SCHOOL OF ARTIFICIAL INTELLIGENCE

24MA602 Computational Mathematics for Data Science

Assignment Set-1

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-1 Fundamentals of Linear Algebra

1.1 Linear Independence

1. Check whether the following set of vectors are independent or not.

$$\left\{ \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} \right\}$$

SOLUTION

Concept used: Three vectors v_1, v_2 and v_3 are linearly dependent then we can find non-zero scalars, α and β such that, $v_3 = \alpha v_1 + \beta v_2$. In matrix form, it can be expressed as:

$$\begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \beta \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

This system can be written as:

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

$$\therefore \quad \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

In Matlab, the `pinv(A)` function can be used to find the pseudo inverse of the matrix A. The Input program 1 will produce the values of α and β.

</> **Input program 1:** *Checking Linear independence* </>

```matlab
v_1 = [1; 2; 3];   % First vector
v_2 = [3; 4; 5];   % Second vector
v_3 = [2; 2; 2];   % Third vector

A = [v_1, v_2];

% Solve the system [alpha; beta] = A^{-1}.v_3
coefficients = pinv(A) * v_3;
```

```
% Extract alpha and beta
alpha = coefficients(1);
beta = coefficients(2);
disp('Solution for the system v_3 = alpha * v_1 + beta * v_2: is');
fprintf('alpha = %.4f\n', alpha);
fprintf('beta = %.4f\n', beta);
```

Output of the program is shown below.

```
Solution for the system v_3 = alpha * v_1 + beta * v_2: is
alpha = -1.0000
beta = 1.0000
```

RESULTS

Since the scalars α and β are nonzero, the vector v_3 can be expressed as a linear combination of v_1 and v_2. So the given vectors are not linearly independent.

1.2 Solution of System of Linear Equations

2. If $A = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 6 & 8 \end{bmatrix}$, then for any random vector b of the form $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, there is a solution $x = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ for Ax = b. Give reason. If it has infinite solution, write a program to list 10 solutions.

SOLUTION

Here the system, $Ax = b$ can be written in matrix form as:

$$\begin{bmatrix} 1 & 2 & 4 \\ 3 & 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

The augmented matrix, $K = \begin{bmatrix} 1 & 2 & 4|b_1 \\ 3 & 6 & 8|b_2 \end{bmatrix}$

This system is consistent if $\rho(A) = \rho(K)$. Using elementary transformations, the augmented matrix, K is reduced into:

$$\begin{aligned} K &= \begin{bmatrix} 1 & 2 & 4|b_1 \\ 3 & 6 & 8|b_2 \end{bmatrix} \\ &\sim \begin{bmatrix} 1 & 2 & 4|b_1 \\ 0 & 0 & -4|b_2 - 3b_1 \end{bmatrix} \quad R_2 \longrightarrow R_2 - 3R_1 \end{aligned}$$

Now the matrix K is in the row reduced echelon form. So rank of the matrix is the number of non-zero rows. Hence $\rho(A) = \rho(K) = 2 <$ number of unknowns. This implies that the system is consistent but many solutions. The matlab code to find rank of A is shown in Input program 2.

Input program 2: *Finding solution of linear system*

```
A = [1, 2, 4;
     3, 6, 8];
rank_A = rank(A);
fprintf('The rank of matrix A is:%.1f', rank_A);
```

```
The rank of matrix A is:2.0
```

So irrespective to the values of b, $\rho(K) = 2$. Hence the system has many solutions.

From the row reduced form, the given system can be reformulated as:

$$\begin{bmatrix} 1 & 2 & 4 \\ 0 & 0 & -4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - 3b_1 \end{bmatrix}$$

This is a simultaneous system of two equations in three variables. So, we can choose 1 variable y (say) can take arbitrary value a.

Solving this reduced system, we get:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = b_1 \begin{bmatrix} -2 \\ 0 \\ \frac{3}{4} \end{bmatrix} + a \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} + b_2 \begin{bmatrix} 1 \\ 0 \\ -\frac{1}{4} \end{bmatrix}$$

Corresponding to different values of b_1, a, and b_2 we will get infinite number of solutions. Computationally we can find 10 such solutions using the `Matlab` code in Input program 3.

Input program 3: *Listing 10 solutions*

```
A = [1, 2, 4;
     3, 6, 8];
num_sol = 10;
X_val = zeros(3, num_sol);
b_val = zeros(2, num_sol);
reconstructed_b_val = zeros(2, num_sol);
errors = zeros(1, num_sol);
for i = 1:num_sol
    b = rand(2, 1);
    disp(['Randomly chosen vector b ' num2str(i) ':']);
    fprintf("%.3f\t",b);
    X = pinv(A) * b;
    X_val(:, i) = X;
    b_val(:, i) = b;
    b_reconstructed = A * X;
    reconstructed_b_val(:, i) = b_reconstructed;
    errors(i) = norm(b - b_reconstructed);
end
T = table((1:num_sol)', ...
```

```
            b_val(1,:)', b_val(2,:)', ...
            reconstructed_b_val(1,:)', reconstructed_b_val(2,:)', ...
            X_val(1,:)', X_val(2,:)', X_val(3,:)', ...
            errors', ...
            'VariableNames', {'Solution_Num', 'Actual_b1', 'Actual_b2',
             ↪  ...
                              'Reconstructed_b1', 'Reconstructed_b2', ...
                              'X1', 'X2', 'X3', 'Error'});


disp('Table of actual b values, reconstructed b values, solutions, and
 ↪  errors:');
```

The 10 random solutions with reconstruction error in b are shown below.

```
Table of actual b values, reconstructed b values, solutions, and errors:

    Solution_Num    Actual_b1    Actual_b2    Reconstructed_b1    Reconstructed_b2       X1           X2           X3          Error
    ------------    ---------    ---------    ----------------    ----------------    ---------    ---------    ---------    ----------

          1            0.2638     0.14554          0.2638             0.14554         -0.076413     -0.15283      0.16147    3.3307e-16
          2           0.13607     0.86929         0.13607             0.86929           0.11943      0.23886     -0.11527    1.2413e-16
          3            0.5797     0.54986          0.5797             0.54986          -0.12191     -0.24382      0.29731    1.1102e-16
          4           0.14495     0.85303         0.14495             0.85303           0.11262      0.22525     -0.10454    1.2413e-16
          5           0.62206     0.35095         0.62206             0.35095          -0.17863     -0.35726       0.3788    1.5701e-16
          6           0.51325     0.40181         0.51325             0.40181          -0.12494     -0.24988      0.28449    3.5108e-16
          7          0.075967     0.23992        0.075967             0.23992          0.017597     0.035193    -0.003004    1.3878e-17
          8           0.12332     0.18391         0.12332             0.18391         -0.012546    -0.025092     0.046512    3.9252e-17
          9           0.23995     0.41727         0.23995             0.41727         -0.012528    -0.025055     0.075648    1.2413e-16
         10          0.049654     0.90272        0.049654             0.90272           0.16068      0.32136     -0.18844    1.5701e-16
```

It is found that the pseudo inverse function does it's job of better least square approximation of inverse of A with minimum error.

RESULTS

(a) Since $\rho(A) = \rho(K)$, the system has a solution.

(b) The 10 random solutions are given in the Table 1.1.

Table 1.1: Random solutions $X = [x_1, x_2, x_3]$.

Sl.No	x_1	x_2	x_3
1.	-0.076413	-0.15283	0.1647
2.	0.11943	0.23886	-0.11527
3.	-0.12191	-0.24382	0.29731
4.	0.11262	0.22525	-0.10454
5.	-0.17863	-0.35726	0.3788
6.	-0.12494	-0.24988	0.28449
7.	0.017597	0.035193	-0.003004
8.	-0.012546	-0.025092	0.046512
9.	-0.012528	-0.025055	0.075648
10.	0.16068	0.32136	-0.18844

1.3 General Solution of System of Linear Equations

1.3.1 General solution from particular solution

3. (a) Write a program to compute 100 solutions to $x + y + z = 100$.

SOLUTION

Here the equation can be written as, $Ax = b$. Where $A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ and $b = [100]$. Since $\rho(A) = 1$ and using rank nullity theorem, $dim(N(A)) = 3 - 1 = 2$. So the basis of $N(A)$ contains two vectors orthogonal to the row space of A. Using Gauss-elimination method, the solution of the system, $AX = 0$ can be written as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = a \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} + b \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}.$$

So for any given particular solution, X_p of $AX = b$, the general solution of the system can be written as $X = X_p + a \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} + b \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$.

Since X_p is a solution of $AX = b$ and orthogonal to the elements in the $N(A)$, $AX = AX_p = b$. Hence it is clear that both X_p and $X \in$ Row Space(A).

The `Matlab` code to find the general solution using a particular solution and the basis of $N(A)$ is shown in Input program 4.

Input program 4: *General solution from particular solution*

```
A = [1, 1, 1];
b = 100;
x_p = pinv(A') * b;
null_space_A = null(A, 'r');
num_sol = 5;
alpha = randn(size(null_space_A, 2), 1);
x = transpose(x_p) + null_space_A * alpha;
disp('Particular solution x_p:');
disp(x_p);
disp('Null space basis N(A):');
disp(null_space_A);
disp('General solutions (x_p + alpha1 * N(A1) + alpha2 * N(A2)) for
    random alpha values:');
dis(x);
```

Output of the code is given below.

```
Particular solution x_p:

33.3333   33.3333   33.3333
Null space basis N(A):

-1    -1
 1     0
 0     1

 General solutions (x_p + alpha1 * N(A1) + alpha2 * N(A2)) for random alpha valu
   35.9393
   31.7465
   32.3142
```

Hundred solutions can be generated using the formula $X = X_p + \alpha N_1 + \beta N_2$ using the `Matlab` code as follows:

Input program 5: *Listing 100 Solutions*

```
A = [1 1 1];
b = 100;
x_p = pinv(A) * b;
null_space = null(A, 'r');
num_solutions = 100;
solutions = zeros(num_solutions, 3);
verification = zeros(num_solutions, 1);
sl_no = (1:num_solutions)';
for i = 1:num_solutions
    alpha = randn(2, 1);
    solutions(i, :) = x_p' + alpha(1) * null_space(:, 1)' + alpha(2) * null_space(:, 2)';
    verification(i) = A * solutions(i, :)';
end
T = table(sl_no, solutions(:, 1), solutions(:, 2), solutions(:, 3), verification, ...
    'VariableNames', {'Sl_No', 'x', 'y', 'z', 'A_x'});
disp('Table of 100 Solutions:');
disp(T);
```

RESULTS

The 100 general solutions of the system $x + y + z = 100$ is given by:

Sl_No	x	y	z	A_x
1	34.655	33.533	31.812	100
2	34.65	32.61	32.74	100
3	31.99	33.735	34.275	100
4	33.406	33.634	32.96	100
5	31.719	34.149	34.132	100
6	32.642	33.454	33.905	100
7	33.907	33.746	32.346	100
8	33.231	34.093	32.676	100
9	33.76	32.729	33.51	100
10	33.773	33.026	33.202	100
11	31.691	33.929	34.38	100
12	33.204	33.135	33.661	100
13	33.342	33.095	33.563	100
14	33.51	33.773	32.716	100
15	32.457	33.608	33.934	100
16	31.511	33.426	35.063	100
17	34.679	32.725	32.596	100
18	34.173	31.583	34.244	100
19	32.546	34.2	33.253	100
20	32.251	34.232	33.517	100
21	32.93	33.624	33.446	100
22	32.792	33.773	33.435	100
23	31.713	36.121	32.167	100
24	36.328	31.479	32.193	100
25	34.86	32.24	32.9	100
26	33.72	33.165	33.115	100

27	32.403	33.875	33.723	100
28	30.804	34.085	35.112	100
29	33.394	34.556	32.05	100
30	34.76	31.004	34.235	100
31	35.102	31.498	33.4	100
32	31.071	33.369	35.561	100
33	33.91	33.264	32.826	100
34	32.852	33.569	33.579	100
35	33.872	33.403	32.725	100
36	34.239	32.111	33.65	100
37	35.708	31.99	32.301	100
38	32.421	34.665	32.914	100
39	32.574	33.193	34.233	100
40	32.604	33.033	34.363	100
41	32.666	32.988	34.346	100
42	32.917	33.963	33.12	100
43	35.242	32.468	32.29	100
44	34.042	33.063	32.895	100
45	32.758	32.925	34.317	100
46	32.487	33.036	34.477	100
47	32.892	32.802	34.306	100
48	33.679	32.811	33.51	100
49	32.777	34.304	32.919	100
50	31.768	32.895	35.337	100
51	32.814	34.284	32.901	100
52	33.044	33.982	32.973	100
53	31.212	34.039	34.749	100
54	33.909	31.729	34.362	100
55	31.828	34.791	33.381	100
56	31.432	35.08	33.489	100
57	36.764	32.096	31.14	100
58	32.953	33	34.047	100
59	32.602	33.651	33.747	100
60	33.766	32.756	33.477	100
61	35.732	31.695	32.573	100
62	33.632	32.515	33.853	100
63	34.503	33.319	32.178	100
64	34.033	33.324	32.644	100
65	33.136	32.667	34.197	100
66	32.822	33.447	33.732	100
67	32.269	34.217	33.514	100
68	32.1	33.884	34.016	100
69	31.687	34.504	33.809	100
70	31.898	34.746	33.356	100
71	31.68	33.285	35.035	100
72	33.846	32.824	33.33	100
73	32.264	34.253	33.483	100
74	30.894	34.738	34.367	100
75	33.819	33.625	32.556	100
76	34.149	33.9	31.951	100
77	32.28	33.578	34.142	100
78	32.241	33.546	34.213	100
79	30.371	35.372	34.257	100

80	32.425	33.6	33.975	100
81	34.223	33.759	32.019	100
82	32.525	32.917	34.558	100
83	32.794	33.29	33.916	100
84	34.275	32.327	33.398	100
85	34.095	33.934	31.972	100
86	33.168	33.681	33.151	100
87	34.31	32.394	33.296	100
88	37.358	31.437	31.205	100
89	35.501	32.156	32.343	100
90	36.232	32.16	31.608	100
91	34.639	33.622	31.739	100
92	32.436	33.444	34.12	100
93	33.242	33.331	33.426	100
94	35.194	32.955	31.851	100
95	32.416	33.29	34.294	100
96	32.025	35.072	32.903	100
97	34.794	31.706	33.5	100
98	33.184	33.71	33.106	100
99	32.458	32.184	35.358	100
100	36.203	30.974	32.823	100

1.3.2 Nature of particular solution

(b) Check whether x_p is in row space or not.

SOLUTION

Solution: For this, we have to check whether $Ax_p = b$. Since X_p is a solution of the given equation, x_p satisfies $Ax_p = b$. Hence $X_p \in$ Row Space(A). We can verify this with the `Matlab` code in Input program 6.

Input program 6: *verification of $AX_p = b$*

```
disp(x_p);
A*transpose(x_p)
```

Output of the code is given below.

```
33.3333    33.3333    33.3333
ans = 100
```

RESULTS

Since $Ax_p = 100$, the particular solution, $x_p \in$ Row Space(A).

1.4 Role of Column Space in Consistency of a System

4. Create a vector b such that

(a) $\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = b$ has a solution.

SOLUTION

The system, $AX = b$ has a solution if b is in the column space of A. Since $\rho(A) = 2$, the column space is of rank 2. Hence the entire columns of A constitute the column space. Now any element in the column space can be generated as a linear combination of columns of A. In summary to create a vector b that satisfies the given condition, it is enough to create b as a linear combination of columns of A. The `Matlab` code for this task is given in Input program 7.

