MATLAB Essentials for Mechanical Engineering

Table of Contents

Basic Operations and Variables

Variable Assignment and Basic Operations

```
% Variable assignment
length = 5.5;           % Length in meters
width = 3.2;            % Width in meters
height = 2.1;           % Height in meters

% Basic arithmetic operations
area = length * width;                 % Area calculation
volume = length * width * height;      % Volume calculation
perimeter = 2 * (length + width);      % Perimeter calculation

% Display results
fprintf('Area: %.2f m²\n', area);
fprintf('Volume: %.2f m³\n', volume);
fprintf('Perimeter: %.2f m\n', perimeter);
```

Constants and Built-in Functions

```matlab
% Important constants
g = 9.81;              % Acceleration due to gravity (m/s²)
pi_val = pi;           % Pi constant
e_val = exp(1);        % Euler's number

% Common mathematical functions
angle = 45;            % Angle in degrees
angle_rad = deg2rad(angle);     % Convert to radians
sin_val = sin(angle_rad);       % Sine function
cos_val = cos(angle_rad);       % Cosine function
sqrt_val = sqrt(25);            % Square root
log_val = log(10);              % Natural logarithm
log10_val = log10(100);         % Base-10 logarithm
```

Vectors and Matrices

Vector Operations

```matlab
% Creating vectors
force_x = [100, 150, 200, 250, 300];    % Force in x-direction (N)
force_y = [50, 75, 100, 125, 150];      % Force in y-direction (N)
time = 0:0.1:5;                         % Time vector (0 to 5 seconds)

% Vector operations
resultant_force = sqrt(force_x.^2 + force_y.^2);  % Resultant force magnitude
mean_force = mean(resultant_force);                % Average force
max_force = max(resultant_force);                  % Maximum force
min_force = min(resultant_force);                  % Minimum force

% Vector indexing
first_force = force_x(1);               % First element
last_force = force_x(end);              % Last element
middle_forces = force_x(2:4);           % Elements 2 to 4
```

Matrix Operations

```
% Creating matrices
stress_tensor = [100, 20, 0;           % Stress tensor (MPa)
                 20, 80, 0;
                 0, 0, 60];

strain_tensor = [0.001, 0.0002, 0;     % Strain tensor
                 0.0002, 0.0008, 0;
                 0, 0, 0.0006];

% Matrix operations
det_stress = det(stress_tensor);        % Determinant
inv_stress = inv(stress_tensor);        % Inverse matrix
trace_stress = trace(stress_tensor);    % Trace (sum of diagonal elements)
eigenvalues = eig(stress_tensor);       % Eigenvalues
```

Mathematical Operations

Solving Linear Systems

```
% Solving system of linear equations: Ax = b
% Example: Truss analysis
A = [1, 0, -0.707;      % Coefficient matrix
     0, 1, -0.707;
     0, 0, 1];

b = [1000; 0; 1414];    % Load vector (N)

% Solve for unknown forces
x = A\b;                % Matrix left division (most efficient)
x_inv = inv(A)*b;       % Using matrix inverse (less efficient)

fprintf('Forces: F1 = %.2f N, F2 = %.2f N, F3 = %.2f N\n', x(1), x(2), x(3));
```

Numerical Integration and Differentiation

```
% Numerical integration (trapezoidal rule)
x = 0:0.1:10;
y = sin(x);
area_trap = trapz(x, y);

% Numerical differentiation
dx = diff(x);
dy = diff(y);
derivative = dy./dx;

% Definite integral using quad function
fun = @(x) x.^2 + 2*x + 1;
integral_result = integral(fun, 0, 5);
```

Control Structures

Conditional Statements

```
% Material property selection based on temperature
temperature = 450;  % Temperature in Celsius

if temperature < 200
    material = 'Aluminum';
    yield_strength = 276; % MPa
elseif temperature < 500
    material = 'Steel';
    yield_strength = 250; % MPa
else
    material = 'Titanium';
    yield_strength = 880; % MPa
end

fprintf('At %.0f°C, use %s with yield strength %.0f MPa\n', ...
        temperature, material, yield_strength);
```

