



LINEAR ALGEBRA
(23MAT117)

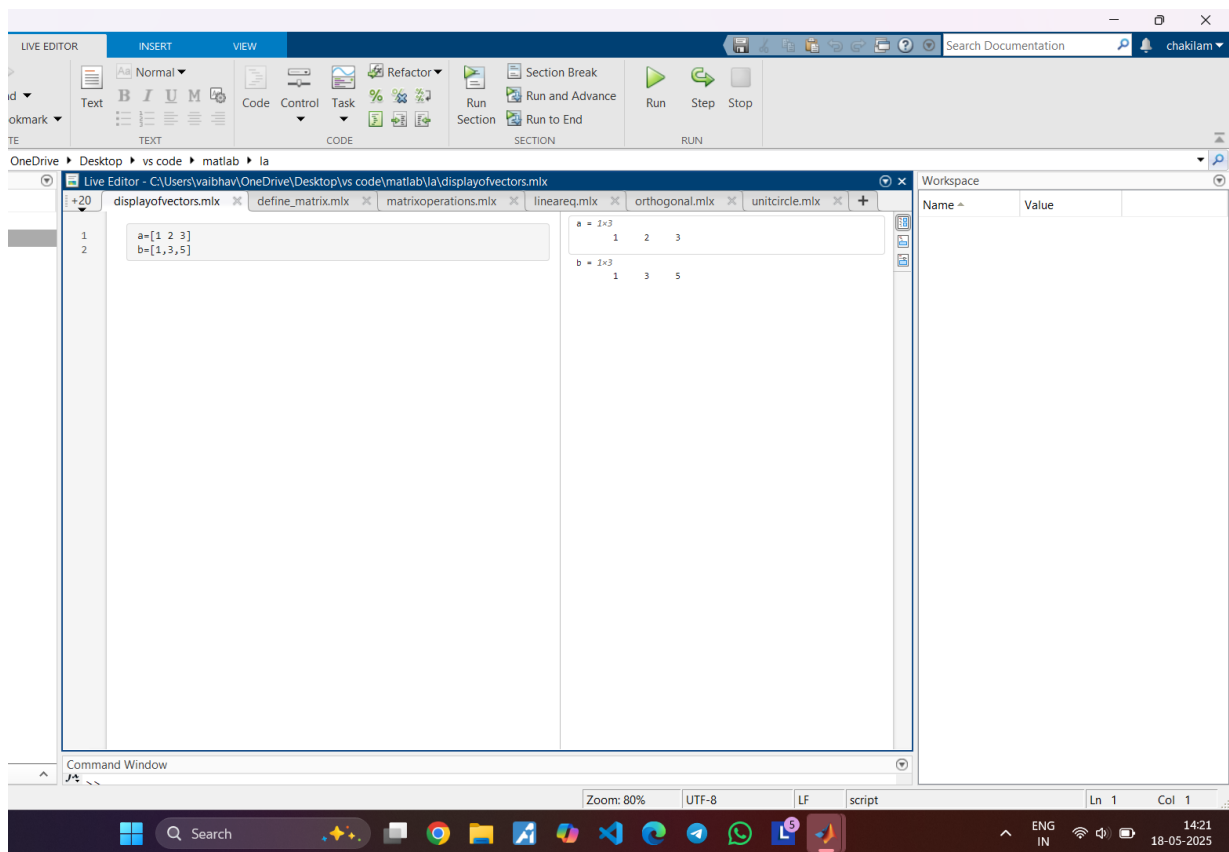
LAB MANUAL

Submitted by	
Name	C VAIBHAV
Roll No	AV.SC.U4CSE24033
Year/Sem/Section	1 st Year/2 nd Sem/CSE-A
Date of Submission	
Submitted to	
Name	Dr. Rashmi Prasad
Department	Mathematics
Designation	Professor

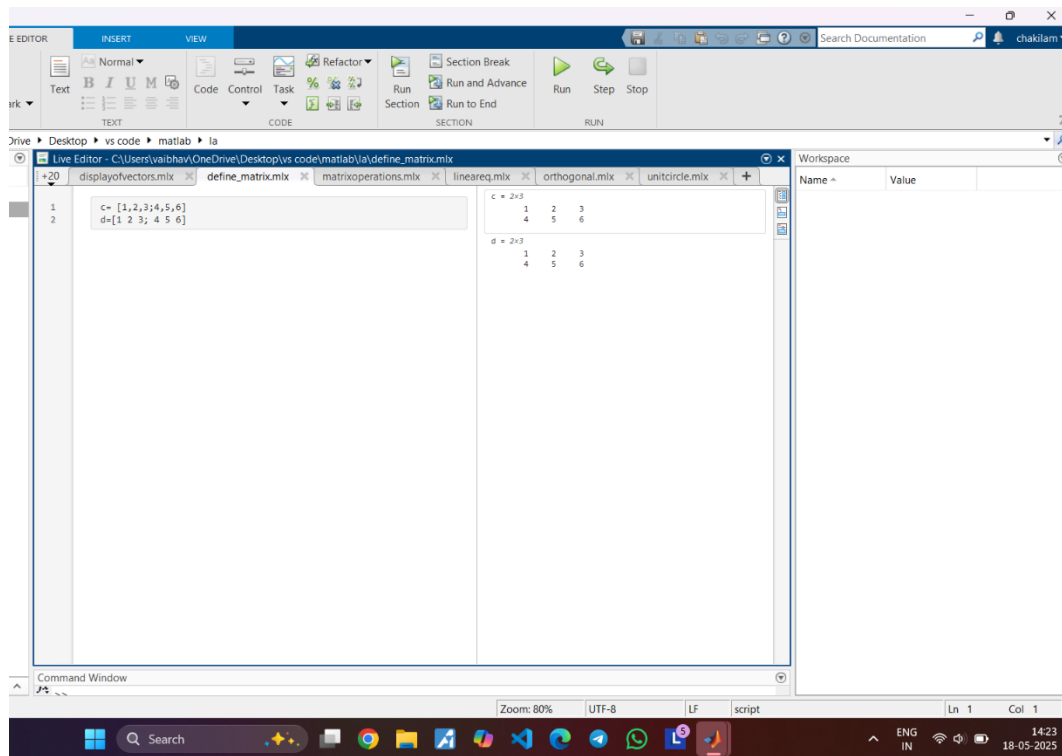
Marks	
-------	--

S.No		Signature
1.	Write a MATLAB code to define and display vectors.	
2.	Write a MATLAB code to define and display matrix	
3.	Write a MATLAB code to define and display a random matrix and their operations.	
4.	Write a MATLAB code to find the solutions of given linear system of equations.	
5.	Write a MATLAB code to draw the vectors (3,4) and (-2,5) in a 2-D plane.	
6.	Write a MATLAB code to draw the vectors (3,4) and (-2,5) in a 3-D plane.	
7.	Using MATLAB, define the vectors $v_1 = [3, 1]$ and $v_2 = [1, 2]$. Write a program to visualize their span in 2D space.	
8.	Using MATLAB, define the vectors $v_1 = (3, 1, 2)$ and $v_2 = (1, 4, -1)$. Write a program to visualize their span in 3D space.	
9.	Using MATLAB, for any given positive integer and for any given basis for \mathbb{R}^n , write a program to find the orthonormal basis corresponding to the given basis.	
10.	Using MATLAB, write a program to find the QR decomposition corresponding to the given	
11.	Using MATLAB, write a program to Visualize how a matrix transforms the unit circle.	
12.	Using MATLAB, write a program to Visualize how a matrix transforms a square.	
13.	Using MATLAB, write a program to Visualize how a matrix transforms a triangle	
14.	Using MATLAB, write a program to find diagonalisation of the matrix.	
15.	Using MATLAB, write a program to find orthogonal diagonalisation of a given matrix.	

