



## LINEAR ALGEBRA

### LAB REPORT

(23MAT117)

Amrita School of Engineering

Amrita Vishwa Vidyapeetham, Amaravati Campus

**Name:** G. Balaji Ajay

**Section:** CSE-B

**Roll No:** AV.SC.U4CSE24124

**Semester:** 2<sup>nd</sup> semester

# MATRICES FUNCTIONS:

## 1.

The screenshot shows the MATLAB R2024b interface with the 'LIVE EDITOR' tab selected. The code editor displays several examples of matrix indexing:

```
%When used as an index, the colon operator (:) specifies all the elements
% in that dimension. eg: this code creates a column vector
% containing all the elements from the first column of A.x = (:,1)
C=rand(5,4)
C(:,2) %column
C(2,:) %row

% 2. You can use the colon operator to specify a range of values.
% This code creates a matrix containing the first, second, and
% third rows of the matrix A.
A=rand(5,6)
A(1:4,:)

%To index into a vector, use a single index value.
% x
% x(2)

%You can use a single range of index values to reference multiple vector
%elements.
% x(3)

% You can change the elements in a vector by combining indexing with
% assignment.
% x
```

The command window shows the results of the operations:

```
C = 5x4
0.3180 0.3326 0.5324 0.3219
0.6086 0.8531 0.7165 0.4039
0.2415 0.7283 0.3164 0.2074
0.0901 0.3044 0.3365 0.0487
0.5916 0.0332 0.1877 0.5527

ans = 5x1
0.3326
0.8531
0.7165
0.4039
0.1877

ans = 1x4
0.6086 0.8531 0.7165 0.4039

A = 5x6
0.2748 0.9357 0.1888 0.5431 0.7624 0.5044
0.2415 0.8187 0.0012 0.4398 0.5761 0.3473
0.2431 0.7283 0.3164 0.2874 0.7477 0.0921
0.1542 0.1758 0.6996 0.5017 0.6455 0.1478
0.9564 0.3604 0.6253 0.7615 0.1232 0.1982

ans = 4x6
0.2748 0.9357 0.1888 0.5431 0.7624 0.5044
0.2415 0.8187 0.0012 0.4398 0.5761 0.3473
0.2431 0.7283 0.3164 0.2874 0.7477 0.0921
0.1542 0.1758 0.6996 0.5017 0.6455 0.1478
```

The screenshot shows the MATLAB R2024b interface with the 'LIVE EDITOR' tab selected. The code editor displays examples of matrix indexing and multiplication:

```
% You can change the elements in a vector by combining indexing with
% assignment.
% x
% x(3)=6

% You can also use indexing to change the elements in a matrix.
A
A(2,3)=0

% Q. Define a random matrix A of order 5x4 and change the first row entries of A to be 1.
A=rand(5,4)
A(1,:)=1

% Q. MATLAB is designed to work naturally with arrays.
% eg: you can add a scalar value to all the elements of an array.
b=(2:6)
b+2
b*2

% Q. The * operator performs matrix multiplication.
```

The command window shows the results of the operations:

```
A = 5x4
0.2748 0.9357 0.1888 0.5431 0.7624 0.5044
0.2415 0.8187 0 0.4398 0.5761 0.3473
0.2431 0.7283 0.3164 0.2874 0.7477 0.0921
0.1542 0.1758 0.6996 0.5017 0.6455 0.1478
0.9564 0.3604 0.6253 0.7615 0.1232 0.1982

A = 5x4
0.6723 0.5323 0.0249 0.4593
0.8415 0.2794 0.0714 0.5925
0.6944 0.9462 0.8372 0.6866
0.2568 0.9864 0.9715 0.7194
0.0098 0.3927 0.0569 0.6500

A = 5x4
1.0000 1.0000 1.0000 1.0000
0.4316 0.2794 0.6714 0.5825
0.6454 0.9462 0.8372 0.6866
0.2568 0.9864 0.9715 0.7194
0.0098 0.3927 0.0569 0.6500

b = 1x5
2 3 4 5 6

ans = 1x5
4 5 6 7 8

ans = 1x5
4 6 8 10 12

P = 2x3
```

Live Editor

```

33 % Q. The * operator performs matrix multiplication.
34 P = rand(2,3)
35 Q = rand(3,4)
36 P*Q
37
38

39 % Q. The .* operator performs element-wise multiplication
40 p=[2;5]
41 q=[3;6]
42 p.*q
43
44

45 A = rand(3,3)
46 B = rand(3,4)
47 C = rand(4,3)
48 size(A)
49 size(B)
50 size(C)
51 length(A)
52 length(B)
53 length(C)
54 size(C,1) %for row
55 size(C,2) %for column

```

p = 1x4  
2 3 4 5

q = 1x4  
3 4 5 6

ans = 1x4  
6 12 20 30

A = 3x3  
0.1311 0.6146 0.7897  
0.4350 0.0110 0.2354  
0.0915 0.5733 0.4488

B = 3x4  
0.5694 0.6423 0.9711 0.2789  
0.8614 0.2213 0.8464 0.7466  
0.4963 0.8371 0.5060 0.2369

C = 4x3  
0.9573 0.0903 0.6996  
0.6203 0.2553 0.7252  
0.6003 0.8586 0.2299  
0.1726 0.9111 0.5761

ans = 1x2  
3 3

ans = 1x2  
3 4

ans = 1x2  
4 3

2.

Live Editor

```

1 %Solving System of Linear Equations using linsolve.

2 syms x y z
3 eqn1=2*x + y + z == 2;
4 eqn2= -x + y - z == 3;
5 eqn3= x + 2*y + 3*z == -10;
6 [A,b]= equationsToMatrix([eqn1,eqn2,eqn3],[x,y,z]);
sol = linsolve(A,b)

8 syms x y z
9 eqn1=x + y + z == 3;
10 eqn2= 2*x + 2*y + 2*z == 6;
11 [A,b]= equationsToMatrix([eqn1,eqn2],[x,y,z]);
sol = linsolve(A,b)

13 syms x y
14 eqn1=x + y == 3
15 eqn2= x + y == 5
16 [A,b]= equationsToMatrix([eqn1,eqn2],[x,y])
sol = linsolve(A,b)

```

$\begin{pmatrix} 3 \\ 1 \\ -5 \end{pmatrix}$

Warning: symbolic:mldivide:RankDeficientSystem#Solution is not unique because the system is rank-deficient.  
sol =  
 $\begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}$

eqn1 = x + y = 3  
eqn2 = x + y = 5  
A =  
 $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$   
b =  
 $\begin{pmatrix} 3 \\ 5 \end{pmatrix}$

Warning: symbolic:mldivide:InconsistentSystem#Solution does not exist because the system is inconsistent.  
sol =  
 $\begin{pmatrix} \infty \\ \infty \end{pmatrix}$

3.

The screenshot shows the MATLAB R2024b interface with the 'LIVE EDITOR' tab selected. The code in the editor is as follows:

```
1 %Solving System of Linear Equations Using backslash operator.
2
3 syms x y z
4 eqn1=2*x + y + z == 2;
5 eqn2= -x + y - z == 3;
6 eqn3= x + 2*y + 3*z == -10;
7 [A,b]= equationsToMatrix([eqn1,eqn2,eqn3],[x,y,z]);
sol = A\b
8
9 syms x y z
10 eqn1=x + y + z == 3;
11 eqn2= 2*x + 2*y + 3*z == 6;
12 [A,b]= equationsToMatrix([eqn1,eqn2],[x,y,z]);
sol = A\b
13
14 syms x y z
15 eqn1=x + y == 3;
16 eqn2= x + y == 5;
17 [A,b]= equationsToMatrix([eqn1,eqn2],[x,y]);
sol = A\b
```

A warning message is displayed on the right side of the editor:

Warning:  
symbolic/ldivide:InconsistentSystem#Solution  
does not exist because the system is  
inconsistent.%  
sol =  
( $\infty$ )  
( $\infty$ )

The status bar at the bottom shows: Zoom: 100% UTF-8 LF script Ln 1 Col 22. The system tray indicates it's 01:10 on 03-06-2025, the weather is 30°C, and the language is ENG IN.