Loops

```
% For loop: Calculate stress for different loads
loads = [1000, 2000, 3000, 4000, 5000];  % Loads in N
area = 0.01;  % Cross-sectional area in m²
stresses = zeros(size(loads));

for i = 1:length(loads)
    stresses(i) = loads(i) / area;  % Stress = Force/Area
    fprintf('Load: %d N, Stress: %.2f Pa\n', loads(i), stresses(i));
end

% While loop: Iterate until convergence
tolerance = 1e-6;
x = 1;
iteration = 0;
while abs(x^2 - 2) > tolerance
    x = (x + 2/x) / 2;  % Newton's method for sqrt(2)
    iteration = iteration + 1;
end
fprintf('Square root of 2 ≈ %.6f (found in %d iterations)\n', x, iteration);
```

Functions

Creating Custom Functions

```
% Function file: beam_deflection.m
function [deflection, max_deflection] = beam_deflection(L, E, I, w)
    % Calculate deflection of simply supported beam with uniform load
    % L: Length (m), E: Young's modulus (Pa), I: Moment of inertia (m^4)
    % w: Uniform load (N/m)

    x = linspace(0, L, 100);
    deflection = (w * x ./ (24 * E * I)) .* (L^3 - 2*L*x.^2 + x.^3);
    max_deflection = max(abs(deflection));
end

% Usage example
L = 5;          % Length: 5 m
E = 200e9;      % Young's modulus: 200 GPa
I = 1e-4;       % Moment of inertia: 1e-4 m^4
w = 10000;      % Uniform load: 10 kN/m

[deflection, max_def] = beam_deflection(L, E, I, w);
fprintf('Maximum deflection: %.6f m\n', max_def);
```

Anonymous Functions

```
% Anonymous functions for quick calculations
stress_fn = @(F, A) F ./ A;                     % Stress function
strain_fn = @(stress, E) stress ./ E;           % Strain function
reynolds_fn = @(rho, v, D, mu) rho * v * D / mu; % Reynolds number

% Usage
force = 5000;     % N
area = 0.02;      % m²
E_steel = 200e9;  % Pa

stress = stress_fn(force, area);
strain = strain_fn(stress, E_steel);
```

Data Visualization

Basic Plotting

```
% Stress-strain curve
strain = 0:0.0001:0.003;
stress = 200e9 * strain;  % Linear elastic region

% Create plot
figure;
plot(strain, stress/1e6, 'b-', 'LineWidth', 2);
xlabel('Strain (m/m)');
ylabel('Stress (MPa)');
title('Stress-Strain Curve for Steel');
grid on;
legend('Linear Elastic Region', 'Location', 'southeast');
```

Multiple Plots and Subplots

```
% Vibration analysis
t = 0:0.01:2;
omega1 = 2*pi*5;    % 5 Hz
omega2 = 2*pi*10;   % 10 Hz
x1 = cos(omega1*t);
x2 = 0.5*cos(omega2*t);
x_total = x1 + x2;

figure;
subplot(3,1,1);
plot(t, x1, 'r-');
title('First Mode (5 Hz)');
ylabel('Displacement');

subplot(3,1,2);
plot(t, x2, 'g-');
title('Second Mode (10 Hz)');
ylabel('Displacement');

subplot(3,1,3);
plot(t, x_total, 'b-');
title('Combined Response');
xlabel('Time (s)');
ylabel('Displacement');
```

3D Plotting

```
% 3D surface plot for heat distribution
[X, Y] = meshgrid(-5:0.5:5, -5:0.5:5);
Z = exp(-(X.^2 + Y.^2)/4);

figure;
surf(X, Y, Z);
xlabel('X Position');
ylabel('Y Position');
zlabel('Temperature');
title('Heat Distribution');
colorbar;
shading interp;
```

