SCHOOL OF ARTIFICIAL INTELLIGENCE

24MA602 Computational Mathematics for Data Science

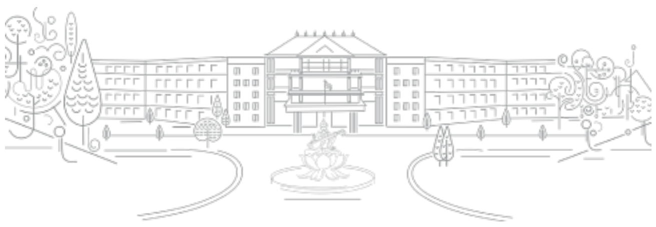
Assignment Set-6

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



SEPTEMBER
2024

Contents

List of Tables

List of Figures

1 | Assignment 65 Support Vector Machines: From Linear to Non-Linear Kernels

Support Vector Machines (SVMs) are super powered supervised learning algorithms commonly used for classification tasks. They work by finding the optimal hyperplane that separates data points of different classes in a feature space. In this chapter, we will explore both linear and non-linear kernels in SVMs, their mathematical formulations, and practical implementations, culminating in the best-fit model for a given dataset.

1.1 Linear Kernel SVM (recap)

1.1.1 The concept of a linear classifier

A linear classifier aims to find a hyperplane that maximizes the margin between two classes. The decision boundary can be expressed mathematically as:

$$f(x) = w^T x - \gamma = 0 \tag{1.1}$$

where w is the weight vector that defines the orientation of the hyperplane, $gamma$ is the bias term, and x is the input feature vector.

1.2 Numerical example

Consider the data given in Table 1.1.

Table 1.1: Data Table

i	x_1	x_2	d
1	1	1	-1
2	-1	0.5	-1
3	0.5	-1	-1
4	-1.5	-1	-1
5	2	3	1
6	-2	2.5	1
7	-2.5	-2.5	1
8	3	-3	1

1.3 Mathematical Formulation

To create a hard margin Support Vector Machine (SVM) model mathematically using the provided data, we can formulate the problem as follows:

1.3.1 Primal formulation

We want to find a weight vector **w** and a bias $gamma$ such that the following conditions hold:

$$d_i(\mathbf{w} \cdot \mathbf{x}_i - \gamma) \geq 1, \quad \forall i$$

where:

- $\mathbf{x}_i$ is the input data point.
- d_i is the corresponding label (-1 or 1).

The objective is to minimize the norm of the weight vector **w**:

$$\text{Minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

Subject to the constraints:

$$d_i(\mathbf{w} \cdot \mathbf{x}_i - gamma) \geq 1 \quad \forall i$$

1.3.2 Lagrangian formulation

To solve this constrained optimization problem, we introduce Lagrange multipliers u_i for each constraint:

$$L(\mathbf{w}, \gamma, \mathbf{u}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} u_i [d_i(\mathbf{w} \cdot \mathbf{x}_i - \gamma) - 1]$$

where $u_i \geq 0$.

1.3.3 Dual formulation

The dual problem can be derived from the Lagrangian by taking derivatives and setting them to zero. The dual problem is:

$$\text{Maximize} \quad \sum_{i=1}^{n} u_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} u_i u_j d_i d_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

Subject to:

$$\sum_{i=1}^{n} u_i d_i = 0 \quad \text{and} \quad u_i \geq 0 \quad \forall i$$

Implementation Steps

1.3.4 Construct the input data matrix

$$X = \begin{bmatrix} 1 & 1 \\ -1 & 0.5 \\ 0.5 & -1 \\ -1.5 & -1 \\ 2 & 3 \\ -2 & 2.5 \\ -2.5 & -2.5 \\ 3 & -3 \end{bmatrix}$$

