
function xdot = msd(t, x)
    % Define parameters and a sample constant input force
    m = 2; c = 5; k = 3; u = 10;

    % Define matrices for clean calculation
    A = [0, 1; -k/m, -c/m];
    B = [0; 1/m];

    % Return column vector of derivatives
    xdot = A*x + B*u;
end

% Call ode45 in the main script
% For 0 ≤ t ≤ 5, with initial conditions x1(0)=0, x2(0)=0
[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)`, `impulse(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.
- `impulse(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 $v(0) = 0$.

```
% v(t) = integral of a(t) from 0 to 20
a_fun = @(t) 5 .* t .* sin(8 .* t);
v_20 = integral(a_fun, 0, 20);

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 ($t = 40$). $T = 48000, m_0 = 2200, r = 0.8, g = 9.81$.

```
T = 48000; m0 = 2200; r = 0.8; g = 9.81; b = 40;

% ODE: dv/dt = T/m(t) - g
% m(t) = m0 * (1 - r*t/b)
dvdt = @(t, v) (T ./ (m0 * (1 - r*t/b))) - g;

[t_sol, v_sol] = ode45(dvdt, [0, b], 0);

disp(['Velocity at burnout: ', num2str(v_sol(end)), ' m/s']);
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 $x = 0.6$.

```
% Define x with fine resolution around 0.6
dx = 0.001;
x = 0 : dx : 1;
y = exp(-2*x) .* sin(4*x) ./ (x.^2 + 3);

% Calculate approximate derivative dy/dx
% diff(y) is difference between adjacent elements
% dividing by dx gives the slope
dydx = diff(y) ./ dx;

% Find index corresponding to x = 0.6
% Note: diff result is 1 element shorter than x
x_diff = x(1:end-1);
[~, idx] = min(abs(x_diff - 0.6));

deriv_val = dydx(idx);

disp(['Approximate derivative at x=0.6: ', num2str(deriv_val)]);

% Analytical check (optional, for verification)
```

```
% y' via Chain/Quotient Rule
x0 = 0.6;
% ... (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.

```
r_tank = 3;
r_drain = 0.02;
A_drain = pi * r_drain^2;
Cd = 0.5; g = 9.81;

% ODE: dh/dt = - (Cd * A * sqrt(2gh)) / (pi * (2rh - h^2))
dhdt = @(t, h) -(Cd * A_drain * sqrt(2*g*h)) ./ (pi * (2*r_tank*h - h.^2));

% Integrate until h is near 0 (event function typically used, or guess time)
% Using ode45 with events to stop at h=0
options = odeset('Events', @stop_event);
[t, h] = ode45(dhdt, [0, 50000], 5, options);

disp(['Time to empty: ', num2str(t(end)/3600), ' hours']);
plot(t, h); title('Tank Draining');

% Event function definition
function [value, isterminal, direction] = stop_event(t, h)
    value = h - 0.01; % Stop when height is 1cm
    isterminal = 1;
    direction = 0;
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 $t \geq 0$.

a. Plot $y(t)$ for $y(0) = \dot{y}(0) = 0$

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

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

```

```
t_span = [0, 5];
f_val = 10;

% a. Zero Initial Conditions
IC_a = [0; 0];
ode_a = @(t, x) [x(2); (1/3)*(f_val - 18*x(2) - 102*x(1))];
[t_a, y_a] = ode45(ode_a, t_span, IC_a);

% b. Non-zero Initial Velocity (y(0)=0, y_dot(0)=10)
IC_b = [0; 10];
ode_b = @(t, x) [x(2); (1/3)*(f_val - 18*x(2) - 102*x(1))];
[t_b, y_b] = ode45(ode_b, t_span, IC_b);

% Plotting
figure;
plot(t_a, y_a(:,1), 'b-', 'LineWidth', 1.5); hold on;
plot(t_b, y_b(:,1), 'r--', 'LineWidth', 1.5);
legend('Case A: Zero ICs', 'Case B: Init Vel = 10');
title('Response of Mass-Spring-Damper');
xlabel('Time (s)'); ylabel('Displacement y(t)');
grid on;
```

Problem 44

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

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

a. Use the `initial` function to plot position x_1 if $x(0) = [5; 3]$.

b. Use the `step` function to plot the response for step input of magnitude 10.