File I/O Operations

Reading and Writing Data

```
% Writing data to file
data = [1:10; (1:10).^2; (1:10).^3]';  % Time, position, velocity data
filename = 'test_data.txt';

% Write to file
dlmwrite(filename, data, 'delimiter', '\t');

% Read from file
imported_data = dlmread(filename, '\t');

% Using save and load for MATLAB variables
save('workspace_data.mat', 'data', 'filename');
load('workspace_data.mat');
```

Excel File Operations

```
% Create sample data
time = (0:0.1:5)';
displacement = sin(2*pi*time);
velocity = 2*pi*cos(2*pi*time);
acceleration = -4*pi^2*sin(2*pi*time);

% Combine data
data_table = [time, displacement, velocity, acceleration];
headers = {'Time (s)', 'Displacement (m)', 'Velocity (m/s)', 'Acceleration (m/s²)'};

% Write to Excel
xlswrite('vibration_data.xlsx', headers, 'Sheet1', 'A1');
xlswrite('vibration_data.xlsx', data_table, 'Sheet1', 'A2');

% Read from Excel
[num_data, txt_data] = xlsread('vibration_data.xlsx');
```

Mechanical Engineering Applications

Thermodynamics

```matlab
% Ideal gas law calculations
function [P, V, T] = ideal_gas_law(n, R, P, V, T)
    % Calculate missing parameter using PV = nRT
    % Input NaN for unknown parameter

    if isnan(P)
        P = n * R * T / V;
    elseif isnan(V)
        V = n * R * T / P;
    elseif isnan(T)
        T = P * V / (n * R);
    end
end

% Example usage
n = 2;          % moles
R = 8.314;      % J/(mol·K)
P = NaN;        % Unknown pressure
V = 0.5;        % m³
T = 300;        % K

[P, V, T] = ideal_gas_law(n, R, P, V, T);
fprintf('Pressure: %.2f Pa\n', P);
```

Fluid Mechanics

```
% Bernoulli's equation for pipe flow
function [v2, P2] = bernoulli_pipe(rho, v1, P1, z1, z2, h_loss)
    % Calculate velocity and pressure at point 2
    % rho: density, v1: velocity at point 1, P1: pressure at point 1
    % z1, z2: elevations, h_loss: head loss

    g = 9.81;

    % Assuming v2 = v1 for constant area pipe
    v2 = v1;

    % Apply Bernoulli's equation with losses
    P2 = P1 + 0.5*rho*v1^2 + rho*g*z1 - 0.5*rho*v2^2 - rho*g*z2 - rho*g*h_loss;
end

% Example
rho = 1000;     % kg/m³ (water)
v1 = 2;         % m/s
P1 = 200000;    % Pa
z1 = 10;        % m
z2 = 5;         % m
h_loss = 0.5;   % m

[v2, P2] = bernoulli_pipe(rho, v1, P1, z1, z2, h_loss);
fprintf('Velocity at point 2: %.2f m/s\n', v2);
fprintf('Pressure at point 2: %.2f Pa\n', P2);
```

Heat Transfer

```
% 1D Heat conduction (finite difference method)
function T = heat_conduction_1d(L, k, T_left, T_right, q_gen, n)
    % Solve 1D heat conduction equation
    % L: length, k: thermal conductivity, T_left/T_right: boundary temps
    % q_gen: heat generation, n: number of nodes

    dx = L / (n - 1);

    % Create coefficient matrix
    A = zeros(n, n);
    b = zeros(n, 1);

    % Boundary conditions
    A(1, 1) = 1;
    b(1) = T_left;
    A(n, n) = 1;
    b(n) = T_right;

    % Interior nodes
    for i = 2:n-1
        A(i, i-1) = 1;
        A(i, i) = -2;
        A(i, i+1) = 1;
        b(i) = -q_gen * dx^2 / k;
    end

    % Solve system
    T = A\b;
end

% Example usage
L = 1;          % Length: 1 m
k = 200;        % Thermal conductivity: 200 W/(m·K)
T_left = 100;   % Left boundary temperature: 100°C
T_right = 50;   % Right boundary temperature: 50°C
q_gen = 1000;   % Heat generation: 1000 W/m³
n = 11;         % Number of nodes

T = heat_conduction_1d(L, k, T_left, T_right, q_gen, n);
x = linspace(0, L, n);

figure;
plot(x, T, 'bo-', 'LineWidth', 2);
xlabel('Position (m)');
```

```
ylabel('Temperature (°C)');
title('1D Heat Conduction');
grid on;
```

