SCHOOL OF ARTIFICIAL INTELLIGENCE

24MA602 Computational Mathematics for Data Science

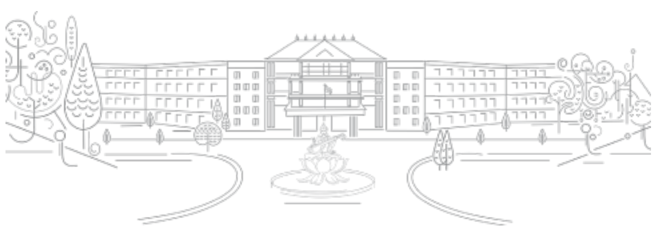
Assignment Set-2

Submitted by

Siju K S
Reg. No. CB.AI.R4CEN24003
Center for Computational Engineering & Networking

Submitted to

Prof. (Dr.) Soman K.P.
Professor & Dean
School of Artificial Intelligence
Amrita Vishwa Vidyapeetham



AUGUST
2024

Contents

List of Tables

List of Figures

1 | Assignment 11 Eigen Vectors and Its Role in Stochastic Process

1.1 Reinterpretation of Matrix- vector Product

Matrix-vector multiplication Ax, can now, be interpreted in three ways

1. Linear Combination of Column vectors in A, with elements in x-vector acting as coefficients
2. A sequence of Dot-products between vector x and row-vectors of A
3. Let Ax result in y, then $A_{m\times n}$ maps x-vector in $\mathbb{R}^n$ space to y-vector in $\mathbb{R}^m$ space

There is another equivalent interpretation for vector_ matrix product.
vector _ matrix multiplication $X^T A$, can be interpreted in three ways

1. Linear Combination of row vectors, with elements in x-vector acting as coefficients.
2. A sequence of Dot-products between vector x and column-vectors of A.
3. if $X^Y A$ result in y, then maps x-vector in $\mathbb{R}^n$ space to y-vector in $\mathbb{R}^m$ space. Note here that y is a row vector.

1.1.1 Creation of stochastic matrices

A **stochastic matrix** is a square matrix used to describe the transitions of a Markov chain. It has two important properties:

1. All elements of the matrix are non-negative:

$$P_{ij} \geq 0 \quad \text{for all } i \text{ and } j$$

2. The sum of the elements in each row or column is 1:

   - For a row-stochastic matrix, each row sums to 1:

$$\sum_j P_{ij} = 1 \quad \text{for all } i$$

   - For a column-stochastic matrix, each column sums to 1:

$$\sum_i P_{ij} = 1 \quad \text{for all } j$$

Example

Here is an example of a column-stochastic matrix:

$$P = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.5 \\ 0.1 & 0.5 & 0.3 \end{pmatrix}$$

Note that each column of this matrix sums to 1:

$$0.5 + 0.4 + 0.1 = 1, \quad 0.3 + 0.2 + 0.5 = 1, \quad 0.2 + 0.5 + 0.3 = 1$$

Logical method to create a stochastic matrix

A simple way to create an $m \times n$ stochastic matrix is here.

- Create random matrix of order $m-1 \times n$ with elements, $0 \leq a_{i,j} \leq 1$.
- Append the m^{th} row as 1- columnsum$(A_{m-1\times n})$.

1. Create a 3×3 stochastic matrix by hand and find its eigen values,

SOLUTION

Step 1: Create a 2×3 matrix as

$$\begin{bmatrix} 0.5 & 0.3 & 0.1 \\ 0.2 & 0.1 & 0.8 \end{bmatrix}$$

Step 2: Create the last row in such a way that column sum becomes 1.

$$\begin{aligned} P &= \begin{bmatrix} 0.5 & 0.3 & 0.1 \\ 0.2 & 0.1 & 0.8 \\ (1-0.7) & (1-0.4) & (1-0.9) \end{bmatrix} \\ &= \begin{bmatrix} 0.5 & 0.3 & 0.1 \\ 0.2 & 0.1 & 0.8 \\ 0.3 & 0.6 & 0.1 \end{bmatrix} \\ Eigen(P) &= \text{Solution of } \lambda^2 - 0.7\lambda + -0.16 = 0 \\ &= (\lambda_1 - 1.0000)(\lambda_2 - 0.2772)(\lambda_3 + 0.5772) = 0 \end{aligned}$$

So the eigen values are $\lambda = 1.0, 0.2772, -0.5772$

`Matlab` code for this task is given below.

```matlab
M1=[0.5 0.3 0.1; 0.2 0.1 0.8;  0.3 0.6 0.1];
% Compute the eigenvalues
eigenvalues = eig(M1);
% Display the eigenvalues
disp('The eigenvalues of the matrix are:');
```

Output of this code is shown below.

```
The eigenvalues of the matrix are:
    1.0000
    0.2772
   -0.5772
```

2. Create a Generic 4x4 stochastic matrix with column sum as one.

SOLUTION

```
A= rand(3,4)*(1/3);
last_row=ones(1,4)- sum(A);
A=[A;last_row];
disp(A)
e=eig(A1);
disp('Eigen Values are:');
disp(abs(e));
```

Output of this code is shown below.

```
    0.0147    0.1040    0.0700    0.2096
    0.1858    0.0597    0.1701    0.0338
    0.2575    0.1130    0.3021    0.1303
    0.5420    0.7234    0.4578    0.6262
Eigen Values are:
    1.0000
    0.0985
    0.1175
    0.1175
```

(a) Explain why it works? Run several times and observe eigen values. What is your conclusion.

SOLUTION

Here first a 3×4 random matrix with elements in $[0, 1)$ is created. Then the last row is calculated as the difference of the column sum from 1. This approach always ensures the column sum of the matrix is 1. Tried many times on different random matrices and verified that column sum is 1 and is independent of the choice of matrix. Also noted that the absolute value of the largest eigen value remains unchanged in all the trials.

(b) Create generic 5x5 stochastic matrix column sums as one.

SOLUTION

Te `Matlab` code for this task is given below.

```
A2= rand(4,5)*(1/4);
last_row=ones(1,5)- sum(A2);
A2=[A2;last_row];
disp(A2)
```

Output of the code is given by.

```
    0.0949    0.1580    0.2124    0.2467    0.0211
    0.2261    0.0608    0.0709    0.0210    0.1328
    0.1701    0.1429    0.1706    0.0626    0.2002
    0.0947    0.2454    0.0895    0.2028    0.1847
    0.4142    0.3929    0.4566    0.4669    0.4612
```

3. Make your own 3x3 stochastic matrix A. Choose your X_0 (its elements must be between 0 and 1 and sum must be one)

SOLUTION

A sample 3×3 stochastic matrix is

$$P = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.5 \\ 0.1 & 0.5 & 0.3 \end{pmatrix}$$

Let $x_0 = [0.2 \quad 0.5 \quad 0.3]$.

(a) Find steady state vector.

SOLUTION

Given the stochastic matrix:

$$P = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.5 \\ 0.1 & 0.5 & 0.3 \end{pmatrix}$$

and the initial probability vector:

$$x_0 = \begin{pmatrix} 0.2 \\ 0.5 \\ 0.3 \end{pmatrix}$$

We need to find the steady-state vector $x = [x_1 \quad x_2 \quad x_3]^T$ such that:

$$P \cdot x = x$$

Step 1: Set up the system of equations
This leads to the following system of equations:

$$\begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.5 \\ 0.1 & 0.5 & 0.3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

This results in the following equations:

$$0.5x_1 + 0.3x_2 + 0.2x_3 = x_1$$

$$0.4x_1 + 0.2x_2 + 0.5x_3 = x_2$$

$$0.1x_1 + 0.5x_2 + 0.3x_3 = x_3$$

Step 2: Rearrange the equations
Rearranging each equation to isolate zero on the right-hand side:

$$0.5x_1 + 0.3x_2 + 0.2x_3 - x_1 = 0$$

$$0.4x_1 + 0.2x_2 + 0.5x_3 - x_2 = 0$$

$$0.1x_1 + 0.5x_2 + 0.3x_3 - x_3 = 0$$

Simplifying further:

$$-0.5x_1 + 0.3x_2 + 0.2x_3 = 0$$

$$0.4x_1 - 0.8x_2 + 0.5x_3 = 0$$

$$0.1x_1 + 0.5x_2 - 0.7x_3 = 0$$

Step 3: Add the constraint
Since $x_1 + x_2 + x_3 = 1$, we add this as the constraint:

$$x_1 + x_2 + x_3 = 1$$

Step 4: Solve the system of equations
The resulting system of equations is:

$$\begin{aligned} -0.5x_1 + 0.3x_2 + 0.2x_3 &= 0 \\ 0.4x_1 - 0.8x_2 + 0.5x_3 &= 0 \\ 0.1x_1 + 0.5x_2 - 0.7x_3 &= 0 \\ x_1 + x_2 + x_3 &= 1 \end{aligned}$$

This system can be solved using Gauss-elimination or by using a computational tool like MATLAB to find the steady-state vector.
`Matlab` code for this task is given below.

```
P = [-0.5 0.3 0.2;
     0.4 -0.8 0.5;
     0.1 0.5 -0.7;
     1 1 1];
b = [0; 0; 0; 1];
x = A0 \ b;
disp('The steady-state vector is:');
```

Output of the code is.

```
The steady-state vector is:
    0.3370
    0.3587
    0.3043
```

The steady-state solution of this discrete Markov process is.$x = \begin{bmatrix} 0.3370 \\ 0.3587 \\ 0.3043 \end{bmatrix}$

(b) show that this solution vector is in the direction of eigenvector of A corresponding to eigenvalue of 1.

SOLUTION

To demonstrate this the following `Matlab` code is used.

```
P = [0.5 0.3 0.2;
     0.4 0.2 0.5;
     0.1 0.5 0.3];
x0 = [0.2; 0.5; 0.3];
[V, D] = eig(P.');
steady_state = V(:, abs(diag(D) - 1) < 1e-8);
steady_state = steady_state / sum(steady_state);
disp('The steady-state vector is:');
```

Output of this code is shown below.

```
The steady-state vector is:
    0.3333
    0.3333
    0.3333
```

4. create 5×5 random integer symmetric matrix A. Compute $A^5 x$ using and eigenvectors of A. Choose x as $[12345]'$. Write a generic Matlab code for the same.

SOLUTION

Let columns of B be eigenvectors of matrix of A . Let D be a diagonal matrix with diagonal elements as eigenvalues. Then

$$A^5 = BD^5B^{-1}$$

Hence

$$A^5 x = BD^5B^{-1}x$$

Matlab code for this task is given below.

```
A_rnd=randi([1,10],5,5);
A= (A_rnd+A_rnd');
[B, D]= eig(A);
x= [1 2 3 4 5]';
A5= B*(D^5)*B' *x;
disp('Action of fifth power of A on x is:');
fprintf('%2.4f\n',A5)
```

Output of the above code is.

```
Action of fifth power of A on x is:
2078947838.0000
2035111588.0000
2498052885.0000
2063157770.0000
2405460277.0000
```

This result is verified using direct calculation

```
%direct computation
A_d5=A^5*x;
disp('Error in Calculations');
fprintf("%2.4f\n",A5-A_d5)
```

```
Error in Calculations
-0.0000
-0.0000
-0.0000
-0.0000
-0.0000
```

From the error table, it is clear that the error in calculation of A^5 using the spectral and modal matrix did a better job.

RESULTS

1. Stochastic matrix and properties of their eigen values are discussed.
2. Steady state solution of a Discrete Markov process is the direction of the largest eigen vector.
3. The power of a matrix is calculated using its spectral and modal matrices.

2 | Assignment 12 Programming Challenge

2.1 Detailed Discussion on K-means Clustering

Data is given in the form of a matrix , $D_{m\times n}$. Each row vector represent a data point. Each data point represent a point in n-dimensional space . Aim is find K clusters (points which are closely located around a point(called cluster center) form a cluster). We Assume number of clusters as K.
The step-by step procedure of K-means clustering is shown below.

2.1.1 Clustering algorithm

Algorithm 1 K-Means Clustering

Input: Dataset $X = \{x_1, x_2, \ldots, x_n\}$, number of clusters k
Output: Cluster centers $C = \{c_1, c_2, \ldots, c_k\}$ and cluster assignments $S = \{S_1, S_2, \ldots, S_k\}$
Initialize: Randomly select k points from the dataset as initial cluster centers $C = \{c_1, c_2, \ldots, c_k\}$
repeat
  Step 1: Assignment step:
  for each point x_i in X **do**
    Assign point x_i to the nearest cluster S_j where:

$$S_j = \arg\min_j \|x_i - c_j\|$$

  end for
  Step 2: Update step:
  for each cluster S_j **do**
    Update the cluster center c_j as the mean of all points in S_j:

$$c_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

  end for
until The cluster centers no longer change or the maximum number of iterations is reached
Return: Final cluster centers C and assignments S

2.1.2 Dataset creation and initialization

In this challenge, a dataset with 5 features is selected using `mvnrnd()` function.In this unsupervised method, a new data point will be assigned into the nearest cluster in terms of feature similarity. In this algorithm, Euclidean distance is used to calculate the distance between data points and the cluster centers. The optimal number of clusters, K is found using the Elbow method. As expected, the optimum number of clusters is 3. `Matlab` code for this task is given below.

```
M = 100;
features = 5;

mu1 = [3, -3, 1, 4, -2];
Sigma1 = eye(features);

mu2 = [1, 2, -1, 3, 0];
Sigma2 = eye(features);

mu3 = [-2, -2, 3, 1, 2];
Sigma3 = eye(features);

r1 = mvnrnd(mu1, Sigma1, M);
r2 = mvnrnd(mu2, Sigma2, M);
r3 = mvnrnd(mu3, Sigma3, M);

D = [r1; r2; r3];
[M, N] = size(D);
max_clusters = 10;
WCSS = zeros(max_clusters, 1);
for K = 1:max_clusters
    C = D(randi(M, K, 1), :);
    iter = 10;
    for j = 1:iter
        dist = zeros(M, K);
        for k = 1:K
            repc = repmat(C(k, :), M, 1);
            sd = (D - repc).^2;
            dist(:, k) = sqrt(sum(sd, 2));
        end

        [~, minIndices] = min(dist, [], 2);

        for k = 1:K
            DIC{k} = find(minIndices == k); % Points assigned to cluster k
            if ~isempty(DIC{k})
                C(k, :) = mean(D(DIC{k}, :), 1); % New cluster centroid
            end
        end
    end
    WCSS_K = 0;
    for k = 1:K
        DIC_k = find(minIndices == k);
        if ~isempty(DIC_k)
            dist_k = sqrt(sum((D(DIC_k, :) - C(k, :)).^2, 2));
            WCSS_K = WCSS_K + sum(dist_k.^2); % Sum of squared distances
        end
    end
    WCSS(K) = WCSS_K;
```

```
end

figure;
plot(1:max_clusters, WCSS, '-o');
xlabel('Number of Clusters (K)');
ylabel('Within-Cluster Sum of Squares (WCSS)');
title('Elbow Plot for Optimal K');
grid on;
```

Above code chunk calculates the within sum of square distance of all the 10 clusters and creates a plot (elbow plot). The elbow plot helps determine the optimal number of clusters where the curve starts to "elbow" or flatten out.

2.1.3 Identifying optimal number of clusters

Elbow plot for one trial of previous `Matlab` code is shown in Figure 2.1 .

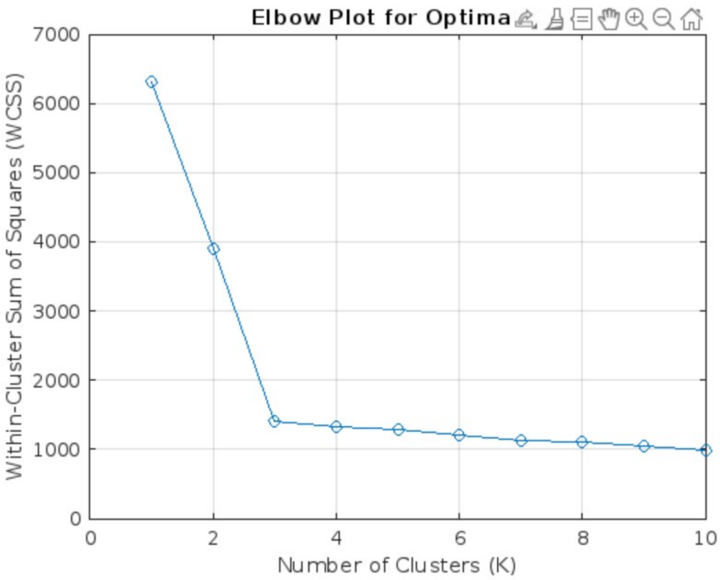


Figure 2.1: Elbow plot to identify optimum number of clusters

From Figure 2.1, $K = 3$ is the optimum number of clusters and this value is read from the user. Following code will do that task, complete the clustering algorithm and produce the progress in the clustering job.

```
optimal_K = input('Choose the optimal number of clusters (K) based
    on the elbow plot: ');

% Define number of clusters and iterations
K = optimal_K;
iter = 10;
C = D(randi(M, K, 1), :);
convergence_threshold = 1e-4;
total_within_cluster_dist = zeros(iter, 1);
centroid_shift = zeros(iter, 1);
convergence_status = false(iter, 1);

for j = 1:iter
    dist = zeros(M, K);
```

```
    for k = 1:K
        repc = repmat(C(k, :), M, 1);
        sd = (D - repc).^2;
        dist(:, k) = sqrt(sum(sd, 2));
    [minValues, minIndices] = min(dist, [], 2);
    prev_C = C;
    within_cluster_dist = 0;
    for k = 1:K
        DIC{k} = find(minIndices == k); % Points assigned to cluster k
        if ~isempty(DIC{k})
            C(k, :) = mean(D(DIC{k}, :), 1);
            within_cluster_dist = within_cluster_dist + sum(minValues(DIC{k}));
        end
    end

    centroid_shift(j) = sum(sqrt(sum((C - prev_C).^2, 2)));
    total_within_cluster_dist(j) = within_cluster_dist;
    convergence_status(j) = centroid_shift(j) < convergence_threshold;

    if convergence_status(j)
        break;
    end
end
T = table((1:iter)', total_within_cluster_dist(1:iter), centroid_shift(1:iter), convergence_status(1:iter), ...
    'VariableNames', {'Iteration_No', 'Within_Cluster_Dist', 'Centroid_Shift', 'Converged'});
disp(T);

figure;
for j = 1:iter
    if isempty(total_within_cluster_dist(j))
        break;
    end
    subplot(2, 5, j);
    scatter(D(:,1), D(:,2), 20, minIndices, 'filled');
    hold on;
    scatter(C(:,1), C(:,2), 100, 'kx', 'LineWidth', 3);
    title(['Iteration ' num2str(j)]);
    xlabel('Feature 1');
    ylabel('Feature 2');
    hold off;
end

support = zeros(optimal_K, 1);
for k = 1:optimal_K
    support(k) = length(DIC{k});
end

```

```
FinalModelTable = table((1:optimal_K)', C(1:optimal_K,1), C(1:
    optimal_K,2), support, ...
    'VariableNames', {'Cluster_No', 'Centroid_X', 'Centroid_Y', '
       Support'});

disp(FinalModelTable);

% Plot final clusters
figure;
scatter(D(:, 1), D(:, 2), 20, minIndices, 'filled');
hold on;
scatter(C(:, 1), C(:, 2), 100, 'kx', 'LineWidth', 3);
title(['Final Clustering with K = ' num2str(optimal_K)]);
xlabel('Feature 1');
ylabel('Feature 2');
legend('Data Points', 'Cluster Centers');
hold off;
```

2.1.4 Progress in clustering

The progress of the algorithm is shown in the following `Matlab` output.

Iteration_No	Within_Cluster_Dist	Centroid_Shift	Converged
1	1097.2	9.3245	false
2	658.33	2.9241	false
3	605.56	0.51781	false
4	602.85	0.33627	false
5	601.49	0.26102	false
6	600.97	0.15001	false
7	600.94	0.10969	false
8	600.84	0	true
9	0	0	true
10	0	0	true

From the above table, on the eighth iteration, the K-means clustering algorithm converges. Change in cluster centers on consecutive iteration is shown in Figure 2.2 choosing the first two features of D.

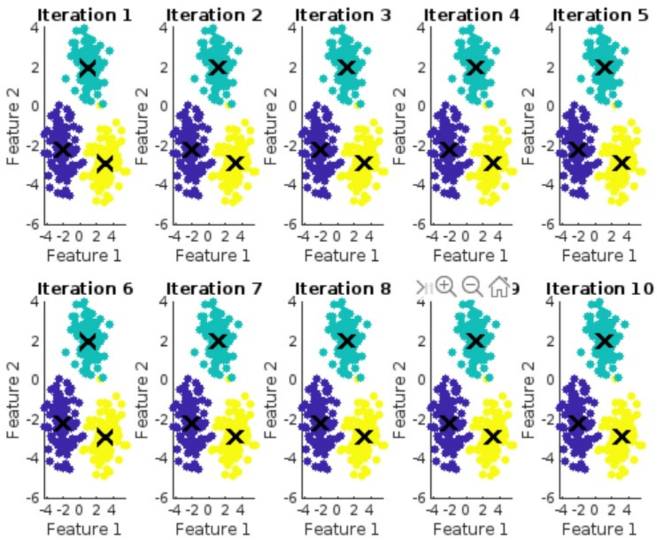


Figure 2.2: Change in cluster centers over iteration

2.1.5 Assessing the skill of the model

The summary table of converged iteration is shown below. Skill of the converged iteration is visualized in Figure 2.3.