4.

The screenshot shows the MATLAB R2024b interface with the 'LIVE EDITOR' tab selected. The code in the editor is as follows:

```
1 %Solving System of Linear Equations Using solve
2
3 % Solve system of linear equations
4 % 3 eqns
5
6 syms x y z
7 eqn1 = x + y == 5;
8 eqn2 = x - y == 1;
9 sol = solve([eqn1,eqn2],[x,y])
sol.x
10
11 syms x y z
12 eqn1 = 2*x+y+z == 2;
13 eqn2 = -x+y-z == 3;
14 eqn3 = x+2*y+3*z == -10;
15 sol=solve([eqn1,eqn2,eqn3],[x,y])
16
17 % solve the equation ax+b=0 for the variable x using the command linsolve or backslash operator
```

The output window on the right shows the results of the solve command:

```
x: 3
y: 2
ans = 3

sol = struct with fields:
    x: [0x1 sym]
    y: [0x1 sym]

sol =

$$\frac{-b}{a}$$


sol = struct with fields:
    x: 3
    y: 0
    z: 0
sol = struct with fields:
    x: 3 - z2 - z1
    y: z2
    z: z1
parameters: [z1 z2]
```

The status bar at the bottom shows: Zoom: 100% UTF-8 LF script Ln 1 Col 1. The system tray indicates it's 01:11 on 03-06-2025, the weather is 30°C, and the language is ENG IN.

Live Editor - C:\Users\bala\OneDrive\Documents\html\javaR\oops ass\linear lab

```

15    eqn = a*x + b==0;
16    sol = solve(eqn,x)

17 % solve the equation ax+b=0 for the variable x using the command linsolve or backslash operator

18 syms x y z
19 eqn1=x + y + z == 3;
20 eqn2= 2*x + 2*y + 2*z == 6;
21 sol = solve([eqn1,eqn2],[x,y,z])
22 sol = solve([eqn1,eqn2],[x,y,z],'ReturnConditions',true)

23 syms x y
24 eqn1 = x + y ==3;
25 eqn2 = x + y ==5;
26 sol = solve([eqn1,eqn2],[x,y])
27 sol.x
28 sol = solve([eqn1,eqn2],[x,y,z],'ReturnConditions',true)

```

sol = struct with fields:

- x: 3 - z2 - z1
- y: 0
- z: 0

sol = struct with fields:

- x: 3 - z2 - z1
- y: z2
- z: z1
- parameters: [z1 z2]
- conditions: symtrue

sol = struct with fields:

- x: [0x1 sym]
- y: [0x1 sym]

ans =

Empty sym: 0-by-1

sol = struct with fields:

- x: [0x1 sym]
- y: [0x1 sym]
- z: [0x1 sym]
- parameters: [1x0 sym]
- conditions: [0x1 sym]

## 5.

Live Editor - C:\Users\bala\OneDrive\Documents\html\javaR\oops ass\linear lab

```

1 Row reduced equivalent form

2 syms x y z
3 eqn1=2*x + y + z == 2;
4 eqn2= -x + y - z == 3;
5 eqn3= x + 2*y + 3*z == -10;
6 B = equationsToMatrix([eqn1,eqn2,eqn3],[x,y,z]);
R = rref(B)

8 % For any 5 sys of linear eqn that we solved
9 % solve it using linsolve backslash solve and also find rref

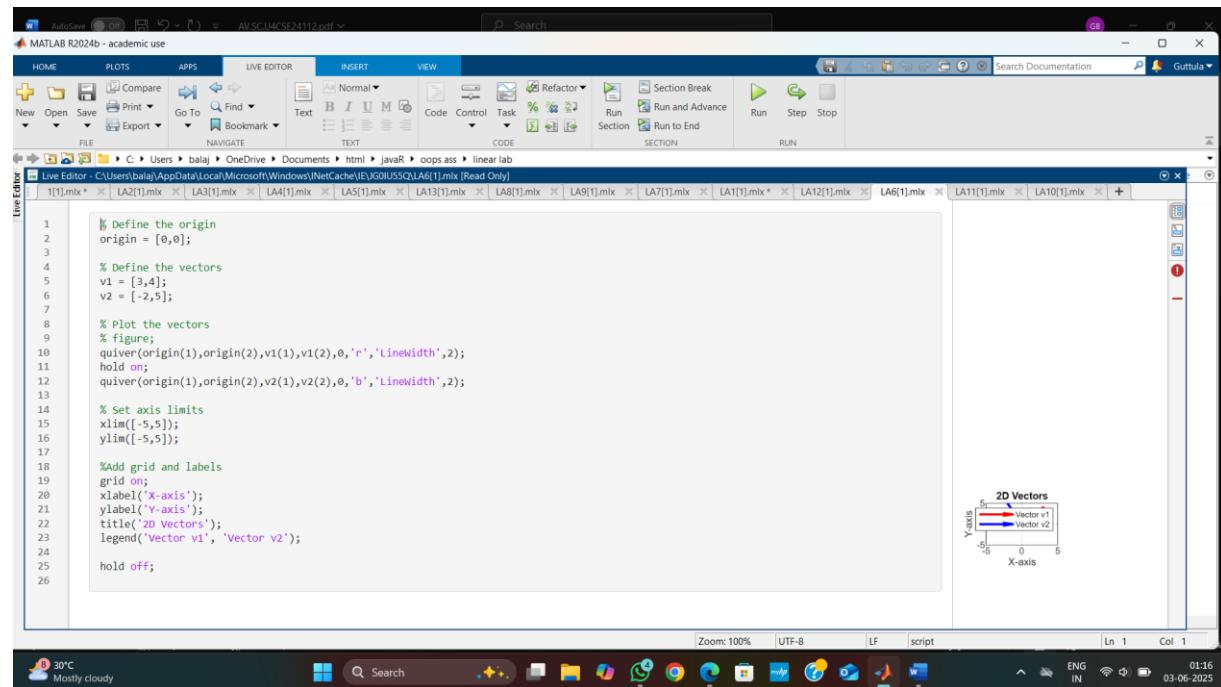
```

