SCHOOL OF ARTIFICIAL INTELLIGENCE

24MA602 Computational Mathematics for Data Science

Assignment Set-3

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 Figures

1 Assignment 21
Newton's Method for the Solution of Equations

1.1 Introduction

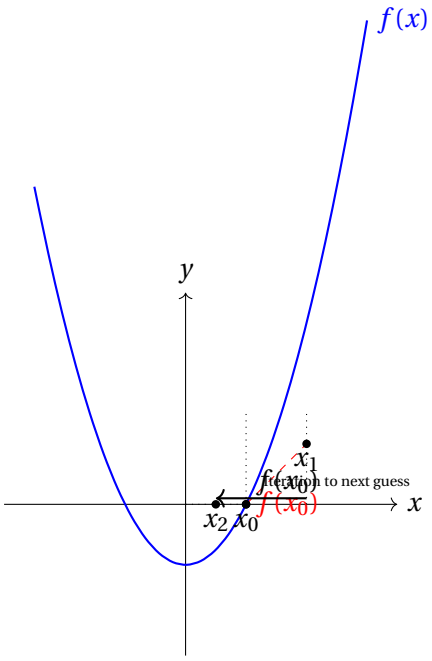
Newton's method is an iterative numerical technique used for finding approximations to the roots of a real-valued function. It is particularly useful when solving nonlinear equations.
Given a function $f(x)$, Newton's method finds successively better approximations to the root x such that $f(x) = 0$.

1.2 Mathematical Derivation

The core idea of Newton's method is to use the tangent line at an initial guess x_0 to approximate the function.

1.2.1 Starting point

Given an initial guess x_0, the goal is to find the root r such that $f(r) = 0$.



1.2.2 Newton's method as the tangent line approximation

The equation of the tangent line to the function $f(x)$ at $x = x_0$ is:

$$y = f(x_0) + f'(x_0)(x - x_0)$$

Setting $y = 0$ to find where the tangent line intersects the x-axis:

$$0 = f(x_0) + f'(x_0)(x - x_0)$$

Solving for x gives:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Thus, the update formula for Newton's method is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

1.2.3 Iterative process

Repeat the process until a sufficiently accurate approximation of the root is obtained. Stop when the difference between successive iterations is less than a tolerance ϵ:

$$|x_{n+1} - x_n| < \epsilon$$

1.3 Order of convergence

Newton's method has **quadratic convergence** near the root, provided that:

- $f(x)$ is sufficiently smooth (twice continuously differentiable),
- The initial guess x_0 is sufficiently close to the actual root,
- $f'(r) \neq 0$, where r is the root of $f(x)$.

1.3.1 Proof of quadratic convergence

Let the error at the n-th step be $\epsilon_n = x_n - r$. Using a Taylor expansion of $f(x)$ around r, we get:

$$f(x_n) = f(r) + f'(r)(x_n - r) + \frac{f''(\xi_n)}{2}(x_n - r)^2$$

where ξ_n lies between x_n and r.
Since $f(r) = 0$, the above simplifies to:

$$f(x_n) = f'(r)\epsilon_n + \mathcal{O}(\epsilon_n^2)$$

Now, using the Newton update formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Substituting the Taylor expansion for $f(x_n)$ into this formula, we can show that:

$$\epsilon_{n+1} \approx C\epsilon_n^2$$

where C is a constant dependent on $f''(r)$ and $f'(r)$. This implies **quadratic convergence**: the error at each step is proportional to the square of the error at the previous step.

1.4 Stopping Criteria

Two common stopping criteria are:

- **Relative Error:**
$$\frac{|x_{n+1} - x_n|}{|x_{n+1}|} < \epsilon$$
- **Absolute Error:**
$$|x_{n+1} - x_n| < \epsilon$$

1.5 Order of Complexity

1.5.1 Per iteration complexity

Each iteration involves evaluating $f(x_n)$ and $f'(x_n)$. The complexity depends on the cost of these evaluations.
For example, if $f(x)$ is a polynomial of degree d, evaluating both $f(x)$ and $f'(x)$ requires $\mathcal{O}(d)$ operations per iteration.

1.5.2 Overall complexity

Since Newton's method converges quadratically, the number of iterations required to achieve a certain accuracy ϵ is $\mathcal{O}(\log\log\frac{1}{\epsilon})$. Therefore, the total complexity is the product of the per-iteration complexity and the number of iterations.
If each iteration is $\mathcal{O}(d)$ and the method requires $\mathcal{O}(\log\log\frac{1}{\epsilon})$ iterations, then the overall complexity is:

$$\mathcal{O}(d \cdot \log\log\frac{1}{\epsilon})$$

1.6 Limitations

Even though the NR method offer fast convergence, there are still potential limitations. Two computationally striking limitations are:

- **Initial Guess Sensitivity:** Newton's method requires a good initial guess. If the guess is far from the actual root, the method may fail to converge.
- **Derivative Calculation:** If $f'(x)$ is zero or very small near the root, the method may fail or converge slowly.
- **Non-quadratic Convergence:** If the root is not simple (e.g., $f(x)$ has a higher-order root), the method may not exhibit quadratic convergence.

1.7 Example

Let's apply Newton's method to find the square root of a number a by solving $f(x) = x^2 - a = 0$.

1. Start with an initial guess x_0.
2. Apply the Newton iteration:
$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

This is the well-known method of **Heron** for computing square roots.

1.7.1 Computation examples

1. Find the square root of 700.

SOLUTION

Let $x = \sqrt{700} \implies x^2 = 700$ and $f(x) = x^2 - 700$.

$$f(x) = x^2 - 700$$
$$f'(x) = 2x$$

By Newton's method,

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{x_n^2 - 700}{2x_n} \\ &= \frac{x_n^2 + 700}{2x_n} \end{aligned}$$

Matlab code for this task is given below.

```
close(gcf)
x=-100:100;
y=x.^2-700;
plot(x,y);
hold on
Xax= [ -101    101];
Xay=[0 0];
plot(Xax,Xay);
hold on
z=100;
plot(z,z^2-700,'*');
sf=0.5 ;
for i= 1:10
    z=z-sf*(z^2-700)/(2*z);
    hold on
    plot(z,z^2-700,'*');
    caption = sprintf('z=%0.2f',z);
    title(caption, 'FontSize', 20);
    pause(.5);
end
hold on
z=-100;
plot(z,z^2-700,'*');
for i= 1:10
    z=z-sf*(z^2-700)/(2*z);
    hold on
    plot(z,z^2-700,'*');
    caption = sprintf('z=%0.2f',z);
    title(caption, 'FontSize', 20);
    pause(.5);
end
caption = sprintf('finished');
title(caption, 'FontSize', 20);
```

output of the code is shown in Figure 1.1

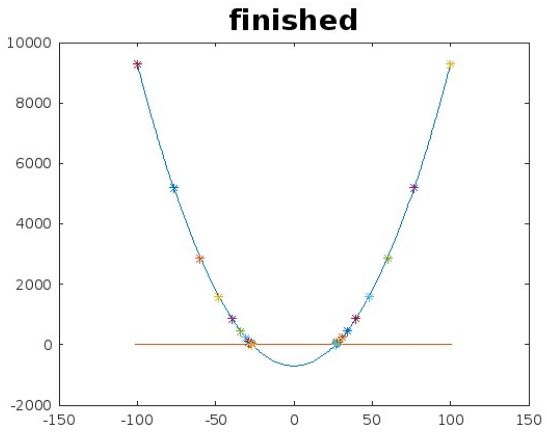


Figure 1.1: Solution path of $x = \sqrt{700}$ in Newton's method.

2. Write a `matlab` code for finding the cube root of a positive number.

SOLUTION

`Matlab` code for this task is given below.

```
a = input('Enter a positive number: ');
if a <= 0
    error('Please enter a positive number.');
end
x = a/2;
tol = 1e-6;
max_iter = 1000;
approximations = zeros(max_iter, 5);
iterations = 0
for iter = 1:
    f_x = x^3 - a;
    f_prime_x = 3*x^2;
    x_new = x - f_x / f_prime_x;
    approximations(iter, 1) = iter;
    approximations(iter, 2) = x;
    approximations(iter, 3) = f_x;
    approximations(iter, 4) = x_new;
    approximations(iter, 5) = x_new^3 - a;
    if abs(x_new - x) < tol
        iterations = iter;
        break;
    end
    x = x_new;
end
approximations = approximations(1:iterations, :);
disp(['The cube root of ', num2str(a), ' is approximately ',
   num2str(x
T = array2table(approximations, 'VariableNames', {'Sl_No', 'X_n
   ', 'f_X_n', 'x_next', 'f_x_next'});
disp(T);
```

Output of the code is shown below.

```
The cube root of 9 is approximately 2.0801

    Sl_No      X_n         f_X_n       x_next      f_x_next
    _____    ______    _________    ______    __________

      1         4.5        82.125    3.1481         22.201
      2      3.1481        22.201    2.4015         4.8493
      3      2.4015        4.8493    2.1212        0.54397
      4      2.1212       0.54397    2.0809       0.010269
      5      2.0809      0.010269    2.0801       3.901e-06
      6      2.0801     3.901e-06    2.0801      5.6488e-13
```

3. Create a dataset (x array and y array) for the step plot and code for getting the plot. Use the floor function $y = 0.5 + [x]$, $0 \le x \le 4$.

SOLUTION

Matlab code for this task is given below.

```
x = linspace(0, 5, 1000);
segments = [0 1; 1 2; 2 3; 3 4; 4 5];
y_values = 0.5 + floor(segments(:,1));
figure;
hold on;
for i = 1:size(segments, 1)
    x_segment = linspace(segments(i, 1), segments(i, 2), 200;
    y_value = 0.5 + floor(segments(i, 1));
    plot(x_segment, y_value * ones(size(x_segment)), 'LineWidth
       ', 1.5);
end
xlabel('x');
ylabel('y');
title('Graph of floor function');
grid on;
hold off;
```

Output of the code is shown in Figure 1.2.

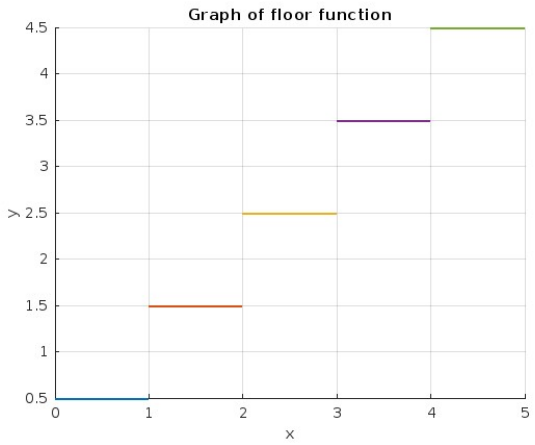


Figure 1.2: Graph of the floor function.

4. Expand a function $f(x) = x^3$ at $x_0 = 1$ using Madhava series and plot all the functions involved in a single diagram. Limit x to -3 to 3 .

SOLUTION

Matlab ocde for this task is given below.

```
x = linspace(-3, 3, 1000);
f_x = x.^3;
x0 = 1;
m_series = 1 + 3*(x - x0) + 3*(x - x0).^2 + (x - x0).^3;
figure;
hold on;
plot(x, f_x, 'k', 'LineWidth', 2, 'DisplayName', 'f(x) = x^3');
plot(x, m_series, 'r--', 'LineWidth', 1.5, 'DisplayName', '
   Madhava Series Expansion');
xlabel('x');
ylabel('y');
title('Function f(x) = x^3 and its Madhava Series Expansion at
   x0 = 1');
legend;
grid on;
hold off;
```

Output of the code is shown in Figure 1.3.

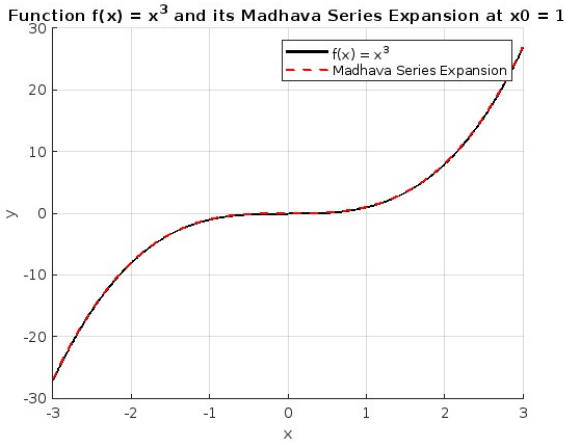


Figure 1.3: Approximation of $f(x) = x^3$ about $x = 1$ using Madhava series.

1.8 Conclusion

Newton's method is a numerical tool for solving nonlinear equations, especially when the function is differentiable and a good initial guess is available. With quadratic convergence, it often outperforms simpler methods like the bisection method in terms of speed, but its sensitivity to initial conditions and the need for derivative evaluation are its main drawbacks.

RESULTS

1. Fast converging numerical method to approximate the root of an algebraic or transcendental equation is revisited.
2. Madhava-Taylor series approximation for continuous function is revisited and compare the series approximation with orginal function.

2 Assignment 22
Concept of Full Rank and Rank Deficient Matrices

2.1 Rank and Independence

In linear algebra, the *rank* of a matrix is a fundamental concept that reveals important information about the matrix's structure, specifically the degree of linear independence of its rows or columns.
A matrix $A \in \mathbb{R}^{m \times n}$ is said to be *full rank* if either:

- All columns of A are linearly independent, or
- All rows of A are linearly independent.

This means that:

$$\text{If } \operatorname{rank}(A) = \min(m, n),$$

then A is full rank.
For example:

- If A has more rows than columns (i.e., $m > n$), full rank implies that all columns are independent, and $\operatorname{rank}(A) = n$.
- If A has more columns than rows (i.e., $m < n$), full rank implies that all rows are independent, and $\operatorname{rank}(A) = m$.

A matrix is said to be *rank deficient* when it does not have full rank, i.e., when both the rows and columns are not linearly independent. For a matrix $A \in \mathbb{R}^{m \times n}$, the matrix is rank deficient if:

$$\operatorname{rank}(A) < \min(m, n)$$

Rank and Invertibility

The concept of matrix rank is crucial for understanding the invertibility of a matrix:

- A square matrix $A \in \mathbb{R}^{n \times n}$ is *invertible* if and only if it is full rank, i.e., $\operatorname{rank}(A) = n$. This implies that all rows and all columns are linearly independent.

For non-square matrices, left and right inverses depend on whether the rows or columns are independent:

- If all *columns* of A are independent, the matrix A is *full column rank* and has a *left inverse*.
- If all *rows* of A are independent, the matrix A is *full row rank* and has a *right inverse*.