1. Write a MATLAB code to define and display vectors.



2. Write a MATLAB code to define and display matrix.



3. Write a MATLAB code to define and display a random matrix and their operations.

LIVE EDITOR

INSERT

VIEW

OneDrive

Desktop

vs code

matlab

la

Live Editor - C:\Users\vaibhav\OneDrive\Desktop\vs code\matlab\la\matrixoperations.mlx

displayofvectors.mlx

define_matrix.mlx

matrixoperations.mlx

lineareq.mlx

orthogonal.mlx

unitcircle.mlx

Workspace

Name	Value

```

1  B=rand(2)           % 2x2 matrix with random values (uniform dist uion)
2  C=rand(4,1)         % 4x1 column vector with random values
3  D=randn(5,6)        % 5x6 matrix with normally distributed random numbe
4
5  zeros(2)            % 2x2 matrix filled with zeros
6  zeros(2,3)          % 2x3 matrix filled with zeros
7  ones(3)             % 3x3 matrix filled with ones
8  ones(4,3)           % 4x3 matrix filled with ones
9  eye(2,3)            % 2x3 Identity matrix (as much as possible)
10 A = [1 2 3; 4 5 6; 7 8 9] % 3x3 matrix
11 A(1,2)              % Element at row 1, column 2--> 2
12 A(end,2)            % Last row, column 2--> 8
13 A(1,end)            % Row 1, last column 3
14 A(end,end)          % Last row, last column + 9
15 g= A(end-1, end-2)  % Second last row, first column 4
16 C = [1 2 3 9; 4 5 6 8; 7 8 9 7]
17 C=rand(3,4)% Overwrites C with a 3x4 randomyretrix
18 C(:,1)              % All rows, first column (column vector)
19 C(2,:)              % Second row, all columns (row vector)
20 C=[
21     1 2 3 9;
22     4 5 6 8;
23     7 8 9 7;
24     2 3 4 5;
25     6 7 8 9;
26     1 2 5 7
27 ]
28 D=rand(5,4)          % New random 5x4 matrix
29 D(1:3,:)             % First 3 rows, all columns
30 D(2:3,:)             % Rows 2 to 3, all columns of matrix D
31 D(:,2:3)            % All rows, columns 2 to 3 of matrix D
32 F=rand(5,1)          % 5x1 column vector with random values
33 F(2,1)              % Value at 2nd row, 1st column (basically F(2)
34 F(2,end)            % Same as above end is 1 here, so F(2,1)
35 F(2:4)              % Elements from 2nd to 4th row
36 F(3)+4              % Assigns value 4 to the 3rd element of F
37 A                   % Display current matrix A
38 A(2,3)=7            % Change the element at row 2.col 3 to 7
39 v=[1;2;3;4;5]       % Column vector
40 v(2)                % Second element-->2
41 v(3,end)            % 3rd row, last column (also element 3)3
42 v                   % View current vector
43 v(3)=6              % Modify 3rd element to 6
44

```

B = 2x2

0.8147	0.1278
0.9058	0.9134

C = 4x1

0.6324
0.8975
0.2785
0.5469

D = 5x6

3.5784	-0.0631	1.4890	1.6302 ...
2.7694	0.7147	1.4172	0.4889
-1.3409	-0.2050	0.6715	1.9347
0.3049	-0.1241	-1.2075	0.7269
0.7254	1.4897	0.7172	-0.3034

ans = 2x2

0	0
0	0

ans = 2x3

0	0	0
0	0	0

ans = 3x3

1	1	1
1	1	1
1	1	1

ans = 4x3

1	1	1
1	1	1
1	1	1
1	1	1

ans = 2x3

1	0	0
0	1	0

A = 3x3

1	2	3
4	5	6
7	8	9

ans = 2

ans = 8

ans = 3

ans = 9

Command Window

Zoom: 80%

UTF-8

LF

script

Ln 1 Col 1

14:42

18-05-2025

```

33 F(2,1) %Value at 2nd row, 1st column (basically F(2)
34 F(2,end) %Same as above end is 1 here, so F(2,1)
35 F(2:4) % Elements from 2nd to 4th row
36 F(3)=4 %Assigns value 4 to the 3rd element of F
37 A % Display current matrix A
38 A(2,3)=7 % Change the element at row 2.col 3 to 7
39 v=[1;2;3;4;5] % Column vector
40 v(2) %Second element-->2
41 v(3,end) % 3rd row, last column (also element 3)3
42 v % View current vector
43 v(3)=6 % Modify 3rd element to 6
44 v=(2:6) % Redefine v as row vector [2 3 4 5 6]
45 v+2 % Add 2 to every element: [4 5 6 7 8]
46 v*2 % Same as above for row vector: [4 6 8 10 12]
47 P=rand(2,3) % 2x3 matrix
48 Q=rand(3,2) % 3x2 matrix
49 P*Q % Matrix multiplication result is 2x2
50 u=2:5 % Row vector[2 3 4 5]
51 v=3:6 % Row vector [3 4 5 6]
52 u.*v % Element-wise multiplication: [6 12 20 30]
53 A %Display matrix A again
54 size(A) % Returns [3 3] if A is a 3x3 matrix
55 size(A,1) %Number of rows 3
56 size(A,2) %Number of columns 3 (yes, same as above in this case)

```

4. Write a MATLAB code to find the solutions of given linear system of equations.

```

1 %Euclidean form of matrix
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 B=equationsToMatrix([eqn1,eqn2, eqn3], [x,y,z])
8 R=rref(B)
9
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 [A,b]=equationsToMatrix([eqn1,eqn2,eqn3], [x,y,z]);
16 sol=linsolve(A,b)
17
18 syms x y z
19 eqn1=2*x+y+z==6;
20 eqn2=-x+y+z==4;
21 [A,b]=equationsToMatrix([eqn1,eqn2],[x,y,z]);
22 sol=linsolve(A,b)
23
24 syms x y z
25 eqn1=2*x+y+z==6;
26 eqn2=-3*x+y+4*z==4;
27 A
28 b
29 [A,b]=equationsToMatrix([eqn1,eqn2],[x,y,z]); sol=linsolve(A,b)
30
31 %using solve command
32 syms x y
33 eqn1=2*x+y==6;
34 eqn2=-3*x+y+4*z==4; sol=solve([eqn1, eqn2],[x,y])
35 x=sol.x
36
37 syms x y z
38 eqn1=2*x+y+z==6;
39 eqn2=-3*x+y+4*z==4;
40 sol=solve([eqn1,eqn2],[x,y,z])
41 x=sol.x
42
43 %using \ operator
44 syms x y z

```

Workspace:

Name	Value
eqn3	$x + 2y + 3z = 10$
B	$\begin{pmatrix} 2 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$
R	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
eqn3	$x + 2y + 3z = -10$
sol	$\begin{pmatrix} 3 \\ 1 \\ -5 \end{pmatrix}$
Warning: symbolic\mldivide:RankDeficientSystem#Solution is not unique because the system is rank-deficient.#	
sol	$\begin{pmatrix} 2 \\ 3 \\ 14 \\ 3 \\ 0 \end{pmatrix}$
A	$\begin{pmatrix} 2 & 1 & 1 \\ -1 & 1 & -1 \end{pmatrix}$
b	$\begin{pmatrix} 6 \\ 4 \end{pmatrix}$
Warning: symbolic\mldivide:RankDeficientSystem#Solution is not unique because the system is rank-deficient.#	

```

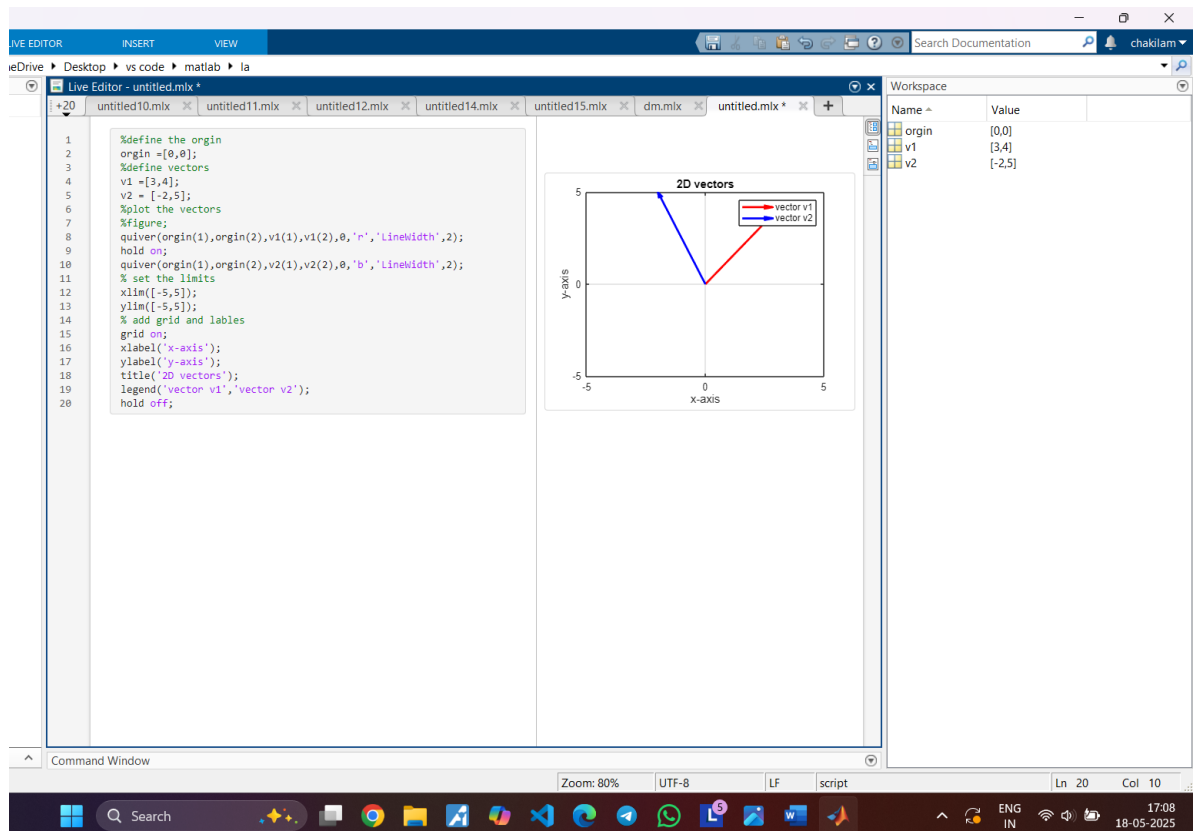
30 %using solve command
31 syms x y
32 eqn1=2*x+y==6;
33 eqn2=-3*x+y+4*z==4; sol=solve([eqn1, eqn2],[x,y])
34 x=sol.x
35
36 syms x y z
37 eqn1=2*x+y+z==6;
38 eqn2=-3*x+y+4*z==4;
39 sol=solve([eqn1,eqn2],[x,y,z])
40 x=sol.x
41
42 %using \ operator
43 syms x y z
44 eqn1 = x + 2*y - z == 5;
45 eqn2 = 3*x - y + 4*z == 10;
46 A
47 B
48 [A,b]=equationsToMatrix([eqn1,eqn2],[x,y,z]);
49 sol=A\b
50 %parametric equation
51 syms x y z
52 eqn1=x+y+z==3;
53 eqn2=2*x+2*y+2*z==6;
54
55 sol=solve([eqn1,eqn2], [x,y, z])
56
57 syms x v z
58
59 eqn1=2*x+y+z==2;
60 eqn2=2*x+2*y+2*z==6;
61 sol=solve([eqn1, eqn2], [x,y, z])
62
63 syms x y z
64 eqn1=2*x+y+z==2;
65 eqn2=-x+y-z==3;
66 sol=solve([eqn1,eqn2], [x,y,z], 'ReturnConditions', true)
67
68 syms x y z
69 eqn1 = x + 2*y - z == 5;
70 eqn2 = 3*x - y + 4*z == 10;
71 sol=solve([eqn1,eqn2], [x,y,z])
72 sol=solve([eqn1,eqn2],[x,y,z], 'ReturnConditions', true)