Input program 7: *Generating b as linear combination of columns of A*

```
A = [1 3; 2 4; 3 5];

alpha = rand(2, 1);
b = A * alpha;
disp('Vector b:');
disp(b);
A_pinv = pinv(A);
x = A_pinv * b;
b_computed = A * x;
disp('Solution x using pseudo-inverse:');
disp(x);
disp('Computed b from A x:');
disp(b_computed);
error_vector = abs(b - b_computed);
T = table(b, b_computed, error_vector, ...
    'VariableNames', {'Original_b', 'Reconstructed_b', 'Error in components'});

disp('Table of Results:');
disp(T);
```

```
Vector b:
    1.5024
    2.3294
    3.1564
Solution x using pseudo-inverse:
    0.4893
    0.3377
Computed b from A x:
    1.5024
    2.3294
    3.1564
Table of Results:
    Original_b    Reconstructed_b    Error in components
    __________    _______________    ___________________

      1.5024          1.5024             8.8818e-16
      2.3294          2.3294             1.3323e-15
      3.1564          3.1564             1.7764e-15
```

In this problem, we created b as a linear combination of columns of A. Using this b, find a solution of the system and verify that reconstructed version of b shows significantly small deviations from the original b. This confirms that the solution using the pseudo inverse of A works well.

RESULTS

The value of b for which the system has a solution is $b = \begin{pmatrix} 1.5024 \\ 2.3294 \\ 3.1564 \end{pmatrix}$.

(b) $\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = b$ has no solution.

SOLUTION

When the system $Ax = b$ has no exact solution, the `b` vector will lie outside the column space of A. We can generate such a `b` by choosing it such that it does not lie in the span of the columns of A. The `Matlab` code for this task is given in Input program 37.

Input program 8: *Find a vector b such that $AX = b$ has no solution*

```matlab
A = [1 3; 2 4; 3 5];
b = [1; 1; 10];
A_pinv = pinv(A);
x = A_pinv * b;
b_computed = A * x;
error_vector = abs(b - b_computed);
T = table(b, b_computed, error_vector, ...
    'VariableNames', {'Original_b', 'Reconstructed_b', 'Error in components'});
disp('Table of Results:');
disp(T);
```

```
Table of Results:
    Original_b    Reconstructed_b    Error in components
    __________    _______________    ___________________

         1             -0.5                  1.5
         1                4                    3
        10              8.5                  1.5
```

Conclusion: Here this significant difference between the original and reconstructed bs are due the the wrong assumption that the system has a solution for the choice of b. Hence it is verified that whenever the column vector b is not in the column space of A, the system has no solution.

RESULTS

The value of b for which the system has no solution is $b = \begin{pmatrix} 1 \\ 1 \\ 10 \end{pmatrix}$.

(c) $\begin{bmatrix} 1 & 3 & 4 \\ 2 & 1 & 3 \\ 3 & 2 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = b$ has a solution.

SOLUTION

As in the case (a) of this problem, the system $AX = b$ has a solution, only if b in in the column space of A. Thus find a vector b such that the system $Ax = b$ has a solution for the matrix, $A = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 1 & 3 \\ 3 & 2 & 5 \end{bmatrix}$, we express b as a linear combination of the columns of A.

Let α, β, γ be scalar values, then b can be written as $b = \alpha \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \beta \cdot \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} + \gamma \cdot \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix}$.

Here we verify our claim by randomly generate b as linear combination of the columns and cross check the same b can be recreated from the solution of the system $Ax = b$ with significantly small errors. The `Matlab` code for this task is shown in Input program 9.

Input program 9: *Find a vector b such that $Ax = b$ has a solution*

```
A = [1 3 4; 2 1 3; 3 2 5];
alpha = rand(3, 1);
b = A * alpha;
disp('Vector b:');
disp(b);
A_pinv = pinv(A);
x = A_pinv * b;
b_computed = A * x;
disp('Solution x using pseudo-inverse:');
disp(x);
disp('Computed b from A x:');
disp(b_computed);
error_vector = abs(b - b_computed);
T = table(b, b_computed, error_vector, ...
    'VariableNames', {'Original_b', 'Reconstructed_b', 'Error in
    components'});
disp('Table of Results:');
disp(T);
```

9

```
Vector b:
    2.4526
    2.5030
    3.9947
Solution x using pseudo-inverse:
    0.5140
   -0.0168
    0.4972
Computed b from A x:
    2.4526
    2.5030
    3.9947
Table of Results:
    Original_b    Reconstructed_b    Error in components
    __________    _______________    ___________________

      2.4526          2.4526              1.3323e-15
       2.503           2.503              4.4409e-16
      3.9947          3.9947                       0
```

In this problem, we created b as a linear combination of columns of A. Using this b, find a solution of the system and verify that reconstructed version of b shows significantly small deviations from the original b. This confirms that the solution using the pseudo inverse of A works well.

RESULTS

The value of b for which the system has a solution is $b = \begin{pmatrix} 2.4536 \\ 3.503 \\ 3.9947 \end{pmatrix}$.

(d) $\begin{bmatrix} 1 & 3 & 4 \\ 2 & 1 & 3 \\ 3 & 2 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = b$ has no solution.

SOLUTION

To find a vector b such that the system $Ax = b$ has no solution, we need to ensure that b does not lie in the column space of A. So, we will write a b such that it will not be a linear combination of column space of A. Then we verify that if such a b is chosen, then the reconstructed column vector, b from the solution of $AX = b$ will be significantly different from the chosen column vector b. This error is due to the wrong assumption that the system has a solution. The `Matlab` code for this task is shown in Input program 10.

</> Input program 10: *Find a vector b such that AX = b has no solution* **</>**

```
A = [1 3 4; 2 1 3; 3 2 5];
b = [10; 15; 20];
disp('Vector b:');
disp(b);
A_pinv = pinv(A);
x = A_pinv * b;
b_computed = A * x;
disp('Solution x using pseudo-inverse:');
disp(x);
disp('Computed b from A * x:');
disp(b_computed);
error_vector = abs(b - b_computed);
T = table(b, b_computed, error_vector, ...
    'VariableNames', {'Original_b', 'Reconstructed_b',
     'Error_in_components'});
disp('Table of Results:');
disp(T);
```

```
Vector b:
    10
    15
    20
Solution x using pseudo-inverse:
    3.7333
   -1.2667
    2.4667
Computed b from A * x:
    9.8000
   13.6000
   21.0000
Table of Results:
    Original_b    Reconstructed_b    Error_in_components
    __________    _______________    ___________________
```

```
10          9.8          0.2
15         13.6          1.4
20           21            1
```

Conclusion: Here this significant difference between the original and reconstructed bs are due the wrong assumption that the system has a solution for the choice of b. Hence it is verified that whenever the column vector b is not in the column space of A, the system has no solution.

RESULTS

The value of b which for the system doesn't have a solution is $b = \begin{pmatrix} 10 \\ 15 \\ 20 \end{pmatrix}$

2 | Assignment-2
Linear Algebra and Optimization for AI and Data Science

2.1 Rank of Matrix

1. For the following matrix $A = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & 2 & 3 \\ 1 & -1 & 0 & 1 \end{bmatrix}$,

   (a) Find, rank of the matrix A.

   SOLUTION

   Rank of a matrix A is the number of linearly independent rows or columns. It is equivalent to the number of non-zero rows in the row reduced Echelon form of A. The row reduced Echelon form of A can be found as follows.

$$
\begin{aligned}
A &= \begin{bmatrix} 1 & 1 & 2 & 3 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & 2 & 3 \\ 1 & -1 & 0 & 1 \end{bmatrix} \\
&\sim \begin{bmatrix} 1 & 1 & 2 & 3 & \\ 0 & -2 & -2 & -2 & R_2 \longrightarrow R_2 - R_1 \\ 0 & 0 & 0 & 0 & R_3 \longrightarrow R_3 - R_1 \\ 0 & -2 & -2 & -2 & R_4 \longrightarrow R_4 - R_1 \end{bmatrix} \\
&\sim \begin{bmatrix} 1 & 1 & 2 & 3 & \\ 0 & -2 & -2 & -2 & R_2 \longrightarrow R_2 - R_1 \\ 0 & -2 & -2 & -2 & R_4 \longrightarrow R_4 - R_1 \\ 0 & 0 & 0 & 0 & R_3 \longleftrightarrow R_4 \end{bmatrix} \\
&\sim \begin{bmatrix} 1 & 1 & 2 & 3 & \\ 0 & 1 & 1 & 1 & R_2 \longrightarrow R_2/-2 \\ 0 & 1 & 1 & 1 & R_3 \longrightarrow R_3/-2 \\ 0 & 0 & 0 & 0 & \end{bmatrix} \\
&\sim \begin{bmatrix} 1 & 1 & 2 & 3 & \\ 0 & 1 & 1 & 1 & \\ 0 & 0 & 0 & 0 & R_3 \longrightarrow R_3 - R_2 \\ 0 & 0 & 0 & 0 & \end{bmatrix} \\
&- \text{A is in RREF}
\end{aligned}
$$

   Hence rank, $\rho(A)$= number of non-zero rows in RREF= 2

2.2 Basis of Rowspace

(b) Create a basis set for row-space from the set of row vectors itself. How many such different basis sets are possible.

SOLUTION

From the first part of the question, it is clear that only first two rows are linearly independent and spans the entire row space. So the basis for row space is $\{\begin{bmatrix}1 & 1 & 2 & 3\end{bmatrix}, \begin{bmatrix}1 & -1 & 0 & 0\end{bmatrix}\}$. In this case, there are only two distinct linearly independent rows after row reduction. Thus, only one basis set is possible for the row space, given that the first two rows themselves determine the unique independent set.

2.3 Basis of Column space

(c) Create a basis set for column-space from the set of column vectors itself. How many such different basis sets are possible. List all such basis sets.

SOLUTION

From the row-reduced Echelon form, we can see that the first and second columns contain pivots. These columns correspond to the linearly independent columns in the original matrix.

Thus, the basis for the column space of A is formed by $\left\{\begin{bmatrix}1\\1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\-1\\1\\-1\end{bmatrix}\right\}$. There is only one basis set for the column space of A and is $\left\{\begin{bmatrix}1\\1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\-1\\1\\-1\end{bmatrix}\right\}$.

(d) Create basis set for right null space and left null-space.

SOLUTION

The right null space is defined as the set of all vectors X such that: $Ax = 0$. This is equivalent to solving the reduced homogeneous system of linear equations: $\begin{bmatrix}1 & 1 & 2 & 3\\0 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\w\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$. Choosing w and z as free variables (choose $w = a, z = b$), the two-parameter family of solution is given by:

$$\begin{bmatrix}x\\y\\z\\w\end{bmatrix} = a\begin{bmatrix}-2\\-1\\0\\1\end{bmatrix} + b\begin{bmatrix}-1\\-1\\1\\0\end{bmatrix}$$

This indicate that the entire right null space, $N(A)$ will be spanned by the linearly independent vectors $\left\{\begin{bmatrix}-2\\-1\\0\\1\end{bmatrix}, \begin{bmatrix}-1\\-1\\1\\0\end{bmatrix}\right\}$. It is the basis for $N(A)$.

Similarly the left null space is generated by the linearly independent solutions of $A^T X = 0$.

The RREF of A^T is given by:

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 2 & 0 & 2 & 0 \\ 3 & 1 & 3 & 1 \end{bmatrix}$$
$$\sim \begin{bmatrix} 1 & 1 & 1 & 1 & \\ 0 & -2 & 0 & -2 & R_2 \longrightarrow R_2 - R_1 \\ 0 & -2 & 0 & -2 & R_3 \longrightarrow R_3 - R_1 \\ 0 & -2 & 0 & -2 & R_4 \longrightarrow R_4 - R_1 \end{bmatrix}$$
$$\sim \begin{bmatrix} 1 & 1 & 1 & 1 & \\ 0 & 1 & 0 & 1 & R_2 \longrightarrow R_2 / -2 \\ 0 & -2 & 0 & -2 & \\ 0 & -2 & 0 & -2 & \end{bmatrix}$$
$$\sim \begin{bmatrix} 1 & 1 & 1 & 1 & \\ 0 & 1 & 0 & 1 & \\ 0 & 0 & 0 & 0 & R_3 \longrightarrow R_3 + 2R_2 \\ 0 & 0 & 0 & 0 & R_4 \longrightarrow R_4 + 2R_2 \end{bmatrix}$$

Now the reduced form of $A^T X = 0$ will be of the form:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Choosing z and w as free variables, $(w = a, z = b)$, the two parameter solution of the system is gien by:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = a \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} + b \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

This means that entire left null space $N(A^T)$ is spanned by the linearly independent vectors $\left\{ \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\}$. So the basis for the left null space is $\left\{ \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\}$.

2.4 Dimensions of Fundamental Subspaces

(e) What is the dimension of (1) row space (2) Column space (3) Left null space (4) Right null space of A.

SOLUTION

Dimension of a vector space is the cardinality of its basis. So, from part (a), (b), (c) and (d) of the solution, it is clear that:

$$dim(\text{row space}) = 2$$
$$dim(\text{column space}) = 2$$
$$\text{dim}(\text{left null space}) = 2$$
$$dim(\text{right null space}) = 2$$

2. (a) Using two Matlab commands, Create a 4×4 symmetric matrix A of rank 2.

SOLUTION

For any linearly independent vectors a, b, $A = aa^T + bb^T$ will be a symmetric matrix of rank 2. Consider two vectors a and b created from random numbers and find $A = aa^t + bb^T$ using following Matlab code.

</> **Input program 11:** *Creating symmetric matrix using outer product of vectors* </>

```matlab
a = randn(4, 1);
b = randn(4, 1);
A = a * a' + b * b';
disp('Symmetric matrix A of rank 2:');
disp(A);
rank_A = rank(A);
disp('Rank of matrix A:');
disp(rank_A);
```

```
Symmetric matrix A of rank 2:

   13.3311    9.8644   -4.3120   10.7115
    9.8644    7.6738   -3.7835    8.4180
   -4.3120   -3.7835    2.3331   -4.2433
   10.7115    8.4180   -4.2433    9.2528
Rank of matrix A:

  2
```

2.5 Dimensions of Subspaces of Symmetric Matrices

(b) Demonstrate that for symmetric matrices, rowspace=column space and leftnullspace=rightnullspace. (show that they share identical basis set).

SOLUTION

Let A be a symmetric matrix. So $A = A^T$. This implies that the entries of A are symmetric across the diagonal, which leads to the equivalence of the row space and column space. This property can be demonstrated through the following example.

Consider the symmetric matrix, $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix}$. Applying the elementary row transformations, A will be reduced into the RREF as:

$$\begin{aligned} A &= \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

So all the rows of A are linearly independent and so the row space of A will be

$$\{\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 1 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}\}$$

Since each column of the RREF(A) contains the pivot element, the column space of A will the entire columns of A itself.

$$\therefore \text{Column Space}(A) = \left\{ \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \right\}$$

So it is clear that both Column Space and Row space of A are identical. Since A is a full rank matrix, using the rank nullity theorem, $N(A) = N(A^T) = 0$. For any random symmetric matrix, this property can be demonstrated using `Matlab` code given below.

Input program 12: *Fundamental subspaces of symmetric matrix*

```
a = randn(4, 1);
b = randn(4, 1);
A = a * a' + b * b';
[~, pivot_rows] = rref(A);
row_space_basis = A(pivot_rows, :);
disp("Row Space Basis:")
disp(row_space_basis)
[~, pivot_cols] = rref(A');
column_space_basis = A(:, pivot_cols);
disp("Column space of A:")
disp(column_space_basis)
right_null_space = null(A);
disp("Right null space:")
disp(right_null_space)
left_null_space = null(A');
disp("Left null space:")
disp(left_null_space)
```

```
Row Space Basis:
    0.5682    1.6159   -0.7292    0.1579
    1.6159    8.0440   -0.3859    0.3621
Column space of A:

    0.5682    1.6159
    1.6159    8.0440
   -0.7292   -0.3859
    0.1579    0.3621

Right null space:

   -0.9239    0.0341
    0.1656   -0.0445
   -0.3338    0.1419
    0.0873    0.9883

Left null space:

   -0.9239    0.0341
    0.1656   -0.0445
   -0.3338    0.1419
    0.0873    0.9883
```

2.6 Verification of Rank Nullity Theorem

(c) compute dim(rowspace)+dim(rightnullspace)., here dim stands for dimension.