Cluster_No	Centroid_X	Centroid_Y	Support
1	-2.1826	-2.1783	100
2	1.0033	2.0338	99
3	3.068	-2.8891	101

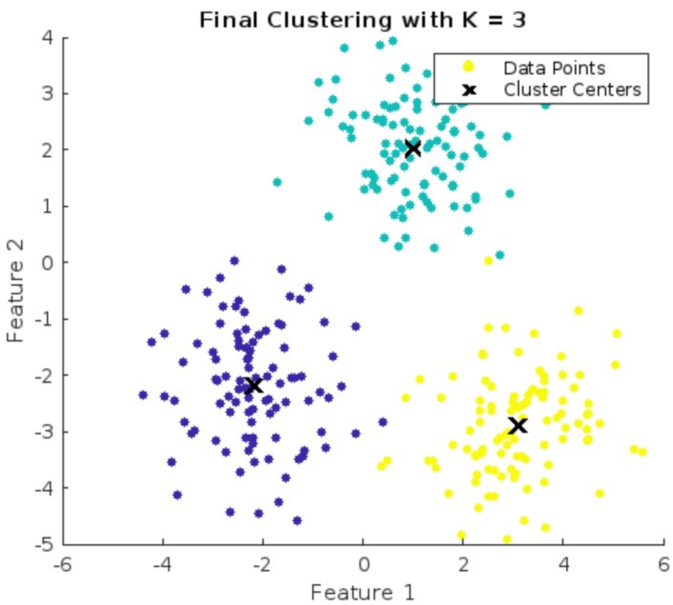


Figure 2.3: Final clusters over first two features

2.2 Best Choice of Features

In this implementation, the first two features are used for visualization. It may not be the right choice to demonstrate the progress in clustering. There are some good methods to choose right features. Some logically sound approaches are listed below.

2.2.1 Variance-based feature selection

Choose two features higher variance. Logic behind this selection is that “Features with high variance tend to contain more useful information for clustering".

2.2.2 Correlation-based feature selection

Another approach is to select two features that are less correlated with each other (to reduce redundancy) but still represent meaningful variability in the data.

2.2.3 Principal Component Analysis (PCA)

PCA is a popular dimensionality reduction technique that can help identify the two principal components (directions with the highest variance) in the data. These components are linear combinations of the original features and can provide a good projection for visualization.

2.2.4 Manual selection

When domain knowledge about the features is available, it is possible to manually select the two most informative features based on relevant business or scientific criteria. This approach aids in identifying features that are likely to provide the most valuable insights and contribute significantly to the analysis.

RESULTS

1. The K-means clustering algorithm is implemented in a more detailed setup.
2. An optimal number of clusters is identified, and an interactive way is provided to use it in clustering.
3. Model skill is evaluated using convergence criteria, support, and visualization.
4. Progress in learning of the model is observed through centroid shift and within cluster distance.

3 | Assignment 13 Spectral Theory and Its Applications

3.1 Background

In modern computational mathematics and applied linear algebra, matrices serve as fundamental tools for representing systems of equations, transformations, and data structures. This article provides a comprehensive analysis of several pivotal matrix concepts, including matrix rank, eigenvalues, eigenvectors, rank-deficient matrices, the Cayley-Hamilton theorem, and spectral decomposition, alongside their applications. It is important to note that these concepts are strictly applicable only to **square matrices**, as they rely on properties unique to square matrices, such as well-defined eigenvalues and characteristic polynomials. These ideas have profound implications for both theoretical advancements and practical applications across science and engineering.

3.1.1 Rank of Matrix products

The *rank of a matrix* is defined as the maximum number of linearly independent rows or columns in the matrix. Formally, it represents the dimension of the matrix's column space or row space. For a product of two matrices A and B, the rank satisfies the inequality:

$$\operatorname{rank}(AB) \leq \min(\operatorname{rank}(A), \operatorname{rank}(B)).$$

This property is crucial for understanding how compositions of linear transformations behave, especially in constrained or overdetermined systems. Rank-deficient matrices, which possess fewer linearly independent rows or columns than their dimensions, are particularly important in systems with redundancy or under-constrained problems. However, when A or B is non-square, this approach does not hold, and other matrix decomposition techniques such as singular value decomposition (SVD) are required.

3.1.2 Eigenvalues and eigenvectors

The *eigenvalue problem* for a square matrix A is defined by the equation $A\mathbf{v} = \lambda\mathbf{v}$, where $\mathbf{v}$ is an eigenvector and λ is the corresponding eigenvalue. Eigenvalues characterize how a linear transformation scales vectors along certain directions (eigenvectors), which remain invariant under that transformation. The spectrum of eigenvalues provides insights into the stability and dynamics of systems modeled by matrices, playing a pivotal role in matrix diagonalization and system analysis.

3.1.3 Creating low-rank matrices using eigenvalues

A *low-rank matrix* can be constructed from an existing square matrix A using its eigenvalues. Let A be an $n \times n$ matrix, and let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be its eigenvalues. The process of reducing the rank step-by-step involves subtracting the identity matrix scaled by an eigenvalue.
For instance, consider $A - \lambda_1 I$, where I is the identity matrix. The matrix $A - \lambda_1 I$ reduces the rank by one, yielding:

$$\operatorname{rank}(A - \lambda_1 I) = n - 1.$$

This result can be generalized by successively subtracting each eigenvalue from the matrix. The product of these terms yields a matrix with progressively decreasing rank. Specifically:

$$\text{rank}((A-\lambda_1 I)(A-\lambda_2 I)\cdots(A-\lambda_n I)) = 0.$$

This construction is applicable only to square matrices, as eigenvalues and identity matrices are not defined for non-square matrices. In the case of non-square matrices, low-rank approximations would rely on techniques like *singular value decomposition (SVD)*, which generalizes eigenvalue decomposition.

3.1.4 Cayley-Hamilton theorem

The *Cayley-Hamilton theorem* asserts that every square matrix satisfies its own characteristic equation. If $p(\lambda) = \det(A-\lambda I)$ is the characteristic polynomial of matrix A, then:

$$p(A) = 0.$$

This theorem is instrumental in reducing computational complexity by expressing powers of matrices in terms of lower powers and scalar multiples. Like eigenvalues, the Cayley-Hamilton theorem is applicable **only to square matrices**, as the characteristic polynomial and determinant are defined only for square matrices. In cases involving non-square matrices, the Cayley-Hamilton theorem does not hold, and alternative approaches like SVD must be used.

Nilpotent Matrices, Eigenvalues, and Rank

A matrix A is called **nilpotent** if there exists a positive integer k such that $A^k = 0$. The smallest such k is called the index of nilpotency.

Eigenvalues of nilpotent matrices

For a nilpotent matrix A, all eigenvalues are zero. To see why, consider an eigenvalue λ of A with corresponding eigenvector v. Then:

$$Av = \lambda v$$

Applying A^k to v:

$$A^k v = \lambda^k v$$

Since $A^k = 0$, it follows that:

$$\lambda^k v = 0$$

Since v is non-zero, $\lambda^k = 0$. Thus, $\lambda = 0$. Therefore, all eigenvalues of a nilpotent matrix are zero.

3.1.5 Rank of nilpotent matrices

The rank of a nilpotent matrix A is related to the size of the largest Jordan block associated with the eigenvalue 0. Specifically, the rank of A is less than the size of the matrix.
To compute the rank, find the number of non-zero rows (or columns) in its row echelon form.

Example

Consider the matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Eigenvalues

Since A is a nilpotent matrix, its eigenvalues are all zero.

Rank

To find the rank, compute the reduced row echelon form of A:

$$\text{rref}(A) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The rank of A is the number of non-zero rows, which is 3. This rank represents the dimension of the column space of A, which is the number of linearly independent columns.

Summary

- **Eigenvalues:** All eigenvalues of a nilpotent matrix are zero.
- **Rank:** The rank of a nilpotent matrix is less than its size and is equal to the number of non-zero rows in its row echelon form.

This connection helps in understanding the structure and properties of nilpotent matrices.

3.2 Spectral Decomposition and Its Applications

Spectral decomposition is the process of decomposing a square matrix into its eigenvalues and eigenvectors. For a diagonalizable matrix A, spectral decomposition is expressed as:

$$A = V\Lambda V^{-1},$$

where V is the matrix of eigenvectors and Λ is a diagonal matrix containing the eigenvalues. Spectral decomposition simplifies matrix computations, especially in solving systems of linear equations, performing principal component analysis (PCA) for data reduction, and understanding quantum mechanical systems. It is essential to note that spectral decomposition is strictly applicable only to **square matrices**. For non-square matrices, *singular value decomposition (SVD)* generalizes this idea, allowing for factorization into orthogonal matrices and singular values.

3.2.1 Applications

The concepts discussed here have far-reaching applications in various fields, including:

- *Data Science and Machine Learning*: Eigenvalue decomposition and low-rank matrix construction are essential in techniques like PCA for dimensionality reduction, which extracts significant features from high-dimensional datasets. In the case of non-square data matrices, SVD is employed.
- *Control Systems*: The Cayley-Hamilton theorem simplifies system analysis, particularly in designing feedback control for linear time-invariant systems, provided the system matrices are square.
- *Quantum Mechanics*: Spectral decomposition aids in understanding operators that describe the evolution of quantum states, with eigenvalues representing measurable quantities.
- *Signal Processing*: Low-rank approximations are used in signal denoising, image compression, and data recovery, making these concepts critical in modern communication systems.

3.3 Tasks

3.3.1 Rank of product of matrices

1. Create two matrix, A and B, with varying ranks. Find the relation between rank of A, B, and AB.

SOLUTION

Let's construct two matrices A and B with varying ranks, and then compute the rank of their product AB.

1. **Matrix A (Rank 2):**

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

This matrix has rank 2 because its rows are linearly dependent ($R_3 = 2R_2 - R_1$).

2. **Matrix B (Rank 3):**

$$B = \begin{bmatrix} 2 & 3 & 1 & 5 \\ 6 & 7 & 4 & 8 \\ 1 & 2 & 9 & 3 \end{bmatrix}$$

This matrix has rank 3 because its columns are linearly dependent.

3. **Rank of Product AB**

Now, let's compute the product AB:

$$AB = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 & 5 \\ 6 & 7 & 4 & 8 \\ 1 & 2 & 9 & 3 \end{bmatrix} = \begin{bmatrix} 17 & 23 & 36 & 30 \\ 44 & 59 & 78 & 78 \\ 71 & 95 & 120 & 126 \end{bmatrix}$$

Using a rank function from MATLAB, we can compute the rank of AB.

$$\text{rank}(AB) = 2$$

RESULTS

- $\rho(A) = 2$
- $\rho(B) = 3$
- $\rho(AB) = 2$

From the inequality $\rho(AB) \leq \min(\rho(A), \rho(B))$, we observe that:

$$\rho(AB) \leq \min(2,3) = 2$$

This satisfies the inequality, as the rank of the product AB is indeed less than or equal to the minimum of the ranks of A and B. `Matlab` code for the above task is given below.

```
A = [1 2 3; 4 5 6; 7 8 9];
B = [2 3 1 5; 6 7 4 8; 1 2 9 3];
AB = A * B;
rank_A = rank(A);
rank_B = rank(B);
rank_AB = rank(AB);
fprintf('Rank of A: %d\n', rank_A);
fprintf('Rank of B: %d\n', rank_B);
fprintf('Rank of AB: %d\n', rank_AB);
```

Output of the above code is shown below.

```
Rank of A: 2
Rank of B: 3
Rank of AB: 2
```

3.3.2 Creating low rank matrices

2. Create a 4×4 matrix with eigenvalues as $1, 2, 3, 4$. Find the following,

   (a) Rank of $A - I, A - 2I, A - 3I$ and $A - 4I$.

   (b) Rank of $(A - I)(A - 2I)$.

   (c) Rank of $(A - I)(A - 2I)(A - 3I)$.

   (d) Rank of $(A - I)(A - 2I)(A - 3I)(A - 4I)$. And note down the observation.

SOLUTION

A square matrix with eigen values $1, 2, 3, 4$ can be created using the method,

$$A = B\text{diag}([1234])\,(B)^{-1} \tag{3.1}$$

Where B is a non singular matrix. Here let's choose $B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$ Using the above relation (3.1),

$$A = \begin{bmatrix} 4 & -7 & 4 & 3 \\ 1 & 0 & 0 & 0 \\ 1 & -2 & 2 & 0 \\ 0 & -2 & 2 & 4 \end{bmatrix}$$

The characteristic polynomial of A is given by;

$$\lambda^4 - 10\lambda^3 + 35\lambda^2 - 50\lambda + 24 = 0 \tag{3.2}$$

Solving (3.2), we get $\lambda = 1, 2, 3, 4$.

(a)

$$A - I = \begin{bmatrix} 3 & -7 & 4 & 3 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & -2 & 2 & 3 \end{bmatrix}$$

$$\rho(A - I) = 3 \quad (\text{ Since, } C_3 = -(C_2 + C_1))$$

$$A - 2I = \begin{bmatrix} 2 & -7 & 4 & 3 \\ 1 & -2 & 0 & 0 \\ 1 & -2 & 0 & 0 \\ 0 & -2 & 2 & 2 \end{bmatrix}$$

$$\begin{aligned} &\rho(A - 2I) = 3 \\ \text{Similarly,}\quad &\rho(A - 3I) = 3 \\ &\rho(A - 4I) = 3 \end{aligned}$$

(b)

$$(A-I)(A-2I)=\begin{bmatrix}3&-21&18&15\\1&-5&4&3\\1&-5&4&3\\0&-6&6&6\end{bmatrix}$$

$$\rho((A-I)(A-2I)=2$$

(c)

$$(A-I)(A-2I)(A-3I)=\begin{bmatrix}0&-24&24&24\\0&-6&6&6\\0&-6&6&6\\0&-6&6&6\end{bmatrix}$$

$$\therefore\rho((A-I)(A-2I)(A-3I))=1$$

(d)

$$(A-I)(A-2I)(A-3I)(A-4I)=\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

$$\therefore\rho((A-I)(A-2I)(A-3I)(A-4I))=0$$

`Matlab` code for this task is given below.

```
B = [1 2 3 4; 1 1 1 1; 1 0 1 1; 0 1 0 1];
D = diag([1 2 3 4]);
A1 = B * D * pinv(B);
lambda1 = 1;
lambda2 = 2;
lambda3 = 3;
lambda4 = 4;

rank_A_1I = rank(A1 - lambda1 * eye(4));
rank_A_2I = rank(A1 - lambda2 * eye(4));
rank_A_3I = rank(A1 - lambda3 * eye(4));
rank_A_4I = rank(A1 - lambda4 * eye(4));
rank_A_12I=rank(((A1-lambda1*eye(4))*(A1-lambda2*eye(4))));
rank_A_123I=rank(((A1-lambda1*eye(4))*(A1-lambda2*eye(4))*(A1-
    lambda3*eye(4))));
rank_A_1234I=rank(round(((A1-lambda1*eye(4))*(A1-lambda2*eye(4)
    )*(A1-lambda3*eye(4))*(A1-lambda4*eye(4))),0));
disp('Rank of (A-lambda_1I):')
disp(rank_A_1I)
disp(rank_A_2I)
disp(rank_A_3I)
disp(rank_A_4I)
disp('Rank of (A-lambda_1I)(A-lambda_2I):')
disp(rank_A_12I)
disp('Rank of (A-lambda_1I)(A-lambda_2I)(A-lambda_3I):')
disp(rank_A_123I)
disp('Rank of (A-lambda_1I)(A-lambda_2I)(A-lambda_3I)(A-
    lambda_4I):')
disp(rank_A_1234I)
```

Output of the above code is shown below.

```
Rank of (A-lambda_1I):
3
3
3
3
Rank of (A-lambda_1I)(A-lambda_2I):
2
Rank of (A-lambda_1I)(A-lambda_2I)(A-lambda_3I):
1
Rank of (A-lambda_1I)(A-lambda_2I)(A-lambda_3I)(A-lambda_4I):
0
```

Observation:

For each λ_i, $A-\lambda_i I$ becomes rank deficient by 1 as one row becomes dependent.When taking the product of these expressions, as many rows as number of terms in the product become dependent. Also by the result $\rho(AB) \leq \min(rho(A), rho(B))$. Hence the characteristic equation of a square matrix will provide an easy method to create a rank deficient matrix with higher size.

RESULTS

(a) A square matrix with given eigen values is created.

(b) All possible low rank matrices from A is created using characteristic polynomial of A.

3. What is the meaning of "Every square matrix satisfies its own characteristic equation"?

SOLUTION

substituting the matrix A in place of λ in its own characteristic equation will always yield the zero matrix. The Cayley-Hamilton theorem ensures that the matrix satisfies this equation, reinforcing the deep connection between eigenvalues and matrix structure and helping on constructing low-rank approximations.

3.3.3 Relationship between rank and eigen values

4. Create matrices with one eigenvalue as 0, two eigenvalues as 0, and three eigenvalues as 0 . Find the rank in each case and comment on the change in rank.

SOLUTION

`Matlab` code for this task is given below.

```
B = [1 2 3 4; 1 1 1 1; 1 0 1 1; 0 1 0 1];
D = diag([1 2 3 0]);
A1 = B * D * pinv(B);
rank_A1 = rank(A1);
disp(['Rank of A1: ', num2str(rank_A1)]);
D = diag([1 2 0 0]);
A2 = B * D * pinv(B);
rank_A2 = rank(A2);
disp(['Rank of A2: ', num2str(rank_A2)]);
eigenvalues (zero matrix)
D = diag([1 0 0 0]);
```

```
A3 = B * D * pinv(B);
rank_A3 = rank(A3);
disp(['Rank of A3: ', num2str(rank(A3))]);
```

Output of the above code is shown below.

```
Rank of A1: 3
Rank of A2: 2
Rank of A3: 1
```

Findings:

As the number of zero eigenvalues in a real non-singular matrix increases, the rank of the matrix decreases. The rank of a non-singular matrix is equal to the number of non-zero eigenvalues, which corresponds to the number of linearly independent rows and columns in the matrix. Thus, a real non-singular matrix with more zero eigenvalues will have a lower rank.

5. Find rank and eigenvalue of the matrix $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$.

SOLUTION

Clearly, there are only two independent rows in A. So $\rho(A) = 2$. The characteristic equation of A is given by

$$\lambda^3 = 0$$

Hence all three eigen values of A are zeros. But A is singular. So the previous relationship between rank and non-zero eigen values will not be valid here. `Matlab` code to find rank and eigen values of A is given below.

```
A = [0 0 0;
     1 0 0;
     1 1 0];
rank_A = rank(A);
disp('Rank of matrix A:');
disp(rank_A);
disp('Eigen values are:')
disp(eig(A))
```

Output of the above code is shown below.

```
Rank of matrix A:
     2
Eigen values are:
     0
     0
     0
```

6. From the answers for question 4 & 5, generate a statement that connects number of zero eigenvalues and rank of a matrix?

SOLUTION

The rank of a non-singular matrix is equal to the number of non-zero eigenvalues, which corresponds to the number of linearly independent rows and columns in the matrix. Thus, a matrix with more zero eigenvalues will have a lower rank.

7. Create basis for column space using 'position of the pivots' return by `rref` command.

SOLUTION

`Matlab` code for this task is given below.

```
A = [1 2 3;
     4 5 6;
     7 8 9];
[R, pivot_columns] = rref(A);
column_basis = A(:, pivot_columns);
disp('Basis for the column space:');
disp(column_basis);
```

Output of the above code is shown below.

```
Basis for the column space:
     1     2
     4     5
     7     8
```

8. Create basis for row space using position of the pivots returned by `rref` command.

SOLUTION

`Matlab` code for this given below.

```
A = [1 2 3;
     4 5 6;
     7 8 9];
[R2,pv2]=rref(A');
row_basis=R2(pv2,:);
disp("Basis for the row space:")
disp(row_basis)
```

Output of the above code is shown below.

```
Basis for the row space:
     1     0    -1
     0     1     2
```

9. Create a 10x10 matrix with rank 6, and create basis sets for column space and row space by selecting vectors from rows and columns. After that, use `rref` again to find null space basis.