```

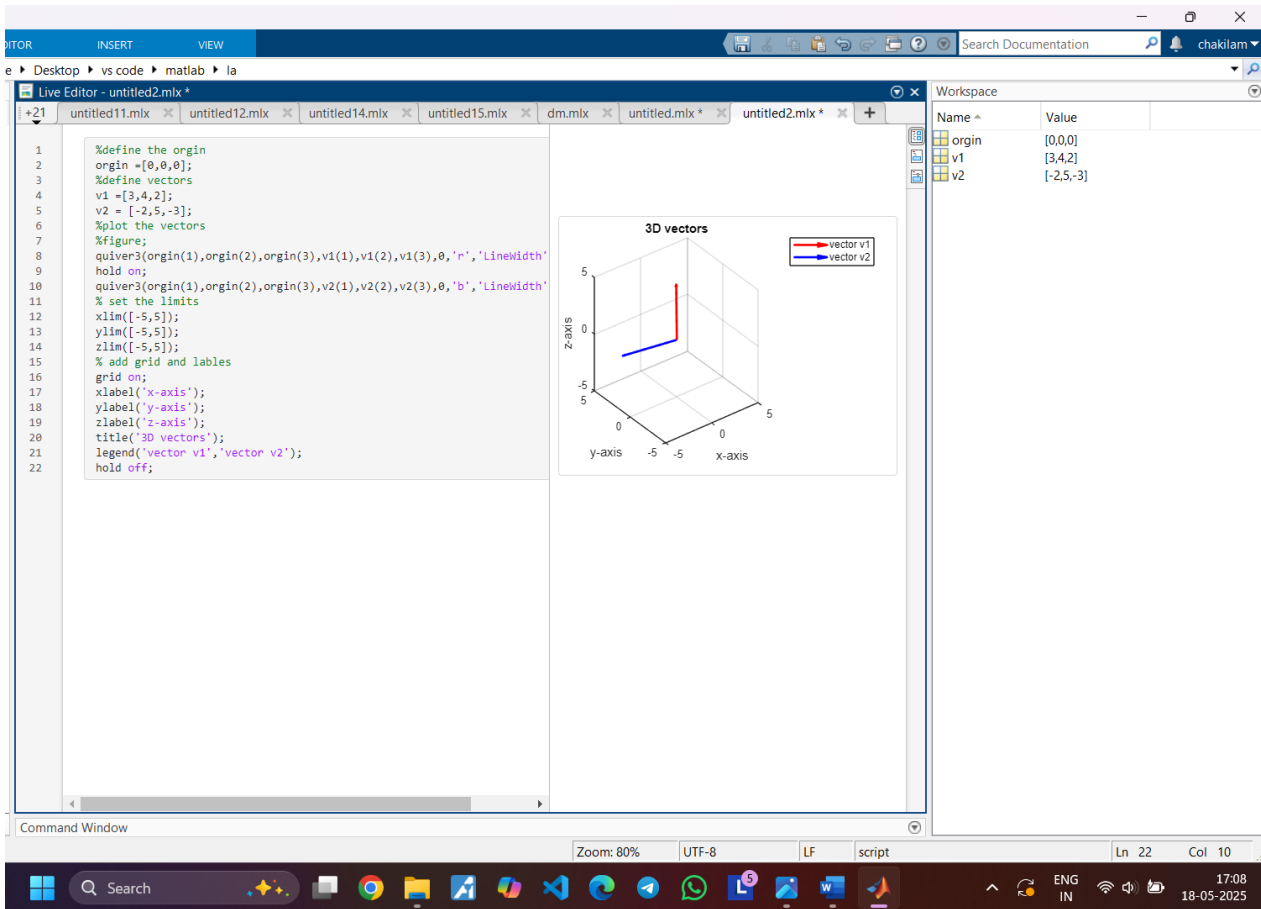
Workspace:

Name	Value
ans	$\begin{pmatrix} -38 \\ -33 \\ 76 \\ 33 \\ 0 \end{pmatrix}$

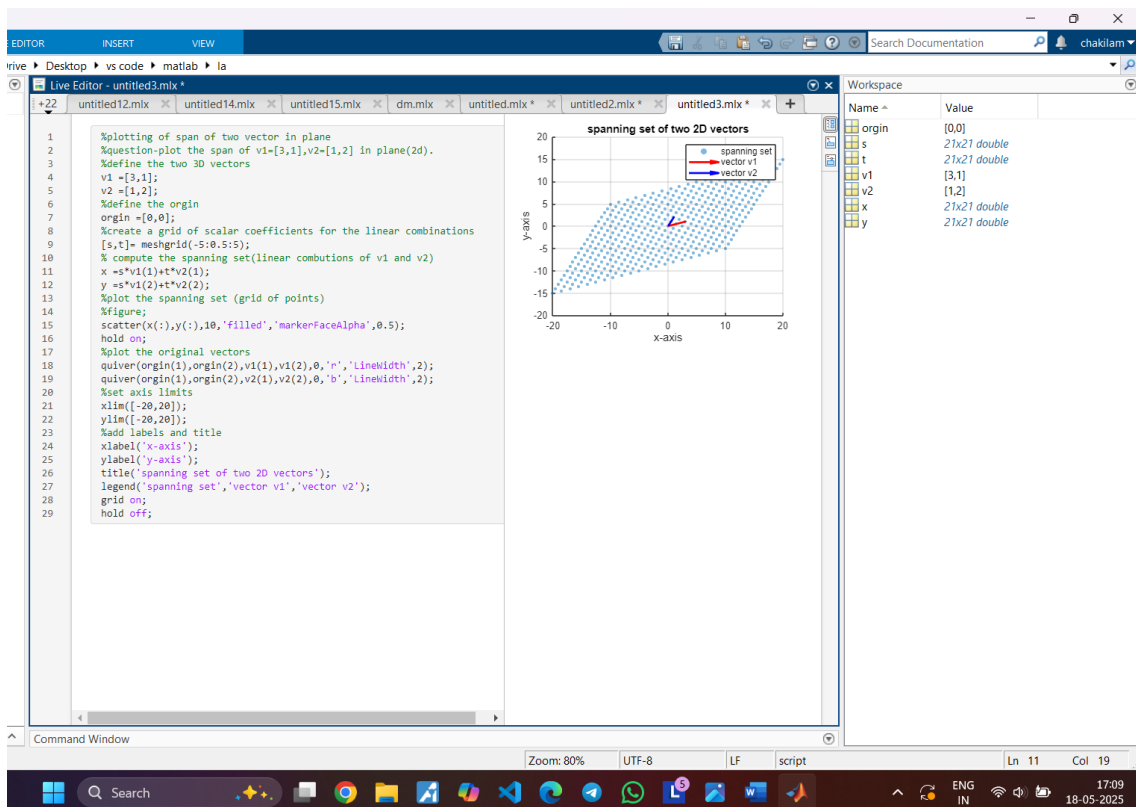
5. Write a MATLAB code to draw the vectors (3,4) and (-2,5) in a 2-D plane.