```
% Define Matrices
A = [0, 1; -5, -2];
B = [0; 1];
C = [1, 0]; % Output y = x1 (Position)
D = 0;

% Create State Space System (requires Control System Toolbox)
sys = ss(A, B, C, D);

% a. Initial Response (Free response to initial conditions)
x0 = [5; 3];
figure;
subplot(2,1,1);
initial(sys, x0);
title('a. Response to Initial Conditions x0=[5; 3]');
grid on;

```

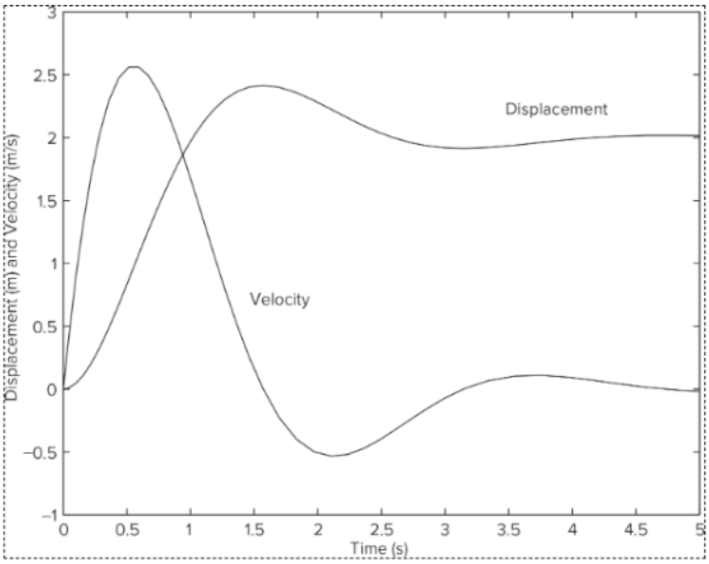

Figure 9.5-1 Displacement and velocity of the mass as a function of time.

```
% b. Step Response (Step magnitude 10)
% The standard step(sys) assumes input is 1.
% For magnitude 10, we scale the system or the input.
sys_scaled = sys * 10; % Scale input channel by 10

subplot(2,1,2);
step(sys_scaled);
title('b. Step Response (Input Magnitude 10)');
grid on;
```

Problem 45

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

a. Free response: $y(0) = 10, \dot{y}(0) = -5$.

b. Step response: Zero ICs, unit step input.

c. Total response superposition.

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

```

% a. Free Response (f=0)
ode_free = @(t, x) [x(2); -0.4*x(2) - 2*x(1)];
[t_free, x_free] = ode45(ode_free, [0, 15], [10; -5]);

% b. Step Response (f=1, IC=0)
ode_step = @(t, x) [x(2); 0.2*1 - 0.4*x(2) - 2*x(1)];
[t_step, x_step] = ode45(ode_step, [0, 15], [0; 0]);

% c. Total Response (f=1, IC=[10, -5])
ode_total = @(t, x) [x(2); 0.2*1 - 0.4*x(2) - 2*x(1)];
[t_tot, x_tot] = ode45(ode_total, [0, 15], [10; -5]);