SOLUTION

`Matlab` code for this task is shown below.

```
A1 = randn(10, 6);
A = A1 * randn(6, 10);
[m,n] = size(A);
[R, pivot_columns] = rref(A);
column_basis = A(:, pivot_columns);
disp('Basis for the column space:');
disp(column_basis);
row_basis = R(any(R, 2), :);
disp('Basis for the row space:');
disp(row_basis);
```

```
null_basis = null(A, 'r');
disp('Basis for the null space:');
disp(null_basis);
num_pivots = length(pivot_columns);
f = setdiff(1:n,pivot_columns);
r=length(pivot_columns)
N = zeros(n,n-r);
N(pivot_columns,:) = -R(1:r,f);
N(f,:) = eye(n-r);
null_basis_rref = N;
disp('Basis for the null space using rref:');
disp(null_basis_rref);
```

Output of the above code is shown below.

```
Basis for the column space:
    2.6351   -5.6726    7.1165   -8.3013    3.6874   -2.8710
   -0.6416    0.7113   -5.4011    2.2625    2.2172   -1.2242
   -1.2105   -1.5501    8.2829  -10.0089   10.9299    1.2231
   -1.3242    2.3562   -0.8481    2.0978   -1.6495    1.2824
    2.1402   -8.5987    2.2781   -6.7352    4.6876   -2.7559
   -2.1138    3.5220    0.6749    0.4911    0.6572    0.9165
    2.5029   -2.1876   -1.4920    3.7029   -7.2587    0.8052
    1.0714   -0.2302    3.9972   -3.4004    1.8138   -1.4352
    2.4791    1.7584    0.1543    2.9156   -6.2803   -2.6390
    0.3093   -0.1423   -4.8204    3.3574   -3.1585   -1.9868
Basis for the row space:
    1.0000         0         0         0         0         0    1.3151    0.8555   -0.881
         0    1.0000         0         0         0         0   -0.0767    1.3757    1.797
         0         0    1.0000         0         0         0    0.2348   -0.7606   -2.619
         0         0         0    1.0000         0         0    1.3806   -1.3644   -5.328
         0         0         0         0    1.0000         0    0.8177   -0.6044   -2.575
         0         0         0         0         0    1.0000    0.2647    0.4235    1.223
Basis for the null space:
   -1.3151   -0.8555    0.8812   -2.2212
    0.0767   -1.3757   -1.7971   -0.3785
   -0.2348    0.7606    2.6193    0.0646
   -1.3806    1.3644    5.3281   -1.0606
   -0.8177    0.6044    2.5756   -1.1018
   -0.2647   -0.4235   -1.2232   -0.7576
    1.0000         0         0         0
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
Basis for the null space using rref:
   -1.3151   -0.8555    0.8812   -2.2212
    0.0767   -1.3757   -1.7971   -0.3785
   -0.2348    0.7606    2.6193    0.0646
   -1.3806    1.3644    5.3281   -1.0606
   -0.8177    0.6044    2.5756   -1.1018
   -0.2647   -0.4235   -1.2232   -0.7576
    1.0000         0         0         0
         0    1.0000         0         0
```

```
        0           0      1.0000           0
        0           0           0      1.0000
```

From the output, it is clear that both the `null(A)` and `rref` approach produced the same result.

10. Create a 3x4 matrix of rank 2 and find the rank of $[AA]$, $\begin{bmatrix} A \\ A \end{bmatrix}$, $\begin{bmatrix} A & A \\ A & A \end{bmatrix}$. Comment your observation.

SOLUTION

Since the rank of a matrix is the number of linearly independent rows or columns. So all the above mentioned constructions from A produce the same rank of A. To demonstrate this fact, the following `matlab` code is used.

```
% Create a 3x4 matrix A with rank 2
A = [1 0 0 0; 0 1 0 0; 0 0 0 0];
disp('Matrix A (Rank 2):');
disp(A);
matrix1 = [A A];
rank_matrix1 = rank(matrix1);
matrix2 = [A; A];
rank_matrix2 = rank(matrix2);
matrix3 = [A A; A A];
rank_matrix3 = rank(matrix3);
disp('Rank of [A A]:');
disp(rank_matrix1);
disp('Rank of [A; A]:');
disp(rank_matrix2);
disp('Rank of [A A; A A]:');
disp(rank_matrix3);
```

Output of the above code is shown below.

```
Matrix A (Rank 2):
     1     0     0     0
     0     1     0     0
     0     0     0     0
Rank of [A A]:
     2
Rank of [A; A]:
     2
Rank of [A A; A A]:
     2
```

Observation:

Concatenating a matrix with itself either horizontally or vertically does not change the rank of the original matrix. The rank is determined by the number of linearly independent rows or columns, which remains constant even when the matrix is repeated.

11. Create a 4×4 matrix A with eigenvalues 1, 2, 3, 4. Compute A^2, A^3, A^4. Find the eigenvalues and eigenvectors. State your observations.

SOLUTION

If A is a square matrix with eigen values λ_i, then eigen values of A^2, A^3, A^4 are respectively λ_i^2, λ_i^3 and λ_i^4 Also all these powers of A share the same eigen vectors for each λ_i. This fact will be demonstrated using following matlab code.

```
D = diag([1 2 3 4]);
B = rand(4,4);
A = B * D * inv(B);
disp('Matrix A:');
disp(A);

% Compute powers of A
A2 = A^2;
A3 = A^3;
A4 = A^4;
disp('Matrix A^2:');
disp(A2);
disp('Matrix A^3:');
disp(A3);
disp('Matrix A^4:');
disp(A4);
[eigA, eigvecA] = eig(A);
disp('Eigenvalues of A:');
disp(diag(eigA));
disp('Eigenvectors of A:');
disp(eigvecA);
[eigA2, eigvecA2] = eig(A2);
disp('Eigenvalues of A^2:');
disp(diag(eigA2));
disp('Eigenvectors of A^2:');
disp(eigvecA2);
[eigA3, eigvecA3] = eig(A3);
disp('Eigenvalues of A^3:');
disp(diag(eigA3));
disp('Eigenvectors of A^3:');
disp(eigvecA3);
[eigA4, eigvecA4] = eig(A4);
disp('Eigenvalues of A^4:');
disp(diag(eigA4));
disp('Eigenvectors of A^4:');
disp(eigvecA4);
```

Output of the above code is shown below.

```
Matrix A:
   -8.4920   19.8787   15.8358  -24.1981
  -13.0700   24.2897   17.3505  -25.6555
   -6.5498   12.5173   10.5737  -14.3897
   -9.8045   16.6059   11.8714  -16.3714
Matrix A^2:
  -54.1719  110.4251   90.6057 -136.2225
  -68.5798  121.3219   93.3579 -136.5475
  -36.1526   67.2393   54.4381  -79.2181
  -51.0229   85.1888   64.0341  -91.5881
Matrix A^3:
```

```
 -241.0916  477.3635  398.9709 -595.7977
 -276.0006  484.6867  385.1182 -560.9995
 -151.6798  280.4856  229.3196 -336.6769
 -201.5707  335.5700  259.8846 -372.9147
Matrix A^4:
   1.0e+03 *

   -0.9635    1.9027    1.6103   -2.4001
   -1.0132    1.7911    1.4512   -2.1136
   -0.5789    1.0773    0.8925   -1.3136
   -0.7201    1.2044    0.9512   -1.3661
Eigenvalues of A:
   -0.8095
    0.6665
   -0.3343
    0.5590
Eigenvectors of A:
    4.0000         0         0         0
         0    3.0000         0         0
         0         0    2.0000         0
         0         0         0    1.0000
Eigenvalues of A^2:
   -0.8095
    0.6665
   -0.3343
    0.5590
Eigenvectors of A^2:
   16.0000         0         0         0
         0    9.0000         0         0
         0         0    4.0000         0
         0         0         0    1.0000
Eigenvalues of A^3:
   -0.8095
    0.6665
   -0.3343
    0.5590
Eigenvectors of A^3:
   64.0000         0         0         0
         0   27.0000         0         0
         0         0    8.0000         0
         0         0         0    1.0000
Eigenvalues of A^4:
    0.8095
   -0.6665
   -0.3343
    0.5590
Eigenvectors of A^4:
  256.0000         0         0         0
         0   81.0000         0         0
         0         0   16.0000         0
         0         0         0    1.0000
```

RESULTS

- **Matrix A:**
  - Matrix A is constructed as $A = BDB^{-1}$, where D is a diagonal matrix with the eigenvalues 1, 2, 3, and 4.
  - The eigenvalues of A are precisely 1, 2, 3, and 4.
- **Powers of A:**
  - When computing A^k, the eigenvalues of A^k are the eigenvalues of A raised to the power k.
  - Specifically:

$$\text{Eigenvalues of } A^2 \text{ are } \{1^2, 2^2, 3^2, 4^2\},$$
$$\text{Eigenvalues of } A^3 \text{ are } \{1^3, 2^3, 3^3, 4^3\},$$
$$\text{Eigenvalues of } A^4 \text{ are } \{1^4, 2^4, 3^4, 4^4\}.$$

- **Eigenvalues and Eigenvectors:**
  - The eigenvectors of A^k are the same as the eigenvectors of A corresponding to the same eigenvalues.
  - The eigenvalues of A^k are computed as λ_i^k, where λ_i are the eigenvalues of A.
- **Observations:**
  - **Eigenvalues of Powers:** The eigenvalues of A^k follow the pattern λ_i^k, directly related to those of A.
  - **Eigenvectors:** The eigenvectors remain constant across different powers of A as they are unchanged.

3.3.4 Nilpotent matrices

12. Create a 4x4 matrix A, for which $A^{10} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

SOLUTION

Here we have to find a nilpotent vector having this characteristics. A matrix A is called **nilpotent** if there exists a positive integer k such that $A^k = 0$. The smallest such k is called the index of nilpotency. `Matlab` code for this task is given below. Some of the important properties of the nilpotent matrix is listed below.

- **Eigenvalues:** All eigenvalues of a nilpotent matrix are zero.
- **Rank:** The rank of a nilpotent matrix is less than its size and is equal to the number of non-zero rows in its row echelon form.

```
A = [0 1 0 0;
     0 0 1 0;
     0 0 0 1;
     0 0 0 0];
disp('Matrix A:');
disp(A);
A10 = A^10;
disp('Matrix A^10:');
disp(A10);
```

Output of the above code is shown below.

```
Matrix A:
     0     1     0     0
     0     0     1     0
     0     0     0     1
     0     0     0     0
Matrix A^10:
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

RESULTS

Spectral theory and its important applications are revisited with suitable examples.

4 Assignment 14 Random Number Generators

Data Scientists are people with mathematical, statistical and computational skills.

4.1 Data Manipulation with Computing Devices

They are people who can do wonders with data and a computing device. They are very much curious like the 'Dennis the Menace' and always ask the question " iske piche kya hai". Dennis the menace once went to a shop and asked for a toy which he cannot break. Later when he got one, he went and asked for a tool by which he can break it.

By this time (after coming to CEN) you must have understood that Random number generation is very important for computational experiments. It is like a telescope to astronomer and microscope to a biologist.

4.2 Random Number Generation

Have you ever wondered the algorithm behind uniform random number generation in `Matlab`?

4.2.1 Algorithm to generate uniform random number

Here is it.

```
uniformrandgen(5,12345)
```

```
Seed = 2822
Seed = 11031
Seed = 21180
Seed = 9861
Seed = 27202
ans = 1x5
    0.0861    0.3366    0.6464    0.3009    0.8301
```

```
function randnum=uniformrandgen(n,Seed)
uniformrandgen(5,12345)
% Seed . If seed is fixed, every time it return same sequence
% Hence it is pseudo-random number generator
C=25173;
D=13849;
M=32768;
for i=1:n
```

```
Seed = mod((C*Seed + D),M )
randnum(i)=Seed/M;
end
end
```

All other kinds of random numbers can be generated based on uniform random generators.

4.2.2 Role of random numbers in Engineering

In many engineering applications, random numbers are crucial for simulations, statistical modeling, and system analysis. For example, pseudo-random numbers are used in areas such as signal processing, cryptography, and Monte Carlo simulations. Computers generate pseudo-random numbers using deterministic algorithms, creating sequences that mimic random behavior. In this assignment, we will implement a Linear Congruential Generator (LCG) for generating uniform random numbers. The goal is to extend this generator to produce random numbers in a user-defined range, which is useful in various real-world applications.

4.3 Linear Congruential Generator

The LCG is defined by the recurrence relation:

$$X_{n+1} = (C \cdot X_n + D) \mod M$$

where C, D, and M are constants, and X_n is the current seed. The generated random numbers are uniform over the interval $[0,1)$. The following MATLAB code implements this generator.

4.3.1 Random numbers over an interval

Question 1. Create a function to Generate uniform random number between a and b. For example, between 5.3 and 8.7.

SOLUTION

Matlab code and its output for the task is given below.

```
function randnum = uniformrandgen_between_a_b(n, Seed, a, b)
    C = 25173;
    D = 13849;
    M = 32768;
    randnum = zeros(1, n);
    for i = 1:n
        Seed = mod((C * Seed + D), M);
        randnum(i) = Seed / M; % This is uniform in [0, 1)
    end
  randnum = a + (b - a) * randnum;
end
```

```
disp("Required random numbers are:")
```

```
Required random numbers are:
```

```
uniformrandgen_between_a_b(10, 1500, 5.3, 8.7)
```

```
ans = 1x10
    7.8538    7.3289    5.9013    6.1558    7.6559
    8.1387    6.5168    6.4111    7.0395    7.3879
```

Question 2

SOLUTION

`Matlab` code for this task and the output is given below.

Create a function to generate an integer uniform random number between k and l. For example, integer numbers between 5 to 10. You may use the round function in `matlab` in addition to the above uniform random number generator.

```
function randnum = integer_uniformrandgen_between_k_l(n, Seed, k, l)
    C = 25173;
    D = 13849;
    M = 32768;

    randnum = zeros(1, n);

    for i = 1:n
        Seed = mod((C * Seed + D), M);
        randnum(i) = Seed / M;
    end

    randnum = round(k + (l - k) * randnum);
end
```

```
rand_integers = integer_uniformrandgen_between_k_l(5, 12345, 5, 10);
disp(rand_integers)
```

```
     5     7     8     7     9
```

4.3.2 Generating random numbers from Exponential distribution

For exponential, we have

$$f(x) = \lambda e^{-\lambda x}, x \geq 0$$

The cumulative probability distribution function F(x) of the exponential function is nothing but the area under the probability density curve from 0 to x. It is obtained by the following integration:

$$F(x) = \int_0^x \lambda e^{-\lambda x} \mathrm{dx}$$

On integration:

$$F(x) = 1 - e^{-\lambda x}$$

$$e^{-\lambda x} = 1 - F(x)$$

$$-\lambda x = \ln(1 - F(x))$$

$$x = -\left(\frac{1}{\lambda}\right) \times \ln(1 - F(x))$$

$F(x)$, for any x is a number between 0 and 1. Like$F(x)$, since $1 - F(x)$ is always a number between 0 and 1, we can use a simplified formula, $x = -\left(\frac{1}{\lambda}\right) \times \ln(F(x))$. where $F(x)$ is taken as a number from uniform distribution. Thus, equation $x = -\left(\frac{1}{\lambda}\right) \times \ln(\text{rand}())$is used to generate the sample values from exponentially distributed random variable X .

4.3.3 Generating random numbers from Gaussian Distribution

The density function of standard normal distribution (denoted as $\mathcal{N}(0,1)$ whose mean is 0 and variance is 1) is given by

$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, -\infty \leq x \leq \infty$. This distribution, however, do not permit direct inversion since $F(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ dx, do not have a closed form expression. F(x) can be computed only by numerical method.

4.3.4 Box muller formula for generating Random numbers from $\mathcal{N}(0,1)$

The Box Muller method is a brilliant trick to overcome the inversion problem of the cumulative Gaussian distribution function by producing two independent standard normal sample values from two independent uniform sample values. It is based on the variable transformation technique that is usually employed for finding the density function of a function of a random variable'. Interested Readers may search google for its derivation. Let U_1 and U_2 be two independent, uniformly distributed random number. Two Gaussian $N(0,1)$ random numbers are

$y_1 = \sqrt{-2}\ln(U_1)\cos(2\pi U_2)$ and

$$y_2 = \sqrt{-2}\ln(U_1)\sin(2\pi U_2)$$

In excel we need to use only one of this formula. We write the formula as

$$= \text{sqrt}(-2 * \ln(\text{rand}())) * \cos\left(2 * \text{pi}() * (\text{rand}())\right)$$

To generate Gaussian random numbers with mean μ and standard deviation σ , we write the formula as

$$= \mu + \sigma * \text{sqrt}(-2 * \ln(\text{rand}())) * \cos\left(2 * \text{pi}() * (\text{rand}())\right)$$

Question 3

Create a function to generate random numbers from an exponential distribution with the given parameter lamdba.

```
function randnum = exp_randgen(n, Seed, lambda)
    C = 25173;
    D = 13849;
    M = 32768;
    randnum = zeros(1, n);
    for i = 1:n
        Seed = mod((C * Seed + D), M);
        U = Seed / M; %
        randnum(i) = -log(U) / lambda;
    end
end
rand_exponential = exp_randgen(5, 12345, 2);
disp(rand_exponential)
```

```
    1.2260    0.5444    0.2182    0.6004    0.0931
```

Question 4

Create a function to Generate random numbers from a normal (Gaussian) distribution with given parameters 'mean' and standard deviation.

SOLUTION

Matlab code for this task and the output is given below.

```
function randnum = normal_randgen(n, Seed, mu, sigma)
    C = 25173;
    D = 13849;
    M = 32768;
    randnum = zeros(1, n);
    for i = 1:ceil(n/2)
        % Generate two independent uniform random numbers, U1 and
           U2
        Seed = mod((C * Seed + D), M);
        U1 = Seed / M; % Uniform in [0, 1)

        Seed = mod((C * Seed + D), M);
        U2 = Seed / M; % Uniform in [0, 1)
         Z0 = sqrt(-2 * log(U1)) * cos(2 * pi * U2);
        Z1 = sqrt(-2 * log(U1)) * sin(2 * pi * U2);
        if 2*i-1 <= n
            randnum(2*i-1) = mu + sigma * Z0;
        end
        if 2*i <= n
            randnum(2*i) = mu + sigma * Z1;
        end
    end
end
rand_normal = normal_randgen(6, 12345, 10, 2);
disp(rand_normal)
```

```
    7.7063   13.7888    9.4122   11.7736    8.7856    9.8797
```

Question 5

Multivariate (n-tuple, correlated) Gaussian random generator.