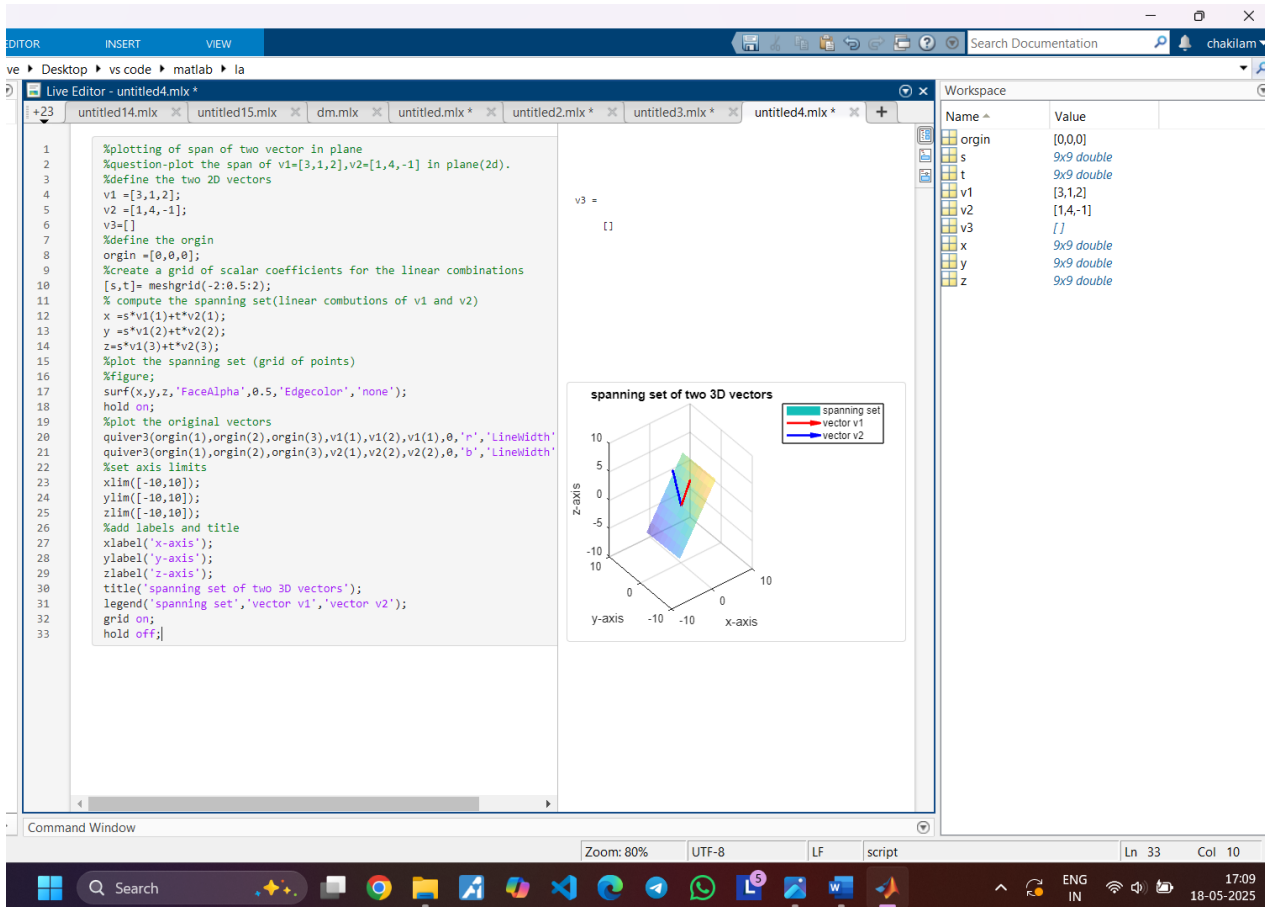
6. Write a MATLAB code to draw the vectors $(3,4,0)$ and $(-2,5,0)$ in a 3-D plane.



7. Using MATLAB, define the vectors $v1 = [3, 1]$ and $v2 = [1, 2]$. Write a program to visualize their span in 2D space.



8. Using MATLAB, define the vectors $v_1=(3,1,2)$ and $v_2=(1,4,-1)$. Write a program to visualize their span in 3D space.



9. Using MATLAB, for any given positive integer and for any given basis for \mathbb{R}^n . write a program to find the orthonormal basis corresponding to the given basis.

The screenshot shows a MATLAB Live Editor window with a script titled 'untitled15.mlx'. The script implements the Gram-Schmidt process to find an orthonormal basis for a given set of basis vectors. The workspace on the right shows the variables defined in the script.

Script Code:

```

1 |
2 v1=[1;1;0];
3 v2=[0;1;1];
4 v3=[1;0;1];
5 % Combine into a matrix where each column is a basic vector
6 A = [v1 v2 v3];
7 fprintf('Original Basis Vectors:\n');
8 disp(A);
9 % Get size
10 [m, n] = size(A);
11 % Initialize Q and R
12 Q= zeros(m, n);
13 % Gram-Schmidt process
14 for k = 1:n
15     v = A(:, k);
16     for j = 1:k-1
17         q=Q(:,j);
18         v = v - (q'*v)/(q'*q)*q;
19     end
20     Q(:, k) = v;
21 end
22 % Display Q and R
23 disp(Q);
24 fprintf('\nOrthogonal check (Q.T q):\n');
25 ortho_check=Q'*Q;
26 disp(ortho_check);
27 fprintf('\nOrthogonal basis vector:\n');
28 Q_normalized=Q./ vecnorm(Q);
29 disp(Q_normalized);
30 %Verification
31 fprintf('\nOrthogonal check (Q_normalized" Q_normalized):\n');
32 disp(Q_normalized'*Q_normalized)

```

Workspace Variables:

Name	Value
origin	[0,0,0]
s	9x9 double
t	9x9 double
v1	[3,1,2]
v2	[1,4,-1]
v3	[]
x	9x9 double
y	9x9 double
z	9x9 double

Output Results:

Original Basis Vectors:

```

1 0 0
0 1 0
1 1 1

```

Orthogonal check (Q.T q):

```

2.0000 0 0
0 1.5000 0.0000
0 0.0000 0.3333

```

Orthogonal basis vector:

```

0.7071 -0.4082 -0.5774
0 0.8165 -0.5774
0.7071 0.4082 0.5774