SOLUTION

Now consider the random symmetric matrix generated by the outer product operation of random vectors, a and b using the formula, $A = a \cdot a^T + b \cdot b^T$. It is a 4×4 matrix.

By rank nullity theorem, dim(Row space)(A)+Nullity(A) = Number of columns of(A). Therefore,

$$\begin{aligned} \text{dim(rowspace)+dim(rightnullspace)} &= 2+2 \\ &= 4 \end{aligned}$$

(d) compute dim(columnspace)+dim(leftnullspace).

SOLUTION

$$\begin{aligned} \text{dim(columnspace)+dim(leftnullspace)} &= 2+2 \\ &= 4 \end{aligned}$$

(e) for a general mxn matrix A, what is

i. dim(rowspace)+dim(rightnullspace) ?

SOLUTION

By rank nullity theorem, for an $m \times n$ matrix,

$$\text{dim(rowspace)+dim(rightnullspace)=number of columns of } A = n.$$

ii. dim(columnspace)+dim(leftnullspace) ?

SOLUTION

By rank nullity theorem, for an $m \times n$ matrix,

$$\text{dim(columnspace)+dim(leftnullspace)=number of rows of } A = m.$$

RESULTS

i. It is verified that, for symmetric matrices,

$$\text{rowspace} = \text{column space}$$

and

$$\text{leftnullspace} = \text{rightnullspace}.$$

ii. Rank nullity theorem for all the four fundamental subspaces are verified for a randomly generated symmetric matrix.

2.7 Creation of Symmetric Matrix

3. Prove that given a general rectangular matrix A , AA^T and $A^T A$ are symmetric.

Proof. Let A be a general rectangular matrix.

Proof that AA^T is symmetric

Let $B = AA^T$. We need to show that $B = B^T$.

$$\begin{aligned} B^T &= \left(AA^T\right)^T \\ &= \left(A^T\right)^T A^T \quad \text{(using the property of transposes)} \\ &= AA^T \end{aligned}$$

Therefore, $B = B^T$, which means AA^T is symmetric.

Proof that $A^T A$ is symmetric

Let $C = A^T A$. We need to show that $C = C^T$.

$$\begin{aligned} C^T &= (A^T A)^T \\ &= A^T (A^T)^T \quad \text{(using the property of transposes)} \\ &= A^T A \end{aligned}$$

Therefore, $C = C^T$, which means $A^T A$ is symmetric. □

Computational proof:

Let A be a 4×3 random matrix. The following `Matlab` code compute both AA^T and $A^T A$. Since the matrix and its order is random, if the property can be generalized to any rectangular matrices.

Input program 13: *For any rectangular matrix A, $A^T A$ and AA^T are symmetric*

```
m = 4;
n = 3;
A = rand(m, n);
AAT = A * A';
ATA = A' * A;
isAATSymmetric = isequal(AAT, AAT');
isATASymmetric = isequal(ATA, ATA');
fprintf('Matrix A:\n');
disp(A);
fprintf('Matrix AA^T:\n');
disp(AAT);
fprintf('Is AA^T symmetric? %d\n', isAATSymmetric);
fprintf('Matrix A^TA:\n');
disp(ATA);
fprintf('Is A^TA symmetric? %d\n', isATASymmetric);
```

```
Matrix A:
    0.2769    0.6948    0.4387
    0.0462    0.3171    0.3816
    0.0971    0.9502    0.7655
    0.8235    0.0344    0.7952
```

```
Matrix AA^T:
    0.7520    0.4005    1.0230    0.6009
    0.4005    0.2483    0.5979    0.3524
    1.0230    0.5979    1.4984    0.7215
    0.6009    0.3524    0.7215    1.3116
Is AA^T symmetric? 1

Matrix A^TA:
    0.7663    0.3277    0.8683
    0.3277    1.4874    1.1806
    0.8683    1.1806    1.5564
Is A^TA symmetric? 1
```

2.8 Properties of Inverse and Transpose

4. If $(PQ)^T = Q^T P^T$ is assumed to be true, show that $\left(A^T\right)^{-1} = \left(A^{-1}\right)^T$ is true for an invertible matrix A.

Proof. Let A be an invertible matrix. We need to show that $\left(A^T\right)^{-1} = \left(A^{-1}\right)^T$.

By definition, if A is invertible, then:

$$AA^{-1} = I \tag{2.1}$$

$$A^{-1}A = I \tag{2.2}$$

where I is the identity matrix with same order of A.

Take the transpose of both sides of (2.1).

$$(AA^{-1})^T = I^T$$
$$\left(A^{-1}\right)^T A^T = I$$

This implies, $(A^{-1})^T$ is the inverse of A^T. So by the definition of the inverse,

$$A^T\left((A^{-1})^T\right) = I$$
$$\implies \left(A^T\right)^{-1} = \left(A^{-1}\right)^T$$

Thus, we have shown that $\left(A^T\right)^{-1} = \left(A^{-1}\right)^T$ for an invertible matrix A. □

2.9 Creation of Orthogonal Matrix from Wave Forms

5. Create a 5×5 orthogonal matrix by sampling 3 cosine waves of wave numbers $0, 1, 2$ and two sine waves of wave numbers 1 and 2. Normalize the row vectors.

SOLUTION

2.9.1 First approach

Consider the partition of interval $[0, 2\pi]$ in to 4 sub-intervals with partitions $x = [0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi]$. So the 5×5 matrix with given wave numbers over this partition is given as:

Original Matrix

$$
\begin{aligned}
M &= \begin{bmatrix} \cos(0\cdot x) \\ \cos(1\cdot x) \\ \cos(2\cdot x) \\ \sin(1\cdot x) \\ \sin(2\cdot x) \end{bmatrix} \\
&= \begin{bmatrix} \cos(0) & \cos(0) & \cos(0) & \cos(0) & \cos(0) \\ \cos(0) & \cos(\pi/2) & \cos(\pi) & \cos(3\pi/2) & \cos(2\pi) \\ \cos(0) & \cos(\pi) & \cos(2\pi) & \cos(3\pi) & \cos(4\pi) \\ \sin(0) & \sin(\pi/2) & \sin(\pi) & \sin(3\pi/2) & \sin(2\pi) \\ \sin(0) & \sin(\pi) & \sin(2\pi) & \sin(3\pi) & \sin(4\pi) \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 0 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

The normalized matrix:

$$
\begin{aligned}
\hat{M} &= \frac{M_i}{||M_i||} \\
&= \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

`Matlab` code for this problem is shown below.

Input program 14: *Creating orthogonal matrix from Fourier components*

```
n = 5;
x = linspace(0, 1, n);
cos_wave_0 = cos(2 * pi * 0 * x);
cos_wave_1 = cos(2 * pi * 1 * x);
cos_wave_2 = cos(2 * pi * 2 * x);

sin_wave_1 = sin(2 * pi * 1 * x);
sin_wave_2 = sin(2 * pi * 2 * x);

matrix = [cos_wave_0; cos_wave_1; cos_wave_2; sin_wave_1; sin_wave_2];
disp("Matrix Created:")
disp(matrix)
for i = 1:n-1
    matrix(i, :) = matrix(i, :) / norm(matrix(i, :));
end
disp('Orthogonal matrix:');
disp(matrix)
```

The normalized matrix (row-wise) is shown below.

```
Matrix Created:

    1.0000    1.0000    1.0000    1.0000    1.0000
```

```
    1.0000    0.0000   -1.0000   -0.0000    1.0000
    1.0000   -1.0000    1.0000   -1.0000    1.0000
         0    1.0000    0.0000   -1.0000   -0.0000
         0    0.0000   -0.0000    0.0000   -0.0000

Normalized matrix:
  0.4472    0.4472    0.4472    0.4472    0.4472
    0.5774    0.0000   -0.5774   -0.0000    0.5774
    0.4472   -0.4472    0.4472   -0.4472    0.4472
         0    0.7071    0.0000   -0.7071   -0.0000
         0    0.0000   -0.0000    0.0000   -0.0000
```

While computing $\hat{M} \cdot \hat{M}'$, we the the following output.

```
  1.0000    0.2582    0.2000   -0.0000   -0.0000
    0.2582    1.0000    0.2582   -0.0000   -0.0000
    0.2000    0.2582    1.0000   -0.0000   -0.0000
   -0.0000   -0.0000   -0.0000    1.0000   -0.0000
   -0.0000   -0.0000   -0.0000   -0.0000    0.0000
```

Which is not the identity matrix!.

In short, in this method, the 5×5 matrix created is row normalized but it is not orthogonal.

2.9.2 Second approach

We will form a 5×5 matrix using cosine and sine waves of different wave numbers and then apply the Gram-Schmidt orthogonalization process to obtain an orthogonal matrix.

Step 1: Construct the Matrix Using Cosine and Sine Functions

We define a set of x-values as follows:

$$x = \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\}$$

For each of these values, we compute the values of the cosine and sine functions. The wave numbers are 0, 1, and 2 for cosine, and 1 and 2 for sine.

Cosine and Sine Functions

$$\begin{aligned}
\cos(2\pi \cdot 0 \cdot x) &= \{1, 1, 1, 1, 1\} \\
\cos(2\pi \cdot 1 \cdot x) &= \{1, 0, -1, 0, 1\} \\
\cos(2\pi \cdot 2 \cdot x) &= \{1, -1, 1, -1, 1\} \\
\sin(2\pi \cdot 1 \cdot x) &= \{0, 1, 0, -1, 0\} \\
\sin(2\pi \cdot 2 \cdot x) &= \{0, 0, 0, 0, 0\}
\end{aligned}$$

Arranging these into a matrix A:

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

The row-wise normalized matrix is:

$$\hat{M} = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 2: Gram-Schmidt Orthogonalization

We now apply the Gram-Schmidt orthogonalization process to convert matrix $\hat{M}$ into an orthogonal matrix.

Gram-Schmidt Process

Given a set of vectors $v_1, v_2, \ldots, v_n$, the Gram-Schmidt process generates orthogonal vectors $u_1, u_2, \ldots, u_n$ using the following steps:

$$\begin{aligned} u_1 &= v_1 \\ u_2 &= v_2 - \text{proj}_{u_1}(v_2) \\ u_3 &= v_3 - \text{proj}_{u_1}(v_3) - \text{proj}_{u_2}(v_3) \\ &\vdots \\ u_n &= v_n - \sum_{i=1}^{n-1} \text{proj}_{u_i}(v_n) \end{aligned}$$

The projection $\text{proj}_{u_i}(v_j)$ is defined as:

$$\text{proj}_{u_i}(v_j) = \frac{u_i \cdot v_j}{u_i \cdot u_i} u_i$$

This process ensure each rows in $\hat{M}$ will be pair-wise orthogonal. The `Matlab` code for this entire process is show below.

</> **Input program 15:** *Complete code to create orthogonal matrix* </>

```
% Number of samples
n = 5;
x = linspace(0, 1, n);

% Define the cosine and sine waves
cos_wave_0 = cos(2 * pi * 0 * x);
cos_wave_1 = cos(2 * pi * 1 * x);
cos_wave_2 = cos(2 * pi * 2 * x);

sin_wave_1 = sin(2 * pi * 1 * x);
```

```
sin_wave_2 = sin(2 * pi * 2 * x);

% Stack the waves into a matrix
matrix = [cos_wave_0; cos_wave_1; cos_wave_2; sin_wave_1; sin_wave_2];
disp('Matrix from wave forms')
disp(matrix)
% Normalize the matrix rows to ensure orthogonality
for i = 1:n-1
    matrix(i, :) = matrix(i, :) / norm(matrix(i, :));
end

% Display the resulting orthogonal matrix
disp('Normalized matrix:');
disp(matrix)
% Gram-Schmidt orthogonalization process
function Q = gram_schmidt(V)
    [rows, cols] = size(V);
    Q = zeros(rows, cols);
    for i = 1:rows
        % Take the i-th vector
        v = V(i, :);
        % Subtract projections of v onto previous vectors in Q
        for j = 1:i-1
            v = v - (dot(Q(j, :), V(i, :)) / dot(Q(j, :), Q(j, :))) *
             ↪  Q(j, :);
        end
        % Normalize the vector
        Q(i, :) = v / norm(v);
    end
end

% Apply the Gram-Schmidt orthogonalization
orthogonal_matrix = gram_schmidt(matrix);
disp('Orthogonal matrix:');
disp(orthogonal_matrix);
% Check orthogonality
orthogonality_check = orthogonal_matrix * orthogonal_matrix';
disp('Orthogonality check (orthogonal_matrix * orthogonal_matrix^T):');
disp(orthogonality_check);
```

```
Matrix from wave forms:
    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    0.0000   -1.0000   -0.0000    1.0000
    1.0000   -1.0000    1.0000   -1.0000    1.0000
         0    1.0000    0.0000   -1.0000   -0.0000
         0    0.0000   -0.0000    0.0000   -0.0000

Normalized matrix:

    0.4472    0.4472    0.4472    0.4472    0.4472
    0.5774    0.0000   -0.5774   -0.0000    0.5774
    0.4472   -0.4472    0.4472   -0.4472    0.4472
         0    0.7071    0.0000   -0.7071   -0.0000
```

```
          0    0.0000   -0.0000    0.0000   -0.0000

Orthogonal matrix:

    0.4472    0.4472    0.4472    0.4472    0.4472
    0.4781   -0.1195   -0.7171   -0.1195    0.4781
    0.2673   -0.5345    0.5345   -0.5345    0.2673
    0.0000    0.7071    0.0000   -0.7071   -0.0000
    0.7071   -0.0000    0.0000    0.0000   -0.7071

Orthogonality check (orthogonal_matrix * orthogonal_matrix^T):

    1.0000    0.0000    0.0000   -0.0000    0.0000
    0.0000    1.0000   -0.0000   -0.0000    0.0000
    0.0000   -0.0000    1.0000    0.0000    0.0000
   -0.0000   -0.0000    0.0000    1.0000   -0.0000
    0.0000    0.0000    0.0000   -0.0000    1.0000
```

Final Orthogonal Matrix

After applying the Gram-Schmidt process, we obtain the orthogonal matrix Q such that:

$$Q \cdot Q^{\top} = I$$

where I is the identity matrix.

2.9.3 Third approach

Directly using Discrete Fourier Transform (DFT), we can create a 5×5 matrix without any additional process of orthogonalization. The `matlab` code for this task is shown below.

Input program 16: *Direct method using DFT*

```
% using DFT
% Set the size of the matrix (5x5)
N = 5;

% Construct the DFT matrix
dft_matrix = zeros(N, N);

% Populate the DFT matrix
for k = 0:N-1
    for n = 0:N-1
        dft_matrix(k+1, n+1) = exp(-2i * pi * k * n / N) / sqrt(N);
    end
end

% Display the DFT matrix
disp('DFT matrix:');
% Check orthogonality: the matrix should satisfy F * F^H = I
orthogonality_check = dft_matrix * dft_matrix';

```

```
disp('Orthogonality check (DFT * DFT^H):');
```

```
DFT matrix:

   0.4472 + 0.0000i   0.4472 + 0.0000i   0.4472 + 0.0000i   0.4472 + 0.0000i   0.4472 + 0.0000i
   0.4472 + 0.0000i   0.1382 - 0.4253i  -0.3618 - 0.2629i  -0.3618 + 0.2629i   0.1382 + 0.4253i
   0.4472 + 0.0000i  -0.3618 - 0.2629i   0.1382 + 0.4253i   0.1382 - 0.4253i  -0.3618 + 0.2629i
   0.4472 + 0.0000i  -0.3618 + 0.2629i   0.1382 - 0.4253i   0.1382 + 0.4253i  -0.3618 - 0.2629i
   0.4472 + 0.0000i   0.1382 + 0.4253i  -0.3618 + 0.2629i  -0.3618 - 0.2629i   0.1382 - 0.4253i

Orthogonality check (DFT * DFT^H):

   1.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 + 0.0000i  -0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 + 0.0000i
  -0.0000 - 0.0000i  -0.0000 + 0.0000i   1.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i
   0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i   1.0000 + 0.0000i  -0.0000 - 0.0000i
   0.0000 - 0.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i   1.0000 + 0.0000i
```

RESULTS

- Constructed a 5×5 matrix using cosine and sine functions with different wave numbers.
- Applied the Gram-Schmidt process to ensure the matrix is orthogonal.
- Constructed the 5×5 orthogonal matrix directly using DFT.