```
function X = mv_gaussian_randgen(n, mu, Sigma)
    % mv_gaussian_randgen generates n multivariate Gaussian random
      vectors.
    %
    % Inputs:
    %   n     - Number of random vectors needed
    %   mu    - Mean vector (k x 1)
    %   Sigma - Covariance matrix (k x k)
    %
    % Output:
    %   X     - Matrix of multivariate Gaussian random numbers (n x
      k)

    % Size of the mean vector
    k = length(mu);

    % Cholesky decomposition of the covariance matrix
    L = chol(Sigma, 'lower');

    % Generate independent standard normal random numbers (n x k)
    Z = randn(n, k);

    % Compute the multivariate Gaussian random numbers
    X = repmat(mu', n, 1) + Z * L';
end
mu = [2; 3];
Sigma = [1 0.8; 0.8 1];
n = 5;

rand_multivariate = mv_gaussian_randgen(n, mu, Sigma);
disp(rand_multivariate)
```

```
    0.9311    1.6919
    1.1905    3.1746
   -0.9443   -0.3823
    3.4384    4.0894
    2.3252    3.1153
```

RESULTS

1. Principles behind random number generation in `matlab` are revisited.
2. Various random numbers with specific distributions are generated.

5 | Assignment 15 Integer Valued Matrices with Integer Eigenvalues

5.1 Background

Suppose you are a professor (of Mathematics Department) and wants to give 3x3 eigenvalue problems to first year undergraduates in an examination. You want to give different individual questions to students. Also assume, deputy COE insists that no question should be a repeated version of previous years questions.

What you will do?.

You can't blindly use random integer matrices and spectral decomposition theorem to create such matrices, because of the following reason.

1. Nobody expects a matrix with non-integer elemental values, in an examination.
2. Characteristic equation of the created matrix should have small coefficients so that students can easily factor the cubical equation obtained. This is necessary, because, there is no direct formula for finding the roots of cubical equations and hence students need to do a trial and error procedure to obtain the first root.

So, the challenge is about creating a matrix with small integer values as elements and at the same time having integer eigen values. How will you create hundreds of such matrices?

The solution is based on following linear algebra concepts that allow us to create **integer matrices** whose **inverse** is again an integer matrix.

The rule can be applied for matrix of any size but for examination, we will be generating only 3x3 matrices.

5.2 Important Theorems

Theorems

1. Given two vectors u and $v \in Z^n$ with $u^T v = \beta$, the matrix $P = I_n + uv^T$ has $\det(P) = 1 + \beta$. Also, if $\beta \neq -1, P^{-1} = I_n - \frac{1}{1+\beta} uv^T$.
2. Given two vectors u and $v \in Z^n$ with $u^T v = 0$ $(\Rightarrow \text{Orthogonality})$, the matrix $P = I_n + uv^T$ has $\det(P) = 1$. Also, $P^{-1} = I_n - uv^T$.
3. Let P be an integer matrix with $\det(P) = \delta \equiv$necessarily be an integer, then its inverse $P^{-1} = \frac{1}{\delta} P^{\text{adj}}$, where the adjoint matrix P^{adj} is an integer matrix

Also if D is a diagonal matrix with all its diagonal entries an integer multiple of δ, then the matrix $A = \mathrm{PD}P^{-1}$ is an integer valued matrix.

4. Let P be an nxn integer matrix with determinant $\delta \neq 0$. Let D be a diagonal matrix whose diagonal entries are all integers that are mutually congruent modulo δ (that is $\lambda_1 = c\,(\mathrm{mod}\,\delta)\,,\lambda_2 = c\,(\mathrm{mod}\,\delta)\ldots\lambda_n = c\,(\mathrm{mod}\,\delta)\,;c = 1,2,\ldots)$. Then $A = \mathrm{PD}P^{-1}$is an integer matrix with diagonal entries of D as eigen values.

First two theorems help us to produce integer matrices with its inverse as again integer matrices.

So one good examination question is : create an nxn integer matrix whose inverse is also a integer matrix.

Proof of theorems 1 and 2

Note that $\left(uv^T\right)$ is a rank one matrix with one non-zero eigen value.

(use matlab to verify this). We show that this eigen value is β . Corresponding eigen vector is u.

$$\left(uv^T\right)u = u\left(v^T u\right) = u\beta = \beta u$$

Eigen values of $\left(uv^T\right)$ are $\{\beta\ 0\,0\ldots 0\}$

Eigen values of $I_n + uv^T$ are $\{1+\beta\ 1\,1\ldots 1\}$

(This is because $\mathrm{Ax} = \lambda x \Rightarrow (A+\mathrm{bI})\,x = (\lambda + b)\,x$)

$$\det\left(I_n + uv^T\right) = \prod_{i=1}^{n} \lambda_i = 1+\beta$$

Proof of the inverse:

$$\left(I_n + uv^T\right)\left(I_n - \tfrac{1}{1+\beta}uv^T\right) = \left(I_n - \tfrac{1}{1+\beta}uv^T\right) + uv^T - \tfrac{u\left(v^T u\right)v^T}{1+\beta}$$
$$= \left(I_n - \tfrac{1}{1+\beta}uv^T\right) + uv^T - \tfrac{\beta uv^T}{1+\beta} = \left(I_n - \tfrac{uv^T}{1+\beta}\right) + \tfrac{uv^T}{1+\beta} = I_n$$

Proof of Theorem 3 is straight forward

$$A = \mathrm{PD}P^{-1} = P\begin{bmatrix} i\delta & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & k\delta \end{bmatrix}\frac{1}{\delta}P^{\mathrm{adj}}; i,j,k \in Z$$

The δ cancels out and all three matrices on RHS become integer valued and hence A is integer valued.

All diagonal values in D must integer multiple of δ . This can be expressed as

diagonal_entry (modδ)$\cong$ 0

Proof of Theorem 4 *(Little tricky)*

We know $A = \mathrm{PD}P^{-1}$ is integer eigen-valued if P is integer valued and all entries of D are congruent to 0 (mod δ). So the matrix $B = P\,(D + \mathrm{bI})$is integer valued if all diagonal entries are congruent to b (modδ).

5.2.1 Creating integer eigen values integer matrix

We use theorem 4 to generate Integer eigen valued Integer matrix.

Steps involved for generating 3x3 matrices

1. Generate 3x3 integer valued matrix P using small integer numbers
2. Find its determinant 'detP'. If detP =0, go back to step 1 , other wise proceed to step 3.
3. Take diagonal matrix D as diag ([1 , 1-detP, 1+detP]) (in general [k, k-detP, k+detP], k is an integer)
4. Compute $A = \mathrm{PD}P^{-1}$

An example will illustrate the third step.

Suppose detP is equal to 5 for a 3x3 matrix.

Then diagonal entries are [1, 1-5, 1+5]= [1, -4, 6].

It satisfy the requirement of mod constraint:

1 (mod 5)=1

-4 (mod 5)=1

6 (mod 5)=1

Mod arithmetic is also called 'clock arithmetic'.

At 12 count become zero and 13 hour is equivalent to 1, and hour 11 equivalent to -1.

Following is the `matlab` code for generating 3x3 matrices

```
while 1
    P=randi([-1 2],3,3); % random number between -1 and 2
    DetP=det(P);
    if DetP > 0
        break
    end
    end

A= P*diag([1, 1-DetP, 1+DetP])*inv(P);
A=round(A)
```

```
A = 3x3
    -1    -6    16
     4     1    -8
     2    -3     3
```

```
round(eig(A))
```

```
ans = 3x1
     1
    -5
     7
```

5.2.2 Tasks

Questions

1. Write a matlab code for generating a 4x4 integer matrix whose inverse is also integer matrix

SOLUTION

Matlab code for this task and its output is given below.

```
function A=generate_integer_matrix_4x4_using_theorem()
    while true
        [u, v] = generate_orthogonal_vectors(4);   %
        I = eye(4);
        P = I + u * v';
        P_inv = I - u * v';
        if all(all(abs(round(P_inv)) - P_inv < 1e-10))
            A= round(P_inv);
            break;
        end
    end
end
function [u, v] = generate_orthogonal_vectors(n)
    while true
        u = randi([-5, 5], n, 1);
        v = randi([-5, 5], n, 1);
        if dot(u, v) == 0
            break;
        end
    end
end
A=generate_integer_matrix_4x4_using_theorem();
disp(A)
```

```
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

2. Write a matlab code for generating a 4x4 integer matrix with integer eigen values.

SOLUTION

Matlab code for this task and its output is given below.

```
while 1
    P=randi([-1 2],4,4);
    DetP=det(P);
    if DetP > 0
        break
    end
    end
A= P*diag([1, 1-DetP, 1+DetP,1+2*DetP])*inv(P);
A=round(A)
```

```
A = 4x4
    17     5   -13    -4
    -2    -4     6     4
    -2   -12    14     4
    12    23   -29    -9
```

```
round(eig(A))
```

```
ans = 4x1
    15
    -6
     1
     8
```

General Method to create a matrix of given order `Size`.

```

function A = generate_integer_matrix_with_theorem(size)
    n = size;
    P = randi([-10, 10], n, n);
    while det(P) == 0
        P = randi([-10, 10], n, n);
    end
    delta = det(P);
    eigenvalues = mod(randi([-10, 10], 1, n) + delta, delta);
    D = diag(eigenvalues);
    A = P * D * inv(P);
    A = round(A);
    disp('Matrix A:');
    disp(A);
    disp('Eigenvalues of Matrix A:');
    disp(round(diag(D)));
end
A = generate_integer_matrix_with_theorem(2);
```

```
Matrix A:
    61   -45
   -94    79

Eigenvalues of Matrix A:
     4
   136
```

RESULTS

1. Theoretical foundations for creating matrices with special spectral properties is revisited.
2. Matrices with given eigen values are created.

6 | Assignment 15-1 Creating Some Special Matrices

6.1 Stochastic Matrices

1. Create a stochastic matrix using `Matlab`.

SOLUTION

`Matlab` function, function call and the output for this task is given below.

```
function generate_column_stochastic_matrix(n)
    A = rand(n);
    S = A ./ sum(A, 1);
    S = round(S, 4);
    disp('Column-Stochastic Matrix:');
    disp(S);
end

generate_column_stochastic_matrix(3)
```

```
Column-Stochastic Matrix:
    0.4380    0.3097    0.4077
    0.4550    0.2817    0.4337
    0.1069    0.4086    0.1586
```

6.1.1 Creating a matrix with one eigen value zero

2. Create a 3×3 matrix with all elements non-zero and one eigen value as zero.

SOLUTION

`Matlab` function, function call and the output for this task is given below.

```
function generate_nonzero_matrix_with_zero_eigenvalue()
    n = 3;
    M=randi([1,10],n,n);
    while any(M(:) == 0) || det(M)~=0
        M = randi([1, 10], n, n);
    end
    disp(rank(M))
    disp('Matrix M:');
    disp(M);
    eigenvalues = eig(M);
```

```
    disp('Eigenvalues of Matrix M:');
    disp(eigenvalues);
end
generate_nonzero_matrix_with_zero_eigenvalue()
```

```
     2

Matrix M:
     8     8     1
     7     7     3
    10    10    10

Eigenvalues of Matrix M:
   19.3007
    0.0000
    5.6993
```

Another method to get the same output with some intentional manipulation is shown below.

```
%alternate method
function generate_nonzero_matrix_with_zero_eigenvalueN()
    n = 3;
    M = randi([1, 10], n, n);
    v = randi([1, 10], n, 1);
    M(3, :) = M(1, :) + M(2, :);
    disp('Matrix M:');
    disp(M);
    eigenvalues = eig(M);
    disp('Eigenvalues of Matrix M:');
    disp(eigenvalues);
end
generate_nonzero_matrix_with_zero_eigenvalue()
```

```
     2

Matrix M:
     9     6     6
    10     6     6
    10     2     2

Eigenvalues of Matrix M:
   19.4659
   -2.4659
    0.0000
```

6.1.2 Creating a positive semi-definite matrix

3. Create a 3×3 positive definite matrix (all eigen values are positive) (with all elements non-zero)

SOLUTION

`Matlab` function, function call and the output for this task is given below.

```
function generate_positive_definite_matrix()
    n = 3;
    A = randi([1, 10], n, n);
    M = A' * A;
    disp('Positive Definite Matrix M:');
    disp(M)
    eigenvalues = eig(M);
    disp('Eigenvalues of Matrix M:');
    disp(eigenvalues);
end
generate_positive_definite_matrix()
```

```
Positive Definite Matrix M:
    65    49    68
    49    66    74
    68    74   101

Eigenvalues of Matrix M:
    6.3604
   17.1983
  208.4413
```

6.1.3 Creating a matrix with a specific null space

4. Construct a matrix whose null space consists of combinations of (2,2,1,0) and (3,1,0,1).

SOLUTION

Matlab function, function call and the output for this task is given below.

```
function A = construct_matrix_with_given_nullspace()
    v1 = [2; 2; 1; 0];
    v2 = [3; 1; 0; 1];
    N = [v1, v2];
    A = null(N', 'r');
    if size(A, 1) < 4
        A = [A; zeros(4 - size(A, 1), size(A, 2))];
    end
    A = A';
    disp('Matrix A whose null space consists of (2,2,1,0) and
       (3,1,0,1):');
    disp(A);
    disp('Verification (should be a zero matrix):');
    disp(N' * A');
end
A = construct_matrix_with_given_nullspace();
```

```
Matrix A whose null space consists of (2,2,1,0) and (3,1,0,1):
    0.2500   -0.7500    1.0000         0
   -0.5000    0.5000         0    1.0000
```

```
Verification (should be a zero matrix):
     0     0
     0     0
```

6.1.4 Creating a matrix with a specific column space

5. Construct a matrix whose column space contains (1,1,1) and whose nullspace is the line of multiples of (1,1,1,1).

SOLUTION

Matlab function, function call and the output for this task is given below.

```
n=[1 1 1 1];
NS=-2*null(n)'
```

```
NS = 3x4
    1.0000   -1.6667    0.3333    0.3333
    1.0000    0.3333   -1.6667    0.3333
    1.0000    0.3333    0.3333   -1.6667
```

6. Construct a matrix whose null space consists of all multiples of (4,3,2,1).

SOLUTION

Matlab function, function call and the output for this task is given below.

```
n=[4 3 2 1];
RS=null(n);
RS'
```

```
ans = 3x4
   -0.5477    0.8266   -0.1156   -0.0578
   -0.3651   -0.1156    0.9229   -0.0385
   -0.1826   -0.0578   -0.0385    0.9807
```

6.1.5 Creating a matrix with specific column space and nullspace

7. Construct a matrix whose column space contains (1,1,5) and (0,3,1) and whose nullspace contains (1,1,2).

SOLUTION

Matlab function, function call and the output for this task is given below.

```
Ma=[ 2 0 0; 0 2 0; 0 0 2];
c_1=[1;1;5];
c_2=[0;3;1];
b=[-1;-4;-6];
X=linsolve(Ma,b);
A=[c_1 c_2 X]
```

```
A = 3x3
    1.0000         0   -0.5000
    1.0000    3.0000   -2.0000
    5.0000    1.0000   -3.0000
```

6.1.6 Creation of basis using outer product

8. For the set of all 2 by 2 matrices, create four 2 by 2 basis matrices using the outer products of vectors (1,1) and (1,-1).

SOLUTION

Matlab function, function call and the output for this task is given below.

```
function create_basis_matrices()
    u1 = [1; 1];
    u2 = [1; -1];
    M1 = u1 * u1';
    M2 = u1 * u2';
    M3 = u2 * u1';
    M4 = u2 * u2';
    basis_matrices = {M1, M2, M3, M4};
    disp('Basis Matrix 1:');
    disp(M1);
    disp('Basis Matrix 2:');
    disp(M2);
    disp('Basis Matrix 3:');
    disp(M3);
    disp('Basis Matrix 4:');
    disp(M4);
end
create_basis_matrices()
```

```
Basis Matrix 1:
     1     1
     1     1

Basis Matrix 2:
     1    -1
     1    -1

Basis Matrix 3:
     1     1
    -1    -1

Basis Matrix 4:
     1    -1
    -1     1
```

6.1.7 Creating unsolvable systems

9. Create a system of equations $Ax = b$ such that b is not in column space of A and hence no solution.

SOLUTION

Matlab function, function call and the output for this task is given below.

```
function construct_no_solution_system()
    A=randi([1,10],3,2);
    b = randi([1,5],3,1);
    x = pinv(A) * b;
    residual = norm(A * x - b);

    if residual < 1e-10
        b = b + [0; 0; 10];
    end
    disp('Matrix A:');
    disp(A);
    disp('Vector b:');
    disp(b);
    disp("Verify whether Ax=0")
    if A*x~=0
        disp("Since Ax\neq 0, b is not in column space")

    else
        disp("b is in column space")
    end

end
construct_no_solution_system()
```

```
Matrix A:
     4     7
     2     7
    10     7

Vector b:
     4
     5
     3

Verify whether $Ax=0$. Since $Ax\neq 0$, b is not in column space
```

RESULTS

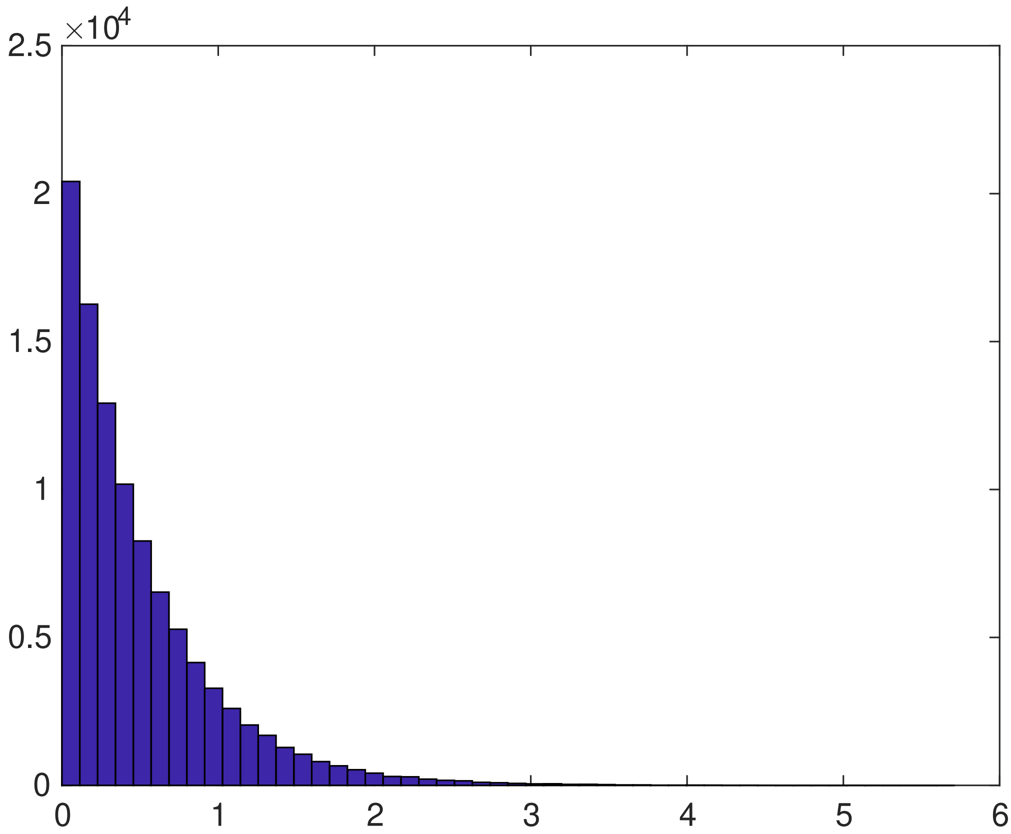
1. Systematic methods to create matrices with specific properties is revisited.
2. Stochastic matrices were created and their important spectral properties are investigated.
3. Long term probabilities of Discrete Markov Process is calculated using spectral operations.
4. Various classes of matrices with specific properties are created.

7 | Assignment 16
Creating Random Numbers Using Matlab

7.1 Random Numbers in Matlab

7.1.1 Creating exponentially distributed random numbers

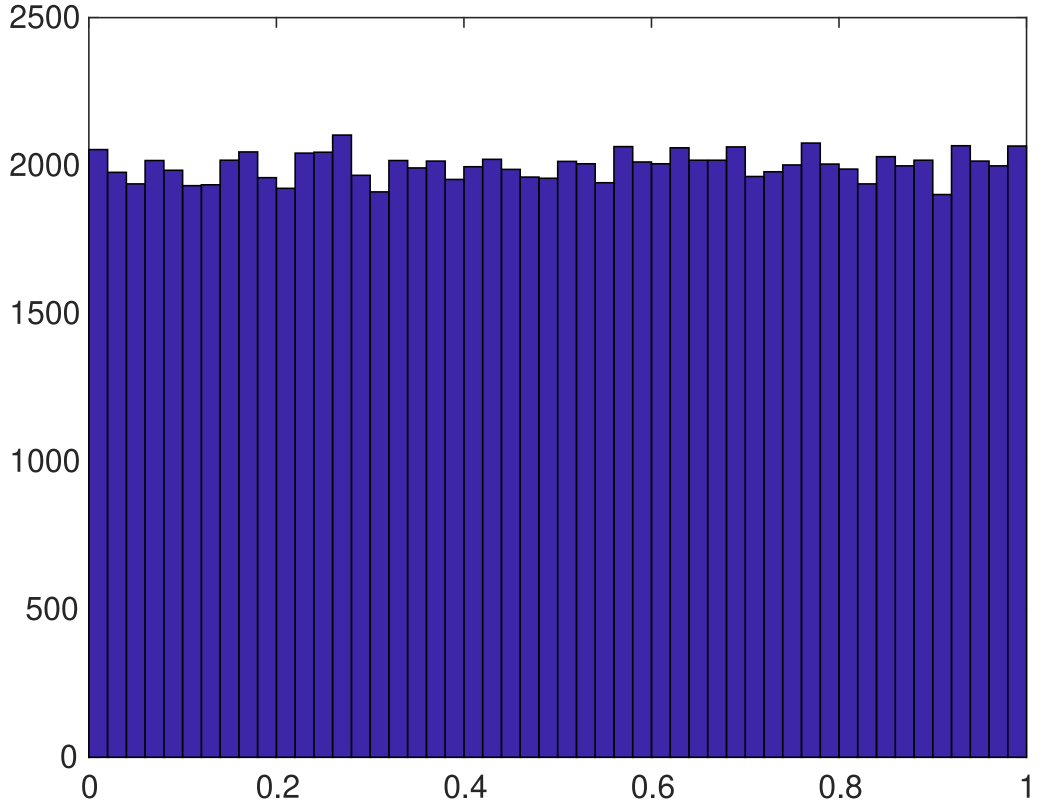
```
figure
lamda=2;
x1=(-1/lamda)*log(rand(100000,1));
hist(x1,50);
```