```

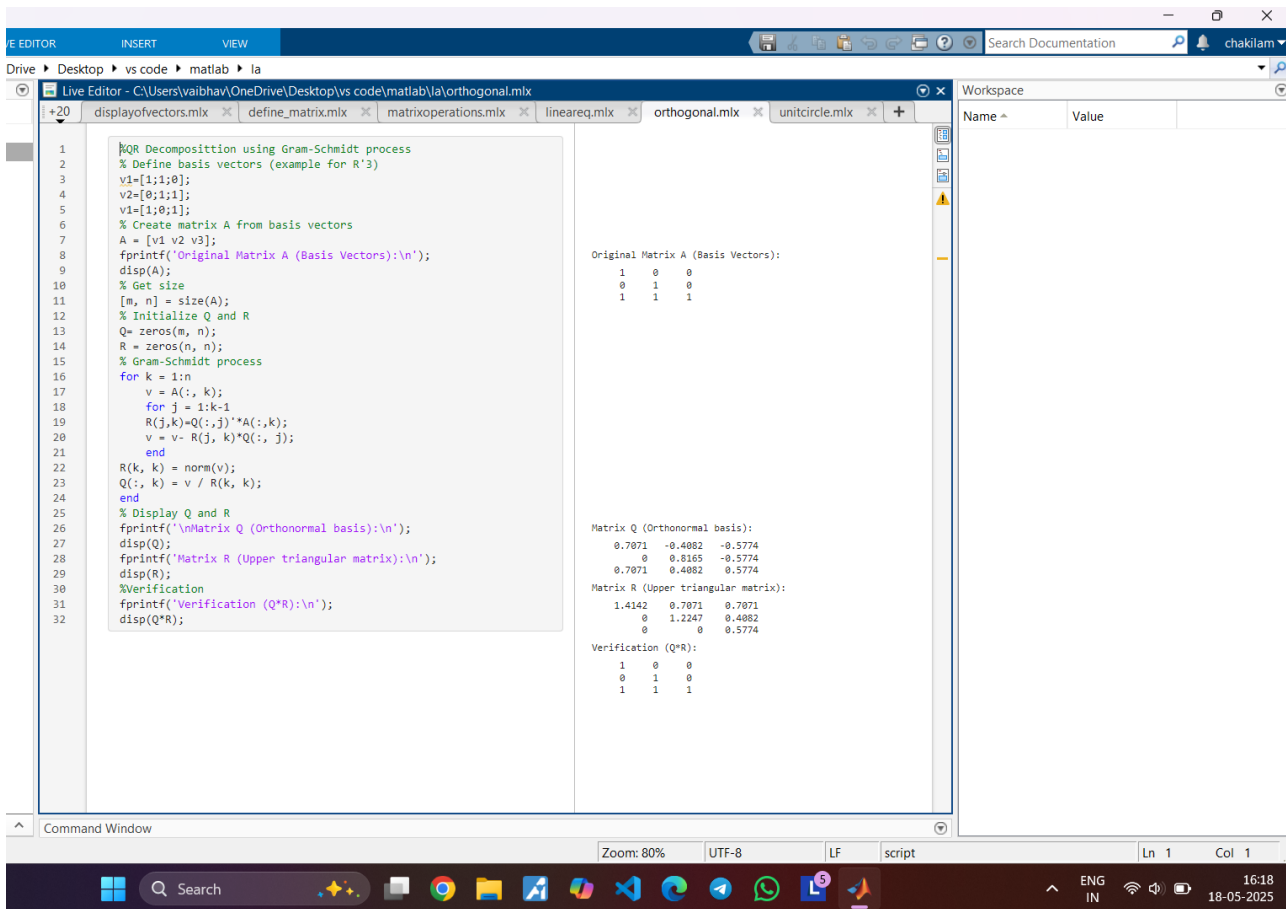
Orthogonal check (Q_normalized" Q_normalized):

```

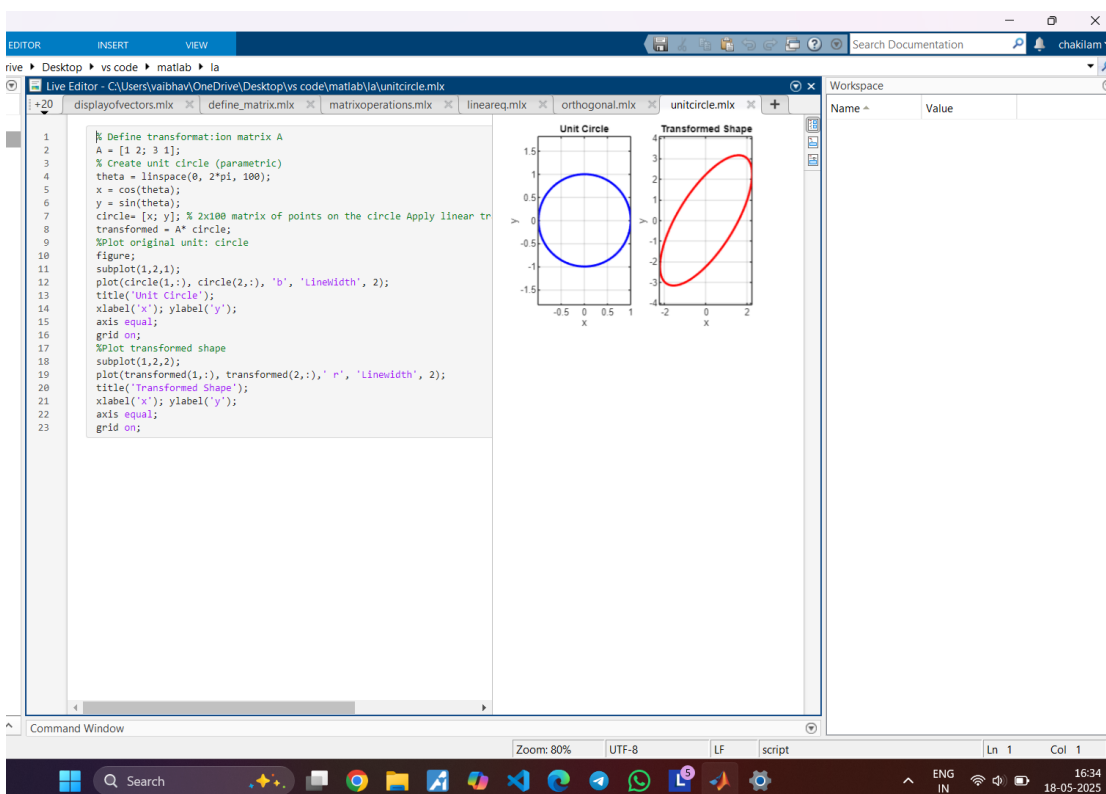
0.0918 -0.8577 -0.5059
-0.4082 0.4310 -0.8047
0.9082 0.2884 -0.3106

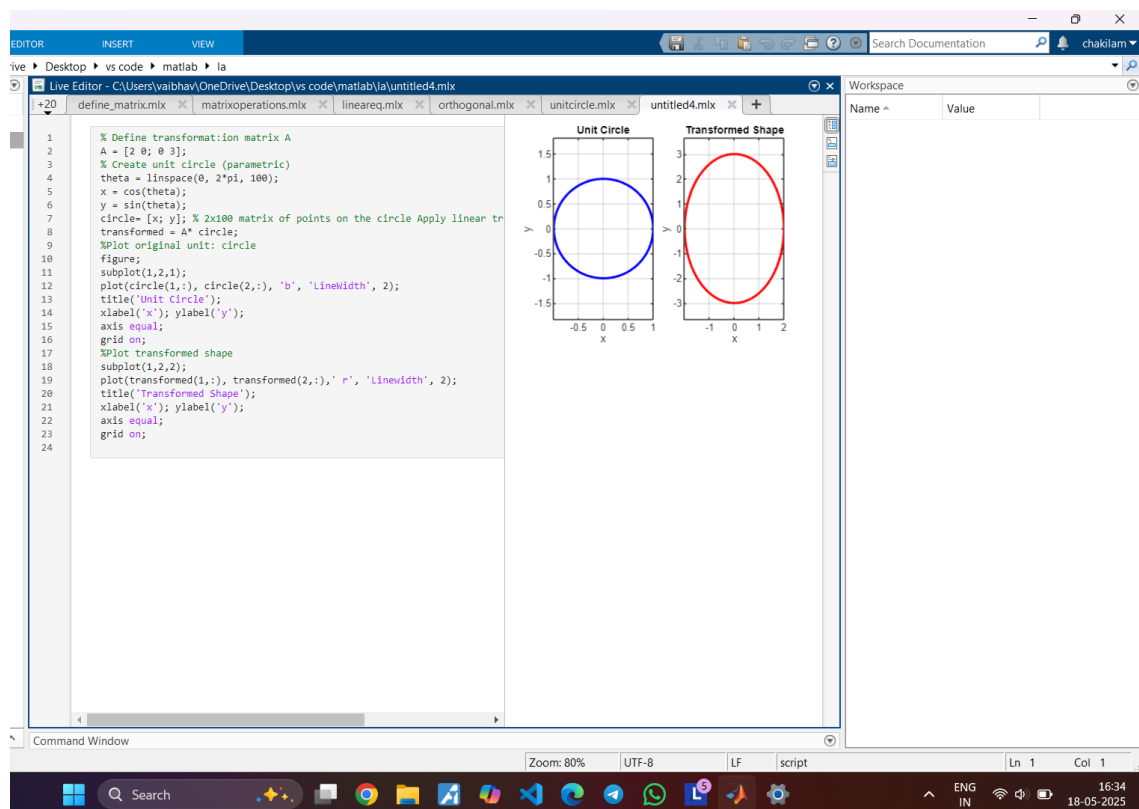
```