Rank-Deficiency in Square Matrices and Zero Eigenvalues

In square matrices $A \in \mathbb{R}^{n \times n}$, the rank also affects the eigenvalues of the matrix. Specifically:

- A square matrix that is *rank deficient* will have one or more *zero eigenvalues*. This is because a rank-deficient matrix has fewer than n linearly independent rows and columns, meaning there are fewer than n non-zero pivot values in its row-echelon form, leading to zero eigenvalues.

If a square matrix is full rank (i.e., $\text{rank}(A) = n$), it has no zero eigenvalues, and thus it is invertible. The rank deficiency directly determines the number of zero eigenvalues:

$$\text{Number of zero eigenvalues} = n - \text{rank}(A)$$

For instance, if a matrix $A \in \mathbb{R}^{4 \times 4}$ has a rank of 3, then it has exactly one zero eigenvalue, since $4-3=1$.

Inverses of a Matrix

In this section, we discuss the conditions and formulas for the left and right inverses of a matrix A.

Conditions for Inverses

1. **Left Inverse**: A matrix A of size $m \times n$ has a left inverse A_{left}^{-1} if:
   - $m \geq n$ (more rows than columns).
   - The columns of A are linearly independent, which implies $\text{rank}(A) = n$.
2. **Right Inverse**: A matrix A of size $m \times n$ has a right inverse A_{right}^{-1} if:
   - $m \leq n$ (more columns than rows).
   - The rows of A are linearly independent, which implies $\text{rank}(A) = m$.

Formulas for Inverses

1. **Left Inverse**: If A has a left inverse A_{left}^{-1}, it satisfies:

$$A_{\text{left}}^{-1} A = I_n$$

The left inverse can be computed as:

$$A_{\text{left}}^{-1} = (A^T A)^{-1} A^T \quad \text{(if columns of } A \text{ are independent)}$$

This is valid when A is $m \times n$ with $m \geq n$.
2. **Right Inverse**: If A has a right inverse A_{right}^{-1}, it satisfies:

$$A A_{\text{right}}^{-1} = I_m$$

The right inverse can be computed as:

$$A_{\text{right}}^{-1} = A^T (A A^T)^{-1} \quad \text{(if rows of } A \text{ are independent)}$$

This is valid when A is $m \times n$ with $m \leq n$.

2.1.1 Summary

- **Left Inverse** exists if $\text{rank}(A) = n$ (columns are independent):

$$A_{\text{left}}^{-1} = (A^T A)^{-1} A^T$$

- **Right Inverse** exists if $\text{rank}(A) = m$ (rows are independent):

$$A_{\text{right}}^{-1} = A^T (A A^T)^{-1}$$

2.2 Assignment

1. Create a 5×4 random integer matrix A with rank 2 with single matlab command. (Also verify).

SOLUTION

A matrix with 2 rows and 4 columns created from random integer will be a matrix of rank 2. Matlab code for this task is given below.

```
A = randi(10, 5, 2) * randi(10, 2, 4);

% Verify the rank of the matrix
rank_A = rank(A);
disp('Matrix A:');
```

Output of the code is shown below.

```
Matrix A:
    25    74    61    61
    24    75    48    48
    18    60    26    26
    10    32    18    18
    17    52    37    37
Rank of Matrix A:
     2
```

2. Create a 4×5 random integer matrix A with rank 4 with single matlab command . (Also verify).

SOLUTION

Matlab code for this task is given below.

```
A = randi(10, 4, 4) * randi(10, 4, 5);

% Verify the rank of the matrix
rank_A = rank(A);
disp('Matrix A:');
```

Output of the code is shown below.

```
Matrix A:
   198   155   154   251   146
   103   153   118   184   126
   183   195   149   259   185
   199   231   191   303   201
Rank of Matrix A:
     4
```

3. if all columns are independent, number of rows in the matrix must be at least equal to number of columns. Prove this result computationally.

SOLUTION

Let us create an arbitrary 4×4 matrix A with 4 rows in such a way that each row is created from random integers. This confirm that all the rows are independent. Then we find the rank. Since the rows are independent, rank will be 4. But the matrix A is arbitrary, so this result can be generalized. Matlab code for this task is given below.

```
nRows = 4;
A = [];
while true
    newRow = randi(2, 1, 4);
    A = [A; newRow];
    if rank(A) == nRows
        break;
    end
end
disp('Matrix A:');
```

Output of the code is shown below.

```
Matrix A:
     1     2     1     2
     1     2     2     1
     2     1     1     1
     1     1     1     1
Rank of Matrix A:
     4
```

4. Using `matlab` create five 5×5 matrix with rank deficiency 2. Then find number of zero-eigen values. Check whether there are at least 2 zero eigenvalues.

SOLUTION

`Matlab` code for this task is given below.

```
num_matrices = 5
matrices = cell(num_matrices, 1);
zero_eigenvalues_count = zeros(num_matrices, 1);
for i = 1:num_matrices
    A = randi(10, 5, 3);
    B = A * A';
    eigenvalues = eig(B);
    zero_eigenvalues_count(i) = sum(abs(eigenvalues) < 1e-10);
    matrices{i} = B;
end
for i = 1:num_matrices
    fprintf('Matrix %d:\n', i);
    disp(matrices{i});
    fprintf('Number of zero eigenvalues: %d\n\n',
       zero_eigenvalues_count(i));
end
all_at_least_two_zeros = all(zero_eigenvalues_count >= 2);
if all_at_least_two_zeros
    disp('All matrices have at least 2 zero eigenvalues.');
else
    disp('Not all matrices have at least 2 zero eigenvalues.');
end
```

Output of the code is shown below.

```
Matrix 1:
    78    82   101    56    41
    82   107   118    55    43
   101   118   166    78    81
    56    55    78    43    37
    41    43    81    37    49
Number of zero eigenvalues: 2
Matrix 2:
    73    50    92    68   115
    50    54    49    33    92
    92    49   129    99   144
    68    33    99    77   108
   115    92   144   108   213
Number of zero eigenvalues: 2
Matrix 3:
    66    67    30    45    29
    67   217   142   144   133
    30   142   120   120   110
    45   144   120   125   110
    29   133   110   110   101
Number of zero eigenvalues: 2
Matrix 4:
   155    99   113    68   118
    99    81    63    75    90
   113    63   107    32   106
    68    75    32    86    76
   118    90   106    76   140
Number of zero eigenvalues: 2
Matrix 5:
    77    92   129    44    48
    92   129   158    44    58
   129   158   217    72    80
    44    44    72    29    27
    48    58    80    27    33
Number of zero eigenvalues: 2

All matrices have at least 2 zero eigenvalues.
```

In all the five cases, number of zero eigen values are 2.

5. Manually create a 5×5 square matrix for which number of zero-eigen values is more than the number of rank-deficiency.

SOLUTION

For this task choose an upper triangular matrix with four zero rows. So number of zero eigen values is 4. But rank is 2. Hence rank deficiency is $5 - 2 = 3$.

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

6. For the matrix, $\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$ find right inverse. Verify the result.

SOLUTION

Given the matrix

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

Now,

$$A^T = \begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

$$AA^T = \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix} = \begin{bmatrix}14 & 32\\ 32 & 77\end{bmatrix}$$

The inverse is:

$$(AA^T)^{-1} = \frac{1}{54}\begin{bmatrix}77 & -32\\ -32 & 14\end{bmatrix}$$

Compute the Right Inverse:

$$A^{-1}_{\text{right}} = A^T(AA^T)^{-1} = \begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}\begin{bmatrix}\frac{77}{54} & -\frac{32}{54}\\ -\frac{32}{54} & \frac{14}{54}\end{bmatrix} = \begin{bmatrix}-\frac{17}{18} & \frac{4}{9}\\ -\frac{1}{9} & \frac{1}{9}\\ \frac{13}{18} & -\frac{2}{9}\end{bmatrix}$$

`Matlab` code for this task is given below.

```
A = [1 2 3; 4 5 6];
format rational;
B = A' * (A * A')^(-1);
identity_matrix = A * B;
disp('Matrix A:');
disp(A);
disp('Right Inverse B:');
disp(B);
disp('Verification (A * B):');
disp(identity_matrix);
```

Output of the code is:

```
Matrix A:
       1             2             3
       4             5             6
Right Inverse B:
     -17/18          4/9
      -1/9           1/9
      13/18         -2/9
Verification (A * B):
       1             *
       *             1
```

Takeaway

- A matrix is *full rank* if all rows or all columns are linearly independent, with $\text{rank}(A) = \min(m, n)$.
- A matrix is *rank deficient* if both rows and columns are linearly dependent, with $\text{rank}(A) < \min(m, n)$.
- For square matrices, rank deficiency implies the presence of zero eigenvalues, and the number of zero eigenvalues equals $n - \text{rank}(A)$.
- Full rank matrices are invertible, and their inverse properties depend on whether the matrix is full row rank or full column rank.

This understanding of rank, linear independence, and eigenvalues provides essential insight into the behavior of matrices in various applications, including solving systems of linear equations and analyzing matrix transformations.

RESULTS

1. Concept of rank deficiency is revisited.
2. Various matrix inverses are discussed.

3 Assignment 23
Newton Fractals

3.1 Concept

Newton Fractals are visual representations derived from Newton's method for finding roots of complex functions. Named after Sir Isaac Newton, this iterative method is used to approximate the roots of real-valued functions. When applied to complex functions, the iterations can produce stunning and intricate patterns, revealing the underlying structure of the function's roots.
In essence, a Newton Fractal is generated by iterating the Newton method for a complex function and coloring the points in the complex plane based on the root they converge to and the speed of convergence.

3.2 The Beauty of Newton Fractals

1. **Visual Complexity:** Newton Fractals exhibit stunning visual complexity. Each fractal can look drastically different depending on the function and the initial conditions chosen. The interplay of colors and shapes forms intricate designs that are both mesmerizing and mathematically rich.

2. **Mathematical Insight:** They provide insights into the behavior of complex functions. The patterns reveal information about the stability of roots, as well as the regions of attraction for different roots.

3. **Aesthetic Appeal:** Artists and mathematicians alike are drawn to the beauty of these fractals, often using them in visual art and educational contexts to illustrate the harmony between mathematics and aesthetics.

3.3 Computational Steps for Creating Newton Fractals Using MATLAB

Here are the key steps to create a Newton Fractal in MATLAB:

1. **Define the Complex Function:** Choose a complex function for which you want to find the roots. For example, $f(z) = z^3 - 1$.

2. **Define the Newton Iteration Function:**

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

   where $f'(z)$ is the derivative of $f(z)$.

3. **Set Up the Grid in the Complex Plane:** Create a grid of complex numbers over a defined range.

4. **Initialize the Iteration:** For each point in the grid, initialize the complex number z to that point and apply the Newton iteration.

5. **Determine Convergence:** Track the number of iterations until convergence to a root or until a maximum number of iterations is reached.
6. **Color the Points:** Assign colors based on the root each point converges to and the speed of convergence.
7. **Visualize the Result:** Use a suitable plotting function to visualize the resulting fractal.

3.4 MATLAB Code Example

Here's a simple MATLAB code to generate a Newton Fractal for the function $f(z) = z^3 - 1$:

```
% Define the function and its derivative
f = @(z) z.^3 - 1;                % The function
df = @(z) 3*z.^2;                 % Derivative of the function

% Set up the grid of complex numbers
x = linspace(-2, 2, 1000);        % Real part
y = linspace(-2, 2, 1000);        % Imaginary part
[X, Y] = meshgrid(x, y);          % Create a grid
Z = X + 1i * Y;                   % Complex grid

% Initialize iteration parameters
max_iter = 50;                    % Maximum number of iterations
colors = zeros(size(Z));          % Array for storing colors

% Perform Newton's method
for k = 1:max_iter
    Z_old = Z;  % Store the previous value
    Z = Z - f(Z)./df(Z);  % Update the values using Newton's method

    % Determine convergence (roots of z^3 - 1 are 1, -0.5 + 0.866i,
        -0.5 - 0.866i)
    converged = abs(Z - 1) < 0.01;  % Root at 1
    colors(converged) = 1;  % Color for root 1
    converged = abs(Z + 0.5 + 0.866i) < 0.01;  % Root at -0.5 +
       0.866i
    colors(converged) = 2;  % Color for root -0.5 + 0.866i
    converged = abs(Z + 0.5 - 0.866i) < 0.01;  % Root at -0.5 -
       0.866i
    colors(converged) = 3;  % Color for root -0.5 - 0.866i
end

% Plot the result
figure;
imagesc(x, y, colors);
colormap(hsv(3));  % Use a colormap for three roots
axis xy;
title('Newton Fractal for $f(z) = z^3 - 1$','Interpreter','latex');
xlabel('Re(z)');
ylabel('Im(z)');
colorbar;
```

Output of this code is shown in Figure 3.1.

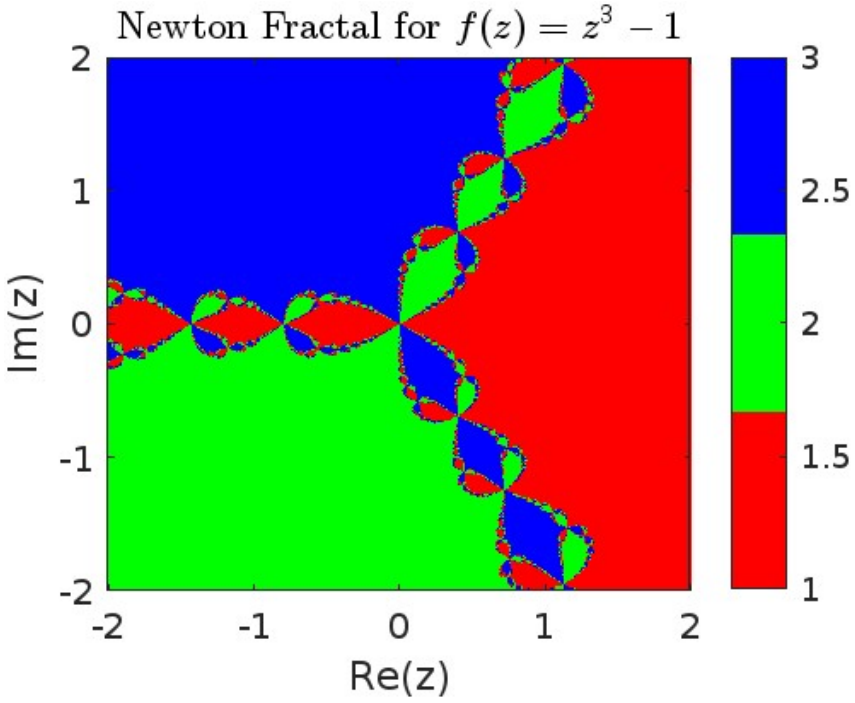


Figure 3.1: Newton fractal for the function $f(z) = z^3 - 1$.