3 | Assignment- 3 Generating and Analyzing Random Matrix

1. Generate a 4×5 random matrix of rank 3. Through a `matlab` program generate all possible combinations of 4×3 matrices from above 4x5 matrices. (it is about selecting 3 columns at a time from given 4×5 matrices. Total there are $\binom{5}{3} = 10$ such matrices) . Find the rank of each matrix.

SOLUTION

1. Generating a 4×5 Matrix of Rank 3

Step 1: Create a Random Matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix}$$

where a_{ij} are randomly chosen entries.

Step 2. Ensure Rank 3: Adjust the matrix to ensure it has rank 3. Modify some columns to be linear combinations of the first three columns: as $C_3 = C_1 + C_2 - C_3, C4 = C_1 - C2 + C3$. The resulting matrix will be:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{11} + a_{12} - a_{13} & a_{11} - a_{12} + a_{13} \\ a_{21} & a_{22} & a_{23} & a_{21} + a_{22} - a_{23} & a_{21} - a_{22} + a_{23} \\ a_{31} & a_{32} & a_{33} & a_{31} + a_{32} - a_{33} & a_{31} - a_{32} + a_{33} \\ a_{41} & a_{42} & a_{43} & a_{41} + a_{42} - a_{43} & a_{41} - a_{42} + a_{43} \end{bmatrix}$$

2. Generating All Possible 4×3 Sub-matrices

Step:1. Determine Column Combinations: Find all combinations of 3 columns out of the 5 columns. There are $\binom{5}{3} = 10$ such combinations:

$$\{1,2,3\}, \{1,2,4\}, \{1,2,5\}, \{1,3,4\}, \{1,3,5\}, \{1,4,5\}, \{2,3,4\}, \{2,3,5\}, \{2,4,5\}, \{3,4,5\}$$

Step 2. Extract Sub-matrices: For each combination, extract the corresponding 4×3 sub-matrix from A:

$$A_{123} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

3. Computing the Rank of Each 4×3 Sub-matrix

Compute the Rank: For each 4×3 sub-matrix, calculate its rank. The rank is the maximum number of linearly independent columns. The rank can be analytically computed using row reduction. The complete `matlab` code is shown below:

Input program 17: *Working with sub matrices of random matrices*

```matlab
rng(0);
A = rand(4, 5);
A(:,4) = A(:,1) + A(:,2) - A(:,3);
A(:,5) = A(:,1) - A(:,2) + A(:,3);
combinations = nchoosek(1:5, 3);
num_combinations = size(combinations, 1);
submatrices = cell(num_combinations, 1);
ranks = zeros(num_combinations, 1);
for i = 1:num_combinations
    cols = combinations(i, :);
    submatrix = A(:, cols);
    submatrices{i} = submatrix;
    ranks(i) = rank(submatrix);
end
disp('Original 4x5 Matrix A:');
disp(A);
for i = 1:num_combinations
    fprintf('Submatrix %d:\n', i);
    disp(submatrices{i});
    fprintf('Rank: %d\n', ranks(i));
end
```

```
Original 4x5 Matrix A:
    0.8147    0.6324    0.9575    0.4896    1.1399
    0.9058    0.0975    0.9649    0.0384    1.7731
    0.1270    0.2785    0.1576    0.2479    0.0061
    0.9134    0.5469    0.9706    0.4897    1.3371
Submatrix 1:
    0.8147    0.6324    0.9575
    0.9058    0.0975    0.9649
    0.1270    0.2785    0.1576
    0.9134    0.5469    0.9706
Rank: 3

Submatrix 2:
    0.8147    0.6324    0.4896
    0.9058    0.0975    0.0384
    0.1270    0.2785    0.2479
    0.9134    0.5469    0.4897
Rank: 3
Submatrix 3:
    0.8147    0.6324    1.1399
    0.9058    0.0975    1.7731
    0.1270    0.2785    0.0061
    0.9134    0.5469    1.3371
Rank: 3
```

```
Submatrix 4:
    0.8147    0.9575    0.4896
    0.9058    0.9649    0.0384
    0.1270    0.1576    0.2479
    0.9134    0.9706    0.4897
Rank: 3
Submatrix 5:
    0.8147    0.9575    1.1399
    0.9058    0.9649    1.7731
    0.1270    0.1576    0.0061
    0.9134    0.9706    1.3371
Rank: 3
Submatrix 6:
    0.8147    0.4896    1.1399
    0.9058    0.0384    1.7731
    0.1270    0.2479    0.0061
    0.9134    0.4897    1.3371
Rank: 2
Submatrix 7:
    0.6324    0.9575    0.4896
    0.0975    0.9649    0.0384
    0.2785    0.1576    0.2479
    0.5469    0.9706    0.4897
Rank: 3
Submatrix 8:
    0.6324    0.9575    1.1399
    0.0975    0.9649    1.7731
    0.2785    0.1576    0.0061
    0.5469    0.9706    1.3371
Rank: 3
Submatrix 9:
    0.6324    0.4896    1.1399
    0.0975    0.0384    1.7731
    0.2785    0.2479    0.0061
    0.5469    0.4897    1.3371
Rank: 3
Submatrix 10:
    0.9575    0.4896    1.1399
    0.9649    0.0384    1.7731
    0.1576    0.2479    0.0061
    0.9706    0.4897    1.3371
Rank: 3
```

RESULTS

(a) 10 rank 3 sub-matrices were generated from the 4×5 random rank 3 matrix.

(b) Rank of each sub-matrix is verified using `matlab` code.

Conclusion:

If the original matrix A is correctly created with rank 3, all 4×3 sub-matrices should have rank 3, indicating that each sub-matrix has only 3 linearly independent columns.

4 | Assignment-4 Rank of Sampled Sinusoidal Functions

4.1 Rank of Sampled Cosine Functions

1. Create sampled version of cosine functions as follows and answer the question regarding the rank.

```
N=8;
theta= (0:N-1)*2*pi/N; % row vector
A= cos( (0:N-1)'*theta); % bases using outer product
rankA=rank(A)
```

What is the rank obtained? Why the rank is less than 8?

SOLUTION

The vector, θ, generated partitioning the interval $[0, 2\pi]$ into 7 equal parts is:

$$\theta = \left[0, \frac{\pi}{4}, \frac{2\pi}{4}, \frac{3\pi}{4}, \frac{4\pi}{4}, \frac{5\pi}{4}, \frac{6\pi}{4}, \frac{7\pi}{4}\right]$$

The Matrix created with outer product operation is given by:

</> **Input program 18:** *Matrix created by the cosine of outer products* </>

```
N = 8;
theta = (0:N-1) * 2 * pi / N;  %
A = cos((0:N-1)' * theta);  %
disp('Matrix A:');
disp(A)
```

```
1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
1.0000   0.7071   0.0000  -0.7071  -1.0000  -0.7071  -0.0000   0.7071
1.0000   0.0000  -1.0000  -0.0000   1.0000   0.0000  -1.0000  -0.0000
1.0000  -0.7071  -0.0000   0.7071  -1.0000   0.7071   0.0000  -0.7071
1.0000  -1.0000   1.0000  -1.0000   1.0000  -1.0000   1.0000  -1.0000
1.0000  -0.7071   0.0000   0.7071  -1.0000   0.7071  -0.0000  -0.7071
1.0000  -0.0000  -1.0000   0.0000   1.0000  -0.0000  -1.0000  -0.0000
1.0000   0.7071  -0.0000  -0.7071  -1.0000  -0.7071  -0.0000   0.7071
```

From the matrix generated, it is clear that four rows (4,6,8) are inter-related (scalar multiple of 2nd). So, there are $8 - 3 = 5$ independent rows. Hence $\rho(A) = 5$.

RESULTS

The rows of the matrix A are likely linearly dependent due to the specific nature of cosine functions. So, it is rank deficient.

4.2 Rank of Sampled Sine Functions

2. Create sampled version of sin functions as follows and answer the question regarding the rank.

```
N=8
theta= (0:N-1)*2*pi/N; % row vector
A= sin( (1:N)'*theta); %bases using outer product
rankA=rank(A)
```

What is the rank obtained?. Why the rank is less than 8?

SOLUTION

The rank of the matrix so created by the `Matlab` code is 3. Reason for this low rank is due to the linear dependence of rows of the matrix A. The `Matlab` code for this task is shown below.

Input program 19: *Matrix created by sines of outer product*

```
N=8
theta= (0:N-1)*2*pi/N; % row vector
A= sin( (1:N)'*theta); %bases using outer product
disp('Matrix created is:')
disp(A)
```

```
Matrix created is:

         0    0.7071    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071
         0    1.0000    0.0000   -1.0000   -0.0000    1.0000    0.0000   -1.0000
         0    0.7071   -1.0000    0.7071    0.0000   -0.7071    1.0000   -0.7071
         0    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000
         0   -0.7071    1.0000   -0.7071    0.0000    0.7071   -1.0000    0.7071
         0   -1.0000    0.0000    1.0000   -0.0000   -1.0000    0.0000    1.0000
         0   -0.7071   -1.0000   -0.7071    0.0000    0.7071    1.0000    0.7071
         0   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000
```

Only independent rows in A are 1, 2 and 4.

RESULTS

The rows of the matrix A are likely linearly dependent due to the specific nature of sine functions. So it is rank deficient.

3. Create sampled version of sin and cosine functions as follows and answer the question regarding the rank.

```
N=8
theta= (0:N-1)*2*pi/N; % row vector
A= cos( (0:N/2)'*theta); % N/2+1 cos bases using outer product
A= [A ; sin((N/2+1: N-1)'*theta) ]; % N/2-1 sin bases using outer product
rankA=rank(A)
```

What is the rank obtained? Why the rank is 8?

SOLUTION

The rank of the matrix so created by the `Matlab` code is 3. Reason for this low rank is due to the linear dependence of rows of the matrix A. The `Matlab` code for this task is shown below.

Input program 20: *Matrix created by* $\frac{N}{2}-1$ *sine bases*

```
theta= (0:N-1)*2*pi/N; % row vector
A= cos( (0:N/2)'*theta); % N/2+1 cos bases using outer product
A= [A ; sin((N/2+1: N-1)'*theta) ]; % N/2-1 sin bases using outer product
rankA=rank(A)
disp('Matrix created is:')
dip(A)
```

```
rankA=8
Matrix created is:
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071   -0.0000    0.7071
    1.0000    0.0000   -1.0000   -0.0000    1.0000    0.0000   -1.0000   -0.0000
    1.0000   -0.7071   -0.0000    0.7071   -1.0000    0.7071    0.0000   -0.7071
    1.0000   -1.0000    1.0000   -1.0000    1.0000   -1.0000    1.0000   -1.0000
         0   -0.7071    1.0000   -0.7071    0.0000    0.7071   -1.0000    0.7071
         0   -1.0000    0.0000    1.0000   -0.0000   -1.0000    0.0000    1.0000
         0   -0.7071   -1.0000   -0.7071    0.0000    0.7071    1.0000    0.7071
```

RESULTS

The rows of the matrix A are linearly independent due to the specific nature of sine functions. So, it is a full rank matrix.

5 | Assignment-5 Role of Orthogonal Transformations in Image Processing Applications

5.1 Linear Algebra and Signal/ Image Processing

Connecting Signal Processing into Linear Algebra: Two-dimensional wave forms are useful in image processing applications. A 2D wave can be created using outer product of 1D wave representations. Orthogonality is a critical concept in signal processing. It ensures that different components of a signal (or image) can be separated and manipulated independently without interference. This property is pivotal in data compression, filtering, and noise reduction. There are different mathematical approaches to create orthogonal transformations (matrices). One such popular method is the Discrete Fourier Transform. This transform decomposes a signal into orthogonal components. So, the signal/ image processing operations becomes just algebraic operations. The mathematical representation of DCT is given in (5.1).

$$\mathrm{DCT}(i,j) = \alpha(i)\cos\left(\frac{\pi}{N}\left(j+0.5\right)i\right) \tag{5.1}$$

where the scaling factor $\alpha(i)$ is defined as:

$$\alpha(i) = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } i = 0 \\ \sqrt{\frac{2}{N}} & \text{if } i > 0 \end{cases}$$

5.2 Discrete Cosine Transform of Basis Vectors

Orthogonal transformations like DCT decompose a signal into orthogonal components, simplifying tasks such as compression and filtering.

A simple example of a 2D wave using `Matlab` (with `dct()` function) is shown below.

Input program 21: *Matlab code to generate 2D wave form*

```
N=64;
theta=(0:N-1)*2*pi/N;
m=2; %vertical frequency
n=4; %horizontal frequency
wav2d1=cos(m*theta')*cos(n*theta); %outer product of waves
surf(wav2d1)
```

The wave form created is shown in Figure 5.1

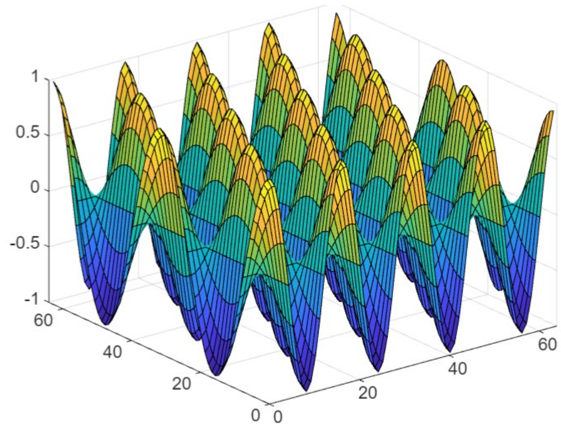

Figure 5.1: Output of 2D wave form

5.3 Orthogonality of Sampled Outer Products

In terms of linear algebra, the outer product of DCT basis vectors creates matrices (2D wave forms) capturing different frequencies in both directions, essential for tasks like filtering and compression. If they are pair-wise orthogonal, then the signal/ image processing becomes an algebraic operation. Fortunately it is true. This task will prove this property using `Matlab` implementation. Formally the problem can be stated as follows.

Problem Statement: Create a 4×4 DCT matrix. Taking two columns at a time, create matrices with outer product. Demonstrate that any such two different matrices will be orthogonal.

SOLUTION

Task 1 Creation of DCT matrix: The `Matlab` code for Task 1 is shown below.

Input program 22: *Code for Task 1*

```
N = 4;                  % Size of the basis vectors
dctMatrix = dct(eye(N));  % 4x4 DCT matrix
disp("4X4 DCT matrix:")
disp(dctMatrix)
```

```
4X4 DCT matrix:
    0.5000    0.5000    0.5000    0.5000
    0.6533    0.2706   -0.2706   -0.6533
    0.5000   -0.5000   -0.5000    0.5000
    0.2706   -0.6533    0.6533   -0.2706
```

Task 2 Creation of outer products The `Matlab` code for Task 2 is shown below.