7.1.2 Creating uniformly distributed random numbers

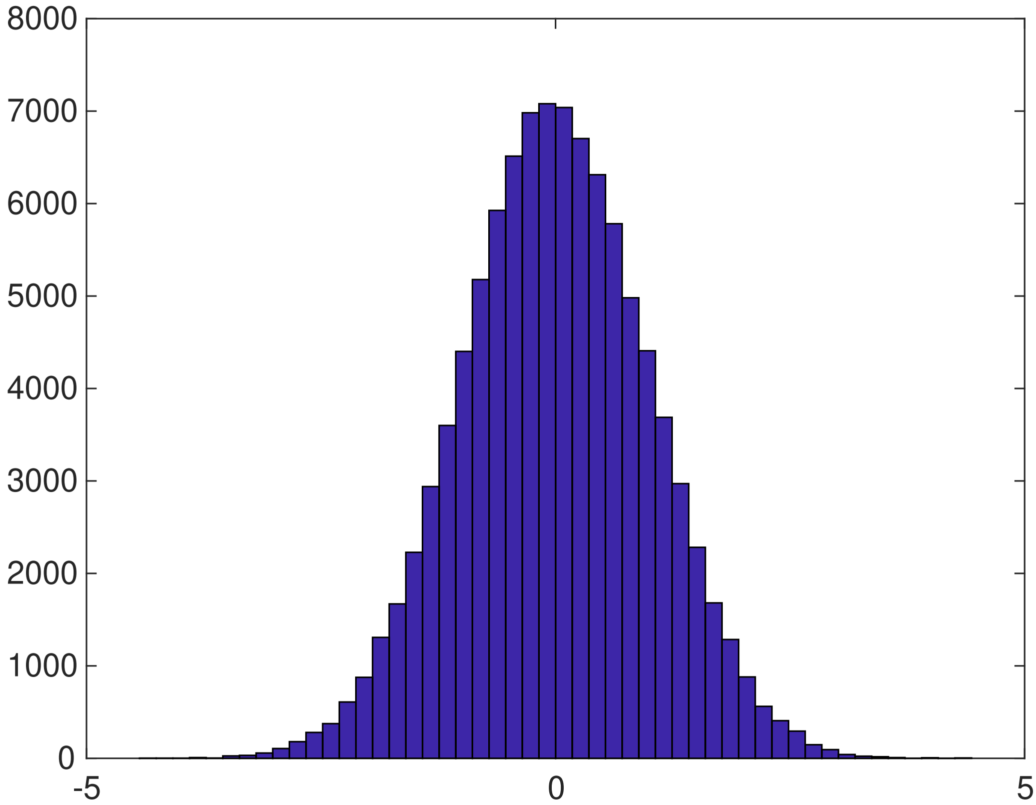
```
% 2. uniform distribution
figure
```

```
x2=rand(100000,1);
hist(x2,50);
```



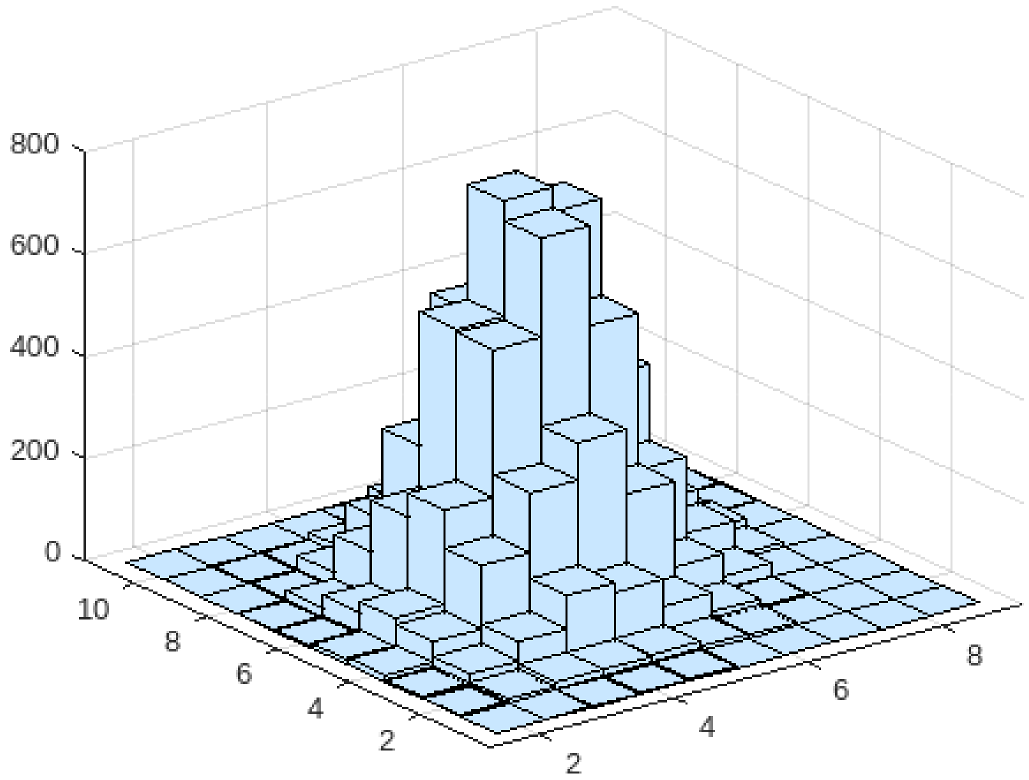
7.1.3 Creating normal random numbers

```
% 3.normal distribution
figure
x3=randn(100000,1);
hist(x3,50);
```



7.1.4 Creating bi-variate normal random numbers

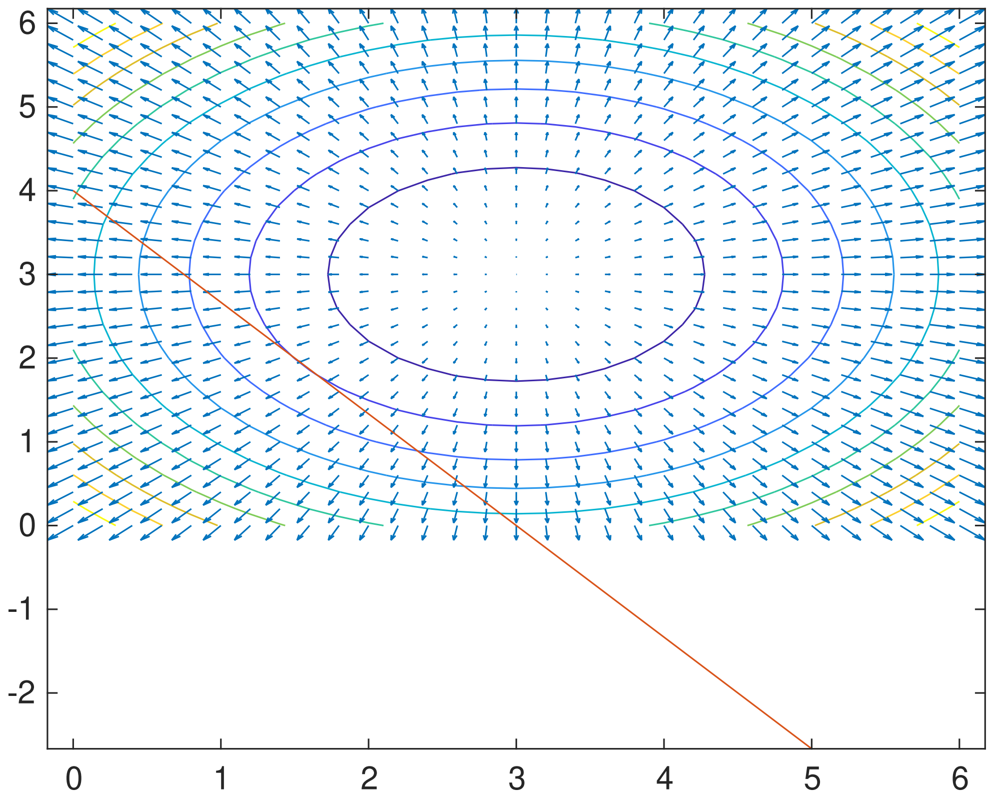
```
figure
%bivariate normal distribution
x4=mvnrnd([5 6],[1 0.4;0.4 2],10000);
hist3(x4);
```



7.2 Visual Tools for Optimization

7.2.1 Plotting Gradients and level sets

```
% plotting gradients and level set  of 2 variable objective
   function and level set of constraint function.

 close(gcf)
[X1,X2] = meshgrid(0:0.2:6);
% Objective function
Z = (X1-3).^2 + (X2-3).^2;
contour(X1,X2,Z,10)
[U,V] = gradient(Z,0.2,0.2);
hold on
quiver(X1,X2,U,V)
% constraint equation 4x1+3x2-12=0
%plotting 4x1+3x2-12=0
x1=0:0.2:5; % x1 is horizonal axis
n=length(x1);
x2=(12*ones(1,n)-4*x1)./3;
plot(x1,x2);
hold off
```



7.3 Tasks

Computational experiments with random variables.

(To teach computational thinking, one should know, computational mathematics and statistics.)

1. Generate 100000 uniform random numbers between 0 and 1.

   SOLUTION

   Matlab code and output for this task is given below.

```
random_numbers = rand(100000, 1);
disp(random_numbers(1:10));
```

```
0.3381
0.1556
0.0693
0.2471
0.5944
0.3356
0.3669
0.6987
0.6347
0.1782
```

(a) Square each value and find average. Report this average value. It should approach 1/3. Why?.

SOLUTION

Matlab code and output for this task is given below.

```
random_numbers = rand(100000, 1);
num2str(average_value)]);
```

```
Average of squared values: 0.3338
```

\Reason: The theoretical average of the square of a uniform random variable X in the interval $[0,1]$ can be derived using the expectation of X^2:

$$\mathbb{E}[X^2] = \int_0^1 x^2\,dx$$

Evaluating this integral:

$$\mathbb{E}[X^2] = \left[\frac{x^3}{3}\right]_0^1 = \frac{1^3}{3} - \frac{0^3}{3} = \frac{1}{3}$$

Thus, the expected value of the squared random variable is $\frac{1}{3}$. This means that, as the number of samples approaches infinity, the average of the squared values should approach $\frac{1}{3}$.

(b) cube each value and find average. Report this average value. It should approach 1/4. Why?.

SOLUTION

Matlab code and output for this task is given below.

```
random_numbers = rand(100000, 1);
cubed_numbers = random_numbers.^3;
average_value = mean(cubed_numbers);
disp(['Average of cubed values: ', num2str(average_value)])
    ;
```

```
Average of cubed values: 0.25119
```

For a uniform random variable X in the interval $[0,1]$, the expected value of X^3 can be derived using the expectation formula:

$$\mathbb{E}[X^3] = \int_0^1 x^3\,dx$$

Evaluating this integral:

$$\mathbb{E}[X^3] = \left[\frac{x^4}{4}\right]_0^1 = \frac{1^4}{4} - \frac{0^4}{4} = \frac{1}{4}$$

Thus, the expected value of the cube of a uniformly distributed random variable on the interval $[0,1]$ is $\frac{1}{4}$.

2. Find variance of the generated 100,000 values. It should approach 1/12. Why ?.

SOLUTION

Matlab code and output for this task is given below.

```
random_numbers = rand(100000, 1);
variance_value = var(random_numbers);
disp(['Variance of generated values: ', num2str(variance_value)
   ]);
```

```
Variance of generated values: 0.083284
```

For a random variable X in the interval $[0,1]$, the variance is given by:

$$\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

We know that:

$$\mathbb{E}[X] = \frac{1}{2}$$

and

$$\mathbb{E}[X^2] = \int_0^1 x^2\, dx = \frac{1}{3}$$

Thus, the variance becomes:

$$\text{Var}(X) = \frac{1}{3} - \left(\frac{1}{2}\right)^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$$

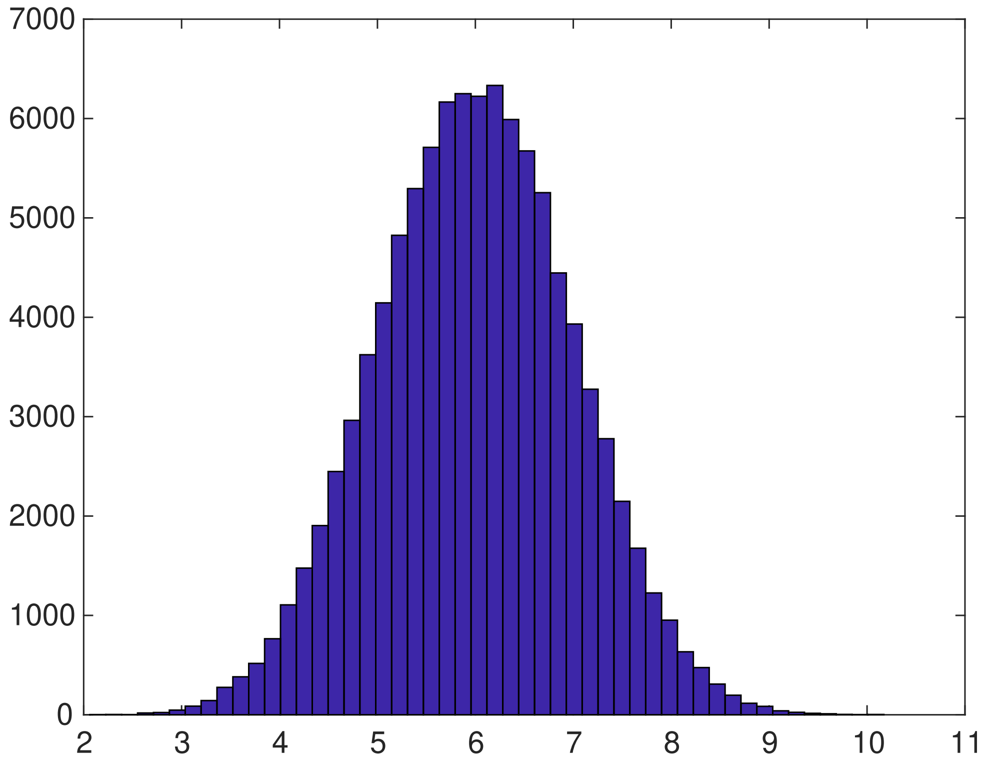
Therefore, the theoretical variance of a uniform random variable between 0 and 1 is $\frac{1}{12}$.

3. Let $X = X_1 + X_2 + \ldots + X_{12}$. Each X is an independent, uniform random variable. By doing simulation. Experiments demonstrate that X approximately follows normal distribution with mean 6 and standard deviation 1.

SOLUTION

Matlab code and output for this task are given below.

```
A=rand(100000,12);
X=sum(A,2);
figure
hist(X,50);
```



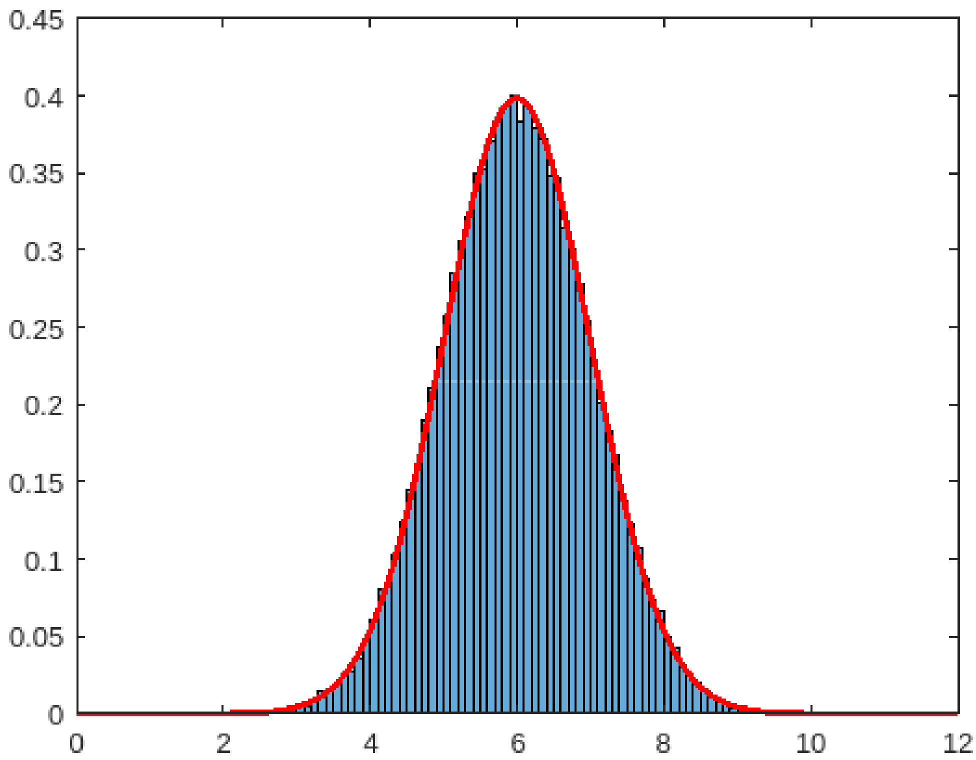

```
n = 100000;
X = sum(rand(n, 12), 2);
histogram(X, 'Normalization', 'pdf');
hold on;
x_values = 0:0.1:12;
normal_pdf = normpdf(x_values, 6, 1);
plot(x_values, normal_pdf, 'r-', 'LineWidth', 2);
mean_X = mean(X);
std_X = std(X);
disp(['Mean of X: ', num2str(mean_X)]);
```

```
Mean of X: 5.9973
```

```
disp(['Standard deviation of X: ', num2str(std_X)]);
```

```
Standard deviation of X: 0.99866
```

```

hold off;
```



By the Central Limit Theorem (CLT), the sum of independent and identically distributed random variables converges to a normal distribution as the number of variables increases.

For uniform random variables $X_i \sim \text{Uniform}(0,1)$, we have:

$$\mathbb{E}[X_i] = \frac{1}{2}, \quad \text{Var}(X_i) = \frac{1}{12}$$

The sum $X = X_1 + X_2 + \cdots + X_{12}$ has:

$$\mathbb{E}[X] = 12 \times \frac{1}{2} = 6, \quad \text{Var}(X) = 12 \times \frac{1}{12} = 1$$

Thus, X is approximately normally distributed with:

$$X \sim \mathcal{N}(6, 1^2)$$

```
n = 100000;
X = sum(rand(n, 12), 2);
mean_X = mean(X);
std_X = std(X);
disp(['Mean of X: ', num2str(mean_X)]);
```

```
Mean of X: 6.0014
```

```
disp(['Standard deviation of X: ', num2str(std_X)]);
```

```
Standard deviation of X: 1.0002
```

4. Repeat above 10 times and report the mean and standard deviation that you get each time.

SOLUTION

Matlab code and output for this task are given below.

```
n = 100000;
num_trials = 30;
means_X = zeros(1, num_trials);
variances_X = zeros(1, num_trials);

for trial = 1:num_trials

    X = sum(rand(n, 12), 2);
    means_X(trial) = mean(X);
    variances_X(trial) = var(X);
end
trial_numbers = (1:num_trials)';
T = table(trial_numbers, means_X', variances_X', ...
          'VariableNames', {'Trial', 'Mean', 'Variance'});
disp(T);
```

```
Trial     Mean      Variance
-----    ------    --------

  1      5.9961     1.0005
  2      6.0008    0.99515
  3      5.9988     1.0033
  4      6.0012     1.0035
  5       6.003     1.0013
  6      5.9925     1.0009
  7      5.9982    0.99934
  8      5.9994      1.001
  9      5.9957     1.0011
 10      5.9995     1.0052
 11      6.0009     0.9957
 12      6.0076     1.0039
 13       6.002     1.0026
 14      5.9999     1.0008
 15      6.0017     1.0036
 16      5.9978    0.99379
 17      6.0029    0.99584
 18      5.9985    0.99654
 19      6.0044     0.9919
 20      5.9984     1.0064
 21      6.0054     1.0016
 22      6.0015      1.001
 23      6.0004     1.0123
 24      6.0049    0.99554
 25      6.0023    0.99251
 26      6.0038    0.99199
 27      5.9983     1.0005
 28      6.0033     1.0048
```

```
     29       5.9971       1.002
     30        6.005     0.99746
```

RESULTS

1. Various types of random numbers are simulated in `matlab`.
2. Distributions are visualized.
3. Central Limit Theorem is computationally verified.

8 | Assignment 16-1 Familiarizing SVD Matrices

8.1 Introduction to Matrix Decomposition

8.1.1 Identifying components of SVD

Do the following computational experiments with `Matlab`.

$$A = \begin{bmatrix} 9 & 10 \\ 10 & 7 \\ 2 & 1 \end{bmatrix}$$

```
format bank        % display results in 2 decimal places
A=[9 10; 10 7; 2 1];
r=rank(A)
```

```
r =
        2.00
```

```
[U S V]=svd(A,'econ')   %  economical SVD
```

```
U = 3x2
       -0.74          0.67
       -0.67         -0.69
       -0.12         -0.28

S = 2x2
       18.18             0
           0          2.13

V = 2x2
       -0.74         -0.67
       -0.67          0.74
```

Note that there are r (=(rank(A)) columns in U, S, and V

Compute $U^T U$and $V^T V$

```
U'*U
```

```
ans = 2x2
         1.00          0.00
         0.00          1.00
```

```
V'*V
```

```
ans = 2x2
         1.00             0
            0          1.00
```

But

```
U*U'
```

```
ans = 3x3
         0.99          0.03         -0.10
         0.03          0.92          0.27
        -0.10          0.27          0.09
```

```
V*V'
```

```
ans = 2x2
         1.00             0
            0          1.00
```

$\mathrm{UU}^T \neq \mathrm{I}$and $\mathrm{VV}^T = \mathrm{I}$

With economical svd, number of columns in U and V matrices , will be same as rank of the matrix. The column vectors in U are orthogonal bases for column space and columns in V (or rows in V^T) form orthogonal bases for row-space.