3.5 Applications of Newton Fractals

Newton Fractals have a variety of applications across different fields. Here are some key areas where they are particularly useful:

- **Visualization of Complex Functions:** Newton Fractals provide a visual representation of the behavior of complex functions, making it easier for students to understand concepts like roots, convergence, and stability.
- **Teaching Tool:** They can be used as teaching aids in mathematics courses, particularly in calculus and complex analysis, to illustrate the concepts of iterative methods and their geometric interpretations.
- **Fractal Generation:** In computer graphics, Newton Fractals can be generated to create visually striking images and animations. They contribute to the field of generative art and are used in artistic applications.
- **Procedural Generation:** Fractals can be used in the procedural generation of textures, landscapes, and other visual elements in video games and simulations.
- **Root-Finding Algorithms:** Newton's method itself is widely used in numerical analysis for solving equations, and visualizing its behavior through fractals can help researchers understand the efficiency and convergence properties of the method.
- **Stability Analysis:** The patterns generated by Newton Fractals can help analyze the stability of numerical algorithms by visualizing the convergence regions for different initial conditions.
- **Bifurcation Analysis:** Newton Fractals can be used to study bifurcations in dynamical systems, providing insights into how systems change behavior with varying parameters.
- **Complex Dynamics:** They are useful in the study of complex dynamics and chaos, helping researchers visualize and analyze the behavior of dynamical systems.

- **Signal Processing:** Concepts from fractal geometry can be applied in signal processing, especially in the analysis of non-linear signals and systems.
- **Material Science:** Fractals can model complex structures in materials, aiding in the understanding of physical properties and behaviors.
- **Digital Art:** Artists leverage the beauty of Newton Fractals in digital art, creating unique pieces that are mathematically inspired.
- **Exhibitions:** Fractals are often featured in mathematical art exhibitions, where they attract attention for their intricate and colorful patterns.

3.5.1 More examples with different approach

Another way of creating fractals from $f(z) = z^4 - 1$ is shown below.

```
function newton_fractal
    NITER = 50;
    threshold = .001;
    [xs, ys] = meshgrid(linspace(-1, 1, 300), linspace(-1, 1, 300))
       ;
    solutions = xs(:) + 1i * ys(:); % Use '1i' for imaginary unit
    select = 1:numel(xs);
    niters = NITER * ones(numel(xs), 1);

    for iteration = 1:NITER
        oldi = solutions(select);
        solutions(select) = oldi - f(oldi) ./ fprime(oldi);

        differ = (oldi - solutions(select));
        converged = abs(differ) < threshold;
        problematic = isnan(differ);
        niters(select(converged)) = iteration;
        niters(select(problematic)) = NITER + 1;
        select(converged | problematic) = [];
    end

    niters = reshape(niters, size(xs));
    solutions = reshape(solutions, size(xs));
    imagesc(niters);
    colormap(jet);
    colorbar;
    xlabel('Real Part');
    ylabel('Imaginary Part');
    title('Newton Fractals: Root Finding for $z^4 - 1$', '
       Interpreter', 'latex');
end
function res = f(x)
    res = (x.^4) - 1;
end
function res = fprime(x)
    res = 4*x.^3;
end
```

Output of the code is shown in Figure 3.2.

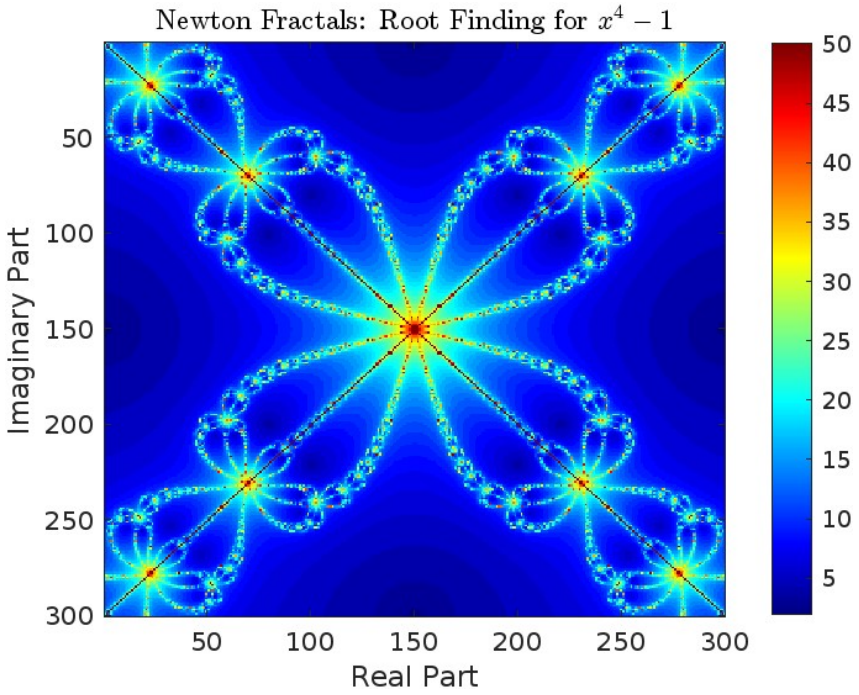


Figure 3.2: Newton Fractal from $f(z) = z^4 - 1$.

3.6 Newton Fractal: Detailed Explanation along with Code

Matlab code given below creates a fractal from a more complicated complex function, $f(z) = z^3 - 1 + \frac{0.1}{(z-0,5)^2}$ with a singular point at $z = 0.5$.

```
function newton_fractal
    NITER = 40;
    threshold = 0.001;
    [xs, ys] = meshgrid(linspace(-1.5, 1.5, 600), linspace(-1.5,
       1.5, 600));
    solutions = xs(:) + 1i * ys(:);
    select = 1:numel(xs);
    niters = NITER * ones(numel(xs), 1);

    for iteration = 1:NITER
        oldi = solutions(select);

        solutions(select) = oldi - f(oldi) ./ fprime(oldi);

        differ = (oldi - solutions(select));
        converged = abs(differ) < threshold;
        problematic = isnan(differ);

        niters(select(converged)) = iteration;
        niters(select(problematic)) = NITER + 1;
        select(converged | problematic) = [];
    end

    niters = reshape(niters, size(xs));
    solutions = reshape(solutions, size(xs));
```

```
25
26      imagesc(niters);
27      colormap(jet);
28      colorbar;
29      xlabel('Real Part');
30      ylabel('Imaginary Part');
31      title('Newton Fractals: Root Finding for $f(x) = x^3 - 1 + \
            frac{0.1}{(x - 0.5)^2}$', 'Interpreter', 'latex');
32      axis equal; % Equal scaling for x and y axes
33  end
34
35  function res = f(x)
36      % Define the complex function
37      res = (x.^3) - 1 + (0.1 ./ (x - 0.5).^2);
38  end
39
40  function res = fprime(x)
41      res = 3 * x.^2 - (0.2 ./ (x - 0.5).^3);
42  end
```

Output of the code is shown in Figure 3.3.

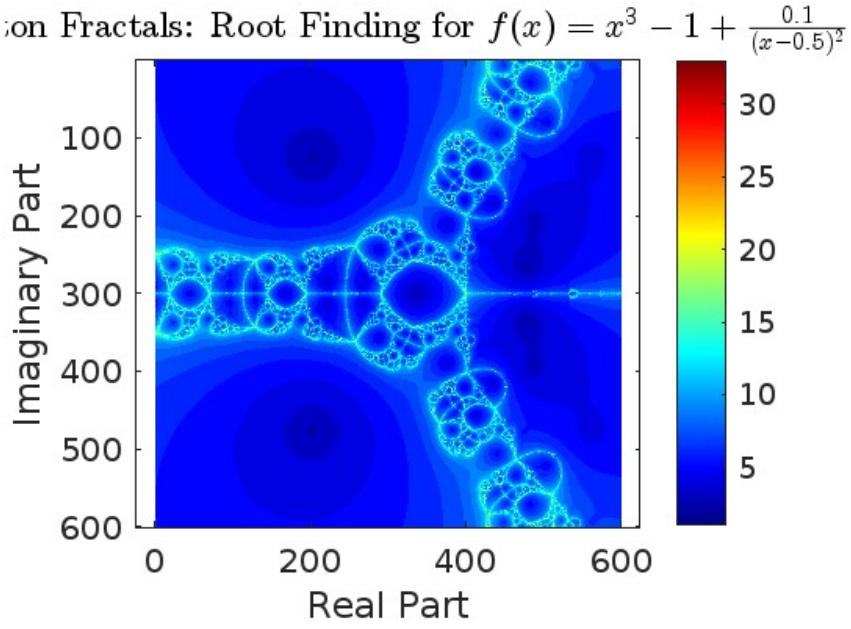


Figure 3.3: Newton Fractal for $f(z) = z^3 - 1 + \dfrac{0.1}{(z-0,5)^2}$.

Following paragraph provides a detailed explanation of the MATLAB code used to generate a Newton fractal based on the function $f(x) = x^3 - 1 + \frac{0.1}{(x-0.5)^2}$.

Function Declaration

```
function newton_fractal
```

This line defines a MATLAB function named `newton_fractal`. When executed, it will run all the code within this function.

Initialization of Parameters

```
NITER = 40; % Maximum number of iterations
threshold = 0.001; % Convergence threshold
```

- `NITER`: Sets the maximum number of iterations for Newton's method. A higher number allows for more refinement in the fractal's detail.
- `threshold`: Specifies the convergence criteria. If the difference between the old and new estimates of a solution is less than this value, we consider the iteration to have converged.

Grid Creation

```
[xs, ys] = meshgrid(linspace(-1.5, 1.5, 600), linspace(-1.5, 1.5, 600));
solutions = xs(:) + 1i * ys(:); % Combine into complex numbers
```

- `meshgrid`: Creates two matrices, `xs` and `ys`, that contain a grid of complex numbers over the specified range $[-1.5, 1.5]$ for both the real and imaginary parts.
- `solutions`: Combines the two matrices into a single vector of complex numbers by converting `xs` and `ys` into column vectors using the `(:)` operator. The imaginary unit `1i` is used to create complex numbers.

Iteration Preparation

```
select = 1:numel(xs);
niters = NITER * ones(numel(xs), 1); % Initialize iteration counts
```

- `select`: An index array that initially selects all the grid points for iteration.
- `niters`: Initializes the count of iterations for each point in the grid to `NITER`, indicating that no points have converged yet.

Newton's Method Iteration

```
for iteration = 1:NITER
    oldi = solutions(select);

    % Newton's method iteration
    solutions(select) = oldi - f(oldi) ./ fprime(oldi);

    % Check for convergence or NaN (division by zero)
    differ = (oldi - solutions(select));
    converged = abs(differ) < threshold;
    problematic = isnan(differ);

    niters(select(converged)) = iteration;
    niters(select(problematic)) = NITER + 1;
    select(converged | problematic) = [];
end
```

- **For Loop**: Runs for up to `NITER` times, applying Newton's method to each point in the `solutions` vector.
- **Newton's Method Calculation**:
  - `oldi`: Stores the previous estimates of the roots for the currently selected points.

- `solutions(select)`: The formula applies Newton's method:

$$z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)}$$

The new estimates are calculated by subtracting the value of the function divided by its derivative from the old estimates.

- **Convergence Check**:
  - `differ`: Computes the difference between the old and new estimates.
  - `converged`: A logical array checks which points have converged based on the `threshold`.
  - `problematic`: Checks for any `NaN` values, which can occur if the derivative is zero (division by zero).
- **Update Iteration Counts**:
  - For those points that have converged, the iteration count is updated to reflect how many iterations it took for convergence.
  - Points that either converged or encountered problems are removed from `select`, reducing the number of points for subsequent iterations.

Reshaping Results

```
niters = reshape(niters, size(xs));
solutions = reshape(solutions, size(xs));
```

After the iterations are complete, the `niters` and `solutions` arrays are reshaped back into the original grid format for visualization. This allows us to plot the number of iterations taken for each point in the complex plane.

Visualization

```
1 imagesc(niters);
2 colormap(jet);
3 colorbar;
4 xlabel('Real Part');
5 ylabel('Imaginary Part');
6 title('Newton Fractals: Root Finding for $f(x) = x^3 - 1 + \frac
     {0.1}{(x - 0.5)^2}$', 'Interpreter', 'latex');
7 axis equal;
```

- `imagesc(niters)`: Displays the `niters` array as an image, where each pixel color corresponds to the number of iterations taken for convergence at that point in the complex plane.
- `colormap(jet)`: Sets the color map to "jet," which uses a gradient of colors.
- `colorbar`: Adds a color bar on the side of the image to indicate how the colors correspond to iteration counts.
- `xlabel` and `ylabel`: Label the axes with "Real Part" and "Imaginary Part."
- `title`: Sets the title of the plot, using LaTeX formatting for a mathematical expression.
- `axis equal`: Ensures that the aspect ratio of the x and y axes is equal, preserving the shape of the fractal.

Function Definitions

```
function res = f(x)
    res = (x.^3) - 1 + (0.1 ./ (x - 0.5).^2);
end

function res = fprime(x)
    res = 3 * x.^2 - (0.2 ./ (x - 0.5).^3);
end
```

These functions define the complex polynomial $f(x)$ and its derivative $f'(x)$. Changing this function based on $f(x)$, we will get different fractals.

RESULTS

Newton fractals and its applications are discussed.

4 Assignment 24
More on Left, Right and Pseudo Inverses

Key Concepts

Let A be an $m \times n$ matrix. Since A is not necessarily a square matrix, it may not have a usual inverse. However, it can have left or right inverses under certain conditions based on its rank:

- **Left Inverse**: If A has full column rank (rank n), then a left inverse A_{left}^{-1} exists such that:

$$A_{\text{left}}^{-1} A = I_n$$

  where I_n is the identity matrix of size $n \times n$.

- **Right Inverse**: If A has full row rank (rank m), then a right inverse A_{right}^{-1} exists such that:

$$A A_{\text{right}}^{-1} = I_m$$

  where I_m is the identity matrix of size $m \times m$.

- **Pseudo-Inverse**: If A is rank deficient, it will have a pseudo-inverse A^+. Every matrix has a pseudo-inverse, which can be computed using Singular Value Decomposition (SVD). The pseudo-inverse can be used to find least-squares solutions for systems of equations.

For the equation $Ax = b$ where b lies in the column space of A, the solution can be obtained using the following conditions:

Independent Columns: If the columns of A are independent, the left inverse A_{left}^{-1} gives a unique solution.

Independent Rows: If the rows of A are independent, the right inverse A_{right}^{-1} provides a solution. Here, the system of equations may have infinite solutions, and the solution obtained is the least-norm solution. Other solutions can be derived by adding a right null space vector to the solution vector obtained using the right inverse.

Pseudo-Inverse: The pseudo-inverse will provide a solution in all cases, corresponding to a least-norm solution from the row space.

In scenarios where b does not lie in the column space, the pseudo-inverse A^+ will provide a solution that corresponds to the least-norm solution, where b is projected onto the column space.