10. Using MATLAB, write a program to find the QR decomposition corresponding to the give

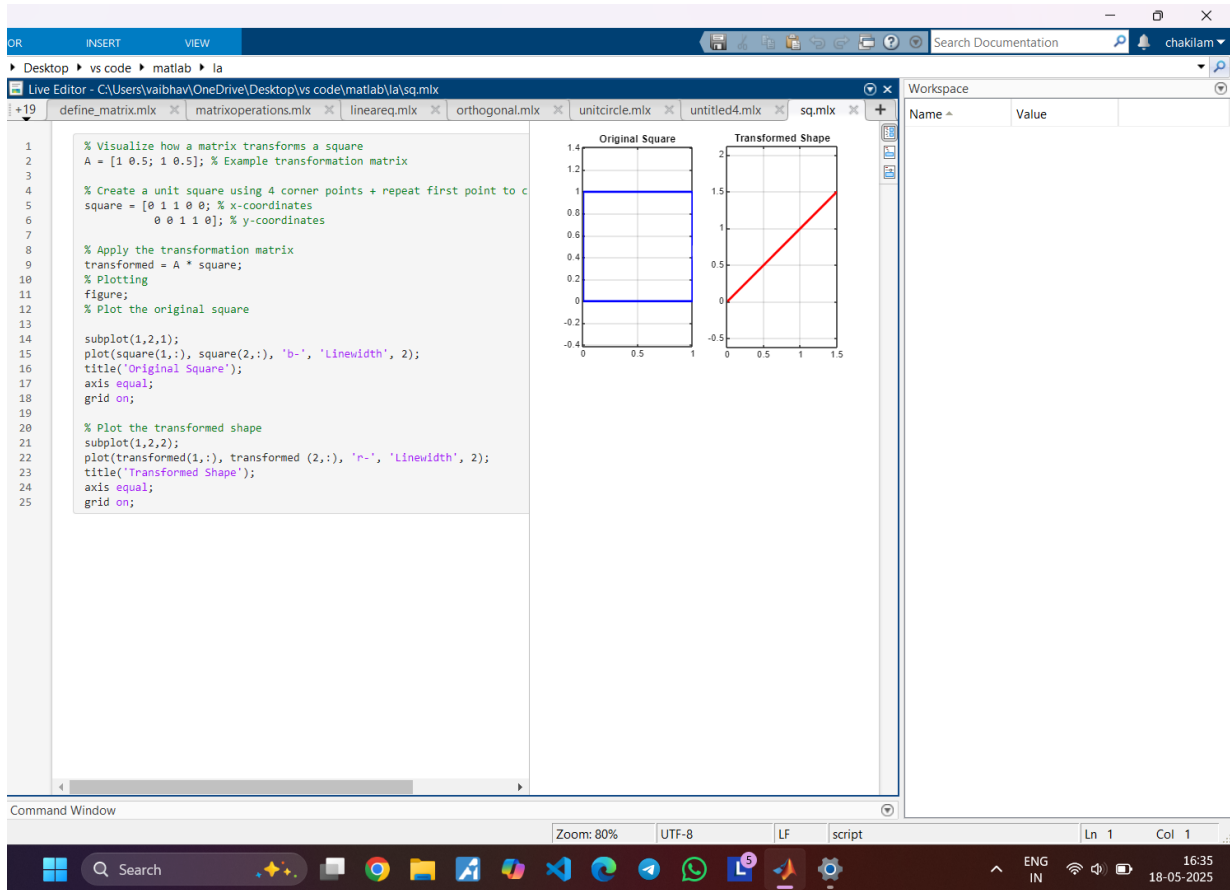


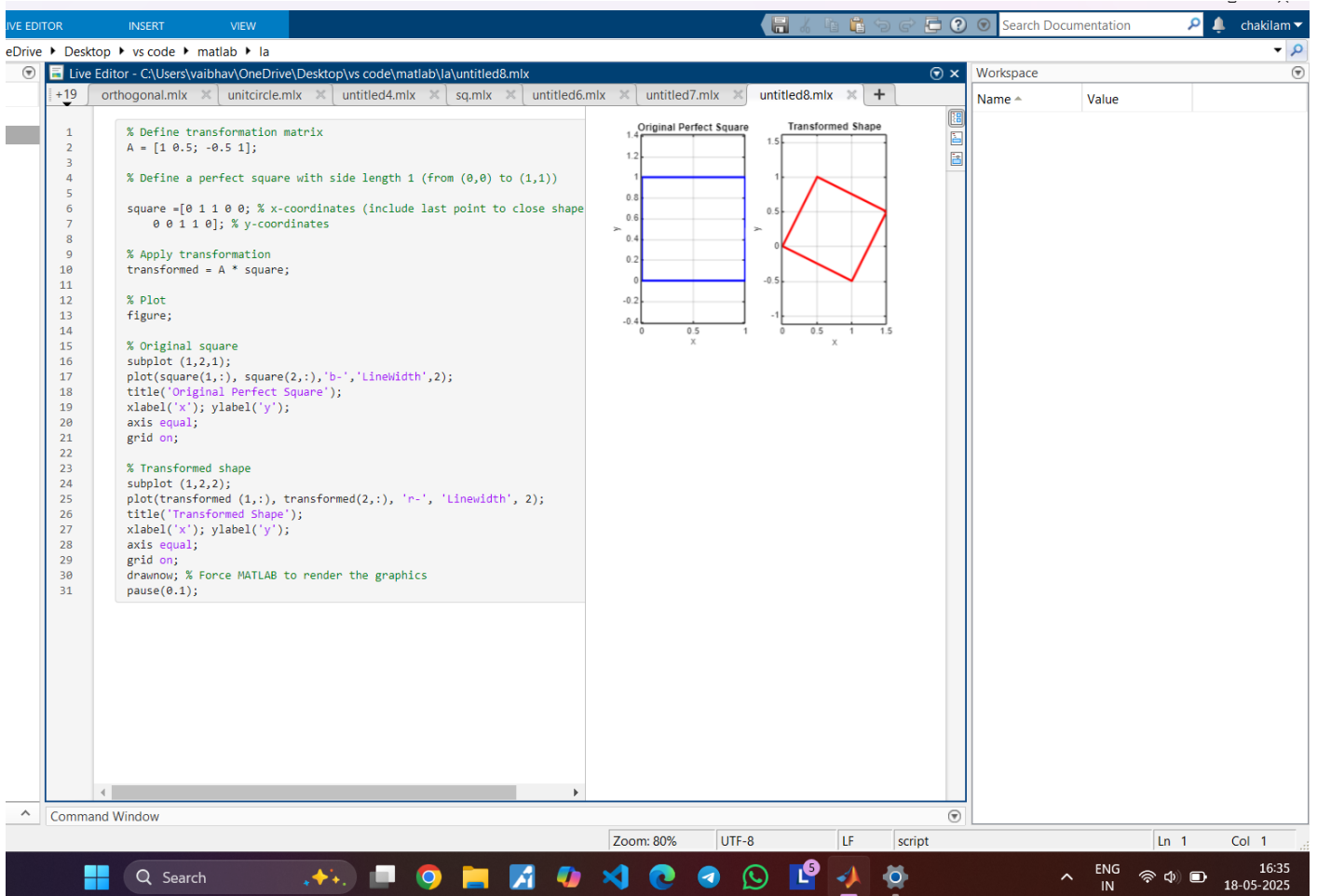
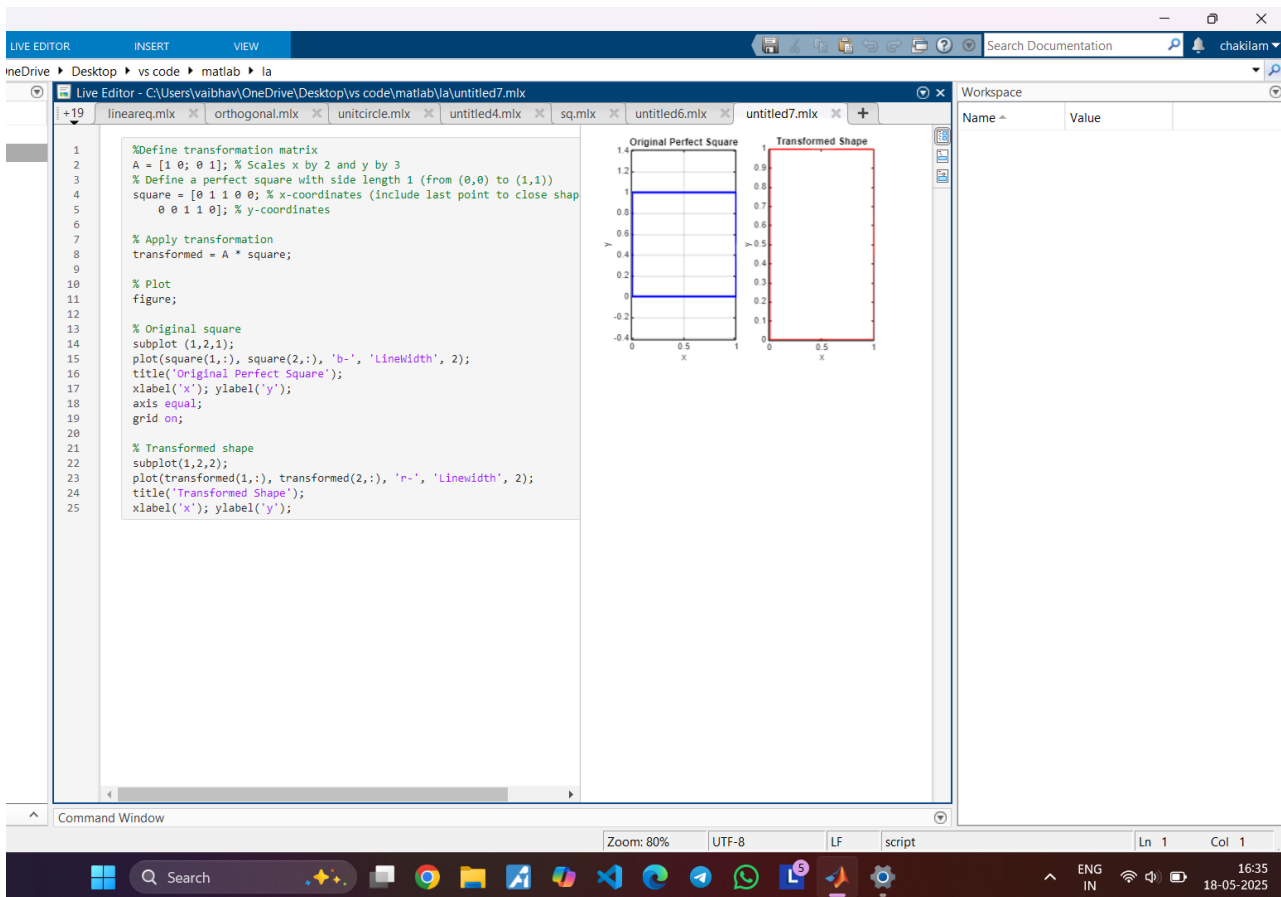
11. Using MATLAB, write a program to Visualize how a matrix transforms the unit circle.