How shall we demonstrate this?.

8.1.2 Basis set for column space of A from U

Prove that columns of U form the basis set for column space.

Method 1. Show that every column of A can be expressed as linear combination of columns of U. Note that columns of U are orthonormal. So if we can express, it can be shown that $U\left(U^T A\right) = A$

Let us first compute

```
U'*A
```

```
ans = 2x2
        -13.53        -12.14
         -1.43          1.59
```

```
U*(U'*A)
```

```
ans = 3x2
          9.00         10.00
         10.00          7.00
          2.00          1.00
```

Or

$$\begin{bmatrix} 9 \\ 10 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.74 & 0.67 \\ -0.67 & 0.69 \\ -0.12 & -0.28 \end{bmatrix} \begin{bmatrix} -13.53 \\ -1.43 \end{bmatrix}; \begin{bmatrix} 10 \\ 7 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.74 & 0.67 \\ -0.67 & 0.69 \\ -0.12 & -0.28 \end{bmatrix} \begin{bmatrix} -12.14 \\ 1.59 \end{bmatrix}$$

That is, each column of A is a linear combination of columns of U.

Rewriting as above relation as $\left(UU^T\right) A = A$, we obtain

```
(U*U')*A
```

```
ans = 3x2
          9.00         10.00
         10.00          7.00
          2.00          1.00
```

What a wonder?

1. Can you figure out why $U\left(U^T A\right) = A$?

   SOLUTION

   Hint. Interpret $U^T A$ as 'dot producting' columns of A (assume each column as a signal) with orthogonal bases in U to get coefficients (as in signal processing). These coefficients are again used for linearly combining columns of U (the leftmost U matrix) to get the signal back.

$$\begin{bmatrix} 9 & 10 \\ 10 & 7 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} -0.74 & 0.67 \\ -0.67 & 0.69 \\ -0.12 & -0.28 \end{bmatrix} \begin{bmatrix} -0.74 & -0.67 & -0.12 \\ 0.67 & 0.69 & -0.28 \end{bmatrix} \begin{bmatrix} 9 & 10 \\ 10 & 7 \\ 2 & 1 \end{bmatrix}$$

Explanation of $U(U^TA) = A$

Key Concepts:

- **Columns of U**: The columns of U, denoted as $u_1, u_2, \ldots, u_m$, form an orthonormal basis for the column space of A. Orthonormal means $u_i^T u_j = \delta_{ij}$ (the Kronecker delta), where δ_{ij} is 1 if $i = j$ and 0 otherwise.
- **Matrix Multiplication and Dot Products**: When we multiply U^TA, we are essentially projecting A onto the basis vectors u_i of U. Each element $(U^TA)_{ij}$ represents the dot product between the i-th column of U and the j-th column of A.

Breakdown of $U(U^TA)$:

(a) **Compute U^TA:**

$$U^TA = \begin{pmatrix} u_1^Ta_1 & u_1^Ta_2 & \cdots & u_1^Ta_n \\ u_2^Ta_1 & u_2^Ta_2 & \cdots & u_2^Ta_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m^Ta_1 & u_m^Ta_2 & \cdots & u_m^Ta_n \end{pmatrix}$$

Each element $(U^TA)_{ij}$ represents the dot product of the i-th column of U with the j-th column of A.

(b) **Multiply U by U^TA:**

$$U(U^TA) = U\left(\sum_{i=1}^{m}(U^TA)_i u_i\right)$$

Here, $(U^TA)_i$ represents the coefficients from the matrix U^TA corresponding to each column u_i of U. Expanding this, we get:

$$U(U^TA) = \sum_{i=1}^{m}\left(u_i \sum_{j=1}^{n}(U^TA)_{ij}\right)$$

(c) **Reconstruction of A**: Since U^TA contains the dot products of A's columns with U's columns, multiplying U by these dot products reconstructs the original columns of A:

$$\text{If } U^TA = B, \text{ then } UB = A$$

Thus:

$$U(U^TA) = A$$

The matrix $U(U^TA)$ effectively reconstructs A by reassembling its columns from the orthogonal projections given by U^TA.

8.1.3 Intuition behind SVD

Intuitive Understanding:

- **Orthonormal Basis**: U forms an orthonormal basis for the space in which A lies. Multiplying by U ensures that each column of A is correctly aligned with this basis.
- **Projection and Reconstruction**: U^TA projects A onto the basis vectors u_i. When multiplied by U, it maps these projections back to the original space, reconstructing A perfectly.

2. A is a 4x3 random integer matrix. Find a Matrix B such that BA=A. B should not be Identity matrix. Write the matlab code for the same.

SOLUTION

`Matlab` code and its output for this task is given below.

```
A = randi([-10, 10], 4, 3);
while true
    new_col = randi([-10, 10], 4, 1);
    U = [A, new_col];
    if rank(U) == rank(A)
        break
    end
end
B = A / (A' * A) * A';
BA = B * A;
disp('Matrix A:');
```

```
Matrix A:
```

```
disp(A);
```

```
-10.00     -8.00      4.00
  9.00      7.00     -4.00
  2.00      4.00     10.00
 10.00      4.00     -1.00
```

```
disp('Matrix B:');
```

```
Matrix B:
```

```
disp(B);
```

```
  0.56     -0.50     -0.02      0.01
 -0.50      0.44     -0.02      0.01
 -0.02     -0.02      1.00      0.00
  0.01      0.01      0.00      1.00
```

```
disp('BA:');
```

```
BA:
```

```
disp(BA);
```

```
-10.00     -8.00      4.00
  9.00      7.00     -4.00
  2.00      4.00     10.00
 10.00      4.00     -1.00
```

```
if norm(BA - A) < 1e-6
    disp('BA is approximately equal to A');
else
    disp('BA is not equal to A');
end
```

```
BA is approximately equal to A
```

Method 2

Make matrix B by Appending A in U as columns and find rank. It must be again r. Already we know r columns in U are independent because of orthogonality. This prove that column space U and A are same.

```
A = randi([-10, 10], 4, 3);
[U, S, V] = svd(A, 'econ');
B = [U, A];
R1 = rank(U);
R2 = rank(B);
R3 = rank(A);
disp(['Rank of U: ', num2str(R1)]);
```

```
Rank of U: 3
```

```
disp(['Rank of B: ', num2str(R2)]);
```

```
Rank of B: 3
```

```
disp(['Rank of A: ', num2str(R3)]);
```

```
Rank of A: 3
```

```
if R2 == R3
    disp('The column space of U and A are the same.');
else
    disp('The column space of U and A are different.');
end
```

```
The column space of U and A are the same.
```

Matlab demonstration

```
B=[U A]
R1=rank(U);
R2=rank(B)    ;
R3=rank (A)   ;
```

How can we show that rows of V^T form an orthonormal basis for rows of A.

We will show that (AV) $V^T = A$

```
        A*V
```

```
ans = 4x3
        -11.31          5.58          1.74
         12.15         -0.30         -1.11
         -6.22         -1.82         -8.18
         -6.29         -8.80          2.83
```

$$\begin{bmatrix} -13.38 & 1.43 \end{bmatrix} \begin{bmatrix} -0.74 & -0.67 \\ -0.67 & 0.74 \end{bmatrix} = \begin{bmatrix} 9 & 12 \end{bmatrix}$$

$$\begin{bmatrix} -12.12 & 1.47 \end{bmatrix} \begin{bmatrix} -0.74 & -0.67 \\ -0.67 & 0.74 \end{bmatrix} = \begin{bmatrix} 10 & 7 \end{bmatrix}$$

$$\begin{bmatrix} -2.16 & 0.59 \end{bmatrix} \begin{bmatrix} -0.74 & -0.67 \\ -0.67 & 0.74 \end{bmatrix} = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

Linear combination of row of V^T produces rows of A

```
(A*V)*V'
```

```
ans = 4x3
         -7.00          8.00         -7.00
          2.00         -8.00          9.00
          0.00         -3.00        -10.00
          8.00          6.00         -5.00
```

```
A
```

```
A = 4x3
         -7.00          8.00         -7.00
          2.00         -8.00          9.00
             0         -3.00        -10.00
          8.00          6.00         -5.00
```

8.2 Full SVD

Do the following computational experiments with Matlab.

$$A = \begin{bmatrix} 9 & 10 \\ 10 & 7 \\ 2 & 1 \end{bmatrix}$$

```
format bank        % display results in 2 decimal places
A=[9 10; 10 7; 2 1];
r=rank(A)
```

```
r =
        2.00
```

```
[U S V]=svd(A)  % full SVD
```

```
U = 3x3
     -0.74        0.67        0.10
     -0.67       -0.69       -0.28
     -0.12       -0.28        0.95

S = 3x2
     18.18           0
         0        2.13
         0           0

V = 2x2
     -0.74       -0.67
     -0.67        0.74
```

Compute

```
U*U'
```

```
ans = 3x3
      1.00        0.00        0.00
      0.00        1.00       -0.00
      0.00       -0.00        1.00
```

```
U'*U
```

```
ans = 3x3
      1.00        0.00        0.00
      0.00        1.00       -0.00
      0.00       -0.00        1.00
```

```
V*V'
```

```
ans = 2x2
      1.00           0
         0        1.00
```

```
V'*V
```

```
ans = 2x2
        1.00           0
           0        1.00
```

Here third column of U form a basis vector for left null space.

How will we verify it?.

Find A' *U(:,3) . It must be a zero vector.

Compute Eigen vectors of AA^T and compare with columns of U.

```
[U1  lamda]  = eig(A*A')
```

```
U1 = 3x3
       -0.10        0.67        0.74
        0.28       -0.69        0.67
       -0.95       -0.28        0.12

lamda = 3x3
        0.00           0           0
           0        4.56           0
           0           0      330.44
```

```
U
```

```
U = 3x3
       -0.74        0.67        0.10
       -0.67       -0.69       -0.28
       -0.12       -0.28        0.95
```

```
U1
```

```
U1 = 3x3
       -0.10        0.67        0.74
        0.28       -0.69        0.67
       -0.95       -0.28        0.12
```

Note that columns of U and U1 are identical except for a column wise permutation.

Compute Eigen vectors of $A^T A$ and compare with columns of V

```
[V1  lamda]  = eig(A'*A)
```

```
V1 = 2x2
          0.67        -0.74
         -0.74        -0.67

lamda = 2x2
          4.56            0
             0       330.44
```

```
V
```

```
V = 2x2
         -0.74        -0.67
         -0.67         0.74
```

```
V1
```

```
V1 = 2x2
          0.67        -0.74
         -0.74        -0.67
```

Note again that columns of V and V1 are identical except for a column wise permutation.

3. A is a 3x4 random integer matrix. Find a Matrix B such that $AB = A$. B should not be Identity matrix. Write the `matlab` code for the same.

SOLUTION

`Matlab` code and its output for this task is given below.

```
A2 = randi([-10, 10], 3, 4);
[U2, S2, V2] = svd(A2, 'econ');
B2 = V2 * diag([1, 1, 1]) * V2';
Result = A2 * B2;
disp('Matrix A:');
```

```
Matrix A:
```

```
disp(A2);
```

```
         -1.00         3.00         1.00        -7.00
          7.00        -1.00         1.00         2.00
         -2.00            0        -5.00        -1.00
```

```
disp('Matrix B:');
```

```
Matrix B:
```

```
disp(B2);
```

```
     1.00     -0.03      0.00     -0.01
    -0.03      0.15      0.08     -0.35
     0.00      0.08      0.99      0.03
    -0.01     -0.35      0.03      0.86
```

```
disp('AB:');
```

```
AB:
```

```
disp(Result);
```

```
    -1.00      3.00      1.00     -7.00
     7.00     -1.00      1.00      2.00
    -2.00      0.00     -5.00     -1.00
```

```
if norm(Result - A2) < 1e-10
    disp('AB is approximately equal to A.');
else
    disp('AB is not equal to A.');
end
```

```
AB is approximately equal to A.
```

Square of Frobenius norm of a matrix and relation with singular values.

```
A3=[9 10; 10 7; 2 1];
[U S V]=svd(A3,'econ');
norm(A3,'fro')^2
```

```
ans =
        335.00
```

```
sum(sum(A3.^2))
```

```
ans =
        335.00
```

```
sum(sum(S.^2))
```

```
ans =
        335.00
```

That is sum of squares of singular values is equal to the square of the Frobenius norm of the matrix.

Relationship between eigen values of AA^T, $A^T A$and Singular values of A

```
eig(A3*A3')
```

```
ans = 3x1
          0.00
          4.56
        330.44
```

```
eig(A3'*A3)
```

```
ans = 2x1
          4.56
        330.44
```

```
S.^2
```

```
ans = 2x2
        330.44              0
             0           4.56
```

Non-zero eigenvalues of $AA^T$and $A^T A$ are same

Square of the singular values is the same as the non-zero eigenvalues of $AA^T$and $A^T A$.

RESULTS

1. Foundations of Singular Value Decomposition is revisited.
2. Properties of the components of SVD is revisited.
3. Role in U and V in determining fundamental subspace of A is investigated.
4. SVD approach of reconstructing A with maximum information with optimum size is investigated.

9 | Assignment 17 Column Independence and Rank of Sub-spaces

Observe the following

1. If in a $A_{m\times n}$ matrix if all columns together are independent, then the following statements are true

   1. $m \geq n$
   2. Rank of the matrix A is n.
   3. Row space of A is R^n
   4. Right null-space of A is empty.
   5. Rank of $\left(A^T A\right)_{n\times n}$ is n (proof requires following property given in 2)
   6. $A^T A$ is invertible

Proof for Matrix A with Independent Columns

Given a matrix $A_{m\times n}$ where all columns are independent, we prove the following statements:
Statement 1: $m \geq n$ If all columns of A are independent, the number of rows m must satisfy $m \geq n$. This is because for n independent columns, there must be enough rows to support this independence. If $m < n$, some columns would become linearly dependent. Thus, the condition $m \geq n$ must hold.
Statement 2: The rank of matrix A is n Since the columns of A are independent, the rank of the matrix A, which is the number of linearly independent columns, is equal to n. Therefore, the matrix A has full column rank. Thus, $\text{rank}(A) = n$.
Statement 3: Row space of A is $\mathbb{R}^n$ The row space of a matrix A is the subspace spanned by its rows. Since the columns of A are independent, the row space is n-dimensional. The dimension of the row space is equal to the rank of A, which is n. Therefore, the row space of A is $\mathbb{R}^n$.
Statement 4: Right null-space of A is empty The right null space of A consists of all vectors $x \in \mathbb{R}^n$ such that $Ax = 0$. Since the columns of A are independent, the only solution to $Ax = 0$ is the trivial solution $x = 0$. Thus, the right null space of A is empty, meaning it only contains the zero vector.
Statement 5: The rank of $A^T A$ is n We want to show that $\text{rank}(A^T A) = n$. - Since A has full column rank n, $A^T A$ is an $n \times n$ matrix. - The rank of $A^T A$ is the same as the rank of A, because $A^T A$ is positive semi-definite and its null space corresponds to the null space of A. - Since the columns of A are independent, the rank of A is n. Therefore, the rank of $A^T A$ is also n. Thus, $\text{rank}(A^T A) = n$.
Statement 6: $A^T A$ is invertible Since the rank of $A^T A$ is n, $A^T A$ is an $n \times n$ matrix with full rank. A square matrix with full rank is invertible. Therefore, $A^T A$ is invertible.

You may generate several matrices and convince yourselves.

1. Generate a $p \times 4$ matrix A for which all columns are independent. What is minimum value of p required? Demonstrate that all the above property is satisfied for A after choosing a suitable but small p.

SOLUTION

Matlab code and output for the above task is given below.

```
% Define number of trials
num_trials = 5;

% Initialize a table to store results
Results = table('Size', [num_trials, 7], ...
    'VariableTypes', {'double', 'cell', 'logical', 'logical', 'logical', 'logical', 'logical'}, ...
    'VariableNames', {'Trial', 'Matrix', 'Property1_FullRank', 'Property2_RankATA', 'Property3_InvertibleATA', 'Property4_p_ge_4', 'Property5_RowSpace'}, ...
    'RowNames', arrayfun(@num2str, (1:num_trials)', 'UniformOutput', false));

% Perform multiple trials
for trial = 1:num_trials
    % Generate a 4x4 random integer matrix A
    A = randi([-10, 10], 4, 4);

    % Step 1: Check if A has full column rank (Property 1)
    rank_A = rank(A);
    property1 = (rank_A == 4);

    % Step 2: Verify that the rank of A^T A is also 4 (Property 2)
    ATA = A' * A;
    rank_ATA = rank(ATA);
    property2 = (rank_ATA == 4);

    % Step 3: Check if A^T A is invertible (Property 3)
    property3 = (det(ATA) ~= 0);

    % Step 4: Property 4 is that p >= 4, which is always true in this case
    property4 = true;

    % Step 5: Check if the row space of A is \(\mathbb{R}^4\) (Property 5)
    % Here, we're checking if A has full row rank
    property5 = (rank_A == 4);

    % Store the matrix and results in the table. Use mat2str to store matrix as a cell.
    Results(trial, :) = {trial, {A}, property1, property2, property3, property4, property5};
end

% Display the table with results
disp(Results);
```

	Trial	Matrix	Property1_FullRank	Property2_RankATA	Property3_InvertibleATA	Property4_p_ge_4	Property5_RowSpace
1	1	{4x4 double}	true	true	true	true	true
2	2	{4x4 double}	true	true	true	true	true
3	3	{4x4 double}	true	true	true	true	true
4	4	{4x4 double}	true	true	true	true	true
5	5	{4x4 double}	true	true	true	true	true

2. A and $A^T A$ share same null space and hence share same row-space and rank.

Let x be any vector in null-space of A so that Ax = 0.

$$\text{Ax} = 0 \Rightarrow A^T\text{Ax} = 0$$

also, $A^T\text{Ax} = 0 \Rightarrow \text{Ax} = 0$(Can you figure out why?)

So A and A^T share same null space.

If $A_{m\times n}$ matrix has rank n, then $A^T A$ is invertible .

2. Give descriptive argument for $A^T\text{Ax} = 0 \Rightarrow \text{Ax} = 0$

SOLUTION

Proof by Contradiction: Assume the contrary: $A^T Ax = 0$ and $Ax \neq 0$. We will derive a contradiction from this assumption.

Let $Ax = b$ where $b \neq 0$.
Since $b = Ax$, it follows that b is in the column space of A.
Therefore, $b \in \text{Col}(A)$.
Since $b \in \text{Col}(A)$, it must be orthogonal to every vector in the null space of A^T.
Thus, b is orthogonal to $\text{N}(A^T)$, and hence b is in $\text{N}(A^T)^\perp$.
If b is in $\text{N}(A^T)^\perp$, then for any vector $y \in \text{N}(A^T)$, the dot product $y^T b = 0$.
Thus, $y^T(Ax) = 0$.
Since $y \in \text{N}(A^T)$, it follows that $A^T y = 0$.
Hence, $(A^T y)^T x = 0$ implies $0^T x = 0$.
However, $A^T Ax = 0$ implies $A^T b = A^T(Ax) = 0$.
Thus, b should be in $\text{N}(A^T)$.
This contradicts our assumption that b is orthogonal to $\text{N}(A^T)$.
Therefore, our assumption is false.
Thus, if $A^T Ax = 0$, it must be that $Ax = 0$.

(use the knowledge that column vectors cannot be orthogonal to itself)

3. Projection of vector $y_{m\times 1}$ onto Column space of $A_{m\times n}$.

If all the column vectors together form an independent set (generally we say if all columns are independent), then we have a projection matrix given by

$P = A(A^T A)^{-1} A^T$ such that $y_{\text{proj}} = P \times y$

In case, if Columns of A are not independent (ie., rank<n) we can do projection in two ways.

a) Let r be rank of A. Generate a matrix B by selecting r independent columns in A and putting it as columns in B. Then $P = B(B^T B)^{-1} B^T$

b) $P = A * \text{pinv}(A)$

3. With suitable examples, demonstrate that both methods produce same projected vector.

SOLUTION

Matlab code and output for this task are given below.

```
% Define matrix A
A = [1 2 3; 4 5 6; 7 8 9; 10 11 12];

% Method a: Projection using selected independent columns
B = A(:, 1:2); % Select first two columns
P1 = B * inv(B' * B) * B'; % Projection matrix

% Method b: Projection using pseudoinverse
P2 = A * pinv(A); % Projection matrix

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

% Project x using both methods
proj_x_a = P1 * x;
proj_x_b = P2 * x;

% Display results
disp('Projection using method a:');
```

```
Projection using method a:
```

```
disp(proj_x_a);
```

```
    1.0000
    1.0000
    1.0000
    1.0000
```

```

disp('Projection using method b:');
```

```
Projection using method b:
```

```
disp(proj_x_b);
```

```
        1.0000
        1.0000
        1.0000
        1.0000
```

4. Following is the code for linear regression. Add code to this to demonstrate that the error vector in linear regression is orthogonal to vector x and vector of ones. Orthogonality of error vectors to vector ones is basically another way of saying that the sum of errors (or deviation from the fitted line) is zero. Use this approach, examine the skill of the linear regression.