4.1 Left Inverse

If the columns of A are independent, there exists a matrix A_{left}^{-1} such that:

$$A_{\text{left}}^{-1} A = I_n$$

Example

Let us create a 3×2 matrix with 2 independent columns:

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

To find the left inverse:

$$A_{\text{left}}^{-1} = A'(AA')^{-1}$$

Matlab code for this task is given below.

```
A = [1 2; 2 3; 3 4];
A_left_inv = A' * inv(A * A');
disp("Left Inverse of A is:");
disp(A_left_inv);
```

Pseudo-inverse

The pseudo-inverse A^+ can be computed as:

$$A^+ = A'(AA')^{-1}$$

Matlab code for this task is given below.

```
disp(pinv(A));
```

Both the left inverse and the pseudo-inverse yield the same result for matrices with independent columns.

4.2 Right Inverse

If the rows of A are independent, we can find a right inverse A_{right}^{-1} such that:

$$AA_{\text{right}}^{-1} = I_m$$

Example

Let:

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

Matlab code for this task is given below.

```
A = [1 2; 3 4];
A_right_inv = inv(A * A') * A';
disp("Right Inverse of A is:");
disp(A_right_inv);
```

Pseudo-inverse

To compute the pseudo-inverse:

```
disp("Pseudo Inverse is:");
disp(pinv(A));
```

Both the right inverse and the pseudo-inverse yield the same result for matrices with independent rows.

4.3 Pseudo-Inverse

What is the Pseudo-Inverse Doing?

The pseudo-inverse A^+ is utilized in scenarios where the matrix is not invertible or where we seek the least-squares solution. The key characteristics of the pseudo-inverse include:

- **Least-Norm Solution**: It finds the solution x that minimizes the norm of the residual $||Ax - b||$.
- **Use in SVD**: The pseudo-inverse is derived from the singular value decomposition (SVD) of the matrix A. If:

  $$A = U\Sigma V'$$

  Then, the pseudo-inverse is given by:

  $$A^+ = V\Sigma^+ U'$$

  where Σ^+ is formed by taking the reciprocal of the non-zero elements of Σ and transposing the matrix.

Example Case 1: columns independent

Assume a case where the columns of A are independent and b is in the column space. The solution x obtained using the pseudo-inverse is the same as that obtained through Gaussian elimination.

Example

Let:

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}$$

`Matlab` code for this task is given below.

```
A = [1 2; 2 3; 3 4];
b = [5; 6; 7];
x_pseudo = pinv(A) * b;
disp("Solution using Pseudo-Inverse:");
disp(x_pseudo);
```

Example Case 2: columns dependent

Assume a case where the columns of A are dependent and b is not in the column space. In this case, while there may not be a solution for $Ax = b$, the pseudo-inverse can still provide a solution corresponding to the least norm.

4.4 Assignment

1. Demonstrate through `matlab` code that `pinv(A)` and $A^+_{left} = \left(A^T A\right)^{-1} A^T$ produce same matrix for any matrix A whose columns are independent.

   SOLUTION

   `Matlab` code for this task is given below.

```
format RAT
A=[1 2 3; 5 7 8; 11 13 17];
disp("Left Inverse of A is:")
AinvR=inv(A'*A)*A';
disp(AinvR)
disp("Pseudo Inverse is:")
disp(pinv(A))
```

Output of the code is shown below.

```
Left Inverse of A is:
      -1            -1/3           1/3
      -1/5          16/15         -7/15
       4/5          -3/5           1/5
Pseudo Inverse is:

      -1            -1/3           1/3
      -1/5          16/15         -7/15
       4/5          -3/5           1/5
```

2. Give 2 more examples with different matrix size to generalize this concept.

SOLUTION

Matlab code for this task is given below.

```
A1=[2 3 11; 0 4 9; 1 7 9]
disp("Left Inverse of A is:")
A1invl=inv(A1'*A1)*A1';
disp(A1invl)
disp("Pseudo Inverse is:")
disp(pinv(A1))

A2=[1 3; 1 5]
disp("Left Inverse of A is:")
A1invl=A2'*inv(A2*A2');
disp(A1invl)
disp("Pseudo Inverse is:")
disp(pinv(A2))
```

Output of the code is shown below.

```
A1 =
       2             3            11
       0             4             9
       1             7             9
Left Inverse of A is:

      27/71        -50/71         17/71
      -9/71         -7/71         18/71
       4/71         11/71         -8/71
Pseudo Inverse is:

      27/71        -50/71         17/71
```

```
      -9/71          -7/71          18/71
       4/71          11/71          -8/71
A2 =
       1              3
       1              5
Left Inverse of A is:
       5/2           -3/2
      -1/2            1/2
Pseudo Inverse is:
       5/2           -3/2
      -1/2            1/2
```

3. Demonstrate through `matlab` code that `pinv(A)` and $A^{+}_{right}A^{T}\left(AA^{T}\right)^{-1}$ produce the same matrix for any matrix A whose rows are independent.

SOLUTION

`Matlab` code for this task is given below.

```
A5=[1 0 1; 0 1 1];
disp("Right Inverse of A is:")
A1invl=A2'*inv(A2*A2');
disp(A1invl)
disp("Pseudo Inverse is:")
disp(pinv(A2))
```

Output of the code is shown below.

```
Right Inverse of A is:
       5/2           -3/2
      -1/2            1/2
Pseudo Inverse is:
       5/2           -3/2
      -1/2            1/2
```

4. Give 2 examples with different matrix size.

SOLUTION

`Matlab` code for this task is given below.

```
A5=randi([0,10],4,5);
disp("Right Inverse of A is:")
A5invl=A5'*inv(A5*A5');
disp(A5invl)
disp("Pseudo Inverse is:")
disp(pinv(A5))
```

Output of the code is shown below.

```
Right Inverse of A is:
     -56/1969      -172/1735       161/979       -397/6136
    -155/1627       309/2915      -613/77919      529/9086
      16/493         21/743        347/155110    -187/7814
     503/7882        79/806       -247/3179      -123/2179
```

```
     155/3272       -82/1067        51/12506      287/2941
Pseudo Inverse is:
     -56/1969      -172/1735       161/979       -397/6136
    -155/1627       309/2915      -613/77919      529/9086
      16/493         21/743        347/155110    -187/7814
     503/7882        79/806       -247/3179      -123/2179
     155/3272       -82/1067        51/12506      287/2941
```

The pseudo-inverse serves as a powerful tool to handle underdetermined or overdetermined systems of equations, providing least-norm solutions and projections onto the relevant subspaces. Understanding the pseudo-inverse requires familiarity with SVD, which is crucial for interpreting solutions in the context of linear algebra applications.

RESULTS

Right, left, and pseudo inverses of a matrix are revisited and computationally verified under what condition the pseudo inverse coincides with the left/right inverse of a matrix.

5 Assignment 25
LU Decomposition of Matrices

5.1 Introduction

This lesson is a revision of the *LU* decomposition. *LU* decomposition of a matrix A is given by $A = LU$, where L is a lower triangular matrix with 1s on the principal diagonal and U is an upper triangular matrix. In terms of the elementary matrices, this L matrix capture the elementary transformation on A that make it reduced in to U.

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

5.2 Elementary Matrices

Let us make 0 at location (2,1) of A. This is accomplished by multiplying first row with (-2) and adding to second row. This is mathematically expressed as pre-multiplying A by an elementary matrix E_1. Elementary matrices are unit diagonal matrix with one additional non-zero element.

$$E_1\text{A} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix};$$

Note how it is obtained:

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

Interpret this as 1 x (first row of A) + 0 x (second row of A) + 0 x (third row of A) = [2 1 1] First and Second row in EA_1 is obtained as

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & 2 \end{bmatrix}$$

In the above equation, second row in RHS is obtained as -2 x (first row of A) + 1 x (second row of A) + 0 x (fthird row of A)= [0 -8 2]. Finally , all the three rows in EA_1 is obtained as:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix}$$

The elementary matrix E_1 has a speciality.

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

Just change the sign of the non-zero off-diagonal element. It does an unwinding (reverse operation) of E_1. It is true for all the elementary matrices that we are about to introduce.

Let us proceed.

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

Let us make 0 at location (3,1) of $E_1 A$. This is obtained by pre-multiplying $E_1\ A$ with E_2 as follows.

$$E_2 E_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Let us now make 0 at location (3,2) of $E_2 E_1 A$.

$$E_3 E_2 E_1 \text{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix} = \text{U}$$

That is $E_3 E_2 E_1$A =U

$A = (\ E_3 E_2 E_1)^{-1}$ U = $(E_1^{-1} E_2^{-1} E_3^{-1})$U

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

Note that in L, the non-zero off-diagonal elements are negative of the multiplying factors used in elementary matrices.

How A=LU is used for solving Ax=b . For solving Ax=b , we solve LUx=b as in following example The x=A\b command in matlab uses LU decomposition for solution.

Ex1. Solve with out multiplying LU.

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix}$$

Solution:

Assume

$$\begin{bmatrix} 2 & 4 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \text{;so}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix}$$

We solve this $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix}$ matrix equation using a process called 'forward substitution'.

First equation gives $c_1 = 2$

Second equation with $c_1 = 2$ gives $c_2 = -2$

Third equation with $c_1 = 2$ gives $c_3 = 0$.

Now we solve the equation

$$\begin{bmatrix} 2 & 4 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} c_1 = 2 \\ c_2 = -2 \\ c_3 = 0 \end{bmatrix}$$

That is $\begin{bmatrix} 2 & 4 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix}$

We solve this using a process called 'backward substitution'. Last equation give $w = 0$.

Second equation give $v = -2$ with $w = 0$.

Third equation give $u = 10$ with $w = 0; v = -2$.

So the final solution is $\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 10 \\ -2 \\ 0 \end{bmatrix}$

5.3 Tasks

Q1. Express A as LU.

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

```
A=[1 -1 0 0; -1 2 -1 0; 0 -1 2 -1; 0 0 -1 2]
```

```
A = 4x4
     1    -1     0     0
    -1     2    -1     0
     0    -1     2    -1
     0     0    -1     2
```

```
[L,U]=lu(A);
disp("LU decomposition of A is:")
```

```
LU decomposition of A is:
```

```
disp("L matrix:")
```

```
L matrix:
```

```
disp(L);
```

```
     1     0     0     0
    -1     1     0     0
     0    -1     1     0
     0     0    -1     1
```

```
disp("U matrix is:")
```

```
U matrix is:
```

```
disp(U)
```

```
     1    -1     0     0
     0     1    -1     0
     0     0     1    -1
     0     0     0     1
```

Q2. Find E^2, E^8 and E^{-1} of the following elementary matrix without computing.

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

```
E=[1 0; 6 1];
E*E
```

```
ans = 2x2
     1     0
    12     1
```

```
E^8
```

```
ans = 2x2
     1     0
    48     1
```

```
inv(E)
```

```
ans = 2x2
     1     0
    -6     1
```

RESULTS

1. *LU* decomposition is revisited.
2. *LU* decomposition is used to solve system of equations.

Spell checking completedNone selected
Skip to content Using SAINTGITS Mail with screen readers
Conversations
Programme Policies Powered by Google Last account activity: 1 minute ago Details Compose: Draft saved MinimisePop-outClose Recipients Subject

6 | Assignment 27
Madhava Series Expansion for Two Variables

In mathematics, a Madhava series is one of the three Taylor series expansions for the sine, cosine, and arctangent functions discovered in 14th or 15th century in Kerala, India by the mathematician and astronomer Madhava of Sangamagrama (c. 1350 – c. 1425) or his followers in the Kerala school of astronomy and mathematics. Using modern notation, these series are:
Madhava series for $\sin\theta$:

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots$$

Or in summation form:

$$\sin\theta = \sum_{n=0}^{\infty} (-1)^n \frac{\theta^{2n+1}}{(2n+1)!}$$

Madhava series for $\cos\theta$:

$$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \cdots$$

Or in summation form:

$$\cos\theta = \sum_{n=0}^{\infty} (-1)^n \frac{\theta^{2n}}{(2n)!}$$

6.1 Madhava Series Expansion

In recognition of Madhava's priority, in recent literature these series are sometimes called the Madhava–Newton series,[4] Madhava–Gregory series, or Madhava–Leibniz series (among other combinations).
No surviving works of Madhava contain explicit statements regarding the expressions which are now referred to as Madhava series. However, in the writing of later Kerala school mathematicians Nilakantha Somayaji (1444 – 1544) and Jyeshthadeva (c. 1500 – c. 1575) one can find unambiguous attributions of these series to Madhava. These later works also include proofs and commentary which suggest how Madhava may have arrived at the series[1].

6.1.1 First degree polynomial approximation of a bi-variate function

Let $f(x,y)$ be a function of two variables. The first-degree Madhava-Taylor series expansion of $f(x,y)$ around a point (x_0,y_0) is:

$$f(x,y) \approx f(x_0,y_0) + \frac{\partial f}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial f}{\partial y}(x_0,y_0)(y-y_0)$$

This gives the linear approximation of $f(x,y)$ around the point (x_0,y_0).

[1] Wikipidia

1. Determine the linear and quadratic approximation of the function $f(x_1,x_2)=2x_1^2+x_2^2+\dfrac{1}{2x_1^2+x_2^2}$ at $(2,2)$ using Madhava series.

SOLUTION

We have to expand the given bi-variate function using the Madhava-Taylor series expansion. Given that

$$f(x_1,x_2)=2x_1^2+x_2^2+\frac{1}{2x_1^2+x_2^2}$$

and $(a_1,a_2)=(2,2)$

The general form of the Madhava series expansion of a function $f(x_1,x_2)$ around a point (a_1,a_2) is given by:

$$f(x_1,x_2)\approx f(a_1,a_2)+\nabla f(a_1,a_2)\cdot\begin{bmatrix}x_1-a_1\\x_2-a_2\end{bmatrix}+\frac{1}{2}\begin{bmatrix}x_1-a_1 & x_2-a_2\end{bmatrix}\nabla^2 f(a_1,a_2)\begin{bmatrix}x_1-a_1\\x_2-a_2\end{bmatrix}$$

where $\nabla f(a_1,a_2)$ is the gradient of f at (a_1,a_2), and $\nabla^2 f(a_1,a_2)$ is the Hessian matrix of second-order partial derivatives at (a_1,a_2).

