```
clear all % Clears all variables and data from memory.
clc
% a1=1:1:5
% a2=1:5
% 1:2:5
% 5:-1:1
% a3=linspace(1,6,5)
% 1:1:5 % Generates [1, 2, 3, 4, 5].
% 1:5   % Same as above (default increment is 1).
% 1:2:5 % Generates [1, 3, 5] (increment of 2).
% 5:-1:1  % Generates [5, 4, 3, 2, 1] (decrement).
% a3=linspace(1,6,5)  % 5 equally spaced points between 1 and 6.
% a4=linspace(1,5,20) % 20 equally spaced points between 1 and 5.
% Array Operations
% a = [1 2 3 4 5] % Row vector
% b = [1; 2; 3; 4; 5] % Column vector
% c = [15 25 35 45 55] % Row vector
% a + b   % Adds corresponding elements of a and b.  (broadcasting)
% a + c   % Adds corresponding elements of a and c.
% e = c'  % Transpose of c (row to column vector).
%
% %Matrix Operations
%
% z = [1 3 4; 4 9 6; 7 8 8] % 3x3 matrix
% y = [2 4 5; 7 5 4; 7 8 7] % 3x3 matrix
% z + y                     % Element-wise addition.
%
% max(z) % Maximum of each column.
% max(max(z)) % Overall maximum of the matrix.
% %
% ac = sort(z, "ascend")   % Sort each column in ascending order.
% des = sort(z, "descend")   % Sort each column in descending order.
% W = [4 5 6 7 8 6 7 8 4 4 5 3 5 4 2 3 6 7]
% D = unique(W) % Returns unique elements in W, sorted in ascending order.
%Complex Numbers
%
% y = [2+5i 3+4i; 4+2i 6+9i] % 2x2 matrix with complex numbers.
% R = [6+6i 7+8i; 7+6i 8+3i] % Another 2x2 matrix with complex numbers.
% y + R % Adds corresponding complex numbers.
%
% clear all;
% clc; % Clear workspace and command window.
%
% % VECTOR AND MATRIX OPERATIONS
%
% a = eye(5)     % Identity matrix of size 5x5.
% b = zeros(3) % 3x3 zero matrix.
% ones(3) % 3x3 matrix of ones.
% 5*ones(7) % 7x7 matrix with all elements as 5.
```

```
% 7*eye(5) % 5x5 identity matrix multiplied by 7.
% diag([1 4 2 5]) % Diagonal matrix with specified diagonal elements.
% rand(3) % 3x3 matrix with random elements between 0 and 1.
%
% A = [14 2 4; 24 4 5; 6 44 6; 4 56 76] % Sample matrix.
% A(1,1) % Element at row 1, column 2.  note: Here index starts from 1.
% A(2,:) % Second row.
% A(:,3) % Third column.
% A(2:4,2:3) % Submatrix from rows 2 to 4 and columns 2 to 3.
% D = [3;4;5;7]
% C = [A, D] % Add a column right to matrix A.
% C = [D,A] % Add a column left to matrix A.
% E = [1,3,4]
% F = [A; E] % Add a row bottom to matrix A.
% [m, n] = size(A) % Get size of matrix A.
% % DETERMINANTS, INVERSES, AND MATRIX POWER
% P = [1 5 6; 5 7 8; 8 6 3]
% det(P) % Determinant of P.
% inv(P) % Inverse of P.
% Q = [3 5 6; 3 7 8; 6 7 8]
% P/Q % P * inv(Q).
% P\Q % inv(P) * Q (left division).
% eig(Q) % Eigenvalues of Q.
% [V, D] = eig(Q) % Eigenvectors (V) and eigenvalues (D) of Q.
% Q^2   % Q*Q
% Q^0.5 % Power of Q (square and square root).
% Q.*Q % Element-wise multiplication.
% % LU AND QR DECOMPOSITIONS
%
%
% A = [1 2 3; 5 3 2; 6 3 2]
% [L,U]=lu(A) % LU decomposition.
% [Q, R] = qr(A) % QR decomposition.
%
%
% AA = diag([1 2 3 4])
% eig(AA) % Eigenvalues of diagonal matrix AA.
% DD = 7*ones(4) % 4x4 matrix of 7s.
% [V, D] = eig(DD) % Eigenvalues and eigenvectors of DD.
% % CONDITIONAL STATEMENTS
% a = input('Enter first value \n')
% b = input('Enter second value \n') % User input.
% if(a > b)
%     fprintf('a is largest \n')
% elseif(b > a)
%     fprintf('b is largest \n')
% else
%     fprintf('Both are equal \n')
% end
```

```
% % NESTED CONDITIONAL STATEMENTS
%
% x = input('Enter the value of x = ') % Input for conditional operations.
% if(x < 0)
%     f = 3*x
% elseif(x > 3)
%     f = sin(x) + x
% else
%     f = sqrt(44 + x)
% end
%
% fprintf("f = %f\n", f)
% % FOR LOOP
% n = input("\n Enter the value of n = ") % Sum of first n numbers.
