```
%% BASIS
%A combination of all the linearly independent vectors that can span an
%entire space
%A set of vectors that generates all elements of the vector space and the
%vectors in the set are linearly independent

clc
A1 =reshape(1:9, 3,3)';
%The vectors corresponfing to the pivot columns in the row reduced echelon
%form are linearly independent and form a basis
R1 = rref(A1);
%The pivot columns in R1 will corresponfd to the basis vectors
basis =A1(:, 1:2);
%Checking whether the vectors span the column space
B1 = A1(:,1);
B2 = A1(:,2);
v = A1(:,3); %this is the third column which should be a linear combination of b1 and
b2

coefficients = [B1 B2] \ v      % v = coefficients(1) * b1 + coefficients(2) * b2


%NB: the first two columns of A1 form a basis for the column space as they
%are linearly independent. The 3rd column can be expressed as a combination
%of these two


% INNER PRODUCT (Mathematical operation that takes two vectors and returns
% a scalar(dot product) , cummulative, distributive and associative
%a.b=mag(a)mag(b)cos(tetha)
%Finding the component of one vetor in direction of another
clc
a1 = [1,2,3];
b1 = [4,5,6];

inner_product = dot(a2,b2);

%Using matrix multiplication
innerProduct = a1 * b1';   %b1 is gtransposed for correct dimension

%% exercise 2
%COMPRESSION: Reducing the size of data while maintaining as much important
%information as possible
clc
```

```
A2  = magic(6);
%perform singular value decomposition
[U, S, V] = svd(A2); %S whose diagonals are singular values, U whose columns are
the left singular vectors, V' whose columns are right singular vectors
k = 3;  %Choose a reduced number of singular values(rank-k approximation)
%Compress the matrix by truncating S,U and V
A2Compressed = U(:, 1:k) *S(1:k, 1:k) * V(:, 1:k)';

%Calculation of compression error
compression_error = norm(A2 - A2Compressed, 'fro');   %Difference between the
matrices elementwise

%TRANSFORMATION
%process of mapping/changing one vector space into another using a matrix
%or a function i.e y=Ax (matrix transformation of x by A where y is the
%transformed vector

% Define a matrix A
At = [2 0; 0 3];

% Define a vector x
x = [1; 1];

% Apply the transformation
y = At * x;

disp('Transformed Vector:');
disp(y);


%CLUSTERING
%grouping data points into clusters based on their similarity

% Generate sample data
rng(1); % For reproducibility
data = [randn(100,2) + 2; randn(100,2) - 2; randn(100,2) + [2 -2]];

% Number of clusters
k = 3;

% Perform K-Means clustering
[idx, C] = kmeans(data, k);

% Plot the clustered data
```

```
figure;
gscatter(data(:,1), data(:,2), idx, 'rgb', 'o', 8);
hold on;
plot(C(:,1), C(:,2), 'kx', 'MarkerSize', 12, 'LineWidth', 2);
title('K-Means Clustering');
xlabel('Feature 1');
ylabel('Feature 2');
legend('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroids');
hold off;

%% exercise 3
%SUBDIAGONAL
%diagonals of a matrix that are parallel to the main diagonal but are
%located below it
clc
A3 = reshape(1:16, 4, 4)'

diagonal = diag(A3);
subdiag1 = diag(A3, -1);
subdiag3 = diag(A3, -3);

supdiag1 = diag(A3, 1);
supdiag2 = diag(A3, 2);

%% Exercise 4
clc
% Define/match the following diagonal patterns
A4 = reshape(1:9, 3, 3)';

% elements a(i,j) with i=j
%Refers to the elements of the main diagonal of a matrix
diag(A4);

% kth subdiagonal of A
%This refres to the diagonal of elements where i=j+k, if k=0(main
%diagonal), if k=1(first lower diagonal) and k=2(2nd subdiagonal)
%For the kth superdiagonal, k diagonals above the main, k=1(i=j-1, first
%superdiagonal
first_subdiag = diag(A4, -1); %k=1
fisrt_superdiag = diag(A4, 1);

% elements of a(i,j) with i \= j+k
% This refers to the elements not on the kth diagonal. For example:
% When k=0k=0, this refers to all off-diagonal elements where i≠ji□=j.
```

```
% When k=1k=1, this refers to all elements not on the 1st subdiagonal.
off_diag_elements = A4 - diag(diag(A4));  % Removing main diagonal elements


% cross diagonal of A(Anti-diagonal of A)
% consists of elements from the top-right corner to the bottom-left corner. they
satisfy the condition i+j=n+1 where n is the matrix dimension
cross_diag = diag(flipud(A4));

% A is banded
%matrix that has non-zero elements concentrated around the main diagonal, with all
other elements zero.
