جَامعة
المَنارة
MANARA UNIVERSITY

البرمجة الاجرائية

تمارين على الأشعة والمصفوفات

Dr. Eng. Essa Alghannam

1. Defining and Manipulating Vectors

%--- Defining Vectors---

```
v_row = [1, 2, 3, 4]; % Row Vector (default)
disp('Row Vector:'); disp(v_row);
v_col = [5; 6; 7; 8];% Column Vector
disp('Column Vector:'); disp(v_col);
```

% Accessing elements (1-based indexing)

```
first_element = v_row(1);
third_element_col = v_col(3);
disp(['First element of row vector: ', num2str(first_element)]);
disp(['Third element of column vector: ', num2str(third_element_col)]);
```

% Slicing vectors

```
sub_vector = v_row(2:3); % Elements from index 2 to 3
disp('Sub-vector (v_row(2:3)):');
disp(sub_vector);
```

%--- Basic Vector Operations---

% Addition/Subtraction (must be same size and orientation)

```
if isequal(size(v_row), size([1, 1, 1, 1])) % Check if sizes are identical
    v_sum = v_row + [1, 1, 1, 1];
```

```
    disp('v_row + [1, 1, 1, 1]:');
    disp(v_sum);
else
    disp('Cannot add/subtract vectors of different sizes or orientations.');
end
% Scalar Multiplication
scalar = 2;
v_scaled = v_row * scalar;
disp(['v_row * ', num2str(scalar), ':']);
disp(v_scaled);

% Element-wise Multiplication (Hadamard product) - requires same size
v_elem_mult = v_row .* [1, 0, 1, 0];
disp('v_row .* [1, 0, 1, 0] (Element-wise):');
disp(v_elem_mult);
% Dot Product (Scalar Product)
dot_product_result = dot(v_row, [1, 1, 1, 1]);
disp(['Dot product of v_row and [1, 1, 1, 1]: ', num2str(dot_product_result)]);
% Alternatively:
dot_product_result_alt = v_row * [1; 1; 1; 1]; %(v_row is row, so it needs column vector)
```

```
>> dot([1 1],[ 1 1])
ans =
     2
>> dot([1 1],[ -1 -1])
ans =
    -2
```

```
>> dot([1 1],[1 0])
ans =
     1
```

- **Dot Product:** Multiply corresponding elements and then sum them up. The result is a scalar (a single number).
- **Element-wise Multiplication:** Multiply corresponding elements without summing. The result is a vector.

```
% Cross Product (Only for 3D vectors)
v_3d_a = [1, 2, 3];
v_3d_b = [4, 5, 6];
```

```
>> cross([1 0 0],[0 1 0])
ans =
     0     0     1
>> cross([1 0 0],[1 0 0])
ans =
     0     0     0
```

```
cross_product_result = cross(v_3d_a, v_3d_b);
disp('Cross product of [1, 2, 3] and [4, 5, 6]:');
disp(cross_product_result);
% --- Vector Properties ---
% Magnitude (Euclidean Norm)
magnitude_v_row = norm(v_row);
magnitude_v_3d = norm(v_3d_a);
disp(['Magnitude of v_row: ', num2str(magnitude_v_row)]);
disp(['Magnitude of v_3d_a: ', num2str(magnitude_v_3d)]);
```

```
>> norm([1 1 1])
ans =
   1.7321
>> sqrt(3)
ans =
   1.7321
```

```
% Normalization (creating a unit vector)
```

1. Magnitude of 1: If you were to measure the length of a unit vector from its tail to its head, it would always be 1 unit long, regardless of the coordinate system or scale.
2. Indicates Direction Only: Because its magnitude is fixed at 1, a unit vector's primary purpose is to specify a direction in space. It tells you "which way" something is pointing without giving any information about its "how much" or "how strong."
3. Notation: Unit vectors are often denoted with a "hat" symbol above them, for example, $\hat{\mathbf{v}}$ (pronounced "v-hat").

```
unit_vector_v_row = v_row / norm(v_row);
disp('Normalized v_row (unit vector):');
disp(unit_vector_v_row);
disp(['Magnitude of normalized v_row: ', num2str(norm(unit_vector_v_row))]); % Should be close to 1
% Angle between two vectors (in radians)
angle_rad = atan2(norm(cross(v_3d_a, v_3d_b)), dot(v_3d_a, v_3d_b));
disp(['Angle between v_3d_a and v_3d_b (rad): ', num2str(angle_rad)]);
% Convert to degrees
angle_deg = rad2deg(angle_rad);
disp(['Angle between v_3d_a and v_3d_b (deg): ', num2str(angle_deg)]);
```

2. Vector Applications in Engineering

a) Representing 2D/3D Points and Displacements

Vectors are fundamental for representing positions and movements in space.

```
% --- Representing Points and Displacements ---
% Define points in 3D space
P1 = [1, 2, 3]; % Position vector of point 1
P2 = [5, 8, 1]; % Position vector of point 2
disp('Point 1:'); disp(P1);
disp('Point 2:'); disp(P2);
% Calculate the displacement vector from P1 to P2
displacement_vector = P2 - P1;
disp('Displacement vector from P1 to P2:');  disp(displacement_vector);
% Calculate the distance between P1 and P2
distance = norm(displacement_vector);
disp(['Distance between P1 and P2: ', num2str(distance)]);
% Represent a velocity vector
velocity_vector = [10; -5; 0]; % m/s (column vector)
disp('Velocity vector:');
disp(velocity_vector);
% Calculate the new position after a time 't'
t = 2; % seconds
new_position = P1' + velocity_vector * t; % P1' makes it a column vector to add velocity_vector
disp(['New position after ', num2str(t), ' seconds:']);
disp(new_position'); % Transpose to display as a row
```

b) Forces and Mechanics

Vectors are used to represent forces, their magnitudes, and directions.

% --- Forces and Mechanics ---

```
% Define two forces acting on a point
F1 = [50, 0, 0];     % Force 1: 50N along the positive x-axis
F2 = [30 * cosd(60), 30 * sind(60), 0]; % Force 2: 30N at 60 degrees to x-axis
disp('Force 1 (N):');   disp(F1);
disp('Force 2 (N):');   disp(F2);
% Calculate the resultant force (vector sum)
F_resultant = F1 + F2;
disp('Resultant Force (N):');   disp(F_resultant);
% Calculate the magnitude and direction of the resultant force
magnitude_F_resultant = norm(F_resultant);
angle_F_resultant_rad = atan2(F_resultant(2), F_resultant(1));
angle_F_resultant_deg = rad2deg(angle_F_resultant_rad);

disp(['Magnitude of resultant force: ', num2str(magnitude_F_resultant), ' N']);
disp(['Direction of resultant force: ', num2str(angle_F_resultant_deg), ' degrees from positive x-axis']);

% Example: Torque calculation (r x F)
% Position vector of point where force is applied
r_vector = [0.5, 0, 0]; % 0.5m along x-axis
force_applied = [0, 100, 0]; % 100N along y-axis
torque = cross(r_vector, force_applied);
disp('Torque vector (Nm):');
disp(torque);
```

c) Electrical Engineering- Phasors

In AC circuits, complex numbers (which can be treated as 2D vectors) are used to represent phasors (voltage, current, impedance).

```
% Define voltage and impedance as complex numbers
V_rms = 120; % RMS Voltage
frequency = 60; % Hz
R = 10; % Resistance (Ohms)
L = 0.05; % Inductance (Henry)
C = 100e-6; % Capacitance (Farad)
% Convert RMS voltage to peak voltage for phasor representation
V_peak = V_rms * sqrt(2);
% Calculate angular frequency
omega = 2 * pi * frequency;
% Calculate inductive and capacitive reactances
XL = omega * L;
XC = 1 / (omega * C);
% Impedance of resistor, inductor, and capacitor
Z_R = R;
Z_L = 1j * XL; % j is the imaginary unit in MATLAB
Z_C = -1j * XC;
% Total impedance for a series RLC circuit
Z_total = Z_R + Z_L + Z_C;
% Voltage phasor (assuming phase of 0 for simplicity)
V_phasor = V_peak * exp(1j * 0); % V_peak * (cos(0) + j*sin(0))
% Current phasor (I = V / Z)
I_phasor = V_phasor / Z_total;
disp('--- AC Circuit Analysis (Phasors) ---');
```

```
disp(['Total Impedance (R+jX): ', num2str(real(Z_total)), ' + j', num2str(imag(Z_total)), ' Ohms']);
disp(['Magnitude of Total Impedance: ', num2str(abs(Z_total)), ' Ohms']);
disp(['Phase Angle of Total Impedance: ', num2str(rad2deg(angle(Z_total))), ' degrees']);

disp(['Voltage Phasor (Peak): ', num2str(real(V_phasor)), ' + j', num2str(imag(V_phasor))]);
disp(['Current Phasor (Peak): ', num2str(real(I_phasor)), ' + j', num2str(imag(I_phasor))]);
disp(['Magnitude of Current (Peak): ', num2str(abs(I_phasor)), ' A']);
disp(['Phase Angle of Current: ', num2str(rad2deg(angle(I_phasor))), ' degrees']);

% Convert peak current to RMS current
I_rms = abs(I_phasor) / sqrt(2);
disp(['Current (RMS): ', num2str(I_rms), ' A']);
```

d) Kinematics and Transformations (e.g., for Robotics)

Homogeneous transformation matrices use vectors (translation) and matrices (rotation) to represent the pose of an object.

```
% --- Kinematics and Transformations (Homogeneous Transformations) ---

% Represent a translation vector
translation_vector = [2; 3; 1]; % meters

% Represent a rotation matrix (e.g., rotation around Z-axis by 90 degrees)
theta = deg2rad(90);
rotation_matrix_Z = [ cos(theta), -sin(theta), 0;
                      sin(theta),  cos(theta), 0;
                      0,           0,          1 ];

disp('Translation Vector (m):');
disp(translation_vector);
disp('Rotation Matrix (Z-axis, 90 deg):');
disp(rotation_matrix_Z);

% Create a 4x4 homogeneous transformation matrix
% T = [ R  t ]
%     [ 0  1 ]
T = [ rotation_matrix_Z, translation_vector;
      0, 0, 0, 1 ];

disp('Homogeneous Transformation Matrix:');
```

```
disp(T);


% Apply transformation to a point
point_before = [1; 1; 1; 1]; % Homogeneous coordinate for a 3D point [1,1,1]


point_after = T * point_before;


disp('Original Point (homogeneous):');
disp(point_before);
disp('Transformed Point (homogeneous):');
disp(point_after);
disp('Transformed Point (Cartesian):');
disp(point_after(1:3)'); % Extracting the Cartesian coordinates
```