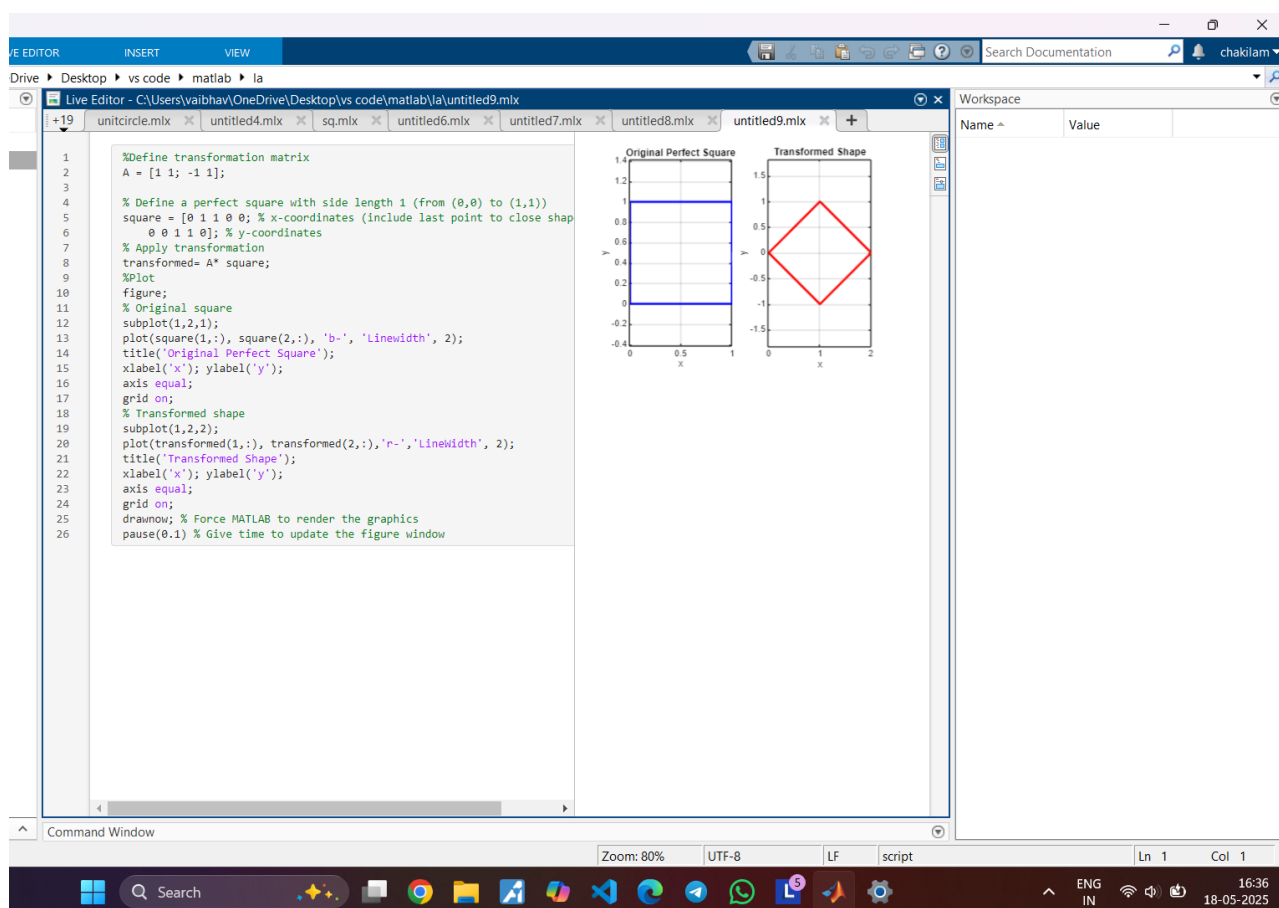




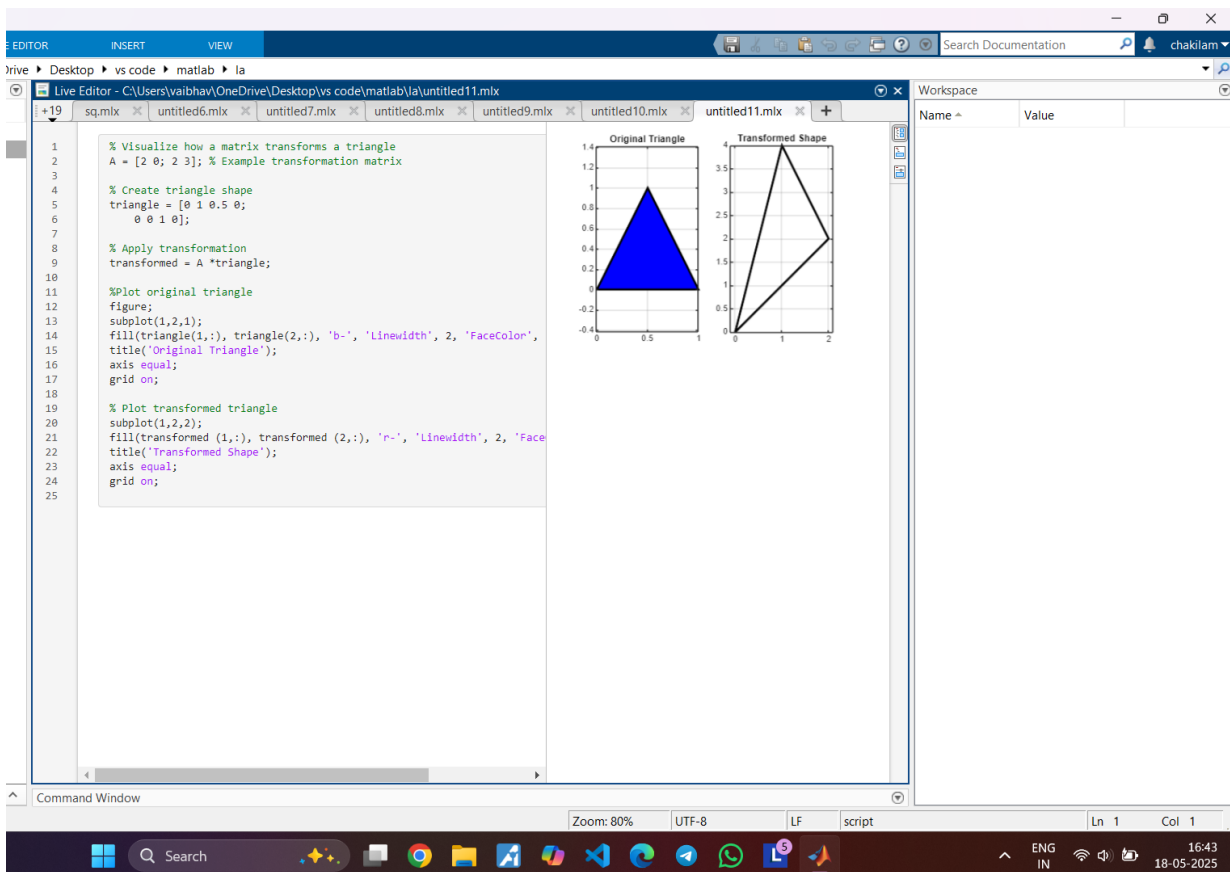
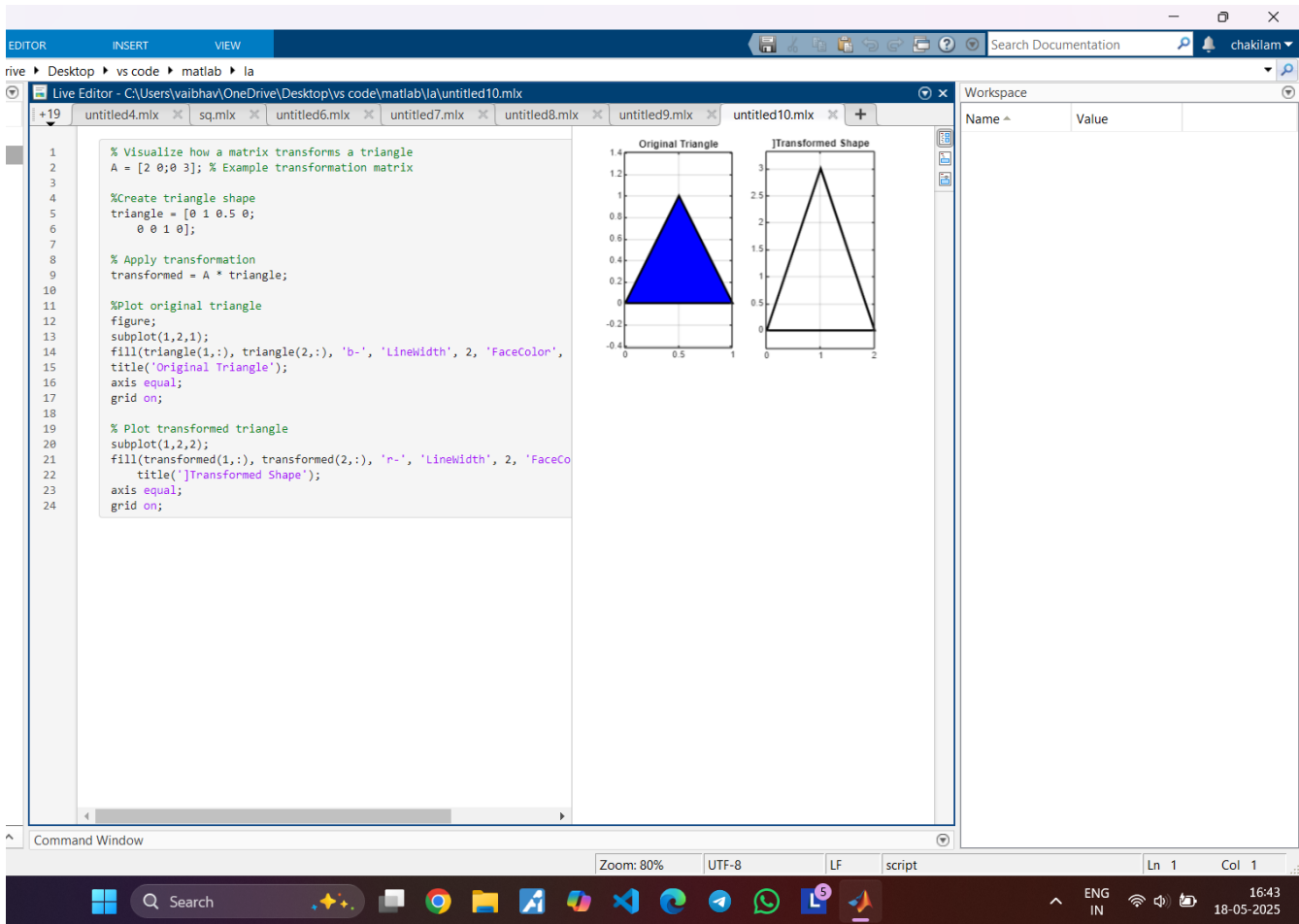
12. Using MATLAB, write a program to Visualize how a matrix transforms a square.







13. Using MATLAB, write a program to Visualize how a matrix transforms a triangle.



14. Using MATLAB, write a program to find diagonalisation of the matrix.

```

1 % diagonalisation of the matrix
2 v1 = [1; 2; 3];
3 v2 = [0; 2; 1];
4 v3 = [0; 0; 1];
5 A = [v1 v2 v3];
6 [m, n] = size(A);
7 if m ~= n
8     error('Matrix must be Square for diagonalisation');
9 end
10 [p, d] = eig(A);
11 eigenvalues = diag(d);
12 disp('eigenvalues');
13 disp(eigenvalues);
14 disp('Eigenvectors(columns of p):');
15 disp(p);
16 if rank(p) == m
17     disp('Matrix is diagonalisable');
18
19     disp('Matrix P (Eigenvectors):');
20 disp(p);
21 disp('Matrix is diagonalisable');
22 disp('Matrix P (Eigenvectors):');
23 disp(p);
24 disp('inverse of p');
25 invp = inv(p);
26 disp(invp);
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 diagonalisable');
32 end
33

```

Workspace

Name	Value
eigenvalues	1 2 1
Eigenvectors(columns of p):	0 0 0.0000 0 0.7071 -0.0000 1.0000 0.7071 -1.0000

Matrix is not diagonalisable

15. Using MATLAB, write a program to find orthogonal diagonalisation of a given matrix.

```

1 % Define the symmetric matrix A