R =

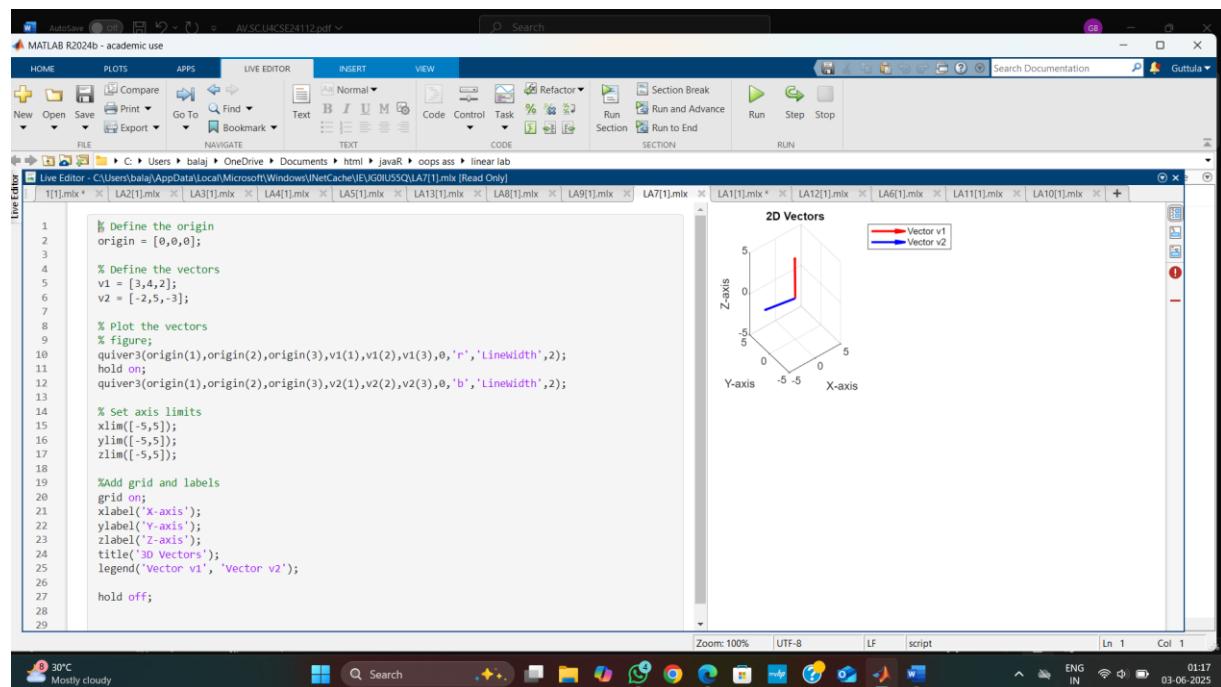
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

## VECTORS GRAPHS

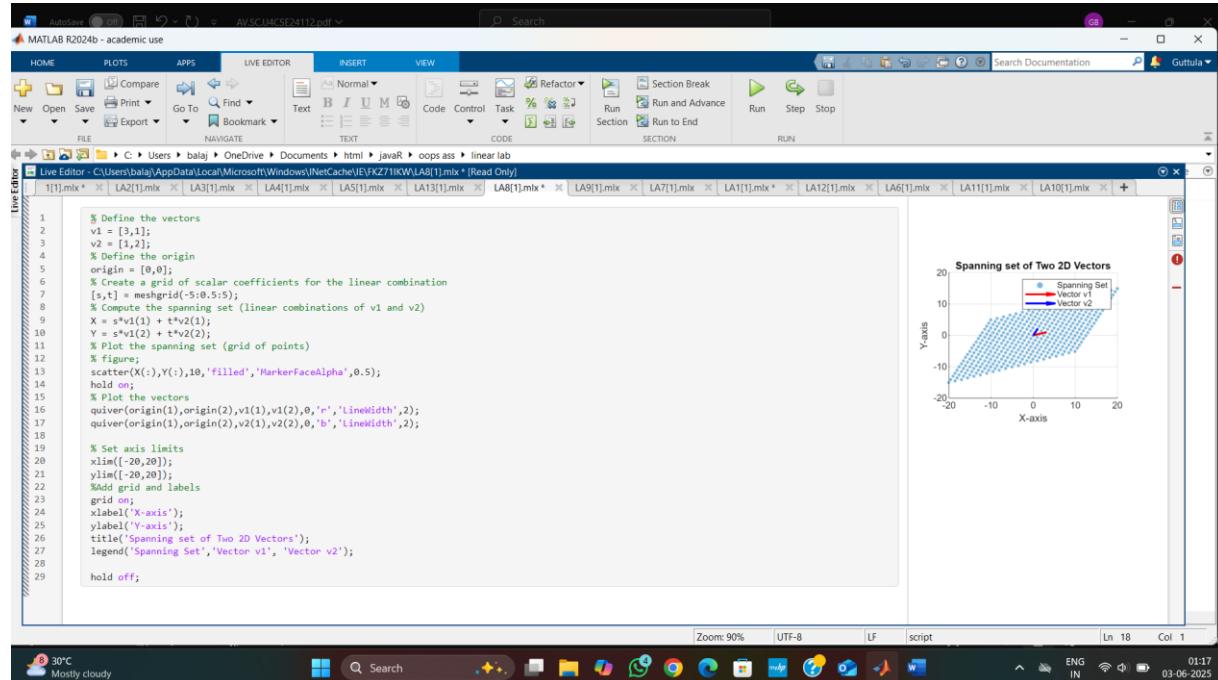
6.



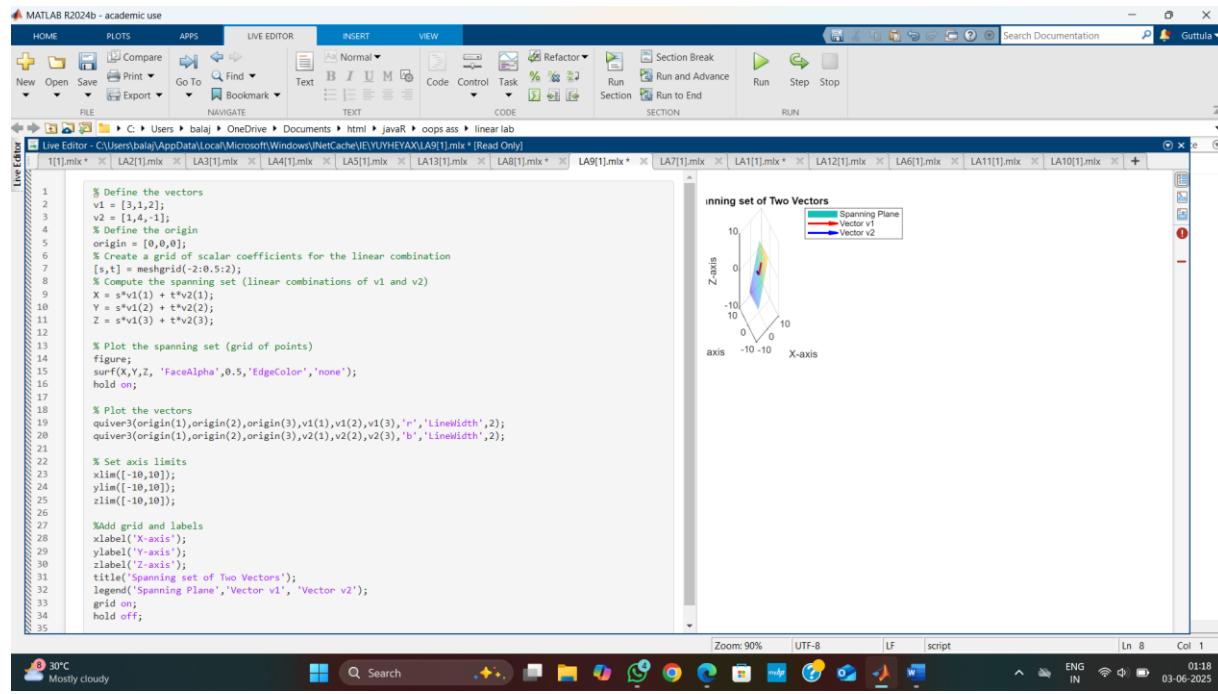
7.