% Plotting
figure;
plot(t_free, x_free(:,1), '--', 'DisplayName', 'Free'); hold on;
plot(t_step, x_step(:,1), ':', 'DisplayName', 'Step');
plot(t_tot, x_tot(:,1), 'k-', 'LineWidth', 1.5, 'DisplayName', 'Total');
legend; title('Superposition of Responses');
```

Tutorial 10: Simulink

Key Concepts and Common Pitfalls (Tutorial 10 Summary)

1. Simulink vs. MATLAB ODE Solvers

Simulink provides a block-diagram environment for simulating dynamic systems. The same problems can be solved programmatically in MATLAB using `ode45`. Key equivalences:

- **Integrator block** $\Leftrightarrow$ state variable (integrate $\dot{x}$ to get x)
- **Gain block** $\Leftrightarrow$ scalar multiplication in the ODE function
- **Sum block** $\Leftrightarrow$ addition inside the ODE function
- **Fcn/Math block** $\Leftrightarrow$ nonlinear expression (e.g., `sin(x)`, `x^3`)

2. Converting to State-Variable Form

Every nth-order ODE must be reduced to n first-order equations before using `ode45`:

1. Solve for the highest derivative.
2. Let $x_1 = y,\ x_2 = \dot{y},\ \ldots,\ x_n = y^{(n-1)}$.
3. Write $\dot{x}_1 = x_2,\ \dot{x}_2 = x_3,\ \ldots,\ \dot{x}_n =$ (expression from step 1).

3. Transfer Function Approach (Linear Systems)

For linear systems with zero initial conditions, use the Control System Toolbox:

- `tf(num, den)`: create a transfer function from coefficient vectors
- `lsim(sys, u, t)`: simulate response to an arbitrary input vector
- `series(sys1, sys2)`: cascade two transfer functions

4. Nonlinear Systems

Nonlinear elements (e.g., $\sin x$, x^3, saturation) require `ode45` because the Control System Toolbox only supports LTI systems. Use `min`/`max` to implement Saturation blocks.

Tutorial Problems

Problem 3

Draw a simulation diagram for the equation:

$$3\dot{y} + 5 \sin y = f(t)$$

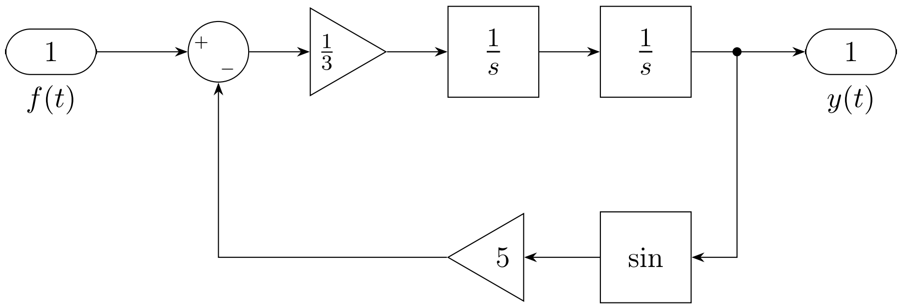

Figure 4: Simulink Block Diagram

Problem 5

Draw a simulation diagram for the model:

$$\dot{x} = -3x + 2y + f(t),$$
$$\dot{y} = 4x - 5y$$

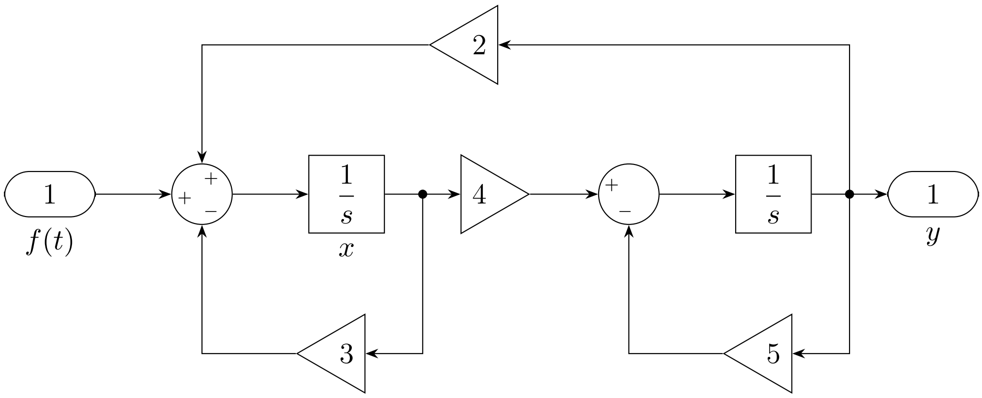

Figure 5: Simulink Block Diagram

Problem 8

Create a Simulink model to plot the solution for $0 \le t \le 6$:

$$10\ddot{y} = 7\sin 4t + 5\cos 3t, \qquad y(0) = 3,\ \dot{y}(0) = 2$$

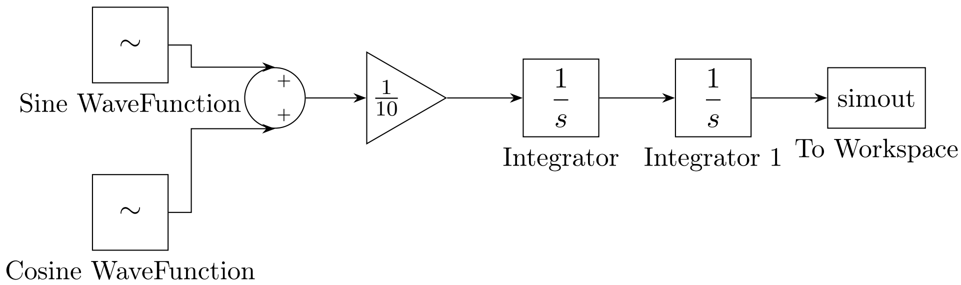

Figure 6: Simulink Block Diagram

Problem 10

The equation $\dot{x} + x = \tan t$, $x(0) = 0$ has no analytical solution. The approximate solution (less accurate for large t) is:

$$x(t) = \tfrac{1}{3}t^3 - t^2 + 3t - 3 + 3\,e^{-t}$$

Compare the numerical and approximate solutions.