Compute $f(2,2)$ Substitute $x_1=2$ and $x_2=2$ into the function $f(x_1,x_2)$:

$$f(2,2)=2(2)^2+(2)^2+\frac{1}{2(2)^2+(2)^2}=2(4)+4+\frac{1}{8+4}=8+4+\frac{1}{12}=12+0.083333=12.083333.$$

Thus,

$$f(2,2)=12.083333.$$

Compute the gradient $\nabla f(x_1,x_2)$

The gradient $\nabla f(x_1,x_2)$ consists of the partial derivatives with respect to x_1 and x_2.

$$\nabla f(x_1,x_2)=\begin{bmatrix}\frac{\partial f}{\partial x_1}\\ \frac{\partial f}{\partial x_2}\end{bmatrix}$$

Compute $\frac{\partial f}{\partial x_1}$:

$$\frac{\partial f}{\partial x_1}=4x_1-\frac{4x_1}{(2x_1^2+x_2^2)^2}$$

At $x_1=2$ and $x_2=2$:

$$\frac{\partial f}{\partial x_1}\Big|_{(2,2)}=4(2)-\frac{4(2)}{(2(2)^2+(2)^2)^2}=8-\frac{8}{(8+4)^2}=8-\frac{8}{144}=8-0.055556=7.944444.$$

$\frac{\partial f}{\partial x_2}$:

$$\frac{\partial f}{\partial x_2}=2x_2-\frac{2x_2}{(2x_1^2+x_2^2)^2}$$

At $x_1=2$ and $x_2=2$:

$$\frac{\partial f}{\partial x_2}\Big|_{(2,2)}=2(2)-\frac{2(2)}{(2(2)^2+(2)^2)^2}=4-\frac{4}{(8+4)^2}=4-\frac{4}{144}=4-0.027778=3.972222.$$

Thus, the gradient, ∇f at $(2,2)$ is:

$$\nabla f(2,2)=\begin{bmatrix}7.944444\\3.972222\end{bmatrix}.$$

Compute the Hessian matrix $H(x_1, x_2)$ The Hessian matrix contains the second-order partial derivatives:

$$\nabla^2(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

$\frac{\partial^2 f}{\partial x_1^2}$**:**

$$\frac{\partial^2 f}{\partial x_1^2} = 4 - \frac{4(6x_1^2 - x_2^2)}{(2x_1^2 + x_2^2)^3}$$

At $x_1 = 2$ and $x_2 = 2$:

$$\frac{\partial^2 f}{\partial x_1^2}\Big|_{(2,2)} = 4 - \frac{4(6(2)^2 - (2)^2)}{(8+4)^3} = 4 - \frac{4(24-4)}{1728} = 4 - \frac{80}{1728} = 4 - 0.046296 = 3.953704.$$

$\frac{\partial^2 f}{\partial x_2^2}$:

$$\frac{\partial^2 f}{\partial x_2^2} = 2 - \frac{2(6x_2^2 - x_1^2)}{(2x_1^2 + x_2^2)^3}$$

At $x_1 = 2$ and $x_2 = 2$:

$$\frac{\partial^2 f}{\partial x_2^2}\Big|_{(2,2)} = 2 - \frac{2(6(2)^2 - (2)^2)}{(8+4)^3} = 2 - \frac{2(24-4)}{1728} = 2 - \frac{40}{1728} = 2 - 0.023148 = 1.976852.$$

Compute $\frac{\partial^2 f}{\partial x_1 \partial x_2}$ **and** $\frac{\partial^2 f}{\partial x_2 \partial x_1}$:

$$\frac{\partial^2 f}{\partial x_1 \partial x_2} = \frac{\partial^2 f}{\partial x_2 \partial x_1} = -\frac{24 x_1 x_2}{(2x_1^2 + x_2^2)^3}$$

At $x_1 = 2$ and $x_2 = 2$:

$$\frac{\partial^2 f}{\partial x_1 \partial x_2}\Big|_{(2,2)} = -\frac{24(2)(2)}{(8+4)^3} = -\frac{96}{1728} = -0.055556.$$

Thus, the Hessian matrix at $(2,2)$ is:

$$\nabla^2(2,2) = \begin{bmatrix} 3.953704 & -0.055556 \\ -0.055556 & 1.976852 \end{bmatrix}.$$

Madhava Series Expansion

Substituting the computed values into the power series formula, we get the following expansions:

(a) **Linear approximation:**

$$f(x_1, x_2) \approx 12.083333 + \begin{bmatrix} 7.944444 & 3.972222 \end{bmatrix} \begin{bmatrix} x_1 - 2 \\ x_2 - 2 \end{bmatrix}$$

(b) **Quadratic approximation:**

$$f(x_1, x_2) \approx 12.083333 + \begin{bmatrix} 7.944444 & 3.972222 \end{bmatrix} \begin{bmatrix} x_1 - 2 \\ x_2 - 2 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x_1 - 2 & x_2 - 2 \end{bmatrix} \begin{bmatrix} 3.953704 & -0.055556 \\ -0.055556 & 1.976852 \end{bmatrix} \begin{bmatrix} x_1 - 2 \\ x_2 - 2 \end{bmatrix}$$

`Matlab` code for this task is given below.

```
syms x1 x2
f = 2*x1^2 + x2^2 + 1/(2*x1^2 + x2^2);
x0 = [2, 2];
taylor_expansion = taylor(f, [x1, x2], 'ExpansionPoint', x0, '
    Order', 3);
taylor_expansion_simplified = simplify(taylor_expansion);
disp('Taylor series expansion up to second degree:');
disp(taylor_expansion_simplified);
point = [x1, x2];
taylor_approx = subs(taylor_expansion_simplified, {x1, x2},
    point);
disp('Taylor series approximation at a point:');
disp(taylor_approx);
```

Output of this code is shown below.

```
(437*x1^2)/216 + (x1*x2)/27 - (2*x1)/9 + (433*x2^2)/432 - x2/9 + sym(1/2)
```

6.1.2 Second degree polynomial approximation of a multi-variate function

2. Expand $f(x_1,x_2,x_3) = x_1^2 + x_2^2 + x_3^2 + 2x_1x_2 + 6x_2x_3 + 4x_1x_3 + 8x_1 + 4x_2 + 6x_3$ using the Madhava series about $(1,2,3)$.

SOLUTION

Given function is:

$$f(x_1,x_2,x_3) = x_1^2 + x_2^2 + x_3^2 + 2x_1x_2 + 6x_2x_3 + 4x_1x_3 + 8x_1 + 4x_2 + 6x_3$$

We have to generate a second degree polynomial that approximates $f(x_1,x_2,x_3)$ about the point $(1,2,3)$.

The general form of the Madhava series expansion of a function $f(x_1,x_2,x_3)$ around a point (a_1,a_2,a_3) up to second order is given by:

$$f(x_1,x_2,x_3) \approx f(a_1,a_2,a_3) + \nabla f(a_1,a_2,a_3) \cdot \begin{bmatrix} x_1-a_1 \\ x_2-a_2 \\ x_3-a_3 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} x_1-a_1 & x_2-a_2 & x_3-a_3 \end{bmatrix} H(a_1,a_2,a_3) \begin{bmatrix} x_1-a_1 \\ x_2-a_2 \\ x_3-a_3 \end{bmatrix}$$

where $\nabla f(a_1,a_2,a_3)$ is the gradient and $\nabla^2 f(a_1,a_2,a_3)$ is the Hessian matrix of f at (a_1,a_2,a_3).

Compute $f(1,2,3)$

Substitute $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$ into the function, $f(x_1,x_2,x_3)$:

$$f(1,2,3) = (1)^2 + (2)^2 + (3)^2 + 2(1)(2) + 6(2)(3) + 4(1)(3) + 8(1) + 4(2) + 6(3)$$

$$= 1 + 4 + 9 + 4 + 36 + 12 + 8 + 8 + 18 = 100.$$

Compute the Gradient $\nabla f(x_1,x_2,x_3)$

The gradient consists of the partial derivatives of f with respect to x_1, x_2, and x_3:

$$\nabla f(x_1,x_2,x_3) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix}$$

Compute $\frac{\partial f}{\partial x_1}$:

$$\frac{\partial f}{\partial x_1} = 2x_1 + 2x_2 + 4x_3 + 8$$

At $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$:

$$\frac{\partial f}{\partial x_1}\Big|_{(1,2,3)} = 2(1) + 2(2) + 4(3) + 8 = 2 + 4 + 12 + 8 = 26.$$

Compute $\frac{\partial f}{\partial x_2}$:

$$\frac{\partial f}{\partial x_2} = 2x_2 + 2x_1 + 6x_3 + 4$$

At $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$:

$$\frac{\partial f}{\partial x_2}\Big|_{(1,2,3)} = 2(2) + 2(1) + 6(3) + 4 = 4 + 2 + 18 + 4 = 28.$$

Compute $\frac{\partial f}{\partial x_3}$:

$$\frac{\partial f}{\partial x_3} = 2x_3 + 6x_2 + 4x_1 + 6$$

At $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$:

$$\frac{\partial f}{\partial x_3}\Big|_{(1,2,3)} = 2(3) + 6(2) + 4(1) + 6 = 6 + 12 + 4 + 6 = 28.$$

Thus, the gradient at $(1,2,3)$ is:

$$\nabla f(1,2,3) = \begin{bmatrix} 26 \\ 28 \\ 28 \end{bmatrix}.$$

Compute the Hessian Matrix $\nabla^2 f(x_1, x_2, x_3)$

The Hessian matrix contains the second-order partial derivatives:

$$\nabla^2 f(x_1, x_2, x_3) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} \end{bmatrix}$$

Compute $\frac{\partial^2 f}{\partial x_1^2}$:

$$\frac{\partial^2 f}{\partial x_1^2} = 2.$$

Compute $\frac{\partial^2 f}{\partial x_2^2}$:

$$\frac{\partial^2 f}{\partial x_2^2} = 2.$$

Compute $\frac{\partial^2 f}{\partial x_3^2}$:

$$\frac{\partial^2 f}{\partial x_3^2} = 2.$$

compute $\frac{\partial^2 f}{\partial x_1 \partial x_2}$ and $\frac{\partial^2 f}{\partial x_2 \partial x_1}$: (Using continuity of f both these values are same)

$$\frac{\partial^2 f}{\partial x_1 \partial x_2} = 2, \quad \frac{\partial^2 f}{\partial x_2 \partial x_1} = 2.$$

compute $\frac{\partial^2 f}{\partial x_2 \partial x_3}$ and $\frac{\partial^2 f}{\partial x_3 \partial x_2}$:

$$\frac{\partial^2 f}{\partial x_2 \partial x_3} = 6, \quad \frac{\partial^2 f}{\partial x_3 \partial x_2} = 6.$$

compute $\frac{\partial^2 f}{\partial x_1 \partial x_3}$ and $\frac{\partial^2 f}{\partial x_3 \partial x_1}$:

$$\frac{\partial^2 f}{\partial x_1 \partial x_3} = 4, \quad \frac{\partial^2 f}{\partial x_3 \partial x_1} = 4.$$

Thus, the Hessian matrix is:

$$\nabla^2 f(1,2,3) = \begin{bmatrix} 2 & 2 & 4 \\ 2 & 2 & 6 \\ 4 & 6 & 2 \end{bmatrix}.$$

Now, we can combine everything to form the second-order Madhava series expansion around $(1,2,3)$.

The Madhava series expansion is:

$$f(x_1,x_2,x_3) \approx 100 + \begin{bmatrix} x_1-1 & x_2-2 & x_3-3 \end{bmatrix} \begin{bmatrix} 26 \\ 28 \\ 28 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x_1-1 & x_2-2 & x_3-3 \end{bmatrix} \begin{bmatrix} 2 & 2 & 4 \\ 2 & 2 & 6 \\ 4 & 6 & 2 \end{bmatrix} \begin{bmatrix} x_1-1 \\ x_2-2 \\ x_3-3 \end{bmatrix}$$

Matlab code for this task is given below.

```
syms x1 x2 x3
f = x1^2 + x2^2 + x3^2 + 2*x1*x2 + 6*x2*x3 + 4*x1*x3 + 8*x1 +
   4*x2 + 6*x3;
x0 = [1, 2, 3];
taylor_expansion = taylor(f, [x1, x2, x3], 'ExpansionPoint', x0
   , 'Order', 3);
taylor_expansion_simplified = simplify(taylor_expansion);
disp('Taylor series expansion up to second degree:');
disp(taylor_expansion_simplified);
point = [x1, x2, x3];
taylor_approx = subs(taylor_expansion_simplified, {x1, x2, x3},
    point);
disp('Taylor series approximation at a point:');
disp(taylor_approx);
```

Output of the above code is shown below.

```
x1^2 + 2*x1*x2 + 4*x1*x3 + 8*x1 + x2^2 + 6*x2*x3 + 4*x2 + x3^2 + 6*x3
```

RESULTS

1. The famous power series expansion of a multi-variate function $f(X)$ about X_0 is revisited with the Mathava's power series expansion method.
2. Both linear and quadratic approximations of given multi-variate functions are generated and computationally verified in Matlab.

7 | Assignment 28
Orthogonal Matrices and Rotation

7.1 Introduction

Classical Rotation matrices Rotation matrices can be created with specific angle of rotation about co-ordinate axes. Let us start with R^2 . The rotation matrix that rotate any vector by an angle theta in anti-clockwise direction is given by

$$R_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

To check whether it is doing the intended task, let us multiply it with a unit vector in the x-axis.

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$

Let us explore its properties.

7.2 Properties of Orthogonal Matrices

1) It is an orthogonal matrix

Proof:

$$R_\theta^T R_\theta = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$R_\theta R_\theta^T = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2. It preserves the norm of vector on which it operates. It is an obvious property that it should satisfy.

It means if $y = R_\theta x$;then $||y|| = ||x||$

Proof:

If $y = R_\theta x$ then

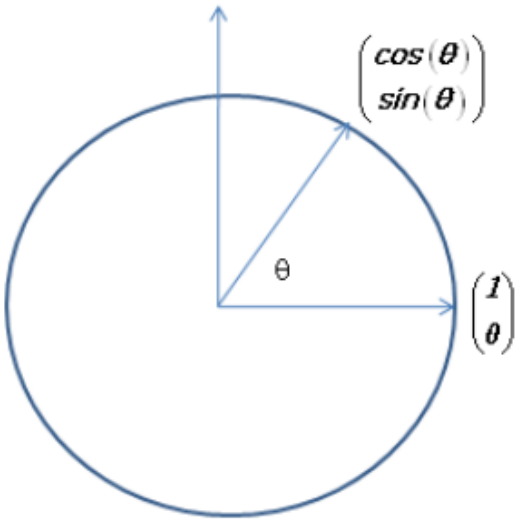


Figure 7.1: Action of classical Rotation matrices on $\hat{i}$ in $\mathbb{R}^2$

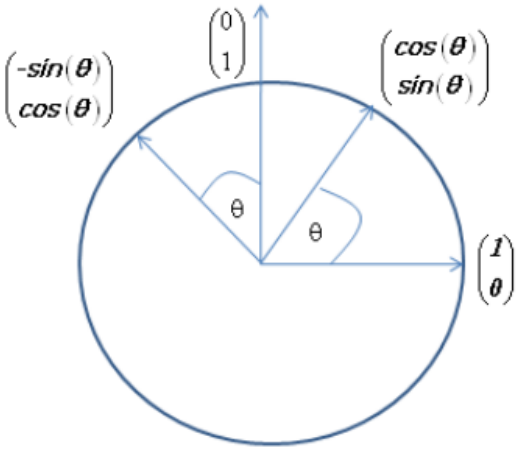


Figure 7.2: Action of classical Rotation matrices on $\hat{i}$ and $\hat{j}$ in $\mathbb{R}^2$

$$y^T y = (R_\theta x)^T (R_\theta x) = x^T R_\theta^T R_\theta x = x^T \left(R_\theta^T R_\theta\right) x = x^T x$$
$$y^T y = x^T x \Rightarrow ||y|| = ||x||$$

3. $R_\theta^T = R_{(-\theta)}$. That is R_θ^T rotate a vector in clockwise direction.