1.3.5 Construct the label vector

$$d = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

`Matlab` code to solve the associated LPP with linear Kernel $K = X^T X$ is given below.

```
% Clear previous data
clear; close all; clc;

% Data
X = [1 1; -1 0.5; 0.5 -1; -1.5 -1; 2 3; -2 2.5; -2.5 -2.5; 3 -3];
d = [-1; -1; -1; -1; 1; 1; 1; 1];

n = length(d);

% Kernel Matrix (Linear Kernel)
K = X * X'; % K(i,j) = <x_i, x_j>

% Solve the Minimization Dual Problem using CVX
cvx_begin quiet
    variable u(n)
    minimize( 0.5 * quad_form(u .* d, K) - sum(u) )
    subject to
        sum(u .* d) == 0;
        u >= 0;
cvx_end

% Compute the weight vector
w = X' * (u .* d);

% Find support vectors and compute bias
support_vector_indices = find(u > 1e-5); % Tolerance to find non-
    zero u
bias = mean(d(support_vector_indices) - X(support_vector_indices,
    :) * w);
```

```matlab

% Display the results
disp('Lagrange multipliers (u):');
disp(u);
disp('Weight vector (w):');
disp(w);
disp('Bias (gamma):');
disp(bias);

% Visualization in 2D
figure;
hold on;

% Plot data points for Class 1 (d = 1) and Class -1 (d = -1)
plot(X(d==1,1), X(d==1,2), 'bo', 'MarkerSize', 10, 'LineWidth', 2);
    % Class 1 (blue)
plot(X(d==-1,1), X(d==-1,2), 'ro', 'MarkerSize', 10, 'LineWidth',
    2); % Class -1 (red)

% Plot the decision boundary
x1Plot = linspace(-4, 4, 100);
x2Plot = -(w(1) * x1Plot + bias) / w(2);
plot(x1Plot, x2Plot, 'k-', 'LineWidth', 2); % Decision boundary (
    black line)

% Highlight support vectors
plot(X(support_vector_indices, 1), X(support_vector_indices, 2), '
    ko', 'MarkerSize', 12, 'LineWidth', 2); % Support Vectors (black
    )

% Label plot
%title('Linear SVM with Support Vectors (2D Visualization)');
xlabel('X_1');
ylabel('X_2');

% Legend with distinct markers
legend({'Class 1 (d=1)', 'Class -1 (d=-1)', 'Decision Boundary', '
    Support Vectors'}, ...
        'Location', 'SouthOutside', 'Orientation', 'vertical');

grid on;
hold off;
```

Result of the code is shown below.

```
Lagrange multipliers (u):
    0.1357
    0.1209
    0.1282
    0.1152
    0.1071
    0.1338
    0.1487
    0.1104
```

```
Weight vector (w):
   1.0e-12 *

   -0.2200
   -0.0435

Bias (gamma):
  -1.6459e-14
```

This result is totally unexpected! All the Lagrangian multipliers are non-zero and the $w_i sim 0; \quad \forall i$. Skill of the model is shown in Figure 1.1. It is not successful in classification.

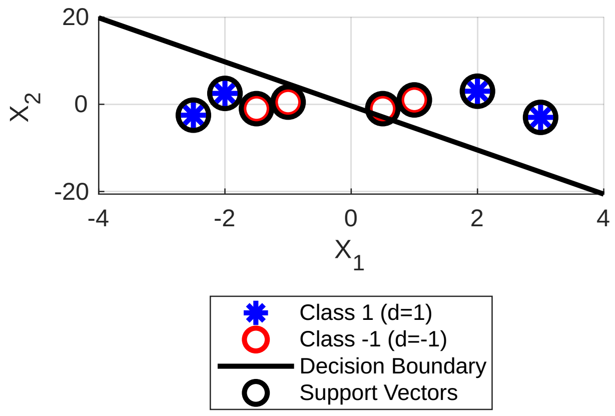


Figure 1.1: Data points and separating plane for SVM with linear Kernel

This can happen if:

- The scaling of data may lead to numerical issues in optimization.
- The problem is not properly constrained, or the data is not separable in the linear space.
- The data could be linearly separable but the regularization may work better.

Let's address this by reviewing the formulation and adding some key checks:

1.3.6 Linear SVM on scaled data

The data X is normalized by dividing each value by the maximum absolute value in X. This helps in stabilizing the optimization. We try this and solve the associated dual LPP as shown in the `Matlab` code given below.

```
% Clear previous data
clear; close all; clc;

% Data (original data)
X = [1 1; -1 0.5; 0.5 -1; -1.5 -1; 2 3; -2 2.5; -2.5 -2.5; 3 -3];
d = [-1; -1; -1; -1; 1; 1; 1; 1];

n = length(d);

% Normalize the data (this often helps with numerical stability)
X = X ./ max(abs(X), [], 'all');