Vibrations and Dynamics

```
% Single degree of freedom vibration system
function [t, x] = sdof_vibration(m, c, k, x0, v0, F0, omega, t_final)
    % Solve equation: m*x'' + c*x' + k*x = F0*cos(omega*t)
    % Initial conditions: x(0) = x0, x'(0) = v0

    % Natural frequency and damping ratio
    omega_n = sqrt(k/m);
    zeta = c / (2*sqrt(k*m));

    % Time vector
    t = linspace(0, t_final, 1000);

    % Check if system is underdamped, critically damped, or overdamped
    if zeta < 1
        % Underdamped
        omega_d = omega_n * sqrt(1 - zeta^2);

        % Homogeneous solution
        A = x0;
        B = (v0 + zeta*omega_n*x0) / omega_d;
        x_h = exp(-zeta*omega_n*t) .* (A*cos(omega_d*t) + B*sin(omega_d*t));

        % Particular solution (for harmonic forcing)
        if F0 ~= 0
            H = 1 / sqrt((k - m*omega^2)^2 + (c*omega)^2);
            phi = atan2(c*omega, k - m*omega^2);
            x_p = (F0/m) * H * cos(omega*t - phi);
        else
            x_p = 0;
        end

        x = x_h + x_p;
    else
        % For simplicity, only implementing underdamped case
        x = zeros(size(t));
    end
end

% Example: Vibrating system
m = 10;         % Mass: 10 kg
c = 50;         % Damping: 50 N·s/m
k = 1000;       % Stiffness: 1000 N/m
x0 = 0.1;       % Initial displacement: 0.1 m
v0 = 0;         % Initial velocity: 0 m/s
```

```
F0 = 100;        % Forcing amplitude: 100 N
omega = 8;       % Forcing frequency: 8 rad/s
t_final = 10;    % Duration: 10 s

[t, x] = sdof_vibration(m, c, k, x0, v0, F0, omega, t_final);

figure;
plot(t, x, 'b-', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Displacement (m)');
title('Single DOF Vibration Response');
grid on;
```

Symbolic Math

Symbolic Calculations

```
% Symbolic variables
syms x y z t F L E I w real positive

% Symbolic expressions
beam_deflection_sym = (w*x/(24*E*I)) * (L^3 - 2*L*x^2 + x^3);

% Differentiation
slope = diff(beam_deflection_sym, x);
moment = diff(slope, x) * E * I;

% Integration
total_deflection = int(beam_deflection_sym, x, 0, L);

% Solve equations
syms omega_n m k
eq1 = omega_n^2 == k/m;
omega_solution = solve(eq1, omega_n);

% Simplification
simplified_expr = simplify(beam_deflection_sym);
```

Symbolic Plotting

```
% Plot symbolic functions
syms x
f = x^3 - 6*x^2 + 11*x - 6;
df = diff(f, x);

figure;
fplot(f, [-1, 5], 'b-', 'LineWidth', 2);
hold on;
fplot(df, [-1, 5], 'r--', 'LineWidth', 2);
xlabel('x');
ylabel('f(x)');
legend('f(x) = x³ - 6x² + 11x - 6', 'f''(x)', 'Location', 'best');
title('Function and its Derivative');
grid on;
```