% sum = 0;
% for i = 1:n
%     sum = sum + i;
% end
% fprintf("Sum of first %d numbers is %f\n", n, sum)
%
%
% % Sum of first m odd numbers.
% m = input("\n Enter the value of m = ")
% sum = 0;
% for i = 1:2:m
%     sum = sum + i
% end
% fprintf("Sum of first %d odd numbers is %f\n", m, sum)
%%%%%% Polynomials in MATLAB %%%%%%
% Polynomials are represented as vectors of their coefficients.
% Example: 2x^3 + 5x + 4 is represented as [2 0 5 4].
%  Key Polynomial Operations:
% conv(a, b) %Multiplies two polynomials.
% [q, r] = deconv(n, d)  % Division of two polynomials.
% roots(p)               % Finds the roots of polynomial p.
% poly(r)                % Forms a polynomial from roots r.
% polyval(a, x)          % Evaluates the polynomial at x.
% polyder(p)             % Derivative of a polynomial.
% polyint(p)             % Integration of a polynomial (constant = 0).
%
% p = [1 -4 4] % Example polynomial (x-2)^2
% polyval(p, 2) % Evaluate p at x = 2
% r = roots(p) % Roots of polynomial p
% p1 = poly(r) % Polynomial from roots
% p2 = [2 0 1 -3]
% q2 = [1 1]
% m = conv(p2, q2) % Polynomial multiplication
% n = deconv(p2, q2) % Polynomial division
% polyder(p) % Derivative of p
```

```
% polyint(p) % Integration of p
%%%%%% section 4: 2D Plotting %%%%%%
% clear all;
% clc;
%
% % Basic 2D Plot
% %
%
% %x1=-2:1:2 %  (from: step: to)
% x = -2 * pi:0.1:2 * pi
% y1 = sin(x)
% plot(x, y1)
% xlim([min(x) max(x)])
% ylim([min(y1) max(y1)])
% title('y1 = sin(x)')
% xlabel('x-axis')
% ylabel('y-axis')
%
% %% Multiple Plots
%
%
% x = -2 * pi:0.1:2 * pi
% y2 = cos(x);
% plot(x, y1, 'r*', x, y2, 'b--')   % r* for red starred line
%                                     % and b-- for blue dashed line
% legend('y1 = sin(x)', 'y2 = cos(x)')
%
% % Styled Plots
%
% x = -2 * pi:0.1:2 * pi
% y3 = sin(1 + x)
% plot(x, y1, 'r--', x, y2, 'b.', x, y3, 'g*')
%%%%%% Section 5: Subplots and Parametric Plots %%%%%%
%
% % clear all;
% % clc;
% x = linspace(0, 10)
% y1 = sin(x)
% y2 = cos(x)
%
% % Tiled Layout
% tiledlayout(2, 2)
%
% ax1 = nexttile
% plot(ax1, x, y1)
%
% ax2 = nexttile
% plot(ax2, x, y2)
%
```

```
% ax3 = nexttile
% plot(ax3, x, y2)
%
% ax4 = nexttile
% plot(ax4, x, y2)
%
%% Subplot Example
% subplot(2, 2, 1) % 2x2 grid, 1st position
% y3 = sin(3 * x)
% plot(x, y3)
%
% subplot(2, 2, 2) % 2x2 grid, 2nd position
% y4 = sin(4 * x)
% plot(x, y4)
%
% subplot(2, 2, 3) % 2x2 grid, 3rd position
% y5 = sin(5 * x)
% plot(x, y5)
%
% subplot(2, 2, 4) % 2x2 grid, 4th position
% y6 = sin(6 * x)
% plot(x, y6)
%% Parametric 2D Plot
%
% t = 0: 0.1: 2*pi
% x = cos(3 * t)
% y = sin(2 * t)
% plot(x, y, 'r')
% grid on
%
% %%  Parametric 3D Plot
%
% z = t .^ 2
% plot3(x, y, z, 'r')
% grid on
% xlabel('x')
% ylabel('y')
% zlabel('z')
%%%%%% Section 6: Solving Systems of Linear Equations (SOLE) %%%%%%
% Method 1: Matrix Method
% clear all;
% clc;
A = [2 1 1; -1 1 -1; 1 2 3]
B = [2; 3; -10]
X = inv(A) * B % Using inverse
X = A \ B % Using left division
% Method 2: Using linsolve
syms x y z
eqn1 = 2 * x + y + z == 2
```

```
eqn2 = -x + y - z == 3
eqn3 = x + 2 * y + 3 * z == -10
[A, B] = equationsToMatrix([eqn1, eqn2, eqn3], [x y z])
X = linsolve(A, B)
% Method 3: Using solve
% sol = solve([eqn1, eqn2, eqn3], [x y z])
% xsol = sol.x % Component-wise solutions
% ysol = sol.y
% zsol = sol.z
```