Further $R_\theta R_{(-\theta)} = 1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Proof

$$R_{(-\theta)} = \begin{bmatrix} \cos(-\theta) & -\sin(\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} = R_\theta^T$$

4. Absolute value (or length) of all Eigen values must be 1.

Proof.

It directly follows from the fact that, it is a norm preserving transformation. It cannot change the length of the vector it operates on. **In this particular case, there is no real input vector for which output vector is same. So what is the eigenvalue of this matrix. It can't take any real value.** If it has, It means that there is a vector for which rotation does not happen. That contradicts the idea of rotation matrix.

Let us find eigenvalue of matrix for $\theta = \frac{\pi}{4}$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}.$$

In matlab eig(R) is obtained as

$$\left\{\frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}, \frac{1}{\sqrt{2}} + \frac{-i}{\sqrt{2}}\right\}$$

The eigenvalues are complex numbers but norm of each eigenvalue is 1.

Orthogonal matrices and properties

Note that **pure rotation matrices** and orthogonal matrices share many properties. We start with popular square orthogonal matrices used in signal processing.

1. It preserve the norm of the vector it operates on

That is $y = \mathrm{Ax} \Rightarrow ||y|| = ||x||$

Proof: $y^T y = (\mathrm{Ax})^T \mathrm{Ax} \Rightarrow x^T \left(A^T A\right) x = x^T x \quad \therefore A^T A = 1$

2. Magnitude of eigenvalues are unity.

Do the following experiment in matlab.

>> A=dct(eye(4))

$$A = \begin{bmatrix} 0.5000 & 0.5000 & 0.5000 & 0.5000 \\ 0.6533 & 0.2706 & -0.2706 & -0.6533 \\ 0.5000 & -0.5000 & -0.5000 & 0.5000 \\ 0.2706 & -0.6533 & 0.6533 & -0.2706 \end{bmatrix}$$

>> E= eig(A)

$$E = \begin{bmatrix} 0.9904 + 0.1379i \\ 0.9904 - 0.1379i \\ -0.9904 + 0.1379i \\ -0.9904 - 0.1379i \end{bmatrix}$$

>> magnitude = $E.^*$ conj (E)

$$\text{magnitude} = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$$

For DCT matrices all eigen values are complex numbers

Let us see eigenvalues of DFT matrix

>>format RAT

>>A=fft(eye(4))

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

>> E=eig(A)

$$E = \begin{bmatrix} 2.0+0.0i \\ -2.0+0.0i \\ 2.0-0.0i \\ 0.0-2.0i \end{bmatrix}$$

>> mag=E.*conj(E)

$$E = \begin{bmatrix} 4.0 \\ 4.0 \\ 4.0 \\ 4.0 \end{bmatrix}$$

Note that some eigenvalues are real some are complex. Their magnitude is same but not unity. This is because, in DFT or FFT , the forward direction uses unnormalized bases. If normalized, magnitude of each eigenvaluer will be one.

Thus Rotation and orthogonal (square matrices) share some common properties. Since Orthogonal matrices preserve norm, it can be considered as rotational matrix. So Gilbert Strang interpret U and V matrices in SVD decomposition as rotation matrices and Σ matrix in $A = U\Sigma V^T$ as a scaling matrix.

7.3 Tasks

Q1. In matlab Create 3 × 3 pure rotation matrices by providing required angles for rotation around different axes.

SOLUTION

Matlab code for this task is given below.

```
% Prompt the user for rotation angles in degrees
theta_x_deg = input('Enter the rotation angle around the x-axis (in degrees): ');
theta_y_deg = input('Enter the rotation angle around the y-axis (in degrees): ');
theta_z_deg = input('Enter the rotation angle around the z-axis (in degrees): ');

% Convert angles from degrees to radians
theta_x = deg2rad(theta_x_deg);
theta_y = deg2rad(theta_y_deg);
theta_z = deg2rad(theta_z_deg);

% Define the original vector
v = [1; 1; 1];

% Compute rotation matrices
R_x = [1, 0, 0;
       0, cos(theta_x), -sin(theta_x);
       0, sin(theta_x), cos(theta_x)];

R_y = [cos(theta_y), 0, sin(theta_y);
       0, 1, 0;
       -sin(theta_y), 0, cos(theta_y)];

R_z = [cos(theta_z), -sin(theta_z), 0;
       sin(theta_z), cos(theta_z), 0;
       0, 0, 1];

% Rotate the vector
v_rot_x = R_x * v;
v_rot_y = R_y * v;
v_rot_z = R_z * v;

% Plotting
figure;
hold on;
grid on;
axis equal;
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Vector Rotation Visualization in 3D Space');

% Plot the original vector
```

```
quiver3(0, 0, 0, v(1), v(2), v(3), 'k', 'LineWidth', 2, '
    DisplayName', 'Original Vector');

% Plot the rotated vectors
quiver3(0, 0, 0, v_rot_x(1), v_rot_x(2), v_rot_x(3), 'r', '
    LineWidth', 2, 'DisplayName', sprintf('Rotated Vector (X-axis,
    %.1f)', theta_x_deg));
quiver3(0, 0, 0, v_rot_y(1), v_rot_y(2), v_rot_y(3), 'g', '
    LineWidth', 2, 'DisplayName', sprintf('Rotated Vector (Y-axis,
    %.1f)', theta_y_deg));
quiver3(0, 0, 0, v_rot_z(1), v_rot_z(2), v_rot_z(3), 'b', '
    LineWidth', 2, 'DisplayName', sprintf('Rotated Vector (Z-axis,
    %.1f)', theta_z_deg));

% Set view angle for better 3D visualization
view(3);

% Add legend
legend('show');
```

Output of the rotation is shown here.

```
Rotation of v in x-direction:
    1.0000
    0.8112
    1.1585
Rotation of v in y-direction:
    1.2817
    1.0000
    0.5977
Rotation of v in z-direction:
    0.2456
    1.3927
    1.0000
```

A 3D visualization of the above code is shown in Figure 7.3.

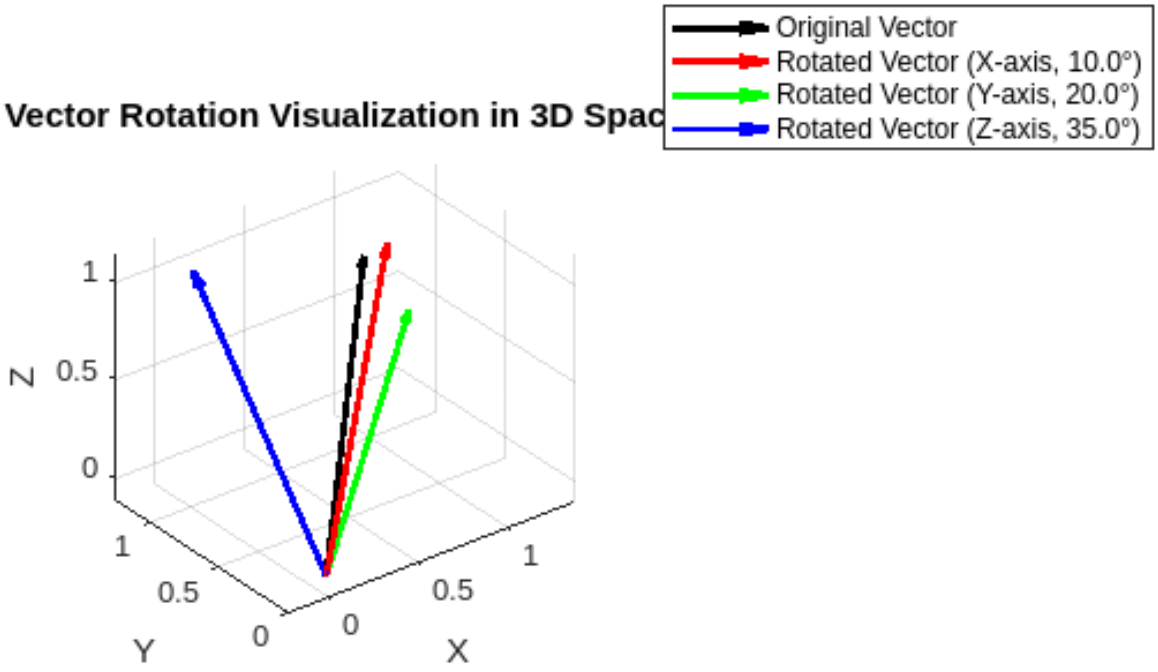


Figure 7.3: Vector rotation in 3D space.

Q2. Create wavelet theory based orthogonal matrices and find its eigenvalues.

```
H2 = (1/sqrt(2)) * [1 1; 1 -1];
H4 = (1/2) * [1 1 1 1; 1 -1 1 -1; 1 1 -1 -1; 1 -1 -1 1];
H8 = (1/4) * [1 1 1 1 1 1 1 1;
              1 -1 1 -1 1 -1 1 -1;
              1 1 -1 -1 1 1 -1 -1;
              1 -1 -1 1 1 -1 -1 1;
              1 1 1 1 -1 -1 -1 -1;
              1 -1 1 -1 -1 1 -1 1;
              1 1 -1 -1 -1 -1 1 1;
              1 -1 -1 1 -1 1 1 -1];

eig_H2 = eig(H2);
eig_H4 = eig(H4);
eig_H8 = eig(H8);

disp('Eigenvalues of 2x2 Haar matrix:');
```

```
Eigenvalues of 2x2 Haar matrix:
```

```
disp(eig_H2);
```

```
    -1
     1
```

```

disp('Eigenvalues of 4x4 Haar matrix:');
```

```
  Eigenvalues of 4x4 Haar matrix:
```

```
disp(eig_H4);
```

```
    -1
    -1
     1
     1
```

```
disp('Eigenvalues of 8x8 Haar matrix:');
```

```
  Eigenvalues of 8x8 Haar matrix:
```

```
disp(eig_H8);
```

```
   -0.7071
   -0.7071
   -0.7071
   -0.7071
    0.7071
    0.7071
    0.7071
    0.7071
```

Q3. Generate 4×4, 8×8, 16×16 walsh-Hadamard matrices(Normalized) and show that their eigenvalues are either +1 or -1. There may be any other orthogonal matrix that ensures real eigenvalues. Normalize each matrix with appropriate numbers and then find eigenvalue.

SOLUTION

Matlab code for this task is given below.

```
A = [1 1; 1 -1];
H4 = kron(A, A); % 4x4 matrix
H8 = kron(H4, A); % 8x8 matrix
H16 = kron(H4, H4); % 16x16 matrix
normFactor4 = sqrt(size(H4, 1));
normFactor8 = sqrt(size(H8, 1));
normFactor16 = sqrt(size(H16, 1));
H4 = H4 / normFactor4;
H8 = H8 / normFactor8;
H16 = H16 / normFactor16;

eig_H4 = eig(H4);
eig_H8 = eig(H8);
eig_H16 = eig(H16);
disp('Eigenvalues of 4x4 Walsh-Hadamard matrix:');
```

Output of the code is shown below.

```
Eigenvalues of 4x4 Walsh-Hadamard matrix:
```

```
disp(eig_H4);
```

```
   -1
   -1
    1
    1
```

```

disp('Eigenvalues of 8x8 Walsh-Hadamard matrix:');
```

```
Eigenvalues of 8x8 Walsh-Hadamard matrix:
```

```
disp(eig_H8);
```

```
   -1.0000
   -1.0000
   -1.0000
   -1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

```

disp('Eigenvalues of 16x16 Walsh-Hadamard matrix:');
```

```
Eigenvalues of 16x16 Walsh-Hadamard matrix:
```

```
disp(eig_H16);
```

```
   -1.0000
   -1.0000
   -1.0000
   -1.0000
   -1.0000
   -1.0000
   -1.0000
   -1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

```
subplot(1, 3, 1);
imagesc(H4);
title('4x4 Walsh-Hadamard Matrix');
colorbar;
subplot(1, 3, 2);
imagesc(H8);
title('8x8 Walsh-Hadamard Matrix');
colorbar;

subplot(1, 3, 3);
imagesc(H16);
title('16x16 Walsh-Hadamard Matrix');
colorbar;
```

Q4. Generate random 4x4 integer matrix with rank 2 and find full SVD.

Then find eigenvalues of U and V matrices. Demonstrate that absolute value of any eigenvalue is 1.