% The number of non-zero diagonals around the main diagonal determines the
bandwidth of the matrix.
% Example of a banded matrix with 1 superdiagonal and 1 subdiagonal


%% Exercise 5
clc
% Define/match the following diagonal patterns
%diagonal of A = elements a(i,j) with i=j
%This refers to the main diagonal of a matrix where the row index i equals the
column index j i.e a11, a22, a33
d5 = [1, 2, 3, 4, 5];
A5_diag = diag(d5);
disp(A5_diag);

% nonzero elements a(i,j) satisfy abs(i-j) <= 1
% This describes a tridiagonal matrix where non-zero elements exist only on the main
diagonal, the first superdiagonal and the first subdiagonal
% Define diagonals
main_diag = [1, 2, 3, 4, 5];
upper_diag = [6, 7, 8, 9];
lower_diag = [10, 11, 12, 13];

% Create the tridiagonal matrix
A5_tridiag = diag(main_diag) + diag(upper_diag, 1) + diag(lower_diag, -1);

% A is upper bidiagonal has non-zero elements on the main diagonal and the
% first upperdiagonal but zeros elsewhere Define main and upper diagonal
maindiag = [1, 2, 3, 4, 5];
upperdiag = [6, 7, 8, 9];

% Create the upper bidiagonal matrix
```

```
A5upperbidiag = diag(maindiag) + diag(upperdiag, 1)


% A is upper Hessenberg matrix where all the elements below the first subdiagonal are zero


mainDiag = [1, 2, 3, 4, 5];
upperDiag = [6, 7, 8, 9];
lowerDiag = [10, 11, 12, 13];

% Create the upper Hessenberg matrix
A5upperhessenberg = diag(mainDiag) + diag(upperDiag, 1) + diag(lowerDiag, -1)

%% Exercise 6
clc
% Define/match the following zero patterns

% elements a(i,j)(value at the ith row and jth column in matrix a) that satisfy i>j must be zero
% describes an upper triangular matrix where all the elements below the main diagonal are zero
% This is because a(i,j) for i>0 hall have zeros hence creating an upper triangular matrix because all the elements below the main diagonal are zero
A6 = reshape(1:9, 3, 3)';
A6upperTriangular = triu(A6);

% elements a(i,j) that satisfy i<j must be zero
%describea a lower triangular matrix where all the elements above the main
%diagonal are zero (The non-zero elements are these below the main diagonal
A6lowerTriangular = tril(A6);

% A is quasi-diagonal
%Refers to a matrix that is almost diagonal but allows some non-zero
%elements in certain off-diagonal positions such as small blocks near the
%diagonal (have BLOCKS of non-zero elements along the diagonal and zero
%elsewhere
b1q = [1 2; 3 4];
b2q = [5 6; 7 8];

Bq = blkdiag(b1q, b2q);

%% Exercise 7
clc
```

```
% Define/match the following zero pattern
% A is block upper triangular = If a(i1,j1) and a(i2,j2) belong to different blocks with
i1<i2, then a(i2,j1) is zero
% Block matrices
A1 = [1 2; 3 4]; % First block
A2 = [5 6; 7 8]; % Second block
A3 = [9 10; 11 12]; % Third block

% Block upper triangular matrix
A7 = [A1, zeros(2,2), zeros(2,2);  % First row of blocks
    zeros(2,2), A2, zeros(2,2);  % Second row of blocks
    zeros(2,2), zeros(2,2), A3];  % Third row of blocks


% A is row echelon
%A matrix is in row echelon form if, any row consisting entirely of zeros
%appears at the bottom of the matrix, the leading entry of each non-zero
%row is strictly to the right of the leading entry of the row above it and
%if the leading entry in any non-zero row is 1 and the columns below each
%pivot contain zeros
% Creating a matrix in row echelon form
B7 = [1 2 3 4; 0 1 5 6; 0 0 0 7; 0 0 0 0]



%% Exercise 8
clc
%
% True or False
% Jordan Block = A Toeplitz bidiagonal matrix with 1's on the superdiaognal

%A jordan block is a special form of a square matrix where the matrix is
%upper triangular with all diagonal elements equal and ones on the
%superdiagonal
lambda = 3; % Eigenvalue
n = 4; % Size of the Jordan block
J = lambda * eye(n) + diag(ones(n-1,1), 1)


%A toeplitz matrix has constant diagonals
%False

% Jordan Form = A block diagonal matrix with Jordan blocks on the diagonal
%The jordan form of a matrix is a block diagonal matrix where each block is
```

```
%a jordan block
%True
J1 = [2 1; 0 2]; % 2x2 Jordan block
