GENG8030-MATLAB
Winter 2026

University of Windsor
Bob Little

Course Notes **February 13, 2026**

Table of Contents

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:

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

```matlab
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 $f = 19.83$.

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

```matlab
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 $b_1 = 200$, $b_2 = 180$, $c_1 = 120$, $A_1 = 120°$, $A_2 = 100°$.

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

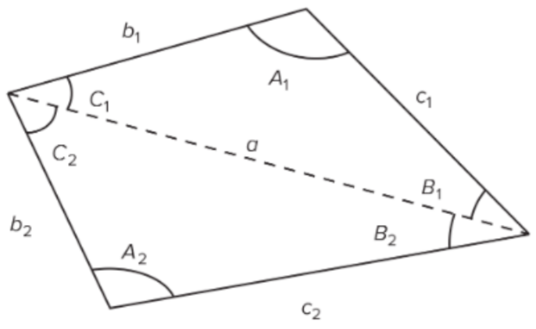

Figure P34

```
% 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 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.

```
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 $\sqrt{x_1^2 + x_2^2 + \dots}$. It is calculated using `norm(x)` or `sqrt(x'*x)`.

Specific Example:

```matlab
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.

```matlab
% 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];
```

```matlab
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 $-2 \leq x \leq 16$.

```matlab
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.

```matlab
% 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.

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

```matlab
% 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 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:

```
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 $h = v_0 t - \frac{1}{2}gt^2$. 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, $g = 9.81 m/s^2$.

```matlab
% --- 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.

```matlab
% --- 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 $0.25 \le a \le 4$ given $b = a + 2$.

```matlab
% --- 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 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
`==`	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) $4, 24, 100$.

```matlab
% 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.

```matlab
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

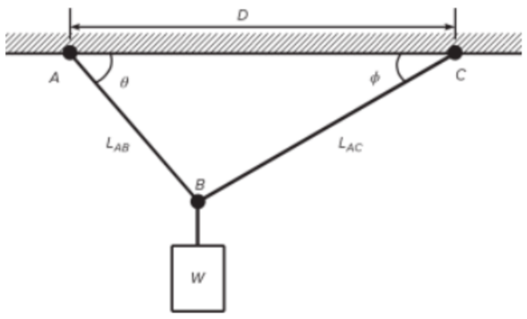

Figure P40

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

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

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

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

```matlab
% --- 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)
```

```matlab
    % 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');
```

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

```matlab
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)`

```matlab
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:

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

```matlab
% 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`.

```matlab
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);
```

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

5. Summary of Functions

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.

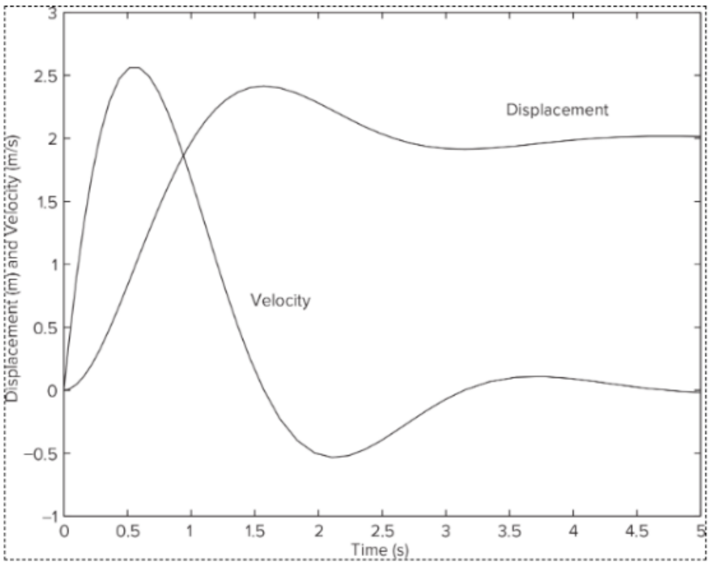

Figure 9.5-1 Displacement and velocity of the mass as a function of time.

```
% 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;

% 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');
```