## 8.



## 9.



## ORTHOGONALIZATION

## 10.

MATLAB R2024b - academic use

```

1 % Gram-Schmidt Orthogonalization in R^3
2
3 clc;
4 clear all;
5 close all;
6
7
8 v1=[1;1;0];
9 v2=[0;1;1];
10 v3=[1;0;1];
11
12 A=[v1,v2,v3];
13 fprintf('Original basis vectors:\n');
14
15 [m,n] = size(A)
16 Q=zeros(m,n);
17
18 for k=1:n
19
20 v=A(:,k);
21
22 for j=1:k-1
23 q=u(:,j);
24 v=v-(q'*v)/(q'*q)*q;
25 end
26
27 Q(:,k)=v;
28 end
29
30 fprintf('\n Orthogonal basis vectors:\n');
31 disp(Q);
32
33 fprintf('\n Orthogonality check:\n');
34 orth_check=Q'*Q;
35 disp(orth_check);

```

Original basis vectors:

m = 3
n = 3

Orthogonal basis vectors:

1.0000	-0.5000	0.6667
1.0000	0.5000	-0.6667
0	1.0000	0.6667

Orthogonality check:

2.0000	0	0
--------	---	---

Zoom: 90% | UTF-8 | LF | script | Ln 1 | Col 1

30°C Mostly cloudy

Search ENG IN 01:19 03-06-2025

MATLAB R2024b - academic use

```

1 % Gram-Schmidt Orthogonalization in R^3
2
3 clc;
4 clear all;
5 close all;
6
7
8 v1=[1;1;0];
9 v2=[0;1;1];
10 v3=[1;0;1];
11
12 A=[v1,v2,v3];
13 fprintf('Original basis vectors:\n');
14
15 [m,n] = size(A)
16 Q=zeros(m,n);
17
18 for k=1:n
19
20 v=A(:,k);
21
22 for j=1:k-1
23 q=u(:,j);
24 v=v-(q'*v)/(q'*q)*q;
25 end
26
27 Q(:,k)=v;
28 end
29
30 fprintf('\n Orthogonal basis vectors:\n');
31 disp(Q);
32
33 fprintf('\n Orthogonality check:\n');
34 orth_check=Q'*Q;
35 disp(orth_check);
36
37 Q_normalized = Q ./vecnorm(Q);
38 fprintf('\n Orthonormal basis vectors :\n');
39 disp(Q_normalized);
40
41 fprintf('Orthogonality check:\n');
42 disp(Q_normalized'*Q_normalized);
43
44 % * Exercise Question
45 % For any given positive integer n write a matlab program to find an
46 % orthonormal basis from a given basis for R^n.

```

Original basis vectors:

m = 3
n = 3

Orthogonal basis vectors:

1.0000	-0.5000	0.6667
1.0000	0.5000	-0.6667
0	1.0000	0.6667

Orthogonality check:

2.0000	0	0
0	1.0000	0.0000
0	0.0000	1.3333

Orthonormal basis vectors :

0.7071	-0.4822	0.5774
0.7071	0.4822	-0.5774
0	0.8165	0.5774

Orthogonality check:

1.0000	0	0
0	1.0000	0.0000
0	0.0000	1.0000

Zoom: 90% | UTF-8 | LF | script | Ln 1 | Col 1

30°C Mostly cloudy

Search ENG IN 01:19 03-06-2025

## 11.

The screenshot shows the MATLAB R2024b interface with a live editor window open. The script content is as follows:

```
% Gram-Schmidt Orthogonalization in R^3 with QR decomposition
clc;
clear all;
close all;

v1=[1;1;0];
v2=[0;1;1];
v3=[1;0;1];

A=[v1,v2,v3];
fprintf('Original basis vectors:\n');

[m,n] = size(A);
Q=zeros(m,n);

for k=1:n
    v=A(:,k);
    for j=1:k-1
        q=Q(:,j);
        v=v-(q'*v)/(q'*q)*q;
    end
    Q(:,k)=v;
end

R=Q' * A;

fprintf('\n Orthogonal basis vectors:\n');
disp(Q);
```

The output window displays the original basis vectors and the orthogonal basis vectors. The orthogonal basis vectors are:

1.0000	-0.5000	0.6667
1.0000	0.5000	-0.6667
0	1.0000	0.6667

System status at the bottom: 30°C, Mostly cloudy, 01:20, 03-06-2025.

The screenshot shows the MATLAB R2024b interface with a live editor window open. The script content is as follows:

```
% Gram-Schmidt Orthogonalization in R^3 with QR decomposition
clc;
clear all;
close all;

v1=[1;1;0];
v2=[0;1;1];
v3=[1;0;1];

A=[v1,v2,v3];
[m,n] = size(A);
Q=zeros(m,n);

for k=1:n
    v=A(:,k);
    for j=1:k-1
        q=Q(:,j);
        v=v-(q'*v)/(q'*q)*q;
    end
    Q(:,k)=v;
end

R=Q' * A;

fprintf('\n Orthogonal basis vectors:\n');
disp(Q);

fprintf('\n Orthogonality check:\n');
orth_check=Q'*Q;
disp(orth_check);

Q_normalized = Q ./vecnorm(Q);
fprintf('\n Orthonormal basis vectors :\n');
disp(Q_normalized);

fprintf('Orthogonality check:\n');
disp(Q_normalized'*Q_normalized);

% Write a matlab program to find a QR decomposition of a given matrix using the formulas we discussed in the th
```

The output window displays the original basis vectors, the orthogonal basis vectors, the orthogonality check, and the orthonormal basis vectors. The orthogonality check is:

2.0000	0	0
0	1.0000	0.0000
0	0.0000	1.3333

The orthonormal basis vectors are:

0.7071	-0.4082	0.5774
0.7071	0.4082	-0.5774
0	0.8165	0.5774

System status at the bottom: 30°C, Mostly cloudy, 01:20, 03-06-2025.

## SHAPES GRAPHS

## 12.

MATLAB R2024b - academic use

LIVE EDITOR

```
% Visualize how a matrix transforms the unit circle
A = [1 2; 3 1]; % Example transformation matrix

% Create unit circle
theta = linspace(0,2*pi,100);
x = cos(theta);
y = sin(theta);
circle = [x; y];

% Apply transformation
transformed = A * circle;

% Plot
figure;
subplot(1,2,1);
plot(circle(:,1),circle(:,2));
title('Unit Circle');
axis equal;

subplot(1,2,2);
plot(transformed(:,1),transformed(:,2));
title('Transformed Shape');
axis equal;
```

% Visualize how a matrix transforms a square
A = [1 0.5; -0.5 1]; % Example transformation matrix

% Create square vertices
square = [0 1 1 0; 0 0 1 1 0];

% Apply transformation
transformed = A \* square;

% Plot

Unit Circle Transformed Shape

Original Square Transformed Shape

Zoom: 90% | UTF-8 | LF | script | ENG IN | 01:21 | 03-06-2025

MATLAB R2024b - academic use

LIVE EDITOR

```
% Visualize how a matrix transforms the unit circle
A = [1 2; 3 1]; % Example transformation matrix

% Create unit circle
theta = linspace(0,2*pi,100);
plot(theta, sin(theta), 'b-', 'LineWidth', 2);
title('Original Circle');
axis equal;
grid on;

% Plot
figure;
subplot(1,2,1);
plot(square(:,1),square(:,2),'b-', 'LineWidth', 2);
title('Original Square');
axis equal;
grid on;

subplot(1,2,2);
plot(transformed(:,1),transformed(:,2),'r-', 'LineWidth', 2);
title('Transformed Shape');
axis equal;
grid on;

% Visualize how a matrix transforms a triangle
A = [1 1; 0 1]; % shear transformation

% Create triangle vertices
triangle = [0 1 0.5 0; 0 0 1 0];

% Apply transformation
transformed = A * triangle;

% Plot
figure;
subplot(1,2,1);
fill(triangle(:,1),triangle(:,2), 'g');
title('Original Triangle');
axis equal;
grid on;

subplot(1,2,2);
plot(transformed(:,1),transformed(:,2), 'm-');
title('Transformed Shape');
axis equal;
grid on;
```