```
% Numerical: x_dot = tan(t) - x
% Note: tan(t) diverges near t = pi/2 ≠ 1.57; limit range
```

```
ode = @(t, x) tan(t) - x;
[t, x_num] = ode45(ode, [0, 1.4], 0);

% Approximate solution
x_approx = (1/3)*t.^3 - t.^2 + 3*t - 3 + 3*exp(-t);

plot(t, x_num, 'b-', t, x_approx, 'r--', 'LineWidth', 1.5);
legend('Numerical (ode45)', 'Approximate');
title('Comparison: \dot{x} + x = tan t');
xlabel('t'); ylabel('x(t)');
grid on;
```

Problem 13

Construct a Simulink model to plot solutions for $0 \leq t \leq 2$:

$$\dot{x}_1 = -6x_1 + 4x_2,$$
$$\dot{x}_2 = 5x_1 - 7x_2 + f(t)$$

where $f(t) = 3t$. Use the Ramp block in the Sources library.

```
% State: s(1) = x1, s(2) = x2
% f(t) = 3t (ramp with slope 3)
f = @(t) 3*t;
ode = @(t, s) [-6*s(1) + 4*s(2);
               5*s(1) - 7*s(2) + f(t)];
[t, sol] = ode45(ode, [0, 2], [0; 0]);

plot(t, sol(:,1), '-', t, sol(:,2), '--', 'LineWidth', 1.5);
legend('x_1(t)', 'x_2(t)');
title('State Response with Ramp Input f(t) = 3t');
xlabel('t'); grid on;
```

Problem 15

Construct a Simulink model to plot solutions for $0 \leq t \leq 10$:

$$\dot{x} = -5x + 3y + 5\sin 2t, \quad x(0) = 0$$

$$\dot{y} = 3x - 4y, \quad y(0) = 0$$

```
% State: s(1) = x, s(2) = y
ode = @(t, s) [-5*s(1) + 3*s(2) + 5*sin(2*t);
               3*s(1) - 4*s(2)];
[t, sol] = ode45(ode, [0, 10], [0; 0]);
```

```
plot(t, sol(:,1), '-', t, sol(:,2), '--', 'LineWidth', 1.5);
legend('x(t)', 'y(t)');
title('State Response with Sinusoidal Forcing');
xlabel('t'); grid on;
```

Problem 18

Construct a Simulink model for $5\dot{x} + \sin x = f(t)$, $x(0) = 0$, where $g(t) = 10 \sin 4t$ and:

$$f(t) = \begin{cases} -5 & \text{if } g(t) \leq -5 \\ g(t) & \text{if } -5 < g(t) < 5 \\ 5 & \text{if } g(t) \geq 5 \end{cases}$$

```
% Saturation: clip g(t) to [-5, 5]
g = @(t) 10 * sin(4*t);
f = @(t) min(5, max(-5, g(t)));

% ODE: x_dot = (f(t) - sin(x)) / 5
ode = @(t, x) (f(t) - sin(x)) / 5;
[t, x] = ode45(ode, [0, 5], 0);

figure;
subplot(2,1,1);
plot(t, f(t)); title('Forcing f(t) [Saturated]');
xlabel('t'); ylabel('f(t)'); grid on;
subplot(2,1,2);
plot(t, x); title('Response x(t)');
xlabel('t'); ylabel('x(t)'); grid on;
```

Problem 25

Use Transfer Function blocks to plot solutions for $0 \leq t \leq 2$:

$$3\ddot{x} + 15\dot{x} + 18x = f(t), \quad x(0) = \dot{x}(0) = 0$$

$$2\ddot{y} + 16\dot{y} + 50y = x(t), \quad y(0) = \dot{y}(0) = 0$$

where $f(t) = 75\, u_s(t)$.

```
% X(s)/F(s) = 1 / (3s^2 + 15s + 18)
% Y(s)/X(s) = 1 / (2s^2 + 16s + 50)
sys_x = tf(1, [3, 15, 18]);
sys_y = tf(1, [2, 16, 50]);

```

```
% Step input: f(t) = 75 * us(t)
t = 0:0.005:2;
f_input = 75 * ones(size(t));

x_out = lsim(sys_x, f_input, t);
y_out = lsim(series(sys_x, sys_y), f_input, t);

figure;
subplot(2,1,1);
plot(t, x_out, 'LineWidth', 1.5);
title('x(t)'); xlabel('t'); grid on;
subplot(2,1,2);
plot(t, y_out, 'LineWidth', 1.5);
title('y(t)'); xlabel('t'); grid on;
```

Problem 28

Create a Simulink model to plot the solution for $0 \leq t \leq 1$:

$$\frac{Y(s)}{F(s)} = \frac{4}{s+5}, \qquad f(t) = u_s(t) - u_s(t-1)$$

```
sys = tf(4, [1, 5]);