Input program 23: *Construction of Outer products*

```
dctMatrix = dct(eye(N));
outerProducts = zeros(N, N, N, N);
for i = 1:N
    for j = 1:N
        outerProducts(:,:,i,j) = dctMatrix(:,i) * dctMatrix(:,j)';
    end
end
for i = 1:N
    for j = 1:N
```

```
        fprintf('Outer Product Matrix %d,%d:\n', i, j);
        disp(outerProducts(:,:,i,j));
        %disp(dot(outerProducts(:,:,i,j),outerProducts(:,:,j,i)))
    end
end
```

```
Outer Product Matrix 1,1:
    0.2500    0.3266    0.2500    0.1353
    0.3266    0.4268    0.3266    0.1768
    0.2500    0.3266    0.2500    0.1353
    0.1353    0.1768    0.1353    0.0732
Outer Product Matrix 1,2:
    0.2500    0.1353   -0.2500   -0.3266
    0.3266    0.1768   -0.3266   -0.4268
    0.2500    0.1353   -0.2500   -0.3266
    0.1353    0.0732   -0.1353   -0.1768
Outer Product Matrix 1,3:
    0.2500   -0.1353   -0.2500    0.3266
    0.3266   -0.1768   -0.3266    0.4268
    0.2500   -0.1353   -0.2500    0.3266
    0.1353   -0.0732   -0.1353    0.1768
Outer Product Matrix 1,4:
    0.2500   -0.3266    0.2500   -0.1353
    0.3266   -0.4268    0.3266   -0.1768
    0.2500   -0.3266    0.2500   -0.1353
    0.1353   -0.1768    0.1353   -0.0732
Outer Product Matrix 2,1:
    0.2500    0.3266    0.2500    0.1353
    0.1353    0.1768    0.1353    0.0732
   -0.2500   -0.3266   -0.2500   -0.1353
   -0.3266   -0.4268   -0.3266   -0.1768
Outer Product Matrix 2,2:
    0.2500    0.1353   -0.2500   -0.3266
    0.1353    0.0732   -0.1353   -0.1768
   -0.2500   -0.1353    0.2500    0.3266
   -0.3266   -0.1768    0.3266    0.4268
Outer Product Matrix 2,3:
    0.2500   -0.1353   -0.2500    0.3266
    0.1353   -0.0732   -0.1353    0.1768
   -0.2500    0.1353    0.2500   -0.3266
   -0.3266    0.1768    0.3266   -0.4268
Outer Product Matrix 2,4:
    0.2500   -0.3266    0.2500   -0.1353
    0.1353   -0.1768    0.1353   -0.0732
   -0.2500    0.3266   -0.2500    0.1353
   -0.3266    0.4268   -0.3266    0.1768
Outer Product Matrix 3,1:
    0.2500    0.3266    0.2500    0.1353
   -0.1353   -0.1768   -0.1353   -0.0732
   -0.2500   -0.3266   -0.2500   -0.1353
    0.3266    0.4268    0.3266    0.1768
Outer Product Matrix 3,2:
    0.2500    0.1353   -0.2500   -0.3266
```

```
    -0.1353   -0.0732    0.1353    0.1768
    -0.2500   -0.1353    0.2500    0.3266
     0.3266    0.1768   -0.3266   -0.4268
Outer Product Matrix 3,3:
     0.2500   -0.1353   -0.2500    0.3266
    -0.1353    0.0732    0.1353   -0.1768
    -0.2500    0.1353    0.2500   -0.3266
     0.3266   -0.1768   -0.3266    0.4268
Outer Product Matrix 3,4:
     0.2500   -0.3266    0.2500   -0.1353
    -0.1353    0.1768   -0.1353    0.0732
    -0.2500    0.3266   -0.2500    0.1353
     0.3266   -0.4268    0.3266   -0.1768
Outer Product Matrix 4,1:
     0.2500    0.3266    0.2500    0.1353
    -0.3266   -0.4268   -0.3266   -0.1768
     0.2500    0.3266    0.2500    0.1353
    -0.1353   -0.1768   -0.1353   -0.0732
Outer Product Matrix 4,2:
     0.2500    0.1353   -0.2500   -0.3266
    -0.3266   -0.1768    0.3266    0.4268
     0.2500    0.1353   -0.2500   -0.3266
    -0.1353   -0.0732    0.1353    0.1768
Outer Product Matrix 4,3:
     0.2500   -0.1353   -0.2500    0.3266
    -0.3266    0.1768    0.3266   -0.4268
     0.2500   -0.1353   -0.2500    0.3266
    -0.1353    0.0732    0.1353   -0.1768
Outer Product Matrix 4,4:
     0.2500   -0.3266    0.2500   -0.1353
    -0.3266    0.4268   -0.3266    0.1768
     0.2500   -0.3266    0.2500   -0.1353
    -0.1353    0.1768   -0.1353    0.0732
```

Task 3 Verification of orthogonality of sub-matrices: To verify orthogonality of sub-matrices, let's try a simple trick. Take different sub-matrices pair-wise and flatten them and apply `dot(Mi,Mj)` operation. If this dot product is zero, then the matrices are pair-wise orthogonal. Since the cosine values are truncated during intermediate calculations, the dot product may not be exactly zero. So set a negligibly small threshold and check that the dot product is less than this threshold. If it is true, then the sub-matrices will be pair-wise orthogonal. The `Matlab` code for this task is shown below.

Input program 24: *Verification of pair-wise orthogonality*

```
tolerance = 1e-10;  % Set a tolerance for floating-point comparisons

for i1 = 1:N
    for j1 = 1:N
        for i2 = 1:N
            for j2 = 1:N

                matrix1 = outerProducts(:,:,i1,j1);  % First matrix
                matrix2 = outerProducts(:,:,i2,j2);  % Second matrix


```

```
                    matrix1_flat = matrix1(:);
                    matrix2_flat = matrix2(:);

                    dotProd = dot(matrix1_flat, matrix2_flat);


                    if (i1 ~= i2 || j1 ~= j2)

                        assert(abs(dotProd) < tolerance, 'Dot product is not
                         ↪  close to zero! Orthogonality failed.');
                    end
                end
            end
        end
    end

disp('All matrices are pairwise orthogonal.');
```

Output of the above `matlab` code is given below.

```
All matrices are pairwise orthogonal.
```

RESULTS

1. A 4×4 DCT matrix from orthogonal basis of $\mathbb{R}^4$ is created.
2. 16 sub-matrices using outer product of 2 rows and columns of the DCT matrix is created.
3. The statement *The outer product matrices derived from DCT basis vectors are pairwise orthogonal* is computationally verified.

6 | Assignment-6 Projection Matrices

6.1 Solution of Rank deficient Linear Systems

6.1.1 Conditions for many solutions

1. Create a random 5×4 matrix A with rank 2 and a 5×1 vector b such that $Ax = b$ has infinite solution. Write the `Matlab` code and also generate infinite solutions using loop?

SOLUTION

The `Matlab` code for this task is given below. For demonstration purpose, 20 random solutions are listed.

Input program 25: *Creating Random* 5×4 *matrix with rank 2*

```
m = 5; % Number of rows
n = 4; % Number of columns
r = 2; % Desired rank
A_base = rand(m, r);
A_expansion = [2*A_base(:,1),3*A_base(:,2)];
A = [A_base, A_expansion];
disp("Random 5X4 matrix with rank 2:")
disp(A)
disp('Rank of A:')
disp(rank(A))
b = A * rand(n, 1);
disp('Vector b:');
disp(b);
x_particular = pinv(A)*b;
null_space = null(A, 'r');
disp('Particular Solution:');
disp(x_particular);
num_solutions = 20;
solutions = zeros(num_solutions, n);

for i = 1:num_solutions

    solutions(i, :) = (x_particular + null_space * rand(size(null_space,
     2), 1))';
end
Sl_No = (1:num_solutions)'; % Column for serial numbers
x = solutions(:, 1);
y = solutions(:, 2);
```

```
z = solutions(:, 3);
w = solutions(:, 4);
% Create the table
solutions_table = table(Sl_No, x, y, z, w);

disp('Table of 20 solutions:');
disp(solutions_table);
```

Output of the code is:

```
Random 5X4 matrix with rank 2:
    0.8813    0.5070    1.7625    1.5209
    0.8274    0.2244    1.6548    0.6733
    0.6143    0.8396    1.2287    2.5187
    0.8138    0.3435    1.6275    1.0305
    0.2156    0.5674    0.4312    1.7021
Rank of A:
2
Vector b:
    2.6567
    2.1730
    2.4728
    2.2940
    1.2161
Particular Solution:
    0.4560
    0.1277
    0.9120
    0.3831
Table of 20 solutions:
    Sl_No        x            y            z          w
    -----    ---------    --------    -------    -------

      1        -1.1646    -0.90335     1.7223     0.7268
      2        -1.4627     -1.9248     1.8713     1.0673
      3        -1.4846     -1.9407     1.8823     1.0726
      4       -0.78983    -0.78774     1.5349    0.68826
      5         -1.452    -0.75718      1.866    0.67808
      6        0.24641     -1.0058     1.0168    0.76096
      7       -0.11672    -0.24587     1.1984    0.50764
      8        -1.4756     -2.2402     1.8778     1.1724
      9       -0.26395     -2.3257      1.272     1.2009
     10       -0.43526     -1.3606     1.3576    0.87922
     11        0.19413     -2.1249     1.0429      1.134
     12       -0.16702     -2.8659     1.2235      1.381
     13         0.3733     -1.3916    0.95334    0.88954
     14       -0.89799    -0.43078      1.589    0.56927
     15        0.19594     -1.8786      1.042     1.0519
     16      -0.070744    -0.20913     1.1754    0.49539
     17        -1.5295     -2.5718     1.9048     1.2829
     18       -0.35079     -2.5891     1.3154     1.2887
     19      -0.029476    -0.75192     1.1547    0.67632
     20        0.35156    -0.19921    0.96421    0.49208
```

6.1.2 Condition for no solution

2. Create a no solution case for the above question. Write the Matlab code?

SOLUTION

A system, $Ax = b$ does not have a solution if $b \not\in$ Column space of A. So, it is enough to create such a b. For this purpose, intentionally choose a vector b which is completely random and in the form of linear combination of a vector in the null space of A! The Matlab code for this task is shown below:

Input program 26: *No solution case*

```
m = 5;
n = 4;
rk = 2;
A_base = rand(m, rk);
A_expansion = [2*A_base(:,1),3*A_base(:,2)];
A = [A_base, A_expansion];
rank_A = rank(A);
fprintf('Rank of matrix A: %d\n', rank_A);
b_random = rand(m, 1);
disp('Vector b (not in the column space of A):');
b_projection = A * (A \ b_random);
b = b_random + (b_random - b_projection);
x_solution = pinv(A)*b;
residual = norm(A * x_solution - b);
disp('Matrix A:');
disp(A);
disp('Vector b (not in the column space of A):');
disp(b);
fprintf('Residual (should be large if no solution exists): %.5f\n', residual);
if residual > 1e-5
    disp('No solution exists for the system Ax = b.');
else
    disp('A solution exists (unexpected for no solution case).');
end
```

Output of the above Matlab code is shown below.

```
Rank of matrix A: 2
Matrix A:
    0.0842    0.3060    0.1683    0.9181
    0.7943    0.1719    1.5886    0.5158
    0.7400    0.7973    1.4800    2.3920
    0.1098    0.3751    0.2195    1.1252
    0.8511    0.4103    1.7023    1.2310
Vector b (not in the column space of A):
    1.2814
    0.8340
    0.3253
    1.3961
    0.0784
Residual (should be large if no solution exists): 1.62012
No solution exists for the system Ax = b.
```

6.1.3 Existence of null spaces

3. Create a 3×4 matrix with rank 3, check whether right null space and left null space exist. Comment. Write a `Matlab` code to verify?

SOLUTION

Let A be a 3×4 matrix with rank 3. So by rank nullity theorem,

$$\begin{aligned} \rho(A) + \text{Nullity}(A) &= \text{no. of columns of} A \\ \implies 3 + \dim(N(A)) &= 4 \\ \implies \dim((N(A)) &= 1 \end{aligned}$$

This implies that $N(A)$, the right null space of A is non empty and contains one basis element.

Similarly, we can use the same theorem for the row space of A.

$$\begin{aligned} \rho(A) + \text{Nullity}(A^T) &= \text{no. of rows of } A \\ \implies 3 + \dim((A^T)) &= 3 \\ \implies \dim(A^T) &= 0 \end{aligned}$$

This implies that $N(A^T)$, the left null space of A is empty. The `Matlab` code to demonstrate this findings is given below.

</> **Input program 27:** *Basis of Null spaces of a Rank deficient Matrix* </>

```
m = 3;
n = 4;
desired_rank = 3;
A_base = rand(m, desired_rank);
A_expansion = [2*A_base(:,2)-.5*A_base(:,1)];
A = [A_base, A_expansion];
rank_A = rank(A);
fprintf('Rank of matrix A: %d\n', rank_A);
right_null_space = null(A);
left_null_space = null(A');
disp('Matrix A:');
disp(A);
disp('Right Null Space of A (null(A)):');
disp(right_null_space);
disp('Left Null Space of A (null(A'')):');
disp(left_null_space);

if isempty(right_null_space)
    disp('Right null space does not exist (only the trivial solution exists).');
else
    disp('Right null space exists (non-trivial solutions exist).');
end

if isempty(left_null_space)
    disp('Left null space does not exist (only the trivial solution exists).');
else
    disp('Left null space exists (non-trivial solutions exist).');
```

```
end
```

Output of the above `Matlab` code is shown below.

```
Rank of matrix A: 3
Matrix A:
    0.2575    0.8143    0.3500    1.4998
    0.8407    0.2435    0.1966    0.0667
    0.2543    0.9293    0.2511    1.7314
Right Null Space of A (null(A)):

    0.2182
   -0.8729
    0.0000
    0.4364
Left Null Space of A (null(A')):
Right null space exists (non-trivial solutions exist).
Left null space does not exist (only the trivial solution exists).
```

6.2 Projection of vectors to Fundamental Subspaces

4. Create a random 4×4 matrix A with rank 2 and a 4×1 vector y. Find the projection of y onto all the four spaces associated with it ?

SOLUTION

For a matrix A and a vector y, the projections of y into the fundamental subspaces are given by:

$$\begin{aligned} y_{col-space} &= \left(AA^{-1}\right)y \\ y_{N(A)} &= y - y_{col-space} \\ y_{row-space} &= \left(A^T(A^T)^{-1}\right)y \\ y_{N(A^T)} &= y - y_{row-space} \end{aligned}$$

where $y_{col-space}, y_{N(A)}, y_{row-space}$ and $y_{N(A^T)}$ are projection of y into column space of A, right null space of A, row space of A and left null space of A respectively. Since A is a rank deficient square matrix, pseudo inverse will be used to find projections. `Matlab` code to demonstrate these operations on a random matrix A is shown below.

</> **Input program 28:** *Projections of vector into Fundamental Sub spaces* </>

```
m = 4;
desired_rank = 2;
A_base = rand(m, desired_rank);
A_expansion = [0.5*A_base(:,1),2*A_base(:,2)-A_base(:,1)];
A = [A_base, A_expansion];
rank_A = rank(A);
fprintf('Rank of matrix A: %d\n', rank_A);
y = rand(m, 1);
col_space_proj = A * (pinv(A)* y);
null_space_proj = y - col_space_proj;
row_space_proj = A' * (pinv(A')* y);
left_null_space_proj = y - row_space_proj;
```

```
disp('Matrix A:');
disp(A);
disp('Vector y:');
disp(y);
disp('Projection of y onto the Column Space (Range of A):');
disp(col_space_proj);
disp('Projection of y onto the Null Space of A:');
disp(null_space_proj);
disp('Projection of y onto the Row Space (Range of A''):');
disp(row_space_proj);
disp('Projection of y onto the Left Null Space (Null space of A''):');
disp(left_null_space_proj);
```

```
Rank of matrix A: 2
Matrix A:
    0.6160    0.5853    0.3080    0.5545
    0.4733    0.5497    0.2366    0.6262
    0.3517    0.9172    0.1758    1.4827
    0.8308    0.2858    0.4154   -0.2592
Vector y:
    0.7572
    0.7537
    0.3804
    0.5678
Projection of y onto the Column Space (Range of A):
    0.6430
    0.5246
    0.5453
    0.7132
Projection of y onto the Null Space of A:
    0.1142
    0.2291
   -0.1649
   -0.1454
Projection of y onto the Row Space (Range of A'):
    0.7830
    0.6911
    0.3915
    0.5991
Projection of y onto the Left Null Space (Null space of A'):
   -0.0258
    0.0627
   -0.0111
   -0.0313
```

6.3 Generation of Basis for Fundamental Subspaces

6.3.1 Creating basis for row space and column space

(a) Method 1- Use 'rref' for getting row space basis and column space basis. Write the `Matlab` code for the same.