```

```matlab
% Kernel Matrix (Linear Kernel)
K = X * X'; % K(i,j) = <x_i, x_j>

% Solve the Minimization Dual Problem using CVX
cvx_begin quiet
    variable u(n)
    minimize( 0.5 * quad_form(u .* d, K) - sum(u) )
    subject to
        sum(u .* d) == 0;
        u >= 0;
cvx_end

% Compute the weight vector
w = X' * (u .* d);

% Find support vectors and compute bias
support_vector_indices = find(u > 1e-5); % Adjusted tolerance to
    capture support vectors
bias = mean(d(support_vector_indices) - X(support_vector_indices,
    :) * w);

% Display the results
disp('Lagrange multipliers (u):');
disp(u);
disp('Weight vector (w):');
disp(w);
disp('Bias (gamma):');
disp(bias);

% Visualization in 2D
figure;
hold on;

% Plot data points for Class 1 (d = 1) and Class -1 (d = -1)
plot(X(d==1,1), X(d==1,2), 'bo', 'MarkerSize', 10, 'LineWidth', 2);
     % Class 1 (blue)
plot(X(d==-1,1), X(d==-1,2), 'ro', 'MarkerSize', 10, 'LineWidth',
    2); % Class -1 (red)

% Plot the decision boundary
x1Plot = linspace(-1, 1, 100); % Adjusted plot range to match
    normalized data
x2Plot = -(w(1) * x1Plot + bias) / w(2);
plot(x1Plot, x2Plot, 'k-', 'LineWidth', 2); % Decision boundary (
    black line)

% Highlight support vectors
plot(X(support_vector_indices, 1), X(support_vector_indices, 2), '
    ko', 'MarkerSize', 12, 'LineWidth', 2); % Support Vectors (black
    )

% Label plot
```

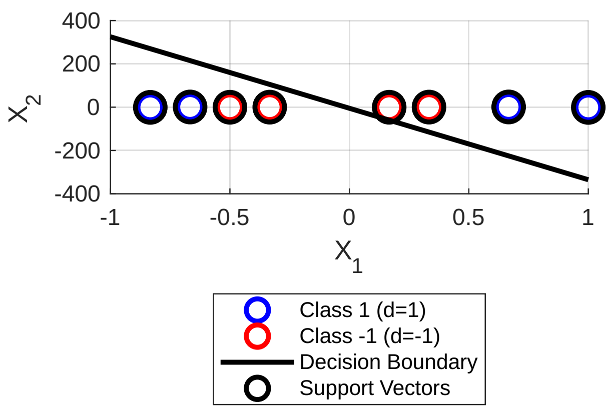


Figure 1.2: Data points and separating hyperplane for normalized linear SVM

```
%title('Linear SVM with Support Vectors (2D Visualization)');
xlabel('X_1');
ylabel('X_2');

% Legend with distinct markers
legend({'Class 1 (d=1)', 'Class -1 (d=-1)', 'Decision Boundary', '
   Support Vectors'}, ...
       'Location', 'SouthOutside', 'Orientation', 'vertical');

grid on;
hold off;
```

Output of the code is shown below.

```
Lagrange multipliers (u):
    0.1310
    0.0936
    0.1304
    0.1450
    0.1046
    0.1264
    0.1597
    0.1093

Weight vector (w):
   1.0e-14 *

   -0.9159
   -0.0028

Bias (gamma):
  -1.3878e-16
```

Data points and the separating hyper plane are shown in Figure 1.2. Again not promising. The first thing to do with the data before building a classification model is to visualize the data and check how the data points are distributed over the class labels. We missed this step. Now let's do its. `Matlab` code for this task is given below.

```
% Clear previous data
clear; close all; clc;
```

```
% Data (original data)
X = [1 1; -1 0.5; 0.5 -1; -1.5 -1; 2 3; -2 2.5; -2.5 -2.5; 3 -3];
d = [-1; -1; -1; -1; 1; 1; 1; 1];

% Visualization in 2D - only data points with class distinction
figure;
hold on;

% Plot data points for Class 1 (d = 1) and Class -1 (d = -1)
plot(X(d==1,1), X(d==1,2), 'b*', 'MarkerSize', 10, 'LineWidth', 2); % Class 1 (blue)
plot(X(d==-1,1), X(d==-1,2), 'ro', 'MarkerSize', 10, 'LineWidth', 2); % Class -1 (red)

% Label plot
%title('Circularly Separable Data Points');
xlabel('X_1');
ylabel('X_2');

% Legend with distinct markers
legend({'Class 1 (d=1)', 'Class -1 (d=-1)'}, ...
       'Location', 'SouthOutside', 'Orientation', 'Horizontal');

grid on;
hold off;
```

Output of the code is shown in Figure 4.2a.