% Pulse input: f(t) = us(t) - us(t-1)
% Over [0,1]: f(t) = 1 for t < 1, drops to 0 at t = 1
t = 0:0.001:1;
f_input = double(t < 1);

y_out = lsim(sys, f_input, t);

plot(t, y_out, 'b-', t, f_input, 'r--', 'LineWidth', 1.5);
legend('y(t)', 'f(t)');
title('Response: Y(s)/F(s) = 4/(s+5), Pulse Input');
xlabel('t'); grid on;
```

Problem 33

Create a Simulink model for a mass supported by a nonlinear hardening spring, $0 \leq t \leq 2$:

$$5\ddot{y} = 5g - (900y + 1700y^3), \qquad y(0) = 0.5, \ \dot{y}(0) = 0$$

Use $g = 9.81$ m/s^2.

```
% State: x(1) = y, x(2) = y_dot
% y_ddot = g - (900*y + 1700*y^3) / 5
g = 9.81;
ode = @(t, x) [x(2);
               g - (900*x(1) + 1700*x(1)^3) / 5];
[t, x] = ode45(ode, [0, 2], [0.5; 0]);

plot(t, x(:,1), 'LineWidth', 1.5);
title('Nonlinear Hardening Spring Response');
xlabel('t (s)'); ylabel('y (m)');
grid on;
```

Problem 35

The equation for water height h in a spherical tank (radius $r = 3$ m) with drain (radius 2 cm, $C_d = 0.5$), $h(0) = 5$ m:

$$\pi(2rh - h^2)\frac{dh}{dt} = -C_d A\sqrt{2gh}$$

Use Simulink to solve the nonlinear equation and plot $h(t)$ until $h(t) = 0$.

```
r = 3; Cd = 0.5; g = 9.81;
A = pi * 0.02^2;  % drain area, radius = 0.02 m

% ODE: dh/dt = -Cd*A*sqrt(2gh) / (pi*(2r*h - h^2))
dhdt = @(t, h) -Cd * A * sqrt(2*g*h) / (pi * (2*r*h - h^2));

opts = odeset('Events', @stop_event, 'RelTol', 1e-6);
[t, h] = ode45(dhdt, [0, 80000], 5, opts);

plot(t/3600, h, 'LineWidth', 1.5);
title('Spherical Tank Draining');
xlabel('Time (hours)'); ylabel('Water Height h (m)');
grid on;
fprintf('Tank empties at t = %.2f hours\n', t(end)/3600);

function [value, isterminal, direction] = stop_event(¬, h)
    value = h - 0.001;    % Stop when h < 1 mm
    isterminal = 1;
    direction = -1;
end
```

Problem 45

Consider the system in Figure P45 with $m_1 = m_2 = 1$, $c_1 = 3$, $c_2 = 1$, $k_1 = 1$, $k_2 = 4$:

$$m_1\ddot{x}_1 + (c_1 + c_2)\dot{x}_1 + (k_1 + k_2)x_1 - c_2\dot{x}_2 - k_2 x_2 = 0$$

$$m_2\ddot{x}_2 + c_2\dot{x}_2 + k_2x_2 - c_2\dot{x}_1 - k_2x_1 = f(t)$$

a. Develop a Simulink model using state-variable representation.

b. Plot $x_1(t)$ for zero initial conditions with piecewise input:

$$f(t) = \begin{cases} t & 0 \leq t \leq 1 \\ 2 - t & 1 < t < 2 \\ 0 & t \geq 2 \end{cases}$$

```
% Parameters
m1=1; m2=1; c1=3; c2=1; k1=1; k2=4;

% Piecewise input (triangular pulse)
f = @(t) (t ≥ 0 & t ≤ 1).*t + (t > 1 & t < 2).*(2 - t);

% State: s = [x1; x1_dot; x2; x2_dot]
% x1_ddot = [-(c1+c2)*x1_dot - (k1+k2)*x1 + c2*x2_dot + k2*x2] / m1
% x2_ddot = [f(t) + c2*x1_dot + k2*x1 - c2*x2_dot - k2*x2] / m2
ode = @(t, s) [
    s(2);
    (-(c1+c2)*s(2) - (k1+k2)*s(1) + c2*s(4) + k2*s(3)) / m1;
    s(4);
    (f(t) + c2*s(2) + k2*s(1) - c2*s(4) - k2*s(3)) / m2
];

[t, sol] = ode45(ode, [0, 10], zeros(4, 1));

plot(t, sol(:,1), 'LineWidth', 1.5);
title('x_1(t) - Two Mass-Spring-Damper System');
xlabel('t (s)'); ylabel('x_1 (m)');
grid on;
```