SOLUTION

Consider a rank deficient matrix $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 5 & 6 & 7 \end{bmatrix}$. Since it is a rank 2 matrix, both the left and right null spaces will be non-empty. Using row-reduced Echelon form, both the row and column bases can be generated. The independent solutions of the systems $Ax = 0$ and $A^T x = 0$ respectively produce the basis for right and left null spaces. `Matlab` code to execute this task is given below.

Input program 29: *Generating Basis for Row and Column spaces*

```
A = [1 2 3 4;2 4 6 8;3 6 9 12;4 5 6 7];
[R, pivot_columns] = rref(A);
row_space_basis = R(any(R, 2), :);
column_space_basis = A(:, pivot_columns);
disp('Matrix A:');
disp(A);
disp('Reduced Row Echelon Form (RREF) of A:');
disp(R);
disp('Pivot columns (indices):');
disp(pivot_columns);
disp('Row Space Basis (Non-zero rows of RREF):');
disp(row_space_basis);
disp('Column Space Basis (Pivot columns of original matrix A):');
disp(column_space_basis);
```

Output of the above code is shown below.

```
Matrix A:
     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     5     6     7
Reduced Row Echelon Form (RREF) of A:
     1     0    -1    -2
     0     1     2     3
     0     0     0     0
     0     0     0     0
Pivot columns (indices):
     1     2
Row Space Basis (Non-zero rows of RREF):
     1     0    -1    -2
     0     1     2     3
Column Space Basis (Pivot columns of original matrix A):
     1     2
     2     4
     3     6
     4     5
```

6.3.2 Creating basis for null spaces

(b) Method 2- Use 'Null' command for getting left and right null space basis sets. Write the `Matlab` code for the same.

SOLUTION

Matlab code for this task is given below.

</> **Input program 30:** *Creating Basis for Null Spaces* </>

```
right_null_space = null(A, 'r');
left_null_space = null(A', 'r');
row_space_basis = R(any(R, 2), :);
disp('Matrix A:');
disp(A);
disp('Right Null Space of A:');
if isempty(right_null_space)
    disp('No right null space exists for matrix A.');
else
    disp('Right null space exists for matrix A.');
    disp(right_null_space);
end
disp('Left Null Space of A^T:');
if isempty(left_null_space)
    disp('No left null space exists for matrix A^T.');
else
    disp('Left null space exists for matrix A^T.');
    disp(left_null_space);
end
```

Output of the above code is shown below.

```
Matrix A:
     1     2     3     4
     2     4     6     8
     1     3     2     5
Right Null Space of A:
Right null space exists for matrix A.
    -5    -2
     1    -1
     1     0
     0     1
Left Null Space of A^T:
Left null space exists for matrix A^T.
    -2
     1
     0
```

6.4 More on projections

5. Create a random 5×4 matrix with rank 2 ,create two vectors one from the rowspace and the other from columnspace and find appropriate projection on to spaces associated with a matrix.

SOLUTION

Matlab code for this task is given below.

Input program 31: *Projecting vectors onto Fundamental Subspaces*

```
m = 5;
desired_rank = 2;
A_base = rand(m, desired_rank);
A_expansion = [0.5*A_base(:,1),2*A_base(:,2)-A_base(:,1)];
A = [A_base, A_expansion];
R = rref(A);
row_space_basis = R(any(R, 2), :);

% Compute the Column Space Basis
[~, pivot_columns] = rref(A);
column_space_basis = A(:, pivot_columns);
row_vector = row_space_basis(1, :)';
column_vector = column_space_basis(:, 1);
P_row = row_space_basis' * (row_space_basis * row_space_basis')^-1 * row_space_basis;
projection_row_vector = P_row * row_vector;
P_col = column_space_basis * (column_space_basis' * column_space_basis)^-1 * column_space_basis';
projection_column_vector = P_col * column_vector;
disp('Original Matrix A:');
disp(A);
disp('Row Space Basis:');
disp(row_space_basis);
disp('Column Space Basis:');
disp(column_space_basis);
disp('Vector from Row Space:');
disp(row_vector);
disp('Vector from Column Space:');
disp(column_vector);
disp('Projection of Row Vector onto Row Space:');
disp(projection_row_vector);
disp('Projection of Column Vector onto Column Space:');
disp(projection_column_vector);
```

```
Original Matrix A:
    0.4841    0.0826    0.2421   -0.3190
    0.1312    0.8189    0.0656    1.5067
    0.9395    0.5762    0.4698    0.2130
    0.0019    0.1814    0.0009    0.3610
    0.4473    0.6044    0.2237    0.7614
Row Space Basis:
    1.0000         0    0.5000   -1.0000
         0    1.0000         0    2.0000
Column Space Basis:
    0.4841    0.0826
    0.1312    0.8189
    0.9395    0.5762
    0.0019    0.1814
    0.4473    0.6044
Vector from Row Space:
    1.0000
         0
```

```
    0.5000
   -1.0000
Vector from Column Space:
    0.4841
    0.1312
    0.9395
    0.0019
    0.4473
Projection of Row Vector onto Row Space:
    1.0000
   -0.0000
    0.5000
   -1.0000
Projection of Column Vector onto Column Space:
    0.4841
    0.1312
    0.9395
    0.0019
    0.4473
```

Observation: It has been observed that the vectors generated from the row space will be projected completely as it is into the row space. A similar result is observed for column vectors too.

6. Explain how bases can be created for row space and column space using `rref`?

SOLUTION

The row-reduced Echelon for of a matrix A provides the basic matrix decomposition, $A = RREF \cdot$ Column Space(A). So the RREF will generate both the basis for rowspace and columnspace. The non-zero rows of $rref(A)$ will provide the rowspace, and the columns of A with pivot elements in $rref(A)$ will generate columnspace of A. In short $rref(A)$ is the *extractor* function that squeeze-out informative part of a linear transformation Ax.

6.5 More on Basis of Fundamental Subspaces

7. If you are given a 5x4 matrix, how will you get bases for all the spaces associated with the matrix? Write a `Matlab` code for the same and put all the bases in column format.

SOLUTION

For a 5×4 matrix A, both the rowspace and columnspace can be generated from its row-reduced Echelon form. The non zero rows (equivalent to rank of A) will generate the rowspace of A and the columns of A corresponding to pivot elements of $rref(A)$ will generate the columnspace. Solutions of the system $Ax = 0$ and $A^T x = 0$ will provide the right and left null spaces of A respectively. `Matlab` code to demonstrate this task is given below.

Input program 32: *Fundamental subspaces of a Matrix*

```
rng('shuffle');
A = rand(5, 4);
R = rref(A);
[~, pivot_columns] = rref(A);
column_space_basis = A(:, pivot_columns);
row_space_basis = R(1:rank(A), :);
```

```
right_null_space = null(A, 'r');
left_null_space = null(A', 'r');
disp('Matrix A:')
disp(A);
disp('Column Space Basis:');
disp(column_space_basis);
disp('Row Space Basis:');
disp(row_space_basis');
disp('Right Null Space:');
disp(right_null_space);
disp('Left Null Space:');
disp(left_null_space);
```

Output of the code is given below.

```
Matrix A:
    0.5897    0.3138    0.6255    0.2887
    0.1635    0.3655    0.8607    0.6471
    0.9634    0.9925    0.3206    0.5262
    0.1342    0.0894    0.3207    0.7774
    0.9570    0.0280    0.9718    0.7735
Column Space Basis:
    0.5897    0.3138    0.6255    0.2887
    0.1635    0.3655    0.8607    0.6471
    0.9634    0.9925    0.3206    0.5262
    0.1342    0.0894    0.3207    0.7774
    0.9570    0.0280    0.9718    0.7735
Row Space Basis:
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
Right Null Space:

Left Null Space:
   -2.4563
    0.9045
    0.5223
   -1.1893
    1.0000
```

RESULTS

1. Revisited the method to generate general solution of a linear system with infinite number of solutions.
2. Basis of all fundamental subspaces of a given random matrix of various dimensions are investigated.
3. Projection of vectors onto various subspaces of a random matrix is investigated.
4. Revisited both abstract and computational approaches to generate basis of all fundamental subspaces of a given matrix A.

7 | Assignment-7 Back to the Origin

7.1 Basis of Fundamental Subspaces

1. For the matrix associated with $x + y = 1$, find set of vectors (of appropriate tuple-size) that span

   (a) row space

   (b) column space

   (c) left null space

   (d) right null space

SOLUTION

The given system can be written as $\begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} = 1$

Since it is linear system with one equation and three variables satisfying

$$\rho(A) = 1 = \rho(K) < \text{ number of variables}$$

it is consistent and has many solutions. Here the rowspace is $\{[1,1]\}$ and the column space is $\{(1),(1)\}$. Using the rank nullity theorem, dimension of right null space is $2-1 = 1$. So the right null space contains a non-trivial element. It is the solution of the equation, $x + y = 0$. So the basis of right null space is $\left\{\begin{bmatrix}-1\\1\end{bmatrix}\right\}$ Similarly dimension of the left null space is $2-2 = 0$. Hence the left null space contains only trivial element, 0. The computational part of this task is given below.

</> **Input program 33:** *Basis of Fundamental subspace of Linear System $x + y = 1$* </>

```

A = [1 1];
b = [1];
R = rref([A]);
column_space_basis = A;
row_space_basis = R(1:rank(A), :);
right_null_space = null(A, 'r');
left_null_space = null(A', 'r');
disp('Column Space Basis:');
disp(column_space_basis);
disp('Row Space Basis:');
disp(row_space_basis);
disp('Right Null Space:');
if isempty(right_null_space)
```

```
    disp('Right null space does not exist (only the trivial solution
     ↪  exists).');
else
    disp('Right null space exists (non-trivial solutions exist).');
    disp(right_null_space);
end
disp('Left Null Space:');
if isempty(left_null_space)
    disp('Left null space does not exist (only the trivial solution
     ↪  exists).');
else
    disp('Left null space exists (non-trivial solutions exist).');
    disp(left_null_space);

end
```

Output of the above code is shown below.

```
Column Space Basis:
     1     1
Row Space Basis:
Right null space exists (non-trivial solutions exist).
    -1
     1
Left Null Space:
Left null space does not exist (only the trivial solution exists).
```

2. For the following matrices, find basis of all fundamental subspaces.

(a) $\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$

(b) $\begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{pmatrix}$

(c) $\begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \end{pmatrix}$

SOLUTION

(a) The `Matlab` code for this task is given below.

Input program 34: *Fundamental subspace of M1*

```
A = [1 4; 2 5; 3 6];
R = rref(A);
[~, pivot_columns] = rref(A);
column_space_basis = A(:, pivot_columns);
row_space_basis = R(1:rank(A), :);
right_null_space = null(A, 'r');
left_null_space = null(A', 'r');
disp('Column Space Basis:');
disp(column_space_basis);
disp('Row Space Basis:');
disp(row_space_basis);
```

```
disp('Right Null Space:');
if isempty(right_null_space)
    disp('Right null space does not exist (only the trivial solution
     exists).');
else
    disp('Right null space exists (non-trivial solutions exist).');
    disp(right_null_space);
end
disp('Left Null Space:');
if isempty(left_null_space)
    disp('Left null space does not exist (only the trivial solution
     exists).');
else
    disp('Left null space exists (non-trivial solutions exist).');
    disp(left_null_space);

end
```

```
Column Space Basis:
     1     4
     2     5
     3     6
Row Space Basis:
     1     0
     0     1
Right Null Space:
Right null space does not exists (only trivial solutions exist).

Left Null Space:
Left null space exists (non-trivial solutions exist).
     1
    -2
     1
```

(b) `Matlab` code for this task is given below.

Input program 35: *Fundamental subspaces of M2*

```
% Define the matrix A
A3 = [1 2; 2 4; 3 6];
R3 = rref(A3);
[~, pivot_columns] = rref(A3);
column_space_basis3 = A3(:, pivot_columns);
row_space_basis3 = R3(1:rank(A3), :);
right_null_space3 = null(A3, 'r');
left_null_space3 = null(A3', 'r');
disp('Column Space Basis:');

disp(column_space_basis3)

disp('Row Space Basis:');
disp(row_space_basis3);

```

```
disp('Right Null Space:');
if isempty(right_null_space3)
    disp('Right null space does not exist (only the trivial solution
     exists).');
else
    disp('Right null space exists (non-trivial solutions exist).');
    disp(right_null_space3);
end

disp('Left Null Space:');
if isempty(left_null_space3)
    disp('Left null space does not exist (only the trivial solution
     exists).');
else
    disp('Left null space exists (non-trivial solutions exist).');
    disp(left_null_space3);

end
```

```
Column Space Basis:
     1
     2
     3
Row Space Basis:
     1     2
Right Null Space:
Right null space exists (non-trivial solutions exist).
    -2
     1
Left Null Space:
Left null space exists (non-trivial solutions exist).
    -2    -3
     1     0
     0     1
```

(c) `Matlab` code for this task is given below.

Input program 36: *Fundamental subspaces of M3*

```
A4 = [1 1 2; 2 1 1];
R4 = rref(A4);
[~, pivot_columns4] = rref(A4);
column_space_basis4 = A4(:, pivot_columns4);
row_space_basis4 = R4(1:rank(A4), :);
right_null_space4 = null(A4, 'r');
left_null_space4 = null(A4', 'r');
disp('Column Space Basis:');
disp(column_space_basis4);

disp('Row Space Basis:');
disp(row_space_basis4);

disp('Right Null Space:');
```

```
if isempty(right_null_space4)
    disp('Right null space does not exist (only the trivial solution exists).');
else
    disp('Right null space exists (non-trivial solutions exist).');
    disp(right_null_space4);
end

disp('Left Null Space:');
if isempty(left_null_space4)
    disp('Left null space does not exist (only the trivial solution exists).');
else
    disp('Left null space exists (non-trivial solutions exist).');
    disp(left_null_space4);

end
```

```
Column Space Basis:
     1     1
     2     1
Row Space Basis:
     1     0    -1
     0     1     3
Right Null Space:
Right null space exists (non-trivial solutions exist).
     1
    -3
     1
Left Null Space:

Left null space does not exist (only the trivial solution exists).
```

7.2 Big Picture of Fundamental Subspaces

3. Summary for $A_{m\times n}$ matrix: verify the following with suitable example

   (a) Appending(joining) row-space basis vectors and Rightnull basis vectors, we obtain a basis set for vector space (note that do not append zero vectors if any space is empty)

   (b) Appending(joining) column-space basis vectors and left-null basis vectors, we obtain basis set for vector space . (note that do not append zero vectors if any space is empty). You can use rank command to verify the independence of joined (appended list of) vectors.

SOLUTION

This task summarizes the big picture of fundamental subspaces of a matrix. For any $m \times n$ matrix A, there are two pairs of orthogonal spaces namely the row space & right null space and the column space & left null space. Furthermore. So, just by the rank nullity theorem, these observation directly follows. This property can be verified using following `matlab` code.