Unit Circle Transformed Shape

Original Square Transformed Shape

Original Triangle Transformed Shape

Zoom: 90% | UTF-8 | LF | script | ENG IN | 01:21 | 03-06-2025

## 13.

MATLAB R2024b - academic use

LIVE EDITOR

FILE

Live Editor - C:\Users\bala\OneDrive\Documents\html\javaR\oops ass\linear lab

```
1 clc;
2 clear;
3
4 A = input('Enter your matrix:');
5
6 [m,n] = size(A);
7 if m ~= n
8     error('Matrix must be square for diagonalization');
9 end
10
11 [P, D] = eig(A);
12 eigenvalues = diag(D);
13
14 disp('Eigenvalues:');
15 disp(eigenvalues);
16 disp('Eigenvectors (columns of P):');
17 disp(P);
18
19 if rank(P) == m
20     disp('Matrix is diagonalizable');
21     disp('Matrix P (eigenvectors):');
22     disp(P);
23     disp('Inverse of P:');
24     invP = inv(P);
25     disp(invP);
26
27     diagonal_matrix = invP * A * P;
28     disp('Diagonal matrix (inv(P)*A*P)');
29     disp(diagonal_matrix);
30 else
31     disp('Matrix is NOT diagonalizable - not enough linearly independent eigenvalues');
32 end
33
34 % Write a matlab code to orthogonally diagonalize a given symmetric matrix
```

Eigenvalues:  
-0.3723 5.3723  
Eigenvectors (columns of P):  
-0.8246 -0.4168  
0.5658 -0.9894  
Matrix is diagonalizable  
Matrix P (eigenvectors):  
-0.8246 -0.4168  
0.5658 -0.9894  
Inverse of P:  
-0.9231 0.4222  
-0.5743 -0.8370  
Diagonal matrix (inv(P)\*A\*P)  
-0.3723 0.0000  
0 5.3723

Zoom: 90% | UTF-8 | LF | script | Ln 1 | Col 1

30°C Mostly cloudy ENG IN 01:23 03-06-2025

MATLAB R2024b - academic use

HOME PLOTS APPS

New New New Open Find Files Import Data Variable Favorites Analyze Code Layout Preferences Add-Ons Help Request Support Resources

FILE VARIABLE ENVIRONMENT RESOURCES

Live Editor

Command Window

```
Enter your matrix:[1 2; 3 4]
f3 >> |
```

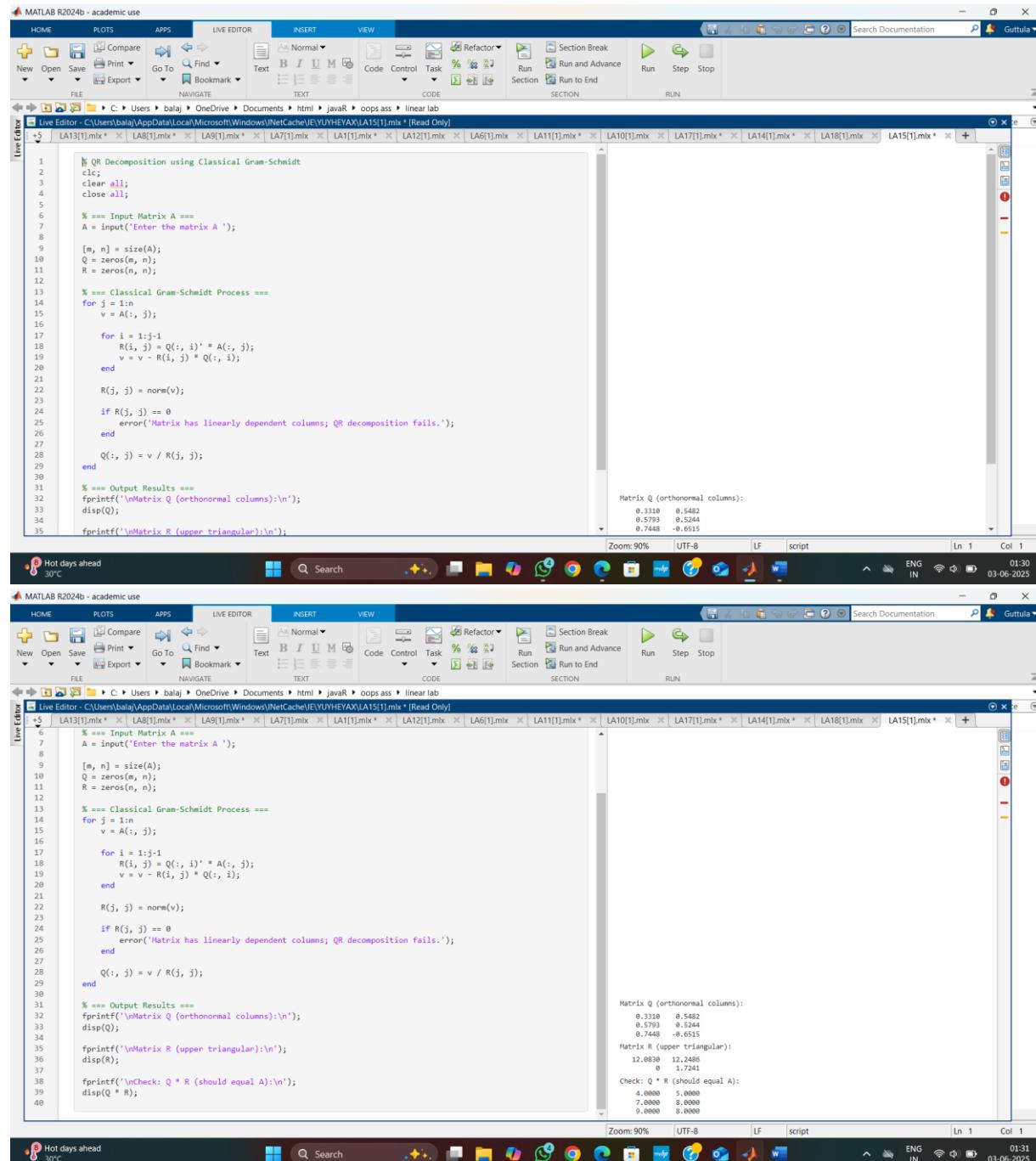
Workspace

Name: A, D, diagonal\_mat, eigenvalues, invP, m, n, P

30°C Mostly cloudy ENG IN 01:24 03-06-2025

## QUESTIONS ASKED:

### 1.WRITE A MATLAB PROGRAM TO FIND A QR DECOMPOSITION OF A GIVEN MATRIX USING THE FORMULAS WE DISCUSSED IN THE THEORY SESSION



The screenshot shows two instances of the MATLAB R2024b interface. Both instances display the same MATLAB script for QR decomposition and its execution results.