SOLUTION

Matlab code for this task is given below.

```
A = randi(10, 4, 2) * randi(10, 2, 4);
[U, S, V] = svd(A);
eig_U = eig(U);
eig_V = eig(V);
disp('Random 4x4 matrix with rank 2:');
```

```
Random 4x4 matrix with rank 2:
```

```
disp(A);
```

```
    70    60   120   160
    32    29    50    74
    26    27    30    62
    40    28    88    88
```

```

disp('Eigenvalues of U matrix:');
```

```
Eigenvalues of U matrix:
```

```
disp(eig_U);
```

```
   1.0000 + 0.0000i
  -0.8422 + 0.5392i
  -0.8422 - 0.5392i
  -1.0000 + 0.0000i
```

```

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

```
Eigenvalues of V matrix:
```

```
disp(eig_V);
```

```
  -0.0559 + 0.9984i
  -0.0559 - 0.9984i
  -0.6060 + 0.7954i
  -0.6060 - 0.7954i
```

```
disp('Absolute values of eigenvalues of U:');
```

```
Absolute values of eigenvalues of U:
```

```
disp(abs(eig_U));
```

```
    1.0000
    1.0000
    1.0000
    1.0000
```

```

disp('Absolute values of eigenvalues of V:');
```

```
Absolute values of eigenvalues of V:
```

```
disp(abs(eig_V));
```

```
    1.0000
    1.0000
    1.0000
    1.0000
```

RESULTS

1. Properties of orthogonal matrices are revisited.
2. Classic orthogonal matrices of practical importance are discussed and computationally executed in `matlab`.

8 Assignment 29
Eigenvalues, Trace and Determinants

8.1 Back to the basics

The concepts of eigenvalues, eigenvectors, similarity matrices, and the Cayley-Hamilton theorem provide powerful tools for matrix analysis, especially in reducing matrices to simpler forms (diagonal or low-rank approximations). These methods are widely used in applications ranging from solving differential equations to machine learning, and understanding the spectral decomposition forms the basis for a deeper exploration into linear algebra and numerical methods.

8.1.1 Characteristic Polynomial

For any square matrix $A \in \mathbb{R}^{n\times n}$, the **characteristic polynomial** is given by:

$$p_A(\lambda) = \det(A - \lambda I),$$

where λ is a scalar (the eigenvalue) and I is the identity matrix of the same dimension as A.

- The characteristic polynomial is a degree n polynomial in λ.
- The roots of this polynomial are the eigenvalues of A.

Example: For a 2×2 matrix A, the characteristic polynomial is:

$$p_A(\lambda) = \det(A - \lambda I) = \det\begin{pmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{pmatrix} = (\lambda_1 - \lambda)(\lambda_2 - \lambda).$$

8.1.2 Cayley-Hamilton Theorem

The **Cayley-Hamilton Theorem** states that every square matrix satisfies its own characteristic polynomial. For a matrix A with characteristic polynomial $p_A(\lambda)$, we have:

$$p_A(A) = 0.$$

For an $n \times n$ matrix A, let the characteristic polynomial be:

$$p_A(\lambda) = \det(A - \lambda I) = \lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0.$$

Then, the Cayley-Hamilton theorem implies:

$$A^n + c_{n-1}A^{n-1} + \cdots + c_1 A + c_0 I = 0.$$

This result has several practical implications:

- **Power of matrices**: Higher powers of A can be expressed as linear combinations of lower powers.
- **Low-rank approximation**: Using the Cayley-Hamilton theorem, it's possible to construct low-rank approximations or reduce the complexity of matrix calculations.

8.1.3 Eigenvalues and Eigenvectors

For a square matrix A, an **eigenvalue** λ and an associated **eigenvector** v satisfy:

$$Av = \lambda v.$$

Here:

- λ is a scalar (eigenvalue).
- v is a non-zero vector (eigenvector).

Properties:

- **Eigenvalues** may be real or complex, even if A is real.
- Eigenvectors corresponding to distinct eigenvalues are linearly independent.
- If A is symmetric ($A = A^T$), its eigenvalues are real, and its eigenvectors form an orthonormal basis for $\mathbb{R}^n$.

8.1.4 Spectral Theorem (for Symmetric Matrices)

If $A \in \mathbb{R}^{n \times n}$ is symmetric ($A = A^T$), then:

$$A = Q\Lambda Q^T,$$

where:

- Q is an orthogonal matrix whose columns are the eigenvectors of A (i.e., $Q^T Q = I$).
- Λ is a diagonal matrix containing the eigenvalues of A.

This decomposition is called **spectral decomposition** or **eigen decomposition**. It allows the matrix A to be understood in terms of its eigenvalues and eigenvectors.

8.1.5 Eigenvalues and Eigenvectors of Similarity Matrices

Given a matrix A and any invertible matrix P, the matrix $B = P^{-1}AP$ is called **similar** to A. Similar matrices have the following properties:

- **Same eigenvalues**: Matrices A and B share the same eigenvalues.

$$\text{If } Av = \lambda v, \text{ then } Bw = \lambda w, \text{ where } w = P^{-1}v.$$

- The eigenvectors of B are related to the eigenvectors of A via the transformation matrix P:

$$w = P^{-1}v.$$

- Similarity preserves the spectrum of the matrix, meaning that diagonalizing A or B gives the same eigenvalues.

8.1.6 Low-Rank Matrix Approximation

Using eigenvalue decomposition, we can approximate A by truncating small eigenvalues:

$$A \approx Q_k \Lambda_k Q_k^T,$$

where Q_k contains the first k eigenvectors and Λ_k the first k eigenvalues. This gives a rank-k approximation of A, reducing computational complexity.

Matrix Powers Using Eigen Decomposition

If A has eigen decomposition $A = Q\Lambda Q^{-1}$, then higher powers of A can be computed as:

$$A^k = Q\Lambda^k Q^{-1},$$

where Λ^k is the diagonal matrix with each diagonal element λ_i^k (the k-th power of the eigenvalue).

8.1.7 Properties of Eigenvalues and Eigenvectors

1. **Sum of eigenvalues**: The sum of the eigenvalues of a matrix is equal to the trace of the matrix.

$$\sum_{i=1}^{n} \lambda_i = \text{Tr}(A).$$

2. **Product of eigenvalues**: The product of the eigenvalues of a matrix is equal to the determinant of the matrix.

$$\prod_{i=1}^{n} \lambda_i = \det(A).$$

3. **Orthogonality of eigenvectors**: If A is symmetric, its eigenvectors corresponding to distinct eigenvalues are orthogonal.

4. **Eigenvalues of powers of matrices**: If λ is an eigenvalue of A, then λ^k is an eigenvalue of A^k.

5. **Real eigenvalues**: If A is symmetric, all eigenvalues are real.

6. **Complex eigenvalues**: For general (non-symmetric) matrices, eigenvalues can be complex.

7. **Zero eigenvalue**: A zero eigenvalue indicates that A is singular, meaning $\det(A) = 0$.

8. **Eigenvector normalization**: Eigenvectors can be normalized, so $\|v_i\| = 1$.

8.2 Taks

1. create two 4×4 matrices A and B with rank 3 such that rank of AB is 2.

   SOLUTION

   Here the key idea is the consequence of cayley Hamilton theorem. If λ_1 and λ_2 are eigen values of A, then $M_1 = A - \lambda_1 I$ and $M_2 = A - \lambda_2 I$ have same shape of A but one less in rank. Also, $M_1 M_2$ will have rank 2 less than that of A.

   To get the desired matrices, M_1 and M_2, let's start with a non singular matrix B and create a new base matrix with specified eigen values 1, 2, 3, and 4 (say). Now create a matrix $A = B^{-1} * D * B$. Here A is called a similarity matrix and has same eigen values as D.

   Create the base matrix, $B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$

   Clearly, $|B| = 16 \neq 0$, so it is nonsingular. Now create a matrix, P such that P has the eigenvalues $1, 2, 3, 4$. This can be achieved by constructing P as a similar matrix to a diagonal matrix D, with diagonal entries 1 , 2 3 and 4.

$$
\begin{aligned}
P &= B^{-1}DB \\
&= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.2500 & 0.2500 & 0.2500 & 0.2500 \\ 0.2500 & -0.2500 & 0.2500 & -0.2500 \\ 0.2500 & 0.2500 & -0.2500 & -0.2500 \\ 0.2500 & -0.2500 & -0.2500 & 0.2500 \end{bmatrix} \\
&= \begin{bmatrix} 5/2 & -1/2 & -1 & 0 \\ -1/2 & 5/2 & 0 & -1 \\ -1 & 0 & 5/2 & -1/2 \\ 0 & -1 & -1/2 & 5/2 \end{bmatrix}
\end{aligned}
$$

Now construct two matrices: $M_1 = P - I$, $M_2 = p - 2I$, and $M_3 = M_1 M_2$. These matrices are respectively,

$$
\begin{aligned}
M_1 &= \begin{bmatrix} 3/2 & -1/2 & -1 & 0 \\ -1/2 & 3/2 & 0 & -1 \\ -1 & 0 & 3/2 & -1/2 \\ 0 & -1 & -1/2 & 3/2 \end{bmatrix} \\
M_2 &= \begin{bmatrix} 1/2 & -1/2 & -1 & 0 \\ -1/2 & 1/2 & 0 & -1 \\ -1 & 0 & 1/2 & -1/2 \\ 0 & -1 & -1/2 & 1/2 \end{bmatrix} \\
M_1 M_2 &= \begin{bmatrix} 3/2 & -1/2 & -1 & 0 \\ -1/2 & 3/2 & 0 & -1 \\ -1 & 0 & 3/2 & -1/2 \\ 0 & -1 & -1/2 & 3/2 \end{bmatrix} \begin{bmatrix} 1/2 & -1/2 & -1 & 0 \\ -1/2 & 1/2 & 0 & -1 \\ -1 & 0 & 1/2 & -1/2 \\ 0 & -1 & -1/2 & 1/2 \end{bmatrix} \\
&= \begin{bmatrix} 2 & -1 & -2 & 1 \\ -1 & 2 & 1 & -2 \\ -2 & 1 & 2 & -1 \\ 1 & -2 & -1 & 2 \end{bmatrix}
\end{aligned}
$$

Now to find the rank of M_1, M_2, and $M_1 M_2$, reduce then into row reduced Echelon form as follows.

$$
\begin{aligned}
rref(M_1) &= \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
rref(M_2) &= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
rref(M_3) &= \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

Counting number of non-zero rows of these row reduces Echelon forms; it is clear that

$$\rho(M_1) = 3$$
$$\rho(M_2) = 3$$
$$\rho(M_3) = 2$$

as desired. Note that all these matrices, M_1, M_2, and M_3, are 4×4 matrices.

`Matlab` code for this task is given below.

```
Base=[1 1;1 -1];
Base=kron(Base,Base);
P=inv(Base)*diag([1,2,3,4])*Base;
M_1=P-eye(4);
M_2=P-2*eye(4);
M_3=M_1*M_2
tol=0.001;
I=eye(4);
r=rank(M_1,tol);
r1=rank(M_2,tol);
r2=rank(M_3,tol);
disp("Rank of M_1:")
disp(r);
disp("rank of M_2:")
disp(r1)
disp("Rank of M_3:")
disp(r2)
```

Output of the code is given below.

```
Rank of M_1:
       3
rank of M_2:
       3
Rank of M_3:
       2
```

2. Create three 4×4 matrices A , B, and C with rank 3 such that rank of ABC is 1.

SOLUTION

As continuation of the previous taks, just create a new matrix $C = P - 3I$, and $A = P - I$ and $B = P - 2I$.

For completion, let's write C.

$$C = P - 3I$$
$$= \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -1 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 & -1 \\ -1 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -1 & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$
$$ABC = \begin{bmatrix} \frac{3}{2} & -\frac{3}{2} & -\frac{3}{2} & \frac{3}{2} \\ -\frac{3}{2} & \frac{3}{2} & \frac{3}{2} & -\frac{3}{2} \\ -\frac{3}{2} & \frac{3}{2} & \frac{3}{2} & -\frac{3}{2} \\ \frac{3}{2} & -\frac{3}{2} & -\frac{3}{2} & \frac{3}{2} \end{bmatrix}$$
$$rref(ABC) = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

So rank of ABC is 1. Matlab code for this task is given below.

```
A=P-eye(4);
B=P-2*eye(4);
C=P-3*eye(4)
ABC=A*B*C
tol=0.001;
I=eye(4);
r=rank(A,tol);
r1=rank(B,tol);
r2=rank(C,tol);
r3=rank(ABC,tol);
disp("rank of ABC is:");
disp(r3);
```

Output of the code is shown below.

```
rank of ABC is:
        1
```

3. Create four 4x4 matrices A , B C, and D with rank 3 such that the rank of ABCD is 0

SOLUTION

This problem can be solved as in the previous task. Create one more matrix, $D = P - 4I$, then $M_4 = ABCD$ is a 4×4 matrix (a zero matrix), but its rank is 0. Matlab code demonstrating this result computationally is given below.

```
M_4=(P-eye(4))*(P-2*eye(4))*(P-3*eye(4))*(P-4*eye(4));
disp("rank of M_4 is:")
disp(rank(M_4,tol))
```

Output of the above code is shown below.

```
rank of M_4 is:
       0
```

4. Consider the 4×4 matrix A created in the previous task, what are the eigenvalues of a) $B = A+2I$, b) $C = A-3I$.

SOLUTION

To find eigen valuues of $A+2I$ and $A-3I$, we will use the properties of eigen values. If λ is an eigen value of A, then $\lambda+a$ and $\lambda-a$ will be the eigen values of $A+a\lambda$ and $A-\lambda I$ respectively. So with out actual calculation of eigen values of $A+2I$ and $A-3I$ we can write them using this property. Since $1,2,3$ and 4 are the eigen values of A,

$$\begin{aligned} \text{EVs}(A+2I) &= \{1+2, 2+2, 3+2, 4+2\} \\ &= \{3,4,5,6\} \\ \text{Evs}(A-3I) &= \{1-3, 2-3, 3-3, 4-3\} \\ &= \{-2,-1,0,1\} \end{aligned}$$

`Matlab` code to verify the result is shown below.

```
B1=Am+2*eye(4)
B2=Am-3*eye(4)
disp("Eigen Values of B=A+2I is:")
disp(eig(B1));
disp("Eigen Values of C=A-3I is")
disp(eig(B2));
```

output of the code is shown below.

```
Eigen Values of B=A+2I is:
       3
       4
       5
       6
Eigen Values of C=A-3I is
      -2
      -1
      -1/3611724390554282
       1
```

5. Create a 3×3 matrix with eigenvalues 0,1,2 using `Matlab`.

SOLUTION

Using the `magic` matrix in `matlab`, a similar matrix with given eigen values 0,1,2 is created and verified. The code for this task is given below.

```
format default
P=magic(3);
D=diag([0 1 2]);
M=inv(P)*D*P;
disp("The similarity matrix with eigen values as that of D
   created is:");
disp(M)
disp("Verification of eigen values:")
disp("Eigen values of M is:")
disp(eig(M))
```

Output of the code is shown below.