Input program 37: *Summary of Fundamental Theorem of Linear Algebra*

```
 A5 = [1 2 3; 4 5 6; 7 8 9];
 R5 = rref(A5);
 row_space_basis5 = R5(1:rank(A5), :);
right_null_space5 = null(A5, 'r');
[~, pivot_columns5] = rref(A5);
column_space_basis5 = A5(:, pivot_columns5);
left_null_space5 = null(A5', 'r');
combined_row_right_null = [row_space_basis5; right_null_space5'];
rank_combined_row_right_null = rank(combined_row_right_null);
disp('Rank of the combined row-space and right-null space basis:');
disp(rank_combined_row_right_null);
combined_column_left_null = [column_space_basis5, left_null_space5];
rank_combined_column_left_null = rank(combined_column_left_null);
disp('Rank of the combined column-space and left-null space basis:');
disp(rank_combined_column_left_null);
disp('Row Space Basis:');
disp(row_space_basis4);
disp('Right Null Space Basis:');
disp(right_null_space4);
disp('Column Space Basis:');
disp(column_space_basis5);
disp('Left Null Space Basis:');
disp(left_null_space5);
```

```
Rank of the combined row-space and right-null space basis:
     3
Rank of the combined column-space and left-null space basis:
     3
Row Space Basis:
     1     0    -1
     0     1     2
Right Null Space Basis:
     1
    -2
     1
Column Space Basis:
     1     2
     4     5
     7     8
Left Null Space Basis:
     1
    -2
     1
```

8 Assignment-8 Projection of Vectors and More

8.1 Creating Vectors with Special Characteristics

1. Write a matlab code for Creating 5x5 matrix with rank 4 and generate 10 vectors each that are not in
   (a) Column space
   (b) Rowspace
   (c) Left null space
   (d) Right null space
2. Explain how will you verify your answer for various cases in problem 1
3. Write a matlab code for Creating 5x5 matrix A with rank 3 and generate 3 different basis set for column space. Form 3 matrices B1, B2 and B3 with above basis set as column vectors. Generate a 5-tuple random vector and project onto column space of above 3 matrices. Check whether the resulting projected vectors are same or not.

SOLUTION

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

Input program 38: *Creation of vectors which is not in column space*

```
    A = rand(5, 5);
A(:,5) = A(:,1) + A(:,2);

fprintf('Rank of A: %d\n', rank(A));

N_col = null(A');
N_row = null(A);

B_col = A * rand(5, 10);
B_null_col = N_col * rand(size(N_col,2), 10);
V_not_colspace = B_col + B_null_col;

B_row = A' * rand(5, 10);
B_null_row = N_row * rand(size(N_row,2), 10);
V_not_rowspace = B_row + B_null_row;

V_not_leftnullspace = A' * rand(5, 10) + N_col * rand(size(N_col,2), 10);

```

```
V_not_rightnullspace = A * rand(5, 10) + N_row * rand(size(N_row,2),
    10);

disp('Vectors not in the column space:');
disp(V_not_colspace);

disp('Vectors not in the row space:');
disp(V_not_rowspace);

disp('Vectors not in the left null space:');
disp(V_not_leftnullspace);

disp('Vectors not in the right null space:');
disp(V_not_rightnullspace);
```

```
Rank of A: 4
Vectors not in the column space:
    1.4253    1.3385    0.3417    1.2150    1.5266    1.1265    1.0978    1.1547    0.3903    0.7681
    1.1292    1.7974    1.0515    1.6832    2.5928    1.1931    1.4905    1.5785    1.1226    0.7584
    0.5729    1.1018    1.0090    0.8495    2.1366    0.7491    0.7416    1.1861    0.8613    0.3918
    1.4182    2.0753    1.0879    1.8023    2.2534    1.1928    1.4218    1.7698    1.0670    1.1175
    1.7534    2.6184    1.3831    2.1079    2.9298    1.3087    1.6965    2.7833    1.8919    1.8757
Vectors not in the row space:

    1.0999    0.5267    0.6793    1.2849    0.4291    0.4519    0.8449    0.8740    1.5127    0.3275
    1.4948    1.0972    1.1181    1.2874    0.6857    0.8225    0.9492    1.2649    1.6643    0.5041
    1.6345    1.5358    1.3294    1.3504    1.3763    1.0939    1.0410    2.0830    2.1758    0.6519
    1.7544    0.8512    1.3530    1.6940    1.0368    1.2674    1.4055    1.1231    1.7540    0.7127
    3.3273    1.7871    2.8341    3.3879    2.3202    2.4866    2.9001    2.1972    3.2962    1.3852
Vectors not in the left null space:
    1.3339    1.3879    0.7386    1.3325    1.3223    1.1102    1.0339    1.0831    1.0392    1.2893
    1.7673    0.8388    0.5309    1.4988    1.7677    0.9378    1.3303    0.9753    0.8891    0.7028
    1.8114    1.2306    1.0515    1.2258    2.0396    1.2375    1.1992    1.3753    1.2292    1.0462
    1.7734    1.2854    0.7131    1.6311    1.7588    1.0463    1.2504    1.2652    0.9125    1.0345
    3.2048    2.6519    1.5484    2.9212    3.2909    2.4058    2.4772    2.4933    2.0713    2.5324
Vectors not in the right null space:
    0.1315    0.7095    0.7113    0.0625    0.7791    0.7399    0.5831    0.4037    0.8070    0.3258
    0.8292    1.8223    1.9263    0.9550    1.8257    1.7839    1.2969    1.6570    2.0329    0.6348
    0.8749    1.8814    1.7061    1.4708    1.8486    1.8256    1.4595    1.7514    1.8458    0.9484
    0.9863    1.7874    1.7066    1.0679    1.8908    1.9795    1.5269    2.0150    1.7832    0.4909
    1.4033    2.5820    2.4448    2.3409    2.5867    2.6257    2.2136    2.7993    2.4770    1.1647
```

8.2 Checking Membership of a Vector in Fundamental Subspaces

2. Verification methods.

Not in column space: A vector v is not in column space, then it will not be orthogonal to the left null space. So, it is enough to prove that $A^T \cdot v \neq 0$. Matlab code for this task is given below.

Input program 39: *Checking for $v \notin$ Col-space(A)*

```
tol = 1e-6;
disp('Checking vectors not in column space:');
for i = 1:10
    if norm(A' * V_not_colspace(:,i)) > tol
        fprintf('Vector %d is NOT orthogonal to the column
            space\n', i);
    else
        fprintf('Vector %d is orthogonal to the column space.\n',
            i);
    end
end
```

Output of this task is shown below.

```
Checking vectors not in column space:
Vector 1 is NOT orthogonal to the column space
Vector 2 is NOT orthogonal to the column space
Vector 3 is NOT orthogonal to the column space
Vector 4 is NOT orthogonal to the column space
Vector 5 is NOT orthogonal to the column space
Vector 6 is NOT orthogonal to the column space
Vector 7 is NOT orthogonal to the column space
Vector 8 is NOT orthogonal to the column space
Vector 9 is NOT orthogonal to the column space
Vector 10 is NOT orthogonal to the column space
```

Not in row space: A vector v is not in row space, then it will not be orthogonal to the right nullspace. So it is enough to prove that $AT \cdot v \neq 0$. `Matlab` code for this task is given below.

Input program 40: *Checking for $v \notin$ Row-space(A)*

```
disp('Checking vectors not in row space:');
for i = 1:10
    if norm(A * V_not_rowspace(:,i)) > tol
        fprintf('Vector %d is NOT orthogonal to the row space\n', i);
    else
        fprintf('Vector %d is orthogonal to the row space\n', i);
    end
end
```

```
Checking vectors not in row space:
Vector 1 is NOT orthogonal to the row space
Vector 2 is NOT orthogonal to the row space
Vector 3 is NOT orthogonal to the row space
Vector 4 is NOT orthogonal to the row space
Vector 5 is NOT orthogonal to the row space
Vector 6 is NOT orthogonal to the row space
Vector 7 is NOT orthogonal to the row space
Vector 8 is NOT orthogonal to the row space
Vector 9 is NOT orthogonal to the row space
Vector 10 is NOT orthogonal to the row space
```

Not in right null space: A vector v is not in right null space, then it will not be orthogonal to the row space. So, it is enough to prove that $A \cdot v \neq 0$. `Matlab` code for this task is given below.

Input program 41: *Checking for $v \notin$ right-Null-space(A)*

```
\end{description}
disp('Checking vectors not in right null space:');
for i = 1:10
    if norm(A * V_not_rightnullspace(:,i)) > tol
        fprintf('Vector %d is NOT orthogonal to the right null space\n', i);
    else
        fprintf('Vector %d is orthogonal to the right null space\n', i);
    end
end
```

Output of the above code is shown below.

```
Checking vectors not in right null space:
Vector 1 is NOT orthogonal to the right null space
Vector 2 is NOT orthogonal to the right null space
Vector 3 is NOT orthogonal to the right null space
Vector 4 is NOT orthogonal to the right null space
Vector 5 is NOT orthogonal to the right null space
Vector 6 is NOT orthogonal to the right null space
Vector 7 is NOT orthogonal to the right null space
Vector 8 is NOT orthogonal to the right null space
Vector 9 is NOT orthogonal to the right null space
Vector 10 is NOT orthogonal to the right null space
```

Not in left null space: A vector v is not in left null space, then it will not be orthogonal to the column space. So, it is enough to prove that $A^T \cdot v \neq 0$. `Matlab` code for this task is given below.

Input program 42: *Checking for $v \not\in$ left-Null-space(A)*

```
disp('Checking vectors not in left null space:');
for i = 1:10
    if norm(A' * V_not_leftnullspace(:,i)) > tol
        fprintf('Vector %d is NOT orthogonal to the left null
        ↪ space\n', i);
    else
        fprintf('Vector %d is orthogonal to the left null space\n',
        ↪ i);
    end
end
```

```
Checking vectors not in left null space:
Vector 1 is NOT orthogonal to the left null space
Vector 2 is NOT orthogonal to the left null space
Vector 3 is NOT orthogonal to the left null space
Vector 4 is NOT orthogonal to the left null space
Vector 5 is NOT orthogonal to the left null space
Vector 6 is NOT orthogonal to the left null space
Vector 7 is NOT orthogonal to the left null space
Vector 8 is NOT orthogonal to the left null space
Vector 9 is NOT orthogonal to the left null space
Vector 10 is NOT orthogonal to the left null space
```

8.3 Some Special Projections

3. Write a `matlab` code for Creating 5x5 matrix, A with rank 3 and generate 3 different basis set for column space. Form 3 matrices B1, B2 and B3 with above basis set as column vectors. Generate a 5-tuple random vector and project onto column space of above 3 matrices. Check whether the resulting projected vectors are same or not.

SOLUTION

`Matlab` code for this task is given below.

Input program 43: *Projections*

```
AR = rand(5, 3) * rand(3, 5);
fprintf('Rank of A: %d\n', rank(AR));
B1 = AR(:, [1 2 3]);
B2 = AR(:, [2 3 4]);
B3 = AR(:, [3 4 5]);
v = rand(5,1);
fprintf('Random 5-tuple vector:\n');
disp(v);
proj_B1 = B1 * (inv(B1' * B1) * B1' * v);
proj_B2 = B2 * (inv(B2' * B2) * B2' * v);
proj_B3 = B3 * (inv(B3' * B3) * B3' * v);
fprintf('Projection onto column space of B1:\n');
disp(proj_B1);
fprintf('Projection onto column space of B2:\n');
disp(proj_B2);
fprintf('Projection onto column space of B3:\n');
disp(proj_B3);

tol = 1e-6;

if norm(proj_B1 - proj_B2) < tol && norm(proj_B2 - proj_B3) < tol
    fprintf('All projected vectors are the same.\n');
else
    fprintf('The projected vectors are different.\n');
end
```

Output of the above code and results of verification is given below.

```
Rank of A: 3
Random 5-tuple vector:
    0.8266
    0.3945
    0.6135
    0.8186
    0.8862
Projection onto column space of B1:
    0.8891
    0.6293
    0.6883
    0.7356
    0.3919
Projection onto column space of B2:
    0.8891
    0.6293
    0.6883
    0.7356
    0.3919
Projection onto column space of B3:
    0.8891
    0.6293
    0.6883
    0.7356
```

```
    0.3919
All projected vectors are the same.
```

9 | Assignment-9
Creating Multivariate Clusters

9.1 Data Simulation for Machine Learning Task

Background: Given mean vector `mu` from and positive definite covariance matrix `sigma` from $\mathbf{R}^{\mathbf{n}\times\mathbf{n}}$, we should be able to generate data. `Matlab` command `mvnrnd(mu, sigma,N)` does it. A sample data generated is given below.

```
mu1 = [3 -3]; Sigma1 = [.9 -.2; -.2 .8];
r1 = mvnrnd(mu1, Sigma1, 100); plot(r1(:,1),r1(:,2),'.');
hold on
mu2 = [1 2]; Sigma2 = [.9 0; 0 .3];
r2 = mvnrnd(mu2, Sigma2, 100);
plot(r2(:,1),r2(:,2),'x');
hold on
mu3 = [-2 -2]; Sigma3 = [1 0; 0 .2];
r3 = mvnrnd(mu3, Sigma3, 100);
plot(r3(:,1),r3(:,2),'o');
```

1. Re-estimate mean and covariance for each dataset generated.
2. Check eigen values of the given covariance matrices are positive definite or not
3. From Wikipedia, understand what is called “k-means” clustering. Write a note about that
4. Challenge yourself by trying to write a code for k-means clustering for above data (all three cluster data put together)

SOLUTION

</> **Input program 44:** *Estimating mean and Covariance* </>

```
estimated_mean_r1 = mean(r1);
r1_centered = r1 - repmat(estimated_mean_r1, size(r1,1), 1);
estimated_cov_r1 = (r1_centered' * r1_centered) / (size(r1,1) - 1);
estimated_mean_r2 = mean(r2);
r2_centered = r2 - repmat(estimated_mean_r2, size(r2,1), 1);
estimated_cov_r2 = (r2_centered' * r2_centered) / (size(r2,1) - 1);
disp('Estimated means');
disp(estimated_mean_r1);
disp(estimated_mean_r2);
disp(estimated_mean_r3);
disp('Estimated covariances');
disp(estimated_cov_r2)
estimated_mean_r3 = mean(r3);
```

```
r3_centered = r3 - repmat(estimated_mean_r3, size(r2,1), 1);
estimated_cov_r3 = (r3_centered' * r3_centered) / (size(r3,1) - 1);
disp(estimated_cov_r3)
```

Output of above code is given below.

```
Estimated means:
    3.0163   -2.9574
    0.8797    1.9849
   -2.0502   -2.0419
Estimated covariance:
    0.7670    0.0040
    0.0040    0.2716

    1.1276   -0.0305
   -0.0305    0.1716

    1.1276   -0.0305
   -0.0305    0.1716
```

Checking positive definiteness: `Matlab` code for this task is given below.