SOLUTION

Matlab code and output for this task are shown below.

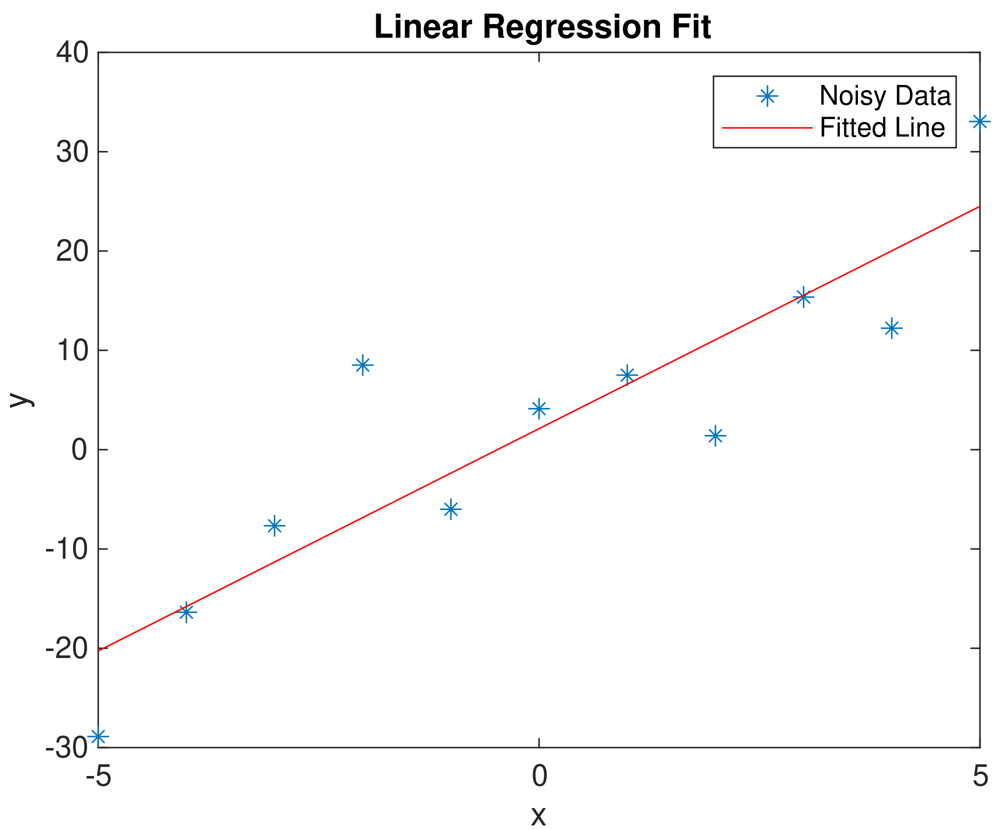
```
close(gcf);
clear all;
m=5;
c=3;
x=-5:5; % a row vector x
n=length(x);
y=m*x+c; % a row vector y representing a line
noise_stdev=6; % noise standard deviation
noise=noise_stdev*randn(1,n);% a row vector
yn=y+noise;
% let us plot
plot(x,yn,'*');
hold on
% fitting a line or resestimating m and c
% create matrix A with x and ones
A=[x' ones(n,1)];
yn=yn'; % make yn a column vector
m_and_c=pinv(A)*yn; % estimate m and c
y_projected=A*pinv(A)*yn; % y estimated
plot(x,y_projected);
m_and_c
```

```
m_and_c = 2x1
    4.4062
    4.5457
```

Projection onto column space is the same as linear regression!

```
close(gcf);
clear all;

% Define parameters
m = 5;
c = 3;
x = -5:5; % a row vector x
n = length(x);
y = m * x + c; % a row vector y representing a line
noise_stdev = 6; % noise standard deviation
```

```
noise = noise_stdev * randn(1, n); % a row vector of noise
yn = y + noise; % observed y with noise

% Plot the noisy data
figure;
plot(x, yn, '*');
hold on;

% Fit a line and estimate m and c
% Create matrix A with x and ones
A = [x' ones(n, 1)];
yn = yn'; % make yn a column vector

% Compute the estimated m and c
m_and_c = pinv(A) * yn;

% Compute the predicted values
y_projected = A * pinv(A) * yn;

% Plot the fitted line
plot(x, y_projected, '-r');
legend('Noisy Data', 'Fitted Line');
xlabel('x');
ylabel('y');
title('Linear Regression Fit');
```




```

```

```
% Compute the error vector
error_vector = yn - y_projected;

% Compute dot products to demonstrate orthogonality
dot_product_x_error = dot(x, error_vector);
dot_product_ones_error = dot(ones(n, 1), error_vector);

% Display results
disp('Dot product of x and error vector:');
```

```
Dot product of x and error vector:
```

```
disp(dot_product_x_error);
```

```
-8.1712e-14
```

```

disp('Dot product of ones and error vector:');
```

```
Dot product of ones and error vector:
```

```
disp(dot_product_ones_error);
```

```
-1.7764e-15
```

```

% Check orthogonality
if abs(dot_product_x_error) < 1e-10
    disp('Error vector is orthogonal to vector x.');
else
    disp('Error vector is not orthogonal to vector x.');
end
```

```
Error vector is orthogonal to vector x.
```

```

if abs(dot_product_ones_error) < 1e-10
    disp('Error vector is orthogonal to vector of ones.');
else
    disp('Error vector is not orthogonal to vector of ones.');
end
```

```
Error vector is orthogonal to vector of ones.
```

RESULTS

1. Relationship between independence of columns of a matrix and rank of subspaces is studied.
2. The quality of the linear regression task is investigated using the orthogonality of the error vector and the one vector.

10 | Assignment 17-1 Projection matrices and SVD

These matrices are going to be used in variety of applications. Suppose we are projecting a vector onto column space of a matrix A in which columns are orthonormal. We know orthonormality imply independence. So $A^T A = I$ and the projection matrix $P = A\left(A^T A\right)^{-1} A^T = AA^T$.

Such matrices are readily obtained using SVD.

Q1. Create rank-3 5x5 matrix X and find projection matrices for projecting onto all subspaces associated with matrix X.

Example.

```
X= randi(10,5,3)*  randi(10,3,5);
r= rank(X);
[U S V]=svd(X);
A= U(:, 1: r) ; %  A is orthogonal basis for columns space of X
B= U(:, r+1:5) ; %  B is orthogonal basis for leftnull space of X
C=V(:, 1: r );  %C is orthogonal basis for row space of X
D=V(:, r+1:5 ); % D is orthogonal basis for rightnull space of X
Prc= A*A'; % projection matrix for projecting into column space
Prln= B*B'; % projection matrix into Left null space
    Prr= C*C'; % projection matrix into row space
Prrn= D*D'; % projection matrix into Right null space
```

Q2. Project vector $y = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$ onto all space associated with the following matrix.

$$X = \begin{bmatrix} 39 & 42 & 39 & 34 & 29 \\ 23 & 24 & 22 & 23 & 18 \\ 28 & 24 & 20 & 28 & 18 \\ 34 & 32 & 28 & 36 & 25 \\ 10 & 10 & 9 & 9 & 7 \end{bmatrix}$$

```
X = [39 42 39 34 29;
     23 24 22 23 18;
     28 24 20 28 18;
     34 32 28 36 25;
     10 10 9 9 7];
y_v = [1 2 3 4 5]';
r = rank(X);
A = U(:, 1:r);
B = U(:, r+1:end);
```

```
C = V(:, 1:r);
D = V(:, r+1:end);
Prc = A * A';
Prln = B * B';
Prr = C * C';
Prrn = D * D';
y_v_column = Prc * y_v;
y_v_left_null = Prln * y_v;
y_v_row = Prr * y_v;
y_v_right_null = Prrn * y_v;
disp('Projection of y_v onto the column space:');
```

```
Projection of y_v onto the column space:
```

```
disp(y_v_column);
```

```
    1.8322
    1.7973
    3.6015
    3.9440
    0.7268
```

```

disp('Projection of y_v onto the left null space:');
```

```
Projection of y_v onto the left null space:
```

```
disp(y_v_left_null);
```

```
   -0.8322
    0.2027
   -0.6015
    0.0560
    4.2732
```

```

disp('Projection of y_v onto the row space:');
```

```
Projection of y_v onto the row space:
```

```
disp(y_v_row);
```

```
    0.5331
    2.5914
    3.1556
    4.5292
    3.9416
```

```

disp('Projection of y_v onto the right null space:');
```

```
Projection of y_v onto the right null space:
```

```
disp(y_v_right_null);
```

```
     0.4669
    -0.5914
    -0.1556
    -0.5292
     1.0584
```

```

% Verify that the projections are orthogonal
disp('Norm of projection onto column space:');
```

```
Norm of projection onto column space:
```

```
disp(norm(y_v - y_v_column));
```

```
    4.3998
```

```

disp('Norm of projection onto left null space:');
```

```
Norm of projection onto left null space:
```

```
disp(norm(y_v - y_v_left_null));
```

```
    5.9700
```

```

disp('Norm of projection onto row space:');
```

```
Norm of projection onto row space:
```

```
disp(norm(y_v - y_v_row));
```

```
    1.4115
```

```

disp('Norm of projection onto right null space:');
```

```
Norm of projection onto right null space:
```

```
disp(norm(y_v - y_v_right_null));
```

```
7.2806
```

Q3. Project vector $Y_1 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$ and $Y_2 = \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}$ onto all appropriate spaces associated with the following matrix.

$$X = \begin{bmatrix} 23 & 41 & 50 & 50 \\ 11 & 21 & 25 & 24 \\ 17 & 25 & 33 & 37 \\ 13 & 23 & 29 & 30 \\ 22 & 36 & 46 & 49 \end{bmatrix}$$

```
X = [23 41 50 50;
     11 21 25 24;
     17 25 33 37;
     13 23 29 30;
     22 36 42 45];
y1 = [1; 2; 3; 4; 5];
y2 = [4; 3; 2; 1];
y2_padded = [y2; zeros(size(X, 1) - length(y2), 1)];
[U, S, V] = svd(X
r = rank(X);
A = U(:, 1:r);
B = U(:, r+1:end);
C1 = V(:, 1:r);
D1= V(:, r+1:end);
Prc = A * A';
Prln = B * B';
Prr = C1 * C1';
Prrn = D1 * D1';
% Make sure projection matrices are of the same size as X
% In this case, X is 5x4, so the projection matrices should be 5x5
% Note that Prr and Prrn should be zero matrices in 5x5 if rank(X)
   < 5
% since they are projecting onto subspaces that do not fully span
   the space.

% Ensure Prc is 5x5
Prc = [Prc, zeros(size(Prc, 1), size(X, 2) - size(Prc, 2))];

% Ensure Prln is 5x5
Prln = [Prln, zeros(size(Prln, 1), size(X, 2) - size(Prln, 2))];

% Ensure Prr is 5x5
Prr = [Prr zeros(4,1);zeros(1,5)];

% Ensure Prrn is 5x5
Prrn = [Prrn zeros(4,1);zeros(1,5)];
```

```

% Project y1 and y2 onto the subspaces
y1_column = Prc * y1; % Projection onto column space
y1_left_null = Prln * y1; % Projection onto left null space
y1_row = Prr * y1; % Projection onto row space
y1_right_null = Prrn * y1; % Projection onto right null space

y2_column = Prc * y2_padded; % Projection onto column space (for
    padded y2)
y2_left_null = Prln * y2_padded; % Projection onto left null space
    (for padded y2)
y2_row = Prr * y2_padded; % Projection onto row space (for padded
    y2)
y2_right_null = Prrn * y2_padded; % Projection onto right null
    space (for padded y2)

% Display the results
disp('Projections for y1:');
```

```
Projections for y1:
```

```
disp('Projection of y1 onto the column space:');
```

```
Projection of y1 onto the column space:
```

```
disp(y1_column);
```

```
    1.8317
    0.7036
    2.6576
    4.0734
    5.0000
```

```

disp('Projection of y1 onto the left null space:');
```

```
Projection of y1 onto the left null space:
```

```
disp(y1_left_null);
```

```
   -0.8317
    1.2964
    0.3424
   -0.0734
    0.0000
```

```

disp('Projection of y1 onto the row space:');
```

```
Projection of y1 onto the row space:
```

```
disp(y1_row);
```

```
    1.0000
    2.0000
    3.0000
    4.0000
         0
```

```

disp('Projection of y1 onto the right null space:');
```

```
Projection of y1 onto the right null space:
```

```
disp(y1_right_null);
```

```
     0
     0
     0
     0
     0
```

```

disp('Projections for y2:');
```

```
Projections for y2:
```

```
disp('Projection of y2 onto the column space:');
```

```
Projection of y2 onto the column space:
```

```
disp(y2_column);
```

```
    4.3914
    2.3899
    1.8388
    1.0345
   -0.0000
```

```

disp('Projection of y2 onto the left null space:');
```

```
Projection of y2 onto the left null space:
```

```
disp(y2_left_null);
```

```
   -0.3914
    0.6101
    0.1612
   -0.0345
    0.0000
```

```

disp('Projection of y2 onto the row space:');
```

```
Projection of y2 onto the row space:
```

```
disp(y2_row);
```

```
    4.0000
    3.0000
    2.0000
    1.0000
         0
```

```

disp('Projection of y2 onto the right null space:');
```

```
Projection of y2 onto the right null space:
```

```
disp(y2_right_null);
```

```
     0
     0
     0
     0
     0
```

```
disp('Norm of projection of y1 onto column space:');
```

```
Norm of projection of y1 onto column space:
```

```
disp(norm(y1 - y1_column));
```

```
    1.5795
```

```

disp('Norm of projection of y1 onto left null space:');
```

```
Norm of projection of y1 onto left null space:
```

```
disp(norm(y1 - y1_left_null));
```

```
7.2460
```

```

disp('Norm of projection of y1 onto row space:');
```

```
Norm of projection of y1 onto row space:
```

```
disp(norm(y1 - y1_row));
```

```
5
```

```

disp('Norm of projection of y1 onto right null space:');
```

```
Norm of projection of y1 onto right null space:
```

```
disp(norm(y1 - y1_right_null));
```

```
7.4162
```

```

disp('Norm of projection of y2 onto column space:');
```

```
Norm of projection of y2 onto column space:
```

```
disp(norm(y2_padded - y2_column));
```

```
0.7433
```

```

disp('Norm of projection of y2 onto left null space:');
```

```
Norm of projection of y2 onto left null space:
```

```
disp(norm(y2_padded - y2_left_null));
```

```
5.4266
```

```

disp('Norm of projection of y2 onto row space:');
```

```
Norm of projection of y2 onto row space:
```

```
disp(norm(y2_padded - y2_row));
```

```
2.6273e-15
```

```

disp('Norm of projection of y2 onto right null space:');
```

```
Norm of projection of y2 onto right null space:
```

```
disp(norm(y2_padded - y2_right_null));
```

```
5.4772
```

RESULTS

Projection of a matrix in to the four fundamental sub spaces are discussed.

11 | Assignment 18 LU Decomposition

11.1 LU Decomposition Using Elementary Matrices

To perform the LU decomposition of a matrix A using elementary matrices, follow these steps:

1. Start with Matrix A

Let A be an $n \times n$ matrix. The goal is to decompose A into the product of a lower triangular matrix L and an upper triangular matrix U, where $A = LU$.

2. Apply Gaussian Elimination

Perform Gaussian elimination to transform A into an upper triangular matrix U. During this process, apply row operations to eliminate the elements below the main diagonal.

3. Track Elementary Matrices

Each row operation can be represented by an elementary matrix E_i. Track these elementary matrices as you perform the row operations.

Elementary Matrices

- **Type I (Row Swaps)**: If you swap rows i and j, the elementary matrix E will have 1's on the diagonal, 0's elsewhere, and 1's in positions (i, j) and (j, i).

- **Type II (Row Scaling)**: If you multiply row i by a scalar k, the elementary matrix E will have k on the diagonal at position (i, i), with 1's elsewhere on the diagonal.

- **Type III (Row Addition)**: If you add k times row j to row i, the elementary matrix E will have 1's on the diagonal and k in position (i, j).

4. Form the LU Decomposition

After transforming A into U using these row operations, the product of all elementary matrices used in the transformation, $E = E_1 \cdot E_2 \cdots E_n$, will be the inverse of the lower triangular matrix L in $A = LU$. Hence, the lower triangular matrix L can be found as $L = E^{-1}$. The upper triangular matrix U is the final matrix obtained after all row operations.

Example

Consider matrix A:

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 2 \\ 8 & 7 & 6 \end{pmatrix}$$

1. Initialize

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 2 \\ 8 & 7 & 6 \end{pmatrix}$$

2. Perform Gaussian Elimination

- **First Elimination Step**:

$$E_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix}$$

  Apply E_1 to A:

$$E_1 \cdot A = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 3 & 2 \end{pmatrix}$$

- **Second Elimination Step**:

$$E_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}$$

  Apply E_2 to the matrix after the first step:

$$E_2 \cdot \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

3. Form Matrices

- **Upper Triangular Matrix** U is:

$$U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

- **Lower Triangular Matrix** L is:

$$L = E_1^{-1} \cdot E_2^{-1}$$

  Compute the inverse of each elementary matrix and their product to get L:

$$E_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix}, \quad E_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{pmatrix}$$

$$L = E_1^{-1} \cdot E_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix}$$

1. Find the LU decomposition of the following matrix,

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix}$$

SOLUTION

Given matrix A:

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix}$$

1. Gaussian Elimination

Perform row operations to convert A into an upper triangular matrix U.

First step: Subtract 2 times the first row from the second row:

$$E_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Apply E_1 to A:

$$E_1 \cdot A = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix}$$

Add the first row to the third row:

$$E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Apply E_2:

$$E_2 \cdot \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix}$$

Add the second row to the third row:

$$E_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Apply E_3:

$$E_3 \cdot \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 5 \end{bmatrix}$$

Thus, the upper triangular matrix U is:

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 5 \end{bmatrix}$$

2. Lower Triangular Matrix L

The lower triangular matrix L is the product of the inverses of the elementary matrices:

$$E_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

$$E_3^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$$

Compute:

$$L = E_1^{-1} \cdot E_2^{-1} \cdot E_3^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}$$

Result

The LU decomposition of A is:

$$A = L \cdot U$$

where:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}$$

and:

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 5 \end{bmatrix}$$

2. Find the LU decomposition of $A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$

SOLUTION

Given the matrix

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix},$$

We perform Gaussian elimination to find its LU decomposition.

Step-by-Step Procedure

1. Upper Triangular Matrix U

Start with the original matrix A:

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Step 1: Eliminate below the first row.

$$E_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Apply E_1 to A:

$$E_1 \cdot A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Step 2: Eliminate below the second row.

$$E_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Apply E_2 to the matrix after the first step:

$$E_2 \cdot \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Eliminate below the third row.

$$E_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Apply E_3 to the matrix after the previous steps:

$$E_3 \cdot \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The upper triangular matrix U is:

$$U = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Lower Triangular Matrix L

The lower triangular matrix L is the product of the inverses of the elementary matrices:

$$E_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$E_3^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

Compute:

$$L = E_1^{-1} \cdot E_2^{-1} \cdot E_3^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

Result

The LU decomposition of A is:

$$A = L \cdot U$$

where:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

and:

$$U = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Find E^2, E^8 and E^{-1} of the following elementary matrix without actual computing, $E = \begin{bmatrix} 1 & 0 \\ 6 & 1 \end{bmatrix}$.