J2 = 4; % 1x1 Jordan block
J = blkdiag(J1, J2) % Block diagonal matrix


% Toeplitz matrix = a matrix where any sub/super/main diagonal has all equal↙
elements
%True
c = [1 2 3]; % First row (first element is the top-left element of the matrix)
r = [1 4 5]; % First column
T = toeplitz(r, c);

% A is upper triangular = elements a(i,j) that satisfy i>j must be zero
%True
A8 = [1 2 3; 0 4 5; 0 0 6];

%% Exercise 9
clc
% Define/match the following patterns
% Hankel matrix = a matrix a matrix where any sub/super/main cross-diagonal has↙
all equal elements
%A hankel matrix is a square/rectangular matrix in which every
%anti-diagonal contains the same (Hij=hi+j-1)
% Define the first column and the last row element
c = [1; 2; 3; 4]; % First column
r = [1, 5, 6, 7]; % First row (excluding the first element)

% Create the Hankel matrix
H = hankel(c, r);


% Vandermonde
%A matrix where each row is a geometric progression of the elements of a
%given vector
v = [1 2 3 4];
V = vander(v)
% Fourier matrix = A vandermonde matrix where the elements are powers of the↙
complex roots of 1
% Define the size of the Fourier matrix
n = 4;

% Create the Fourier matrix
```

```
F = zeros(n, n);
omega = exp(-2 * pi * 1i / n); % Complex root of unity

for i = 1:n
    for j = 1:n
        F(i, j) = omega^((i-1)*(j-1));
    end
end

% Scale the matrix by 1/sqrt(n)
F = F / sqrt(n);

disp(F)

%% Exercise 10
clc
% Define/match the following
% A is Symmetric = elements a(i,j) and a(j,i) are equal for every i and j
A10 = [2 3 4; 3 5 6; 4 6 8];
issymmetric = isequal(A10, A10');

% A is anti-symmetric
    %for a matrix to be anti-symmetric or skew-metric, a(i,j) and a(j,i) must
    %be negatives of each other for all i and j and the diagonal elements must
    %be zero (a(i,j)=-a(j,i)
B10 = [0 2 -1; -2 0 3; 1 -3 0];
isAntiSymmetric = isequal(B10, -B10')

% elements a(i,j) and a(k,i) are equal if k+j=n


% A is positive definite

%% Exercise 11
 clc
% Define/match
% A is semi-definite = a symmetric matrix whose eigenvalues are non-negative
    % A matrix is semi-definite if it is symmetrics and all of its eigenvalues
    % are non-negative (positive semi-definite if all the eigenvalues are
    % greater than or equal to zero, negative semi-definite if all the
    % eigenvalues are less than or equal to zero)
A11 = [4 2 0; 2 3 0; 0 0 0];
eigValues = eig(A11);
isSemiDefinite = all(eigValues >= 0);
```

```
% The transpose of A is the same as the inverse of A
%Describes an othogonal matrix which is a matrix whose transpose is equal
%to the inverse of the matrix
B11 = reshape(1:16, 4, 4)';
t11 = B11';
isOthogonal = isequal(t11, inv(B11));

% A is a projection
%A projection matrix should satisfy A^2=A. this means that applying the
%matrix twice yields the same result as applying it once which
%geometrically represents a projection onto a subspace
C11 = [1 0 0; 0 1 0; 0 0 0];
isprojection = isequal(C11^2, C11);

% x'* A * x is positive for any vector x
%This decribes a positive definite matrix(all positive eigenvalues), x'Ax>0 for all x not
equal to 0
D11 = [2 -1 0; -1 2 -1; 0 -1 2];
eigenvalues = eig(D11);
isPositiveDefinite = all(eigenvalues > 0);

% A and B are equivalent
% Two matrices are equivalent if they represent the same linear transformations
under different bases
% i.e A and B are equivalent if there exists invertible matrices P and Q such that
B=PAQ
E11 = [1 2; 3 4];
P11 = [2 0; 0 1];  %invertible/non-singular/non-degenerate (has an inverse, square)
% The determinant should not be zero, rank should be full and none of the
eigenvalues should be zero
Q11 = [1 1; 0 1];  %Invertible matrix

B11 = P11 * E11 * Q11;

%Check if P11 and Q11 are invertible
isPInvertible = det(P11) ~= 0;
isQInvertible = det(Q11) ~= 0;
disp(isPInvertible && isQInvertible)

%% Exercise 12
clc
% Define/match the following transformations
```

```
% a)the square of A equals A
%This describes a projection matrix
A12 = [1 0 0 ; 0 1 0; 0 0 0];
isprojection = isequal(A12^2, A12);

% Example projection of a vector