</> **Input program 45:** *Checking Positive Definiteness* </>

```
% Check eigenvalues of the covariance matrices
eigenvalues_r1 = eig(estimated_cov_r1);
eigenvalues_r2 = eig(estimated_cov_r2);
eigenvalues_r3 = eig(estimated_cov_r3);

fprintf('Eigenvalues of Covariance Matrix for r1:\n');
disp(eigenvalues_r1);
if all(eigenvalues_r1 > 0)
    fprintf('Covariance matrix for r1 is positive definite.\n');
else
    fprintf('Covariance matrix for r1 is not positive definite.\n');
end

fprintf('Eigenvalues of Covariance Matrix for r2:\n');
disp(eigenvalues_r2);
if all(eigenvalues_r2 > 0)
    fprintf('Covariance matrix for r2 is positive definite.\n');
else
    fprintf('Covariance matrix for r2 is not positive definite.\n');
end

fprintf('Eigenvalues of Covariance Matrix for r3:\n');
disp(eigenvalues_r3);
if all(eigenvalues_r3 > 0)
    fprintf('Covariance matrix for r3 is positive definite.\n');
else
    fprintf('Covariance matrix for r3 is not positive definite.\n');
end
```

Output of this code is given below.

```
Eigenvalues of Covariance Matrix for r1:
    0.6441
    1.0798
Covariance matrix for r1 is positive definite.
Eigenvalues of Covariance Matrix for r2:
    0.3229
    0.8442
Covariance matrix for r2 is positive definite.
Eigenvalues of Covariance Matrix for r3:
    0.1695
    1.1109
Covariance matrix for r3 is positive definite.
```

9.2 Introduction to K-means Clustering

K-means clustering is a widely used unsupervised learning algorithm that partitions a set of n data points into K clusters. The objective is to minimize the sum of squared distances between the data points and the centroid of the clusters they are assigned to.
Given a dataset $\mathcal{X} = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$, the goal is to partition the data into K disjoint subsets $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$, where each subset represents a cluster.

9.2.1 Mathematical Formulation

The K-means algorithm aims to solve the following optimization problem:

$$\min_{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K} \sum_{k=1}^{K} \sum_{x_i \in \mathcal{C}_k} \|x_i - \mu_k\|^2$$

where μ_k is the centroid (mean) of cluster $\mathcal{C}_k$, defined as:

$$\mu_k = \frac{1}{|\mathcal{C}_k|} \sum_{x_i \in \mathcal{C}_k} x_i$$

The objective is to assign each point x_i to the nearest cluster such that the total sum of squared distances to the cluster centroids is minimized.

9.2.2 Distance Metric

K-means uses the Euclidean distance as the distance metric. The distance between a data point x_i and the centroid μ_k of cluster $\mathcal{C}_k$ is given by:

$$d(x_i, \mu_k) = \|x_i - \mu_k\|^2 = (x_i - \mu_k)^T (x_i - \mu_k)$$

9.3 K-Means Algorithm

The K-means algorithm proceeds in an iterative manner, alternating between two steps:

- **Assignment Step:** Assign each data point x_i to the cluster whose centroid μ_k is closest:

$$\mathcal{C}_k = \{x_i : \|x_i - \mu_k\|^2 \le \|x_i - \mu_j\|^2 \quad \forall j, 1 \le j \le K\}$$

- **Update Step:** Update the centroid of each cluster as the mean of the data points assigned to it:

$$\mu_k = \frac{1}{|\mathcal{C}_k|} \sum_{x_i \in \mathcal{C}_k} x_i$$

Algorithm 1 K-Means Algorithm

```
Initialize K centroids μ_1, ..., μ_K randomly from the data points.
repeat
    Assign each data point to the closest centroid.
    Recompute the centroids using the points assigned to each cluster.
until Convergence (i.e., no change in the cluster assignments or centroids).
```

9.4 Elbow Method for Determining Optimal K

The number of clusters K is often not known in advance. A common technique to determine the optimal K is the *elbow method*, which minimizes the *within-cluster sum of squares* (WCSS). WCSS for a cluster $\mathscr{C}_k$ is defined as:

$$\text{WCSS}(\mathscr{C}_k) = \sum_{x_i \in \mathscr{C}_k} \|x_i - \mu_k\|^2$$

The total WCSS is:

$$\text{WCSS}_{\text{total}} = \sum_{k=1}^{K} \text{WCSS}(\mathscr{C}_k)$$

By plotting WCSS as a function of K, the optimal K can often be identified as the point where the reduction in WCSS slows down, forming an "elbow."

9.5 K-Means with Noisy Data from a `mvnrnd()` Simulation

To evaluate the robustness of the K-means algorithm, noise can be added to the data points. Let $\mathscr{X}_{\text{noisy}}$ represent the dataset with Gaussian noise added:

$$\mathscr{X}_{\text{noisy}} = \mathscr{X} + \mathscr{N}(0, \sigma^2)$$

where $\mathscr{N}(0, \sigma^2)$ is Gaussian noise with variance σ^2. The K-means algorithm is then applied to this noisy data, and the clustering performance is evaluated based on the accuracy of the cluster assignments.

The K-means clustering algorithm is tested on a noised version of the simulated data created in the beginning of the section. `Matlab` code for this simple Machine Learning Task is given below.

</> **Input program 46:** *Creating a Noisy Data for K-means Clustering* </>

```matlab
data = [r1; r2; r3];
noise_level = 0.5;
noise = noise_level * randn(size(data));
data_noisy = data + noise;
```

</> **Input program 47:** *Set-up Elbow-method to identify best value of K* </>

```matlab
max_K = 10;
WCSS = zeros(max_K, 1);
for k = 1:max_K
    [~, ~, sumd] = kmeans(data_noisy, k, 'Replicates', 10);
    WCSS(k) = sum(sumd);
end
figure;
plot(1:max_K, WCSS, 'b-o');
title('Elbow Method for Optimal K');
xlabel('Number of Clusters (K)');
ylabel('WCSS');
grid on;
```

The elbow plot created for the simulated noise data is shown in Figure 9.1.

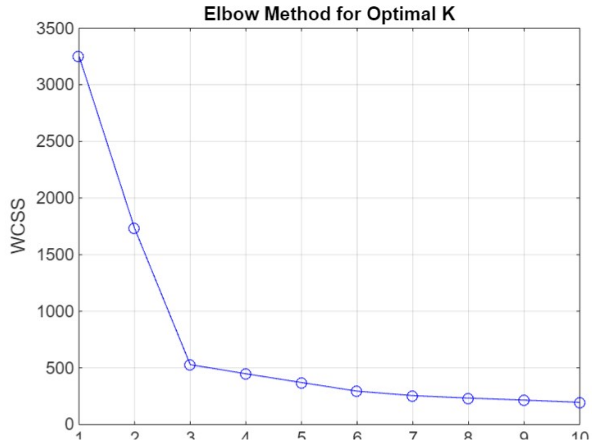

Figure 9.1: Elbow plot for K-means clustering

From Figure 9.1, it is clear that the WCSS is sharply decreases with the increase in number of clusters, k till it reaches 2. After crossing this limit, there is no significantly high decrease in the WCSS. So let's fix $k = 3$ is the optimum number of clusters suit to this dataset.
The algorithm is replicated initially 300 times to get a convergent solution. Specifically, K-means converges when the cluster assignments of data points no longer change between successive iterations or when the centroids stabilize (i.e., they stop moving significantly). In this case the K-means algorithm is 'Does not converge' status. it indicates that one or more of the following may have occurred:

Insufficient Iterations: The algorithm reached the maximum number of iterations without stabilizing. Increasing the 'MaxIter' parameter might help.

Poor Initialization: K-means can be sensitive to the initial placement of centroids. Random initialization might have placed centroids in suboptimal locations, causing difficulty in finding the optimal clusters. You can try using multiple replicates or the 'kmeans++' initialization method.

Noisy or Poorly Separated Data: Adding noise or if clusters are not well-separated, it can be harder for K-means to converge, as the algorithm struggles to find distinct centroids.

Degenerate Cases: Sometimes K-means can result in empty clusters or very small clusters that prevent convergence.

All the above-mentioned ill conditions are checked one by one. MaxIter increased to 6000 and initialization method is updated with suggested methods, but of no use. Finally, the spread of the data and existence of degenerated cases are examined through the scatter plot of the clusters. `Matlab` code for this entire task is given below.

Input program 48: *Implementing K-means Clustering*

```
K = 3;
[idx, centroids,sumd, D] = kmeans(data_noisy, K,'MaxIter',300, 'Replicates',
  10);
support = histcounts(idx, K);
T = table((1:K)', support', centroids, 'VariableNames', {'Cluster',
  'Support', 'ClusterCenters'});
disp(T);
opts = statset('MaxIter', 6000);
[~, ~, sumd_final] = kmeans(data, K, 'Options', opts, 'MaxIter', 6000,
  'Replicates', 5,'Start','plus');
if isequal(sumd, sumd_final)
```

```
    fprintf('Converged\n');
else
    fprintf('Did not converge\n');
end
```

The cluster centers and support for each cluster is shown in the following output.

```
    Cluster    Support      ClusterCenters
    -------    -------    ------------------

       1          99        2.8699    -2.9934
       2         100       0.91757     1.9222
       3         101       -1.8739     -1.973
```

A scatter plot of clusters with cluster centers are generated.`Matlab` code for this task is given below.

Input program 49: *Scatter plot of Clusters*

```
figure;
gscatter(data_noisy(:,1), data_noisy(:,2), idx);
hold on;
plot(centroids(:,1), centroids(:,2), 'kx', 'MarkerSize', 15, 'LineWidth',
    3);
legend('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroids');
title('K-Means Clustering with Noisy Data');
hold off;
```

Output of this code is shown in Figure 9.2.

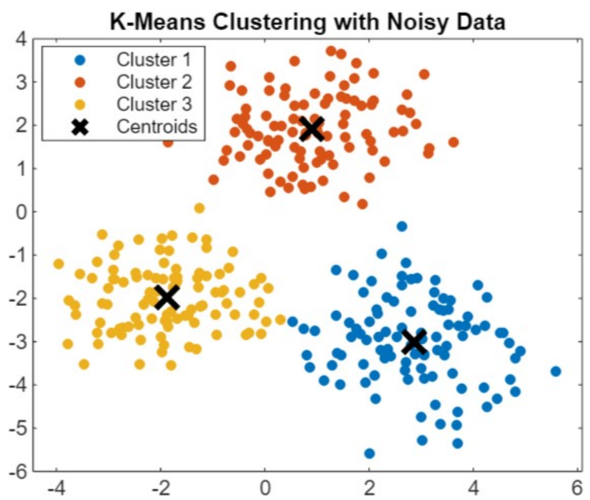

Figure 9.2: Visualization of Clusters with respective centroids

From Figure 9.2, it is clear that there are more noise so that some of the data points may misclassified and there is no degenerated clusters (Equal number of data points for three different classes are simulated. But from the support for final clustering iteration, one point from cluster one is allotted to cluster 3). So, a scaling of the data and/or use of Gaussian K-means clustering or Hierarchical clustering may be getting a convergent solution.

RESULTS

1. A bi-variate dataset is generated using the `mvnrnd()` with given mean and covariance.

2. Estimated Mean and covariance of the simulated data is calculated and compared with the original values used for data generation

3. Positive definiteness of the covariance matrix it confirmed using eigen values. So the distribution has a meaningful spread and the multinomial normal random variable has an ellipsoidal contours and prevent the points collapse into a lower-dimensional subspace.

4. The unsupervised Machine Learning algorithm- K means clustering- is reviewed.

5. Using `Matlab`, a K-means clustering model is developed on the simulated data, optimum number of clusters is identified using Elbow method, centroids & supports of each cluster is identified

6. Convergence of the algorithm is tested and is found to be failed.

10 Assignment-10 Unsupervised to Supervised (*MIMO Linear Classifier)*

10.1 Simulating a Labelled Dataset

Background:

Three clusters of data are being Generated, each with a multivariate normal distribution, representing three classes along with class labels. The data points are spread based on the mean and covariance matrices that provided for each cluster. The `Matlab` code for the data simulation is given below.

Input program 50: *Simulated labelled Data*

```
N=100
mu1 = [3 -3]; Sigma1 = [.9 -.2; -.2 .8];
r1 = mvnrnd(mu1, Sigma1, N);
l1=repmat([1 0 0], N,1);
mu2 = [1 2]; Sigma2 = [.9 0; 0 .3];
r2 = mvnrnd(mu2, Sigma2, N);
l2=repmat([0 1 0], N,1);
mu3 = [-2 -2]; Sigma3 = [1 0; 0 .2];
r3 = mvnrnd(mu3, Sigma3, 100);
l3=repmat([0 0 1], N,1);
plot(r1(:,1),r1(:,2),'.');
hold on
plot(r2(:,1),r2(:,2),'x');
hold on
plot(r3(:,1),r3(:,2),'o');
```

Three distributions created are shown in Figure 10.1.

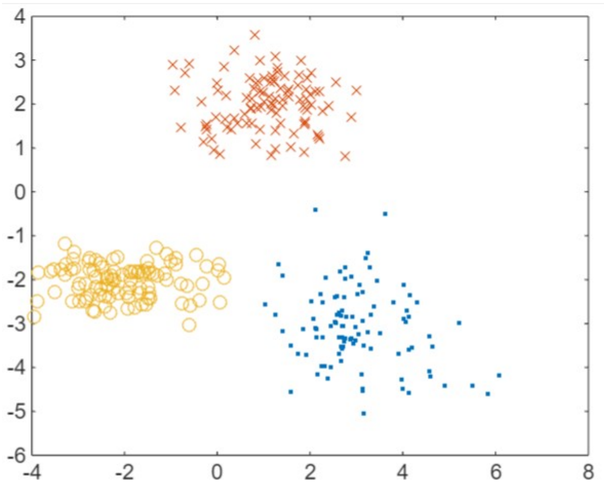

Figure 10.1: Distribution of input dataset created

From Figure 10.1, it is clear that there are three distinct clusters of points such that they are not overlapping. This provides linear separability to some extent.
Using these three multi variable normal distribution with different parameters, a new labelled dataset is created. The `Matlab` code for this task is given below.

10.2 Creation of a Linear Map

Input program 51: *Labelled Data Set for Classification Task*

```
X= [r1;r2;r3]; %append as rows
Y=[l1;l2;l3]; %append as rows
W=pinv(X)*Y;
```

10.3 Insights from the `Matlab` Code

This code implicitly assumes a linear relationship between the input and target as

$$Y = XW$$

Now under the assumption that there exist a linear map between X and Y, the mapping matrix, W is calculated using

$$W = X^{-1}Y$$

.
While comparing the nature of data generated and the type of association in the model assumption are not mathematically matching. Since the data is non-linear but the mapping assumed a linear association!

10.4 Checking the Validity of the Model Assumption

Next code chunk checks the validity of the model assumption by regenerating the labels with

$$Y_e = XW$$

Input program 52: *Regenerating class labels*

```
Ye=X*W
[val,idx] = max(Ye,[],2) ;
[~,j] = ind2sub(size(Ye(1,:)),idx) ;
i = [1:size(Ye,1)]' ;
[i j]
```

The matrix, Y_e is a 300×3 matrix with decimal values. At the first sight, it is clear that there is higher regeneration error (in terms of components of Y_e). A practical solution to get back the label is to choose the index of rows of Y_e with maximum value. For example from the second row, $[0.6824 - 0.0858 - 0.0781]$, the index with maximum value is 1. So, the class label is assigned as 1. Interestingly second row of Y is $[1, 0, 0]$ itself!
Accuracy is the most important measure to assess the skill of a classification model. Following code will calculate the classification accuracy of this model.

Input program 53: *Accuracy of the MIMO Linear Model*

```
[val, idx] = max(Ye, [], 2);
[~, actual_idx] = max(Y, [], 2);
correct_classifications = sum(idx == actual_idx);
total_points = length(actual_idx);
accuracy = (correct_classifications / total_points) * 100;
fprintf('Model accuracy:%2.3f',accuracy)
```

Output of the above code chunk is shown below.

```
Model accuracy:98.667
```

RESULTS

1. A two feature random multi-class dataset is created using `mvnrnd()` function.
2. A linear classifier is modelled using simple matrix operation
3. Accuracy in classification is calculated
4. Skill of the model is found to very good with an average accuracy of 98%.
5. The main reasons for fitting a linear classification model for a non-linear data may be:
   (a) The lower dimensional input data
   (b) Higher linear separability of the data due to positive definite covariance matrix and highly separated means
   (c) Use of pseudo inverse in the estimate of the model W, which minimize the least squares error
   (d) Relatively large number of samples
6. Simplicity of a MIMO Linear model is practically verified in an almost linearly separable dataset