**Script Content:**

```
% QR Decomposition using Classical Gram-Schmidt
clc;
clear all;
close all;

% *** Input Matrix A ***
A = input('Enter the matrix A ');

[m, n] = size(A);
Q = zeros(m, n);
R = zeros(n, n);

% *** Classical Gram-Schmidt Process ***
for j = 1:n
    v = A(:, j);
    R(j, j) = norm(v);
    if R(j, j) == 0
        error('Matrix has linearly dependent columns; QR decomposition fails.');
    end
    Q(:, j) = v / R(j, j);
end

% *** Output Results ***
fprintf('\nMatrix Q (orthonormal columns):\n');
disp(Q);
fprintf('\nMatrix R (upper triangular):\n');

```

**Execution Results (Visible in the first window):**

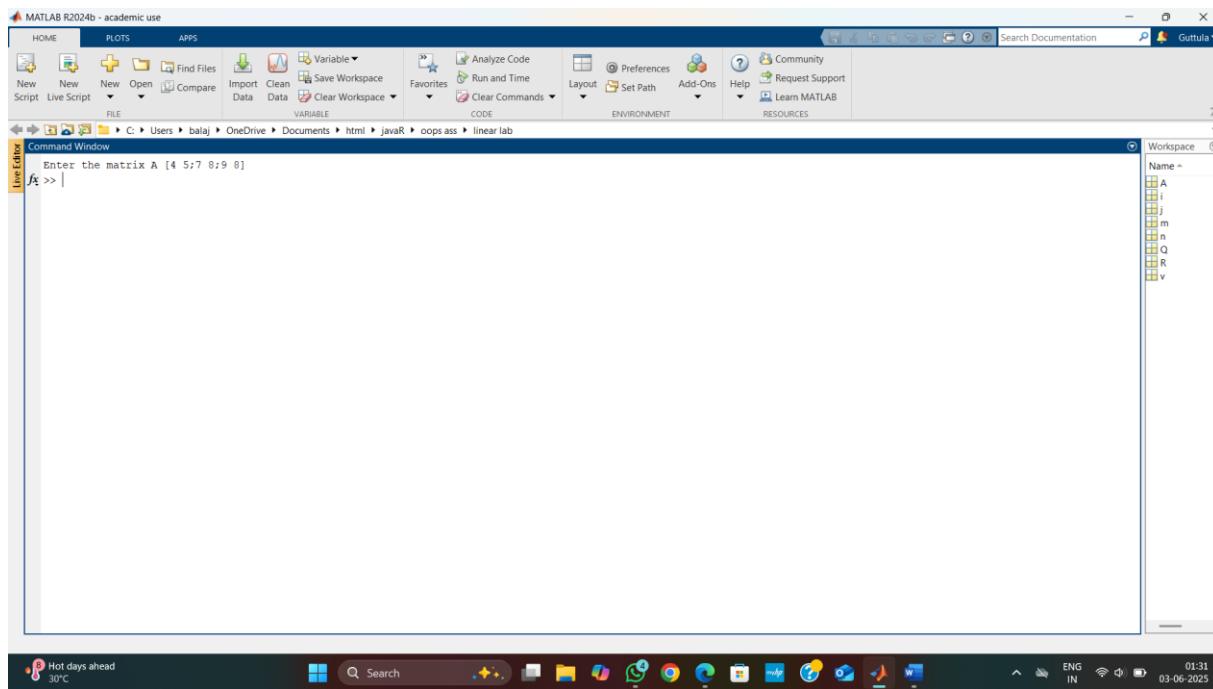
```
Matrix Q (orthonormal columns):
0.3310 0.5482
0.5793 0.5244
0.7448 -0.6515
```

**Execution Results (Visible in the second window):**

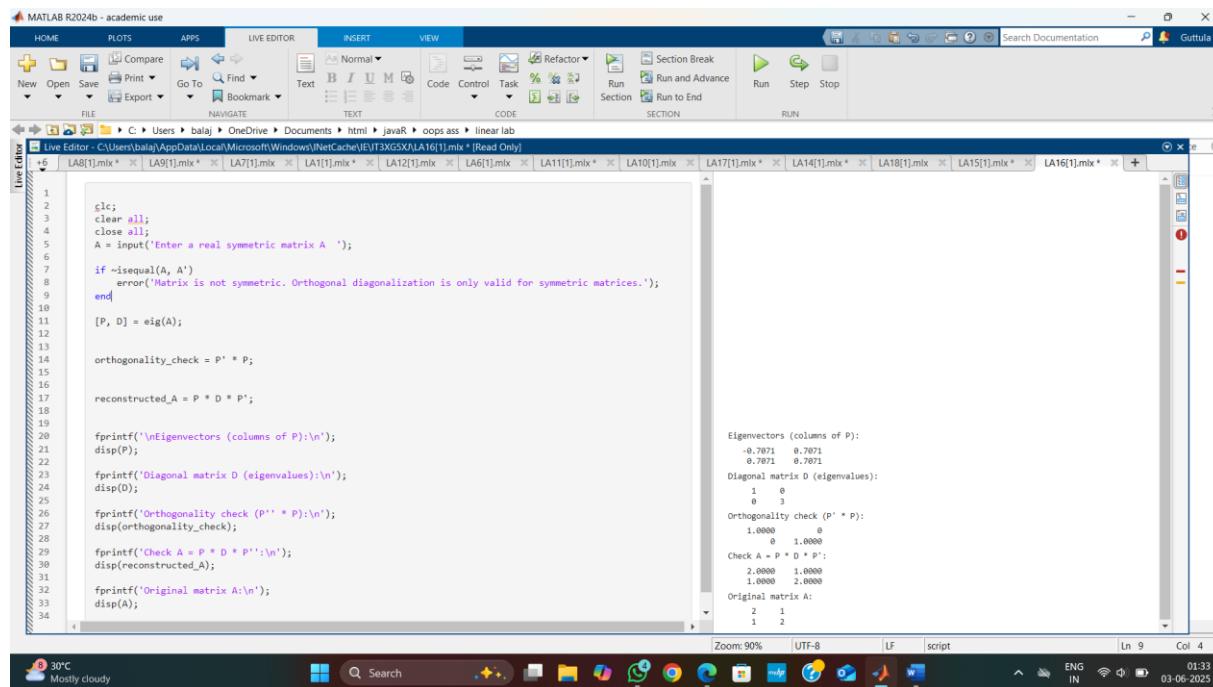
```
Matrix Q (orthonormal columns):
0.3310 0.5482
0.5793 0.5244
0.7448 -0.6515

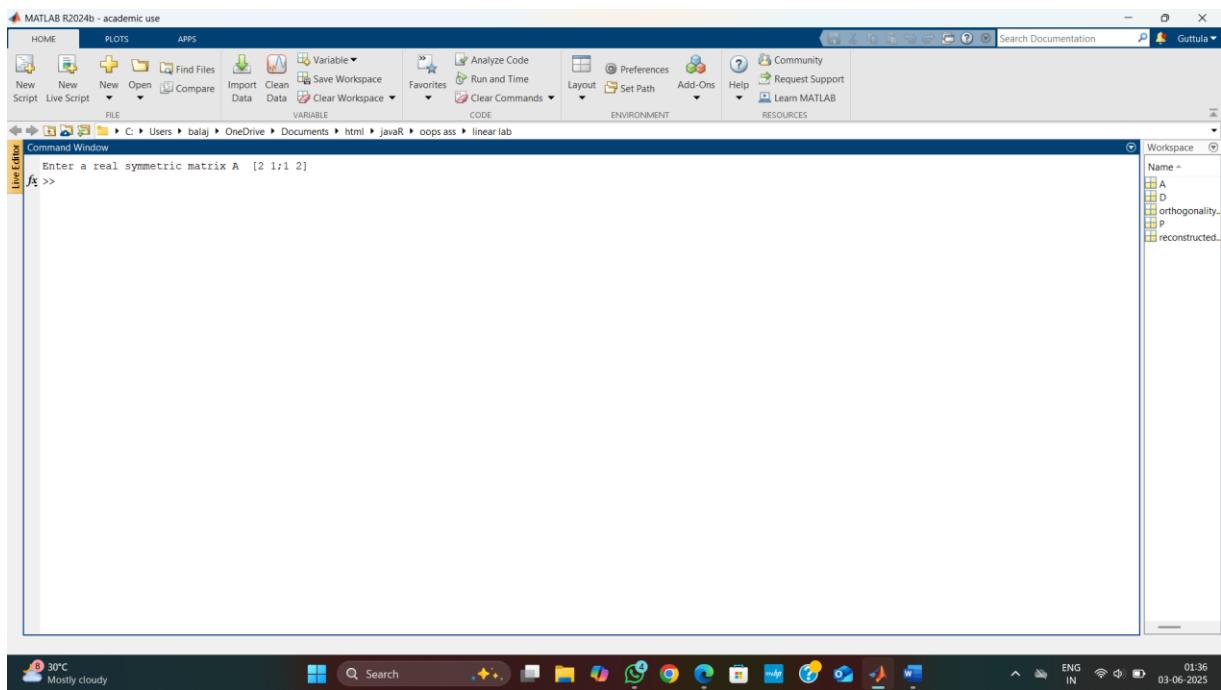
Matrix R (upper triangular):
12.0830 12.2486
0 1.7241

Check: Q * R (should equal A):
4.0000 5.0000
7.0000 8.0000
9.0000 9.0000
```