```
The similarity matrix with eigen values as that of D created is:
    0.0778    0.4278   -0.7556
    0.9111    2.0111    0.5778
   -0.2556   -0.9056    0.9111
Verification of eigen values:
Eigen values of M is:
   -0.0000
    2.0000
    1.0000
```

RESULTS

Matrices with given properties related to rank and eigenvalues are created, and results are verified computationally using `matlab`.

9 Assignment 30
Eigen Values and Eigen Vectors

9.1 Introduction

This lesson focused on revisiting the topics eigen values, eigen vectors, its properties and relationship with null space.

1. Find eigenvalues and vectors of 2x2 matrix with rank 2.
Example

$A = \begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix}$. Note that rank is 2

$\lambda_1 + \lambda_2 = \mathrm{Tr}(A) = 3$ (1)

$$\lambda_1\lambda_2 = \det(A) = -4$$
$$(\lambda_1 - \lambda_2)^2 = (\lambda_1 + \lambda_2)^2 - 4\lambda_1\lambda_2 = 9 + 16 = 25$$

$\lambda_1 - \lambda_2 = 5$ (2)

(1) and (2) $\Rightarrow \lambda_1 = 4, \lambda_2 = -1$

In case of 2x2 , you need to use only first row of A matrix for finding eigen vectors.

$$\begin{bmatrix} 1-\lambda & 2 \\ 3 & 2-\lambda \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} \perp \begin{bmatrix} 1-\lambda \\ 2 \end{bmatrix}$$

$\therefore \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -2 \\ 1-\lambda \end{bmatrix}$ Can u figure out Why?

$$\mathrm{EV}(\lambda_1 = 4) = \begin{bmatrix} -2 \\ 1-\lambda \end{bmatrix} = \begin{bmatrix} -2 \\ 1-4 \end{bmatrix} = \begin{bmatrix} -2 \\ -3 \end{bmatrix} \cong \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$\mathrm{EV}(\lambda_2 = -1) = \begin{bmatrix} -2 \\ 1-\lambda \end{bmatrix} = \begin{bmatrix} -2 \\ 1+! \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$$

You can verify the result

1. Find basis set for all the vector spaces associated with 2x2 matrix with rank 2.

Here you can write the result blindly. Any two independent vectors in $\mathbb{R}^2$ will suffice for rows pace and column space.

All the basis set vectors must be written as a set of column vectors. That is the standard.

$$C(A) = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

$$R(A) = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

$$\text{RN}(A) = \text{null}$$
$$\text{LN}(A) = \text{null}$$

2. Find basis set for all the vector spaces associated with 2x2 matrix with rank 1. Also find eigenvalues and eigen vectors.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}$$

Answers.

All the basis set vectors must be written as a set of column vectors. That is the standard.

Row space is spanned by $\left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\}$

Right Null space is spanned by $\left\{ \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\}$

column space is spanned by $\left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix} \right\}$

Left Null space is spanned by $\left\{ \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$

Eigen values

Since rank deficiency is 1, one eigen value is zero . Corresponding eigen vector is Right null space vector. The other eigenvalue is 6+1= 7,

Corresponding eigen vector is $\begin{bmatrix} -2 \\ 1-7 \end{bmatrix} \cong \begin{bmatrix} 2 \\ 3 \end{bmatrix}$

Dealing with 3x3 matrices

As per Gilbert Strang, asking eigenvalue and eigen vector computation of general $n \times n$ matrix , $n \geq 3$ in examination is a crime (unless it is a special matrix). But unfortunately in India our teachers test only numerical (or clerical) ability of Engineering students rather than the conceptual and visualization capability of students. (Conceptual do not mean asking straight forward definitions, advantages and disadvantages etc).

Here we give below how to deal with eigen value computation of 3x3 matrices for a quick answer. We also deal with special cases.

9.1.1 Writing Characteristic equation of a matrix

3. Characteristic polynomial of 3x3 matrix can be computed very fast except the last term which is determinant .

Let $A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

$$|A - \lambda I| = \left| \begin{bmatrix} a-\lambda & b & c \\ d & e-\lambda & f \\ g & h & i-\lambda \end{bmatrix} \right|$$
$$= (-1)^3\lambda^3 + (-1)^2\lambda^2(a+e+i) + (-1)\lambda\left((\mathrm{ae}+\mathrm{ei}+\mathrm{ai}) - \left(\mathrm{bd}+\mathrm{cg}+\mathrm{fh}\right)\right) + \det(A)$$
$$= (-1)^3\lambda^3 + (-1)^2\lambda^2\mathrm{Tr}(A) + (-1)\lambda\left((\mathrm{ae}+\mathrm{ei}+\mathrm{ai}) - \left(\mathrm{bd}+\mathrm{cg}+\mathrm{fh}\right)\right) + \det(A)$$

Find characteristic equation of the following matrix

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

It is a special matrix. Column sum is same.

Characteristic equation is given by

$$0 = (-1)^3\lambda^3 + (-1)^2\lambda^2\mathrm{Tr}(A) + (-1)\lambda\left((\mathrm{ae}+\mathrm{ei}+\mathrm{ai}) - \left(\mathrm{bd}+\mathrm{cg}+\mathrm{fh}\right)\right) + \det(A)$$
$$= -\lambda^3 + 3\lambda^2 + (-1)\lambda((-1) - (35)) + 32$$
$$= -\lambda^3 + 3\lambda^2 + 36\lambda + 32$$

Or $\lambda^3 - 3\lambda^2 - 36\lambda - 32 = 0$

Note that all column sum is 8. So, one eigen value is 8.

(if all column sum or all row sum is same , that value (the sum) is an eigen value.)

$$\lambda_1 = 8$$

Immediately we infer that

$$8 + \lambda_2 + \lambda_3 = \mathrm{Tr}(A) = (1+3+-1) = 3$$
$$8\lambda_2\lambda_3 = \det(A) = 32$$
$$\Rightarrow \lambda_2 = -4; \lambda_3 = -1;$$

But there is no short cut for eigen-vector computation. Another 3x3 special matrix that can be easily handled are block diagonal matrices of the form

$$A = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & i \end{bmatrix} \text{ or } B = \begin{bmatrix} a & 0 & 0 \\ 0 & e & f \\ 0 & h & i \end{bmatrix}$$

Eigen values of A are eigen values of two blocks namely and eig$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Eigen values of B are eigen values of two blocks namely and eig$\begin{bmatrix} e & f \\ h & i \end{bmatrix}$

Another special matrix.

Lower and upper triangular matrices

Here Eigen values are same as diagonal elements.

9.1.2 Creating integer matrices with integer inverse matrix

Creating integer matrices with integer inverse matrix

$$A = \begin{bmatrix} {}^3C_0 & {}^4C_0 & {}^5C_0 \\ {}^3C_1 & {}^4C_1 & {}^5C_1 \\ {}^3C_2 & {}^4C_2 & {}^5C_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 5 \\ 3 & 6 & 10 \end{bmatrix}$$

$$B = \begin{bmatrix} {}^4C_0 & {}^5C_0 & {}^6C_0 & {}^7C_0 \\ {}^4C_1 & {}^5C_1 & {}^6C_1 & {}^7C_1 \\ {}^4C_2 & {}^5C_2 & {}^6C_2 & {}^7C_2 \\ {}^4C_3 & {}^5C_3 & {}^6C_3 & {}^7C_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 4 & 5 & 6 & 7 \\ 6 & 10 & 15 & 21 \\ 4 & 10 & 20 & 35 \end{bmatrix}$$

Assignment

1. Find the eigen values and eigen vectors of the matrix

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

```
A = [1 -1; 2 4];
[eigVec, eigVal] = eig(A);
disp('Eigenvalues:');
```

```
Eigenvalues:
```

```
disp(diag(eigVal));
```

```
   2
   3
```

```
disp('Eigenvectors:');
```

```
Eigenvectors:
```

```
disp(eigVec);
```

```
   -0.7071    0.4472
    0.7071   -0.8944
```

9.1.3 Eigen values of diagonal matrices

2. Find the eigen values of A, B and C:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 3 & 0 & 0 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

SOLUTION

1. Given matrix A is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

Eigenvalues:

The eigenvalues of the matrix A are the diagonal elements of this upper triangular matrix:

$$\lambda_1 = 1, \quad \lambda_2 = 4, \quad \lambda_3 = 6$$

Eigenvectors:

To find the eigenvectors, we solve $(A - \lambda I)x = 0$ for each eigenvalue λ.

For $\lambda_1 = 1$:

$$A - I = \begin{bmatrix} 0 & 2 & 3 \\ 0 & 3 & 5 \\ 0 & 0 & 5 \end{bmatrix}$$

Solving $(A - I)x = 0$ yields the eigenvector:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

For $\lambda_2 = 4$:

$$A - 4I = \begin{bmatrix} -3 & 2 & 3 \\ 0 & 0 & 5 \\ 0 & 0 & 2 \end{bmatrix}$$

Solving $(A - 4I)x = 0$ yields the eigenvector:

$$\mathbf{v}_2 = \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix}$$

For $\lambda_3 = 6$:

$$A - 6I = \begin{bmatrix} -5 & 2 & 3 \\ 0 & -2 & 5 \\ 0 & 0 & 0 \end{bmatrix}$$

Solving $(A - 6I)x = 0$ yields the eigenvector:

$$\mathbf{v}_3 = \begin{bmatrix} 16 \\ 25 \\ 10 \end{bmatrix}$$

`matlab` code for this task is given below.

```
% question 1
A = [1 2 3; 0 4 5; 0 0 6];
[eigVec, eigVal] = eig(A);
disp('Eigenvalues:');
```

Output of the code is shown below.

```
Eigenvalues:
```

```
disp(diag(eigVal));
```

```
     1
     4
     6
```

```
disp('Eigenvectors:');
```

```
Eigenvectors:
```

```
disp(eigVec);
```

```
    1.0000    0.5547    0.5108
         0    0.8321    0.7982
         0         0    0.3193
```

2. Here the matrix B is given by

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

Eigenvalues:

To find the eigenvalues, solve the characteristic polynomial:

$$\det(B - \lambda I) = (\lambda - 2)(\lambda - \sqrt{3})(\lambda + \sqrt{3}) = 0$$

The eigenvalues of matrix B are:

$$\lambda_1 = 2, \quad \lambda_2 = \sqrt{3}, \quad \lambda_3 = -\sqrt{3}$$

Eigenvectors:

To find the eigenvectors, solve $(B - \lambda I)x = 0$ for each eigenvalue λ.

For $\lambda_1 = 2$:

$$B - 2I = \begin{bmatrix} -2 & 0 & 1 \\ 0 & 0 & 0 \\ 3 & 0 & -2 \end{bmatrix}$$

Solving $(B - 2I)x = 0$ yields the eigenvector:

$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

For $\lambda_2 = \sqrt{3}$:

$$B - \sqrt{3}I = \begin{bmatrix} -\sqrt{3} & 0 & 1 \\ 0 & 2-\sqrt{3} & 0 \\ 3 & 0 & -\sqrt{3} \end{bmatrix}$$

Solving $(B - \sqrt{3}I)x = 0$ yields the eigenvector:

$$\mathbf{v}_2 = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

For $\lambda_3 = -\sqrt{3}$:

$$B - (-\sqrt{3})I = \begin{bmatrix} \sqrt{3} & 0 & 1 \\ 0 & 2+\sqrt{3} & 0 \\ 3 & 0 & \sqrt{3} \end{bmatrix}$$

Solving $(B - (-\sqrt{3})I)x = 0$ yields the eigenvector:

$$\mathbf{v}_3 = \begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

values of a, b , c, d, e and f can be found using `matlab`. `Matlab` code and output for this task is given below.

```
% question 2
B = [0 0 1; 0 2 0; 3 0 0];
[eigVec, eigVal] = eig(B);

disp('Eigenvalues:');
```

```
Eigenvalues:
```

```
disp(diag(eigVal));
```

```
    1.7321
   -1.7321
    2.0000
```

```
disp('Eigenvectors:');
```

```
Eigenvectors:
```

```
disp(eigVec);
```

```
    0.5000   -0.5000         0
         0         0    1.0000
    0.8660    0.8660         0
```

3. Matrix C is given as a row redundant matrix,

$$C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

Eigenvalues:

$$\det(C - \lambda I) = \begin{vmatrix} 2-\lambda & 2 & 2 \\ 2 & 2-\lambda & 2 \\ 2 & 2 & 2-\lambda \end{vmatrix} = 0$$

The eigenvalues are:

$$\lambda_1 = 6, \quad \lambda_2 = 0, \quad \lambda_3 = 0$$

Eigenvectors:

For $\lambda_1 = 6$:

$$C - 6I = \begin{bmatrix} -4 & 2 & 2 \\ 2 & -4 & 2 \\ 2 & 2 & -4 \end{bmatrix}$$

The eigenvector corresponding to $\lambda_1 = 6$ is:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

For $\lambda_2 = 0$ and $\lambda_3 = 0$:

$$C - 0I = C$$

The eigenvectors corresponding to $\lambda_2 = 0$ and $\lambda_3 = 0$ are orthogonal to $\mathbf{v}_1$, are in the null space of C and are the solutions of $x + y + z = 0$ given by

$$v_1 = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

$$v_2 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

`Matlab` code and its output are shown below.

```
% question 3
C = [2 2 2; 2 2 2; 2 2 2];
[eigVec, eigVal] = eig(C);
disp('Eigenvalues:');
```

```
Eigenvalues:
```

```
disp(diag(eigVal));
```

```
-0.0000
-0.0000
 6.0000
```

```

disp('Eigenvectors:');
```

```
Eigenvectors:
```

```
disp(eigVec);
```

```
   -0.3094   -0.7556    0.5774
   -0.4996    0.6458    0.5774
    0.8091    0.1098    0.5774
```

9.1.4 Eigen values of block matrix

3) If B has eigenvalues 1 ,2 ,3, C has eigenvalues 4, 5, and 6, and D has eigenvalues 7,8,9, what are the eigenvalues of 6 by 6 matrix $A = \begin{bmatrix} B & C \\ 0 & D \end{bmatrix}$?

SOLUTION

Given the block matrix:

$$A = \begin{bmatrix} B & C \\ 0 & D \end{bmatrix}$$

Choose B , C and D to produce the eigen values:

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}, \quad C = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{bmatrix}, \quad D = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

Eigenvalues of Matrix A:
The eigenvalues of a block matrix of the form:

$$A = \begin{bmatrix} B & C \\ 0 & D \end{bmatrix}$$

are the union of the eigenvalues of the diagonal blocks B and D. This is because A is an upper block triangular matrix, and for such matrices, the eigenvalues are those of the diagonal blocks.
Thus, the eigenvalues of matrix A are the eigenvalues of B and D.
The eigenvalues of matrix B are:

$$\lambda_B = \{1, 2, 3\}$$

The eigenvalues of matrix D are:

$$\lambda_D = \{7, 8, 9\}$$

Combining these, the eigenvalues of matrix A are:

$$\lambda_A = \{1, 2, 3, 7, 8, 9\}$$

Matlab code to verify above claim is given below.

```
% Define matrices B, C, and D with the given eigenvalues
B = [1 0 0; 0 2 0; 0 0 3];
C = [0 0 0; 0 0 0; 0 0 0];
D = [7 0 0; 0 8 0; 0 0 9];

```

```
A = [B, C;
     zeros(size(D)), D];
eigenvalues_A = eig(A);
disp('Eigenvalues of A:');
```

```
Eigenvalues of A:
```

```
disp(eigenvalues_A);
```

```
     1
     2
     3
     7
     8
     9
```

```
eigenvalues_B = eig(B);
eigenvalues_D = eig(D);
disp('Eigenvalues of B:');
```

```
Eigenvalues of B:
```

```
disp(eigenvalues_B);
```

```
     1
     2
     3
```

```
disp('Eigenvalues of D:');
```

```
Eigenvalues of D:
```

```
disp(eigenvalues_D);
```

```
     7
     8
     9
```

```
combined_eigenvalues = sort([eigenvalues_B; eigenvalues_D]);
disp('Combined eigenvalues of B and D:');
```

```
Combined eigenvalues of B and D:
```

```
disp(combined_eigenvalues);
```

```
1
2
3
7
8
9
```

RESULTS

1. Eigen values and eigen vectors of the given problems are found.
2. Properties of eigen values of block matrix are verified mathematically and computationally.