v = [1; 2; 3];  % Vector to be projected
v_projected = A12 * v;
disp(v_projected);  % The result will be [1; 2; 0], projected onto the xy-plane

% b)A is rotation matrix
% A rotation matrix rotates vectors in euclidean space without changine their length
% It must be  orthogonal(the transpose is equal to the inverse) and the detreminant
of the matrix is 1

theta = pi/4;  % Rotation by 45 degrees (pi/4 radians)

% 2D rotation matrix
B12 = [cos(theta) -sin(theta);
    sin(theta) cos(theta)]  % 2D rotation matrix

% Check if A is orthogonal (A^T * A = I)
isOrthogonal = isequal(A12' * A12, eye(2));
disp(isOrthogonal)  % This should return 1 (true)

% Check if det(A) = 1 (rotation matrix determinant must be 1)
detB12 = det(B12);
disp(detB12)  % This should return 1

% Rotate a vector
v = [1; 0];  % Vector along the x-axis
v_rotated = B12 * v;
disp(v_rotated)  % This will give [sqrt(2)/2; sqrt(2)/2] (rotated 45 degrees)


% c)The angle between x and y is the same as the angle between A*x

% and A*y and det(A) = -1
% d)A and B are  similar
```

```
%BANDED MATRIX: matrix where most of the elements are zero except for those
%within a certain band around the diagonal

% Given the matrix A=reshape(1:21, 3, 7) Exercise 19
% %Write a matlab function that reads the matrix starting at the northeest
% %corner down the diagonal  then down the first superdiagonal and so forth
% %diagonals until you reach the southwest corner. you should get the
% %sequence 1, 5, 9, 4, 8
%
%
% %Step 1: Define a function read_diagonal
% function output = read_diaganol(A19)
%    A19 = reshape(1:21, 3, 7);   %Define the matrix
%    [m19, n19] = size(A19);      %Get the size of the matrix
%    result = [];                 %Initialze an empty array to store the result
%
%
% %% number 14
%
% clc
% A = toeplitz([1, 0, 0, 0, 5], [0, 0, -3, 0, -5]);
% A(10:20);
% s = sparse(A);
% [rowIndices, colIndices, valArray] = find(s);
%
% %Calculating column pointers
% numCols = size(s,2);
% colPointers = zeros(numCols + 1, 1);
%
%
% % The for loop iterates over each column.
% % nnz(col < i) counts the number of non-zero elements in columns before column
i, effectively giving the starting index for each column in the values and row arrays.
% % The last element of colPointers is set to the total number of non-zero elements
in the matrix, which is the total length of the values array.
%
% for i = 1:numCols
%    colPointers(i)=nnz(colIndices<i);
% end
% colPointers(end) = nnz(colIndices)
%
% %% % Define the matrix
% clc
```

```
% A = [1 0 -3 0 -5;
%      0 1 0 -3 0;
%      0 0 1 0 -3;
%      0 0 0 1 0;
%      5 0 0 0 1];
%
% % Convert to sparse matrix format
% S = sparse(A);
%
% % Get the values, row indices, and column indices of non-zero elements
% [row, col, values] = find(S);
%
% % Initialize Row Pointers Array
% numRows = size(S, 1);
% rowPointers = zeros(numRows + 1, 1);
%
% % Fill the Row Pointers Array
% for i = 1:numRows
%     rowPointers(i) = nnz(row < i); % Number of elements in rows before the
current row
% end
% rowPointers(end) = nnz(row); % Total number of non-zero elements
%
% % Display the results
% disp('Values Array (V):');
% disp(values');
%
% disp('Column Indices Array (C):');
% disp(col');
%
% disp('Row Pointers Array (R):');
% disp(rowPointers);
%
% %%
%
% clc
% % Define the matrix
% A = [1 0 -3 0 -5;
%      0 1 0 -3 0;
%      0 0 1 0 -3;
%      0 0 0 1 0;
%      5 0 0 0 1];
%
%
```

```
% % Convert to sparse matrix format
% S = sparse(A);
%
% % Get the values, row indices, and column indices of non-zero elements
% [row, col, values] = find(S);
%
% % Display the results
% disp('Values Array (V):');
% disp(values');
%
% disp('Row Indices Array (R):');
% disp(row');
%
% disp('Column Indices Array (C):');
% disp(col');
%
% % Calculate column pointers
% numCols = size(S, 2);
% colPointers = zeros(numCols + 1, 1);
% for i = 1:numCols
%     colPointers(i) = nnz(col < i);
% end
% colPointers(end) = nnz(col);
%
% % Display column pointers
% disp('Column Pointers Array (C):');
% disp(colPointers);
```