## 2. WRITE A MATLAB CODE TO ORTHOGONALLY DIAGONALIZE A GIVEN SYMMETRIC MATRIX





### 3. FOR ANY 5 SYS OF LINEAR EQN THAT WE SOLVEDSOLVE IT USING LINSOLVE BACKSLASH SOLVE AND ALSO FIND RREF

The screenshot shows the MATLAB R2024b interface with a Live Editor script. The script generates random coefficient matrices A and right-hand side vectors b for systems 1 through 17. It then solves each system using linsolve, backslash (\), and symbolic solve, and finds the reduced row echelon form (RREF) of the augmented matrices.

```

1 glc;
2 clear all;
3 close all;
4
5
6 for system_num = 1:5
7
8 fprintf('System %d\n', system_num);
9
10 A = randi([-10, 10], 3, 3);
11 while rank(A) < 3
12     A = randi([-10, 10], 3, 3);
13 end
14 b = randi([-10, 10], 3, 1);
15
16 fprintf('\nCoefficient matrix A:\n');
17 disp(A);
18
19 fprintf('Right-hand side vector b:\n');
20 disp(b);
21
22 x1 = linsolve(A, b);
23 fprintf('Solution using linsolve:\n');
24 disp(x1);
25
26 x2 = A \ b;
27 fprintf('Solution using backslash (\ ) operator:\n');
28 disp(x2);
29
30 syms x y z;
31 vars = [x; y; z];
32 eqs = A * vars == b;
33 S = solve(eqs);
34 x3 = [S.x; S.y; S.z];
35

```

Live Editor

```

1 Live Editor - C:\Users\bala\OneDrive\Documents\html\javaR\oops ass> linear lab
2 [LA2[1].mlx] [LA3[1].mlx] [LA4[1].mlx] [LAS[1].mlx] [LA13[1].mlx*] [LAB[1].mlx*] [LA9[1].mlx*] [LA7[1].mlx*] [LA1[1].mlx*] [LA12[1].mlx*] [LA6[1].mlx*] [LA11[1].mlx*] [LA10[1].mlx*] [LA17[1].mlx*]
3
4 A = randi([-10, 10], 3, 3);
5
6 end
7 b = randi([-10, 10], 3, 1);
8
9 fprintf('\nCoefficient matrix A:\n');
10 disp(A);
11
12 fprintf('Right-hand side vector b:\n');
13 disp(b);
14
15 x1 = linsolve(A, b);
16 fprintf('Solution using linsolve:\n');
17 disp(x1);
18
19 x2 = A \ b;
20 fprintf('Solution using backslash (\backslash) operator:\n');
21 disp(x2);
22
23 syms x y z;
24 vars = [x; y; z];
25 eqs = A * vars == b;
26 S = solve(eqs);
27 x3 = [S.x; S.y; S.z];
28 fprintf('Solution using symbolic solve:\n');
29 disp(x3);
30
31 Ab = [A, b];
32 R = rref(Ab);
33 x4 = R(:, end);
34 fprintf('Solution using rref of augmented matrix:\n');
35 disp(x4);
36
37 end

```

Solution using linsolve:

$$\begin{pmatrix} 0.1893 \\ -0.5355 \\ 1.1967 \end{pmatrix}$$

Solution using backslash (\) operator:

$$\begin{pmatrix} -0.1893 \\ -0.5355 \\ 1.1967 \end{pmatrix}$$

Solution using symbolic solve:

$$\begin{pmatrix} 20 \\ 183 \\ 98 \\ -183 \\ 73 \\ 61 \end{pmatrix}$$

Solution using rref of augmented matrix:

$$\begin{pmatrix} 0.1893 \\ -0.5355 \\ 1.1967 \end{pmatrix}$$

System #3

Coefficient matrix A:

$$\begin{pmatrix} 6 & -2 & 4 \\ -4 & 2 & 4 \\ 1 & -5 & 5 \end{pmatrix}$$

Right-hand side vector b:

$$\begin{pmatrix} -1 \\ -9 \\ -6 \end{pmatrix}$$

Solution using linsolve:

$$\begin{pmatrix} -0.0909 \\ -0.7727 \\ -1.9545 \end{pmatrix}$$

Zoom: 90% | UTF-8 | LF | script | L1 | Col 1

30°C Mostly cloudy

Live Editor

```

1 Live Editor - C:\Users\bala\OneDrive\Documents\html\javaR\oops ass> linear lab
2 [LA2[1].mlx] [LA3[1].mlx] [LA4[1].mlx] [LAS[1].mlx] [LA13[1].mlx*] [LAB[1].mlx*] [LA9[1].mlx*] [LA7[1].mlx*] [LA1[1].mlx*] [LA12[1].mlx*] [LA6[1].mlx*] [LA11[1].mlx*] [LA10[1].mlx*] [LA17[1].mlx*]
3
4 A = randi([-10, 10], 3, 3);
5
6 end
7 b = randi([-10, 10], 3, 1);
8
9 fprintf('\nCoefficient matrix A:\n');
10 disp(A);
11
12 fprintf('Right-hand side vector b:\n');
13 disp(b);
14
15 x1 = linsolve(A, b);
16 fprintf('Solution using linsolve:\n');
17 disp(x1);
18
19 x2 = A \ b;
20 fprintf('Solution using backslash (\backslash) operator:\n');
21 disp(x2);
22
23 syms x y z;
24 vars = [x; y; z];
25 eqs = A * vars == b;
26 S = solve(eqs);
27 x3 = [S.x; S.y; S.z];
28 fprintf('Solution using symbolic solve:\n');
29 disp(x3);
30
31 Ab = [A, b];
32 R = rref(Ab);
33 x4 = R(:, end);
34 fprintf('Solution using rref of augmented matrix:\n');
35 disp(x4);
36
37 end