SOLUTION

Since the given matrix is elementary, whenever we apply this matrix from left, the second row becomes $6R_1 + R_2$. This relationship can be summarized as:

$$E = \begin{bmatrix} 1 & 0 \\ 6 & 1 \end{bmatrix}^{2n} = E = \begin{bmatrix} 1 & 0 \\ 12n & 1 \end{bmatrix}$$

(a)

$$\begin{aligned} E^2 &= \begin{bmatrix} 1 & 0 \\ 6 & 1 \end{bmatrix}^{2\times 1} \\ &= \begin{bmatrix} 1 & 0 \\ 12 & 1 \end{bmatrix} \end{aligned}$$

(b)

$$\begin{aligned} E^8 &= \begin{bmatrix} 1 & 0 \\ 6 & 1 \end{bmatrix}^{2\times 4} \\ &= \begin{bmatrix} 1 & 0 \\ 48 & 1 \end{bmatrix} \end{aligned}$$

(c)

$$E^{-1} = \begin{bmatrix} 1 & 0 \\ -6 & 1 \end{bmatrix}$$

`Matlab` code for verifying these results are given below.

```
E=[1 0; 6 1];
E2=E^2;
disp("Square of E is:");
disp(E2);
```

```
E8=E^8;
disp("Eighth power of E is:");
disp(E8);
Einv=inv(E);
disp("Inverse of E is:");
disp(Einv);
```

Output of the above code chunk is given below.

```
Square of E is:
         1.00             0
        12.00          1.00
Eighth power of E is:
         1.00             0
        48.00          1.00
Inverse of E is:
         1.00             0
        -6.00          1.00
```

RESULTS

LU decomposition is discussed using elementary matrices.

12 | Assignment 19 QR Decomposition

The QR factorization is a decomposition of A into QR, where Q is an orthogonal matrix and R is an upper triangular matrix. There are three ways to compute this decomposition.

1. Using Householder matrices
2. Using Givens rotations, also known as Jacobi rotations
3. Using Gram-Schmidt orthogonalization

QR decomposition using Gram – Schmidt Orthogonalization.

Let be an matrix with full column rank.

$$\mathrm{A} = \begin{bmatrix} | & | & | \\ a & b & c \\ | & | & | \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$$

Let a = $\left(a^T \ q_1\right) q_1$ where $q_1 = \frac{a}{||a||}$ so that $||q_1|| = 1$ and $a^T q_1 = \frac{a^T a}{||a||} = ||a||$

Let $\lambda q_2 = b - \left(b^T q_1\right) q_1$ with$||q_1|| = 1 = \frac{(b-(b^T q_1) q_1)}{||b-(b^T q_1) q_1||}$, $q_2^T \ q_1 = 0$,

Vector b can be written as $\therefore b = \left(b^T q_1\right) q_1 + \left(b^T q_2\right) q_2$

Similarly $\beta q_3 = c - \left(c^T q_1\right) q_1 - \left(c^T q_2\right) q_2$with $||q_3|| = 1, q_3 q_1 = 0 = q_3 q_2$ where $\beta = c^T q_3$

$$\therefore c = \left(c^T q_1\right) q_1 + \left(c^T q_2\right) q_2 + \left(c^T q_3\right) q_3$$

$$\therefore A = \begin{bmatrix} | & | & | \\ a & b & c \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ q_1 & q_2 & q_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} a^T q_1 & b^T q_1 & c^T q_1 \\ 0 & b^T q_2 & c^T q_2 \\ 0 & 0 & c^T q_3 \end{bmatrix}$$

Every $m \times n$ matrix with independent columns can be factored into A=QR.

The column of Q are orthogonal. And R is an upper triangular matrix.

From the nonorthogonal a, b, c, find orthonomal vectors q_1, q_2, q_3:

$$a = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Ans:

$$q_1 = \frac{a}{||a||} = \frac{\begin{bmatrix} 1 & 1 & 0 \end{bmatrix}}{\sqrt{1+1+0}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}$$

$$q_2 = \frac{B}{||B||}$$

$$B = b - \left(q_1^T b\right) q_1$$

$$= \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} - \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} - \frac{1}{\sqrt{6}} + \sqrt{\frac{2}{3}} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{2} \\ 0 \end{bmatrix}$$

$$q_2 = \frac{\begin{bmatrix} \frac{1}{2} & \frac{-1}{2} & 1 \end{bmatrix}}{\sqrt{\frac{3}{2}}} = \sqrt{\frac{2}{3}} \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{6}} \\ \sqrt{\frac{2}{3}} \end{bmatrix}$$

$$q_3 = \frac{C}{||C||}$$

$$C = c - \left(q_1^T c\right) q_1 - \left(q_2^T c\right) q_2$$

$$= \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & \sqrt{\frac{2}{3}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{6}} \\ \sqrt{\frac{2}{3}} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{\sqrt{6}} + \sqrt{\frac{2}{3}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{6}} \\ \sqrt{\frac{2}{3}} \end{bmatrix} =$$

$$Q = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{6}} & \frac{1}{\sqrt{2}} \\ 0 & \sqrt{\frac{2}{3}} & 0 \end{bmatrix}$$

Questions:

Q1. Find the third column so that Q = $\begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{14}} & X_1 \\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{14}} & X_2 \\ \frac{1}{\sqrt{3}} & \frac{-3}{\sqrt{14}} & X_3 \end{bmatrix}$ is orthogonal. it must be a unit vector that is orthogonal to other columns.

Hint : The problem has nothing to do with QR decomposition.

Let A = $\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & -3 \end{bmatrix}$. Its left null space vector is orthogonal to both columns. Use Matlab to get that and then do necessary things to get the answer.

```
% Define the matrix A
format bank
A = [1 1; 1 2; 1 -3];

% Perform Singular Value Decomposition (SVD)
[U, S, V] = svd(A);

% Display the results of SVD
disp('U matrix:');
```

```
U matrix:
```

```
disp(U);
```

```
-0.27        -0.58        -0.77
-0.53        -0.58         0.62
 0.80        -0.58         0.15
```

```
disp('Singular values (diagonal of S):');
```

```
Singular values (diagonal of S):
```

```
disp(diag(S));
```

```
3.74
1.73
```

```
disp('V matrix:');
```

```
V matrix:
```

```
disp(V);
```

```
    0        -1.00
-1.00            0
```

```

% Left null space corresponds to the columns of U associated with
   zero singular values
% We identify the zero singular values from the S matrix (diagonal
   values)
% We then find the corresponding columns of U, which span the left
   null space.

% In this case, look at the last column of U (because it's
   associated with zero singular value)
```

```
% Since S is a diagonal matrix, we check the corresponding column
    in U

% Left null space vector
left_null_space_vector = U(:, end);

% Display the left null space vector
disp('Left null space vector (orthogonal to both columns of A):');
```

```
Left null space vector (orthogonal to both columns of A):
```

```
disp(left_null_space_vector);
```

```
-0.77
 0.62
 0.15
```

Q2. (It is a simple question with a simple answer, but visualization is required.)

If the vectors q_1, q_2, q_3 are orthogonal, what linear combinations of q_1 and q_2 are closest to q_3.

```
% Define the vectors q1, q2, and q3
q1 = [1; 0; 0];  % Unit vector along x-axis
q2 = [0; 1; 0];  % Unit vector along y-axis
q3 = [1; 1; 1];  % Vector we want to approximate

% Compute the projection coefficients
a = dot(q1, q3) / dot(q1, q1);
b = dot(q2, q3) / dot(q2, q2);

% Find the closest vector v as a linear combination of q1 and q2
v = a * q1 + b * q2
```

```
v = 3x1
        1.00
        1.00
           0
```

```

% Plot the vectors
figure;
hold on;
grid on;

% Plot q1, q2, and q3
quiver3(0, 0, 0, q1(1), q1(2), q1(3), 'r', 'LineWidth', 2);
quiver3(0, 0, 0, q2(1), q2(2), q2(3), 'g', 'LineWidth', 2);
quiver3(0, 0, 0, q3(1), q3(2), q3(3), 'b', 'LineWidth', 2);
```

```
% Plot the projection vector v
quiver3(0, 0, 0, v(1), v(2), v(3), 'k--', 'LineWidth', 2);

% Labels and Legend
xlabel('X');
ylabel('Y');
zlabel('Z');
legend('q_1', 'q_2', 'q_3', 'Closest projection v');
title('Projection of q_3 onto the plane spanned by q_1 and q_2');
axis equal;
view(3);  % Set to 3D view
grid on;

% Adjust the axis for better visibility
xlim([-1 2]);
ylim([-1 2]);
zlim([-1 2]);

hold off;
```

Visualization of the vector with desired property is shown below.

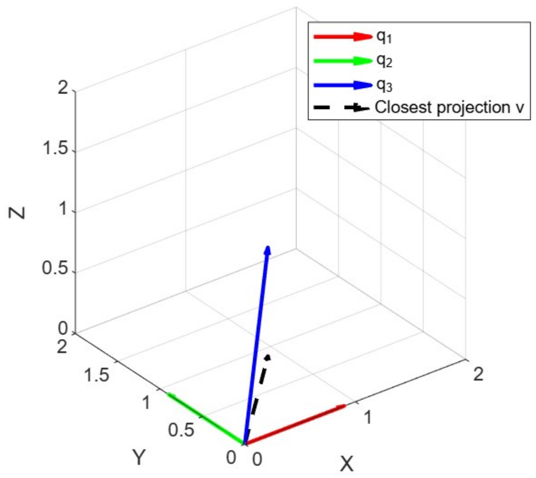


Figure 12.1: The vector $\hat{v}$ projected in the span of q_1 and q_2

Q3. Write a matlab code for generating random 3×3 integer matrices and also for making corresponding Q and R matrices. Verify the answer with built in 'qr' command.

```
% Clear previous variables and close figures
clear;
clc;
close all;

% Generate a random 3x3 integer matrix (random integers between -10
    and 10)
A = randi([-10, 10], 3, 3);

```

```
% Display the generated matrix A
disp('Random 3x3 Integer Matrix A:');
```

```
Random 3x3 Integer Matrix A:
```

```
disp(A);
```

```
     4.00      -10.00        4.00
   -10.00       -8.00       -4.00
    -5.00        7.00        9.00
```

```

% Perform QR decomposition using the built-in function
[Q_builtin, R_builtin] = qr(A);

% Display Q and R from the built-in qr function
disp('Q matrix from built-in qr function:');
```

```
Q matrix from built-in qr function:
```

```
disp(Q_builtin);
```

```
    -0.34       -0.70        0.64
     0.84       -0.52       -0.13
     0.42        0.49        0.76
```

```
disp('R matrix from built-in qr function:');
```

```
R matrix from the built-in qr function:
```

```
disp(R_builtin);
```

```
   -11.87       -0.42       -0.93
        0       14.59        3.74
        0           0        9.91
```

```

% Gram-Schmidt process to manually compute Q and R
% Initialize Q and R
Q_manual = zeros(size(A));
R_manual = zeros(size(A));

% Perform Gram-Schmidt orthogonalization
for i = 1:3
    v = A(:, i);
    for j = 1:i-1
        R_manual(j, i) = Q_manual(:, j)' * A(:, i);
        v = v - R_manual(j, i) * Q_manual(:, j);
```

```
    end
    R_manual(i, i) = norm(v);
    Q_manual(:, i) = v / R_manual(i, i);
end

% Display manually computed Q and R
disp('Manually computed Q matrix:');
```

```
Manually computed Q matrix:
```

```
disp(Q_manual);
```

```
     0.34       -0.70        0.64
    -0.84       -0.52       -0.13
    -0.42        0.49        0.76
```

```
disp('Manually computed R matrix:');
```

```
Manually computed R matrix:
```

```
disp(R_manual);
```

```
    11.87        0.42        0.93
        0       14.59        3.74
        0           0        9.91
```

```

% Verifying if Q_manual and Q_builtin are close
disp('Difference between Q_manual and Q_builtin:');
```

```
Difference between Q_manual and Q_builtin:
```

```
disp(Q_manual - Q_builtin);
```

```
     0.67        0.00           0
    -1.68       -0.00        0.00
    -0.84       -0.00           0
```

```

% Verifying if R_manual and R_builtin are close
disp('Difference between R_manual and R_builtin:');
```

```
Difference between R_manual and R_builtin:
```

```
disp(R_manual - R_builtin);
```

```
   23.75      0.84      1.85
       0     -0.00      0.00
       0         0         0
```

```

% Verifying A = Q*R for both methods
disp('Verification of A = Q_builtin * R_builtin:');
```

```
Verification of A = Q_builtin * R_builtin:
```

```
disp(Q_builtin * R_builtin);
```

```
    4.00    -10.00      4.00
  -10.00     -8.00     -4.00
   -5.00      7.00      9.00
```

```

disp('Verification of A = Q_manual * R_manual:');
```

```
Verification of A = Q_manual * R_manual:
```

```
disp(Q_manual * R_manual);
```

```
    4.00    -10.00      4.00
  -10.00     -8.00     -4.00
   -5.00      7.00      9.00
```

```

% Check orthogonality of Q_manual (Q should be orthogonal)
disp('Q_manual'' * Q_manual (Should be close to identity matrix):')
    ;
```

```
Q_manual' * Q_manual (Should be close to identity matrix):
```

```
disp(Q_manual' * Q_manual);
```

```
    1.00      0.00     -0.00
    0.00      1.00         0
   -0.00         0      1.00
```

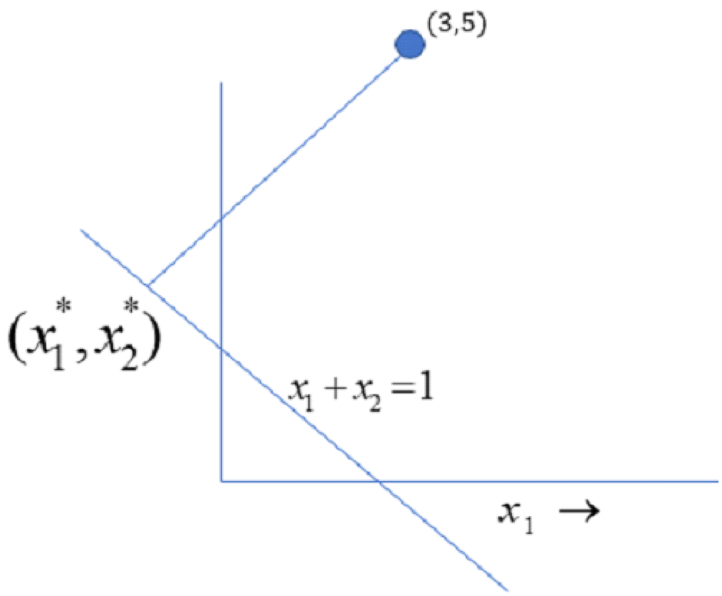
RESULTS

QR decomposition of a matrix A is discussed in detail.

13 | Assignment 20 One Problem And Three Solution Methodology

From point (3,5) a line is drawn perpendicular to the line $x_1 + x_2 = 1$. What is the foot of the perpendicular?

In other words, which point on line $x_1 + x_2 = 1$ is nearest to the point (3,5)?



Three solution from different domains

Co-ordinate Geometry

Slope m_1 of the line joining the points (3,5) with (x_1^*, x_2^*), is

$m_1 = \frac{5-x_2^*}{3-x_1^*}$

Slope of the line $x_1 + x_2 = 1$ is $m_2 = -1$

$m_1 * m_2 = -1 == \frac{5-x_2^*}{3-x_1^*}$ (-1) = -1

$5 - x_2^* = 3 - x_1^*$; also $x_1^* + x_2^* = 1$

$x_1^* + x_2^* = 1$

$x_1^* - x_2^* = -2$

== $x_1^* = \frac{-1}{2}$; $x_2^* = \frac{3}{2}$

Calculus

$$x^* = (x_1^*, x_2^*) = \arg\min_x (x_1 - 3)^2 + (x_2 - 5)^2$$

$$\textit{subject} \text{ to } x_1 + x_2 = 1$$

It is of the form

$$x^* = \arg\min_x f(x)$$

$$\textit{subject} \text{ to } g(x) = 0$$

$$f(x) = (x_1 - 3)^2 + (x_2 - 5)^2$$

$$g(x) = x_1 + x_2 - 1$$

At the optimal point, $x^* = \begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix}$

$$\nabla f(x^*) = \begin{pmatrix} 2(x_1^* - 3) \\ 2(x_2^* - 5) \end{pmatrix}; \quad \nabla g(x^*) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\nabla f(x^*) = \lambda \nabla g(x^*) \Rightarrow \begin{pmatrix} 2(x_1^* - 3) \\ 2(x_2^* - 5) \end{pmatrix} = \lambda \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Rightarrow \begin{cases} x_1^* - 3 = \lambda / 2 \\ x_2^* - 5 = \lambda / 2 \end{cases}$$

Also $x_1^* + x_2^* = 1 \Rightarrow x_2^* = 1 - x_1^*$

On substituting $\lambda = -7$;

$$x_1^* = \frac{-1}{2}; \quad x_2^* = \frac{3}{2}$$

Linear Algebra

From the diagram, it can be observed that the required solution vector (point) is the sum of two vectors.

Let the constrained be expressed as $Ax = b$.

The two vectors are given by.

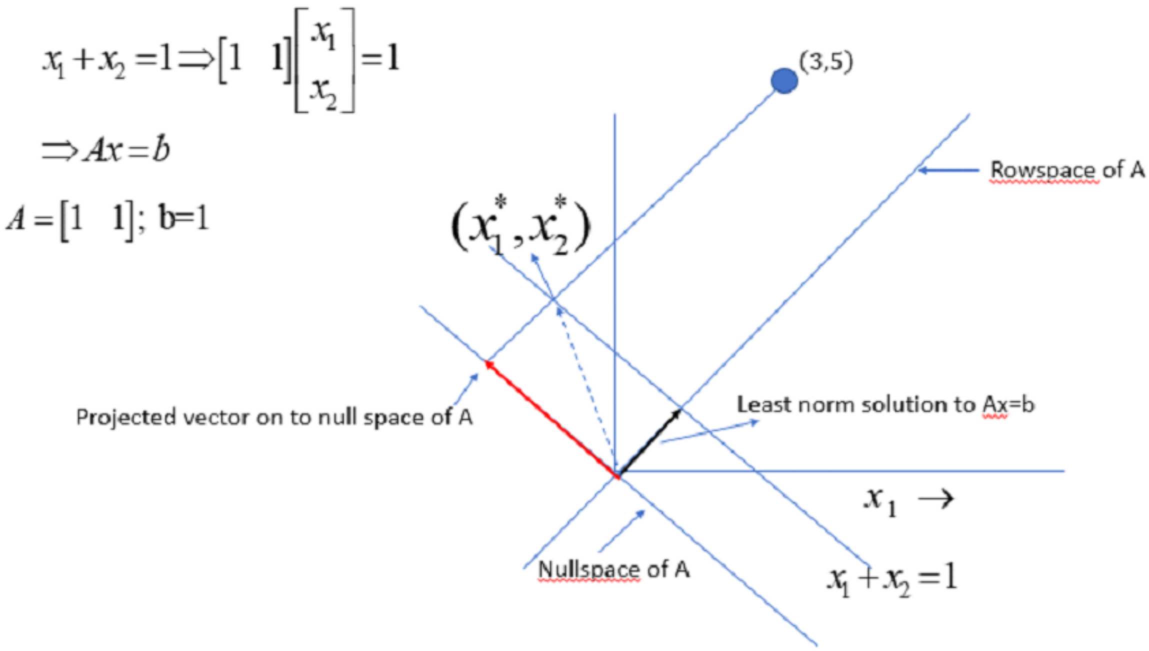
Least norm solution to $Ax = b$. This is given by `pinv(A)*b`.

In the diagram, it is the black-colored vector from the origin to the line $x_1 + x_2 = 1$.

Projected vector of the given point $y = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$ (also a vector) on to null-space of A . Null space is spanned by vector $[1 - 1]^T$. This can be easily checked. Let it be represented by column vector B. B= $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ .Projecting onto null-space of A is equivalent to projecting onto the column space of B. The projection matrix is given by $P = B\left(B^T B\right)^{-1} B^T$

So the projected vector is given by P*y.

In the diagram, it is the red vector.



Let us now compute.

$$x_{ls} = pinv(A) * b = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$$

$$y_{proj} = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 5 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}; \text{ y is } \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = x_{ls} + y_{proj} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{pmatrix} -1/2 \\ 3/2 \end{pmatrix}$$

Assignment Problem:

Now suppose the problem is:

$$x^* = \arg\min_x \|y - x\|^2$$

$$subject\ to\ Ax = b$$

$$where \quad y = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}; \ A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 4 \end{bmatrix}; \ x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}; \ b = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

We will find that the first two methods are very tedious. For this problem, Lagrangian formulation gives only the condition to be satisfied at the optimal point. It does not give a solution.

Linear algebra readily gives a solution.

But you check whether you can geometrically visualize the problem in 3D.

In this problem, $Ax = b$ has an infinite solution (on the meeting edge of two planes corresponding to two equations) . These solutions lie on a line. You are asked to find the solution point nearest to point y.

You may use the `matlab` command null command to get null space basis vectors of A in B.

Find the solution.

```
M=[1 1 2; 1 2 4];
b=[4;7];
y=[3; 4; 5];
x_is=pinv(M)*b;
B=null(A);
y_proj=B*inv(B'*B)*B'*y;
xt=x_is+y_proj
```

```
xt = 3x1
          1.00
          0.60
          1.20
```

RESULTS

The problem of projection and shortest distance from a point to a plane is viewed through three lenses: coordinate geometry, linear algebra, and constrained optimization.