2 A = [4, 1, 2;
3      1, 3, 0;
4      2, 0, 5];
5 % Check if A is square
6
7 [m, n] = size(A);
8 if m ~= n
9     error('Matrix must be square for diagonalization. ');
10 end
11 % Compute eigenvalues and eigenvectors
12 [P, D] = eig(A);
13 eigenvalues = diag(D);
14 % Display eigenvalues and eigenvectors
15 disp('Eigenvalues:');
16 disp(eigenvalues);
17 disp('Eigenvectors (columns of P):');
18 disp(P);
19 % Check orthogonality of P using P' * P == I
20 if isequal(round(P'*P, 10), eye(size(P)))
21     disp('Matrix is orthogonally diagonalizable. ');
22     disp('Orthogonal matrix P:');
23     disp(P);
24
25     disp('Diagonal matrix D:');
26     disp(D);
27     % Reconstruct A using A = P * D * P'
28     A_reconstructed = P*D*P';
29     disp('Reconstructed A from P* D * P\'');
30     disp(A_reconstructed);
31 else
32     disp('Matrix is NOT orthogonally diagonalizable (P is not orthogonal).');
33 end
34

```

Workspace

Name	Value
Eigenvalues:	1.8549 3.4760 6.6691
Eigenvectors (columns of P):	-0.6793 -0.3744 0.6312 0.5932 -0.7864 0.1720 0.4320 0.4913 0.7563
Matrix is orthogonally diagonalizable.	
Orthogonal matrix P:	-0.6793 -0.3744 0.6312 0.5932 -0.7864 0.1720 0.4320 0.4913 0.7563
Diagonal matrix D:	1.8549 0 0 0 3.4760 0 0 0 6.6691
Reconstructed A from P* D * P\'	4.0000 1.0000 2.0000 1.0000 3.0000 -0.0000 2.0000 -0.0000 5.0000