Advanced Topics

Optimization

```matlab
% Minimize weight of a beam subject to constraints
function [x_opt, fval] = optimize_beam_design()
    % Design variables: [width, height] of rectangular beam
    % Objective: minimize weight (proportional to area)
    % Constraints: maximum stress, maximum deflection

    % Objective function (area to minimize)
    objective = @(x) x(1) * x(2);  % width * height

    % Constraints
    function [c, ceq] = constraints(x)
        w = x(1);  % width
        h = x(2);  % height

        % Given parameters
        L = 3;      % Length: 3 m
        P = 10000;  % Load: 10 kN
        E = 200e9;  % Young's modulus: 200 GPa
        sigma_max = 250e6;  % Maximum stress: 250 MPa
        delta_max = 0.01;   % Maximum deflection: 1 cm

        % Calculated values
        I = w * h^3 / 12;                   % Moment of inertia
        sigma = P * L * h / (2 * I);        % Maximum stress
        delta = P * L^3 / (3 * E * I);      % Maximum deflection

        % Inequality constraints (must be <= 0)
        c = [sigma - sigma_max;             % Stress constraint
             delta - delta_max];            % Deflection constraint

        % Equality constraints
        ceq = [];
    end

    % Initial guess
    x0 = [0.1, 0.2];  % Initial width and height

    % Bounds
    lb = [0.05, 0.05];  % Lower bounds
    ub = [0.5, 0.5];    % Upper bounds

    % Optimization
    options = optimoptions('fmincon', 'Display', 'iter');
    [x_opt, fval] = fmincon(objective, x0, [], [], [], [], lb, ub, @constraints, options);
```

```
    fprintf('Optimal dimensions: width = %.3f m, height = %.3f m\n', x_opt(1), x_opt(2));
    fprintf('Minimum cross-sectional area: %.6f m²\n', fval);
end

% Run optimization
[x_opt, fval] = optimize_beam_design();
```

Numerical Methods

```
% Newton-Raphson method for nonlinear equations
function [root, iterations] = newton_raphson(func, dfunc, x0, tol, max_iter)
    % Find root of function using Newton-Raphson method
    % func: function handle, dfunc: derivative function handle
    % x0: initial guess, tol: tolerance, max_iter: maximum iterations

    x = x0;
    for i = 1:max_iter
        fx = func(x);
        dfx = dfunc(x);

        if abs(dfx) < eps
            error('Derivative is zero. Cannot continue.');
        end

        x_new = x - fx / dfx;

        if abs(x_new - x) < tol
            root = x_new;
            iterations = i;
            return;
        end

        x = x_new;
    end

    root = x;
    iterations = max_iter;
    warning('Maximum iterations reached. May not have converged.');
end

% Example: Find root of x^3 - 2x - 5 = 0
func = @(x) x^3 - 2*x - 5;
dfunc = @(x) 3*x^2 - 2;
x0 = 2;
tol = 1e-10;
max_iter = 100;

[root, iters] = newton_raphson(func, dfunc, x0, tol, max_iter);
fprintf('Root found: x = %.10f (in %d iterations)\n', root, iters);
```

Signal Processing

```
% FFT analysis of vibration signal
function analyze_vibration_signal(signal, fs)
    % Analyze vibration signal using FFT
    % signal: time domain signal, fs: sampling frequency

    N = length(signal);
    t = (0:N-1) / fs;

    % Compute FFT
    Y = fft(signal);
    f = (0:N-1) * fs / N;

    % Single-sided spectrum
    P = abs(Y/N);
    P = P(1:N/2+1);
    P(2:end-1) = 2*P(2:end-1);
    f = f(1:N/2+1);

    % Plot results
    figure;
    subplot(2,1,1);
    plot(t, signal);
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Time Domain Signal');
    grid on;

    subplot(2,1,2);
    plot(f, P);
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('Frequency Domain (FFT)');
    grid on;

    % Find dominant frequencies
    [peaks, locs] = findpeaks(P, 'MinPeakHeight', max(P)*0.1);
    dominant_freqs = f(locs);

    fprintf('Dominant frequencies: ');
    for i = 1:length(dominant_freqs)
        fprintf('%.2f Hz ', dominant_freqs(i));
    end
    fprintf('\n');
end
```

```
% Example usage
fs = 1000;              % Sampling frequency: 1000 Hz
t = 0:1/fs:1-1/fs;      % Time vector
signal = sin(2*pi*50*t) + 0.5*sin(2*pi*150*t) + 0.1*randn(size(t));

analyze_vibration_signal(signal, fs);
```

Tips and Best Practices

Code Organization

```
% 1. Use clear variable names
young_modulus = 200e9;    % Good
E = 200e9;                % Acceptable for well-known symbols
x = 200e9;                % Poor

% 2. Add comments for complex calculations
% Calculate maximum stress in beam using bending formula
sigma_max = (M * c) / I;  % M: moment, c: distance to neutral axis, I: moment of inertia

% 3. Use functions for repeated calculations
function stress = calculate_stress(force, area)
    % Calculate normal stress
    stress = force / area;
end

% 4. Vectorize operations when possible
forces = [100, 200, 300, 400, 500];
areas = [0.01, 0.02, 0.03, 0.04, 0.05];
stresses = forces ./ areas;  % Vectorized operation
```

Error Handling

```
function result = safe_division(numerator, denominator)
    % Safe division with error checking
    if denominator == 0
        error('Division by zero is not allowed');
    end

    if ~isnumeric(numerator) || ~isnumeric(denominator)
        error('Both inputs must be numeric');
    end

    result = numerator / denominator;
end

% Using try-catch for error handling
try
    result = safe_division(10, 0);
catch ME
    fprintf('Error: %s\n', ME.message);
end
```

Performance Optimization

```
% 1. Pre-allocate arrays
n = 1000;
data = zeros(n, 1);  % Pre-allocate
for i = 1:n
    data(i) = i^2;
end

% 2. Use vectorized operations
x = 1:1000;
y = x.^2;  % Vectorized (faster than loop)

% 3. Use appropriate data types
int_data = int32(1:1000);    % Use integer when appropriate
double_data = double(1:1000); % Use double for calculations
```

Common MATLAB Commands for Mechanical Engineers

Quick Reference

```
% Mathematical operations
sin(), cos(), tan()           % Trigonometric functions
asin(), acos(), atan()        % Inverse trigonometric functions
exp(), log(), log10()         % Exponential and logarithmic functions
sqrt(), abs(), sign()         % Square root, absolute value, sign
round(), ceil(), floor()      % Rounding functions

% Matrix operations
det()                         % Determinant
inv()                         % Matrix inverse
eig()                         % Eigenvalues and eigenvectors
trace()                       % Trace (sum of diagonal elements)
rank()                        % Matrix rank
transpose() or '              % Matrix transpose

% Statistics
mean(), median(), mode()      % Central tendency measures
std(), var()                  % Standard deviation and variance
min(), max()                  % Minimum and maximum values
sum(), prod()                 % Sum and product of elements

% Plotting
plot(), plot3()               % 2D and 3D line plots
scatter(), scatter3()         % 2D and 3D scatter plots
bar(), histogram()            % Bar charts and histograms
surf(), mesh()                % 3D surface plots
contour(), contourf()         % Contour plots
xlabel(), ylabel(), zlabel()  % Axis labels
title(), legend()             % Title and legend
grid(), axis()                % Grid and axis control

% File operations
load(), save()                % Load and save MATLAB variables
xlsread(), xlswrite()         % Excel file operations
csvread(), csvwrite()         % CSV file operations
importdata()                  % Import various data formats

% Control flow
if, elseif, else, end         % Conditional statements
for, while, end               % Loop statements
break, continue               % Loop control
try, catch, end               % Error handling
```

This comprehensive guide covers the essential MATLAB concepts and applications specifically relevant to mechanical engineering. Each section includes practical examples that can be directly used or modified for your specific engineering problems.