```

Right-hand side vector b:

$$\begin{pmatrix} -1 \\ -9 \\ -6 \end{pmatrix}$$

Solution using linsolve:

$$\begin{pmatrix} -0.0909 \\ -0.7727 \\ -1.9545 \end{pmatrix}$$

Solution using backslash (\) operator:

$$\begin{pmatrix} -0.0909 \\ -0.7727 \\ -1.9545 \end{pmatrix}$$

Solution using symbolic solve:

$$\begin{pmatrix} -1 \\ 17 \\ -23 \\ 43 \end{pmatrix}$$

Solution using rref of augmented matrix:

$$\begin{pmatrix} -0.0909 \\ -0.7727 \\ -1.9545 \end{pmatrix}$$

System #4

Coefficient matrix A:

$$\begin{pmatrix} 9 & 1 & -1 \\ -7 & 10 & -8 \\ 7 & -9 & 10 \end{pmatrix}$$

Right-hand side vector b:

$$\begin{pmatrix} -10 \\ 6 \\ 7 \end{pmatrix}$$

Solution using linsolve:

$$\begin{pmatrix} -0.9963 \\ 3.6447 \end{pmatrix}$$

Zoom: 90% | UTF-8 | LF | script | L1 | Col 1

30°C Mostly cloudy

```

13 A = randi([-10, 10], 3, 3);
14 end
15 b = randi([-10, 10], 3, 1);
16
17 fprintf('\nCoefficient matrix A:\n');
18 disp(A);
19
20 fprintf('Right-hand side vector b:\n');
21 disp(b);
22
23 x1 = linsolve(A, b);
24 fprintf('Solution using linsolve:\n');
25 disp(x1);
26
27 x2 = A \ b;
28 fprintf('Solution using backslash (\backslash) operator:\n');
29 disp(x2);
30
31 syms x y z;
32 vars = [x, y, z];
33 eqs = A * vars == b;
34 S = solve(eqs);
35 x3 = [S.x; S.y; S.z];
36 fprintf('Solution using symbolic solve:\n');
37 disp(x3);
38
39 Ab = [A, b];
40 R = rref(Ab);
41 x4 = R(:, end);
42 fprintf('Solution using rref of augmented matrix:\n');
43 disp(x4);
44
45
46
47 end

```

Solution using rref of augmented matrix:

$$\begin{pmatrix} -272 \\ 273 \\ 995 \\ 273 \\ 1277 \\ 273 \end{pmatrix}$$

System #5

Coefficient matrix A:

$$\begin{pmatrix} 8 & -1 & -7 \\ -9 & 6 & -7 \\ -2 & -1 & -5 \end{pmatrix}$$

Right-hand side vector b:

$$\begin{pmatrix} -7 \\ -8 \\ 8 \end{pmatrix}$$

Solution using linsolve:

$$\begin{pmatrix} 7.8958 \\ 4.0288 \\ -5.5625 \end{pmatrix}$$

Solution using backslash (\) operator:

$$\begin{pmatrix} 7.8958 \\ 4.0288 \\ -5.5625 \end{pmatrix}$$

Solution using symbolic solve:

$$\begin{pmatrix} 379 \\ 48 \\ 193 \\ 48 \\ 89 \\ 16 \end{pmatrix}$$

## 4. SOLVE THE EQUATION AX+B=0 FOR THE VARIABLE X USING THE COMMAND LINSOLVE OR BACKSLASH OPERATOR

```

1 %le;
2 clear all;
3 close all;
4
5 a = input('Enter the coefficient a: ');
6 b = input('Enter the constant b: ');
7
8 if a == 0
9     if b == 0
10         fprintf('Infinite solutions: The equation reduces to 0 = 0.\n');
11     else
12         fprintf('No solution: The equation reduces to 0 = %g.\n', -b);
13     end
14 else
15     x1 = (-b) / a;
16     fprintf('\nSolution using backslash (manual): x = %g\n', x1);
17
18 A = a;
19 M_B = b;
20 x2 = linsolve(A, rhs);
21 fprintf('Solution using linsolve: x = %g\n', x2);
22
23 end

```

Solution using backslash (manual):  $x = -1$

Solution using linsolve:  $x = -1$

## 5. FOR ANY GIVEN POSITIVE INTEGER N WRITE A MATLAB PROGRAM TO FIND AN ORTHONORMAL BASIS FROM A GIVEN BASIS FOR R<sup>n</sup>.

The screenshot shows the MATLAB R2024b interface with a script named 'linearlab.m' open in the Live Editor. The script prompts the user for the dimension n, initializes a matrix A with zeros, and then iterates through each column k. For each column, it finds the original basis vector v, calculates its norm q, and then projects v onto q. The resulting vector v is then subtracted from v to get the orthogonal component, which is then normalized to form the k-th column of the orthogonal matrix Q. Finally, the script prints the original basis vectors, the orthogonal basis vectors, and the orthogonality check (Q' \* Q).

```

1 %
2 %
3 %clc;
4 %clear all;
5 %close all;
6 %
7 n = input('Enter the dimension n (for R^n): ');
8 %
9 %
10 A = zeros(n, n);
11 for i = 1:n
12     A(:, i) = input(sprintf('Enter vector %d: ', i));
13 end
14 %
15 fprintf('\nOriginal basis vectors:\n');
16 disp(A);
17 %
18 Q = zeros(n, n);
19 %
20 for k = 1:n
21     v = A(:, k);
22     for j = 1:k-1
23         q = Q(:, j);
24         proj = (q' * v) / (q' * q) * q;
25         v = v - proj;
26     end
27     Q(:, k) = v;
28 end
29 %
30 fprintf('\nOrthogonal basis vectors:\n');
31 disp(Q);
32 %
33 fprintf('\nOrthogonality check (Q'*Q):\n');
34 %
35 fprintf(Q'*Q);
36 %
37 %
38 
```

This screenshot shows the same MATLAB session after adding normalization code. The script now includes a line 'Q\_normalized = Q ./ vecnorm(Q);' before printing the orthogonal basis vectors. This results in unit vectors. The rest of the script remains the same, including the orthogonality check and the final orthonormality check.

```

1 %
2 %
3 %clc;
4 %clear all;
5 %close all;
6 %
7 n = input('Enter the dimension n (for R^n): ');
8 %
9 %
10 A = zeros(n, n);
11 for i = 1:n
12     A(:, i) = input(sprintf('Enter vector %d: ', i));
13 end
14 %
15 fprintf('\nOriginal basis vectors:\n');
16 disp(A);
17 %
18 Q = zeros(n, n);
19 %
20 for k = 1:n
21     v = A(:, k);
22     for j = 1:k-1
23         q = Q(:, j);
24         proj = (q' * v) / (q' * q) * q;
25         v = v - proj;
26     end
27     Q(:, k) = v;
28 end
29 %
30 fprintf('\nOrthogonal basis vectors:\n');
31 disp(Q);
32 %
33 fprintf('\nOrthogonality check (Q'*Q):\n');
34 disp(Q'*Q);
35 %
36 fprintf('\nOrthonormal basis vectors:\n');
37 disp(Q_normalized);
38 %
39 Q_normalized = Q ./ vecnorm(Q);
40 %
41 fprintf('\nOrthonormality check (Q_normalized'*Q_normalized):\n');
42 disp(Q_normalized'*Q_normalized);
43 %
44 fprintf('\nOrthonormality check (Q_normalized'*Q_normalized):\n');
45 disp(Q_normalized'*Q_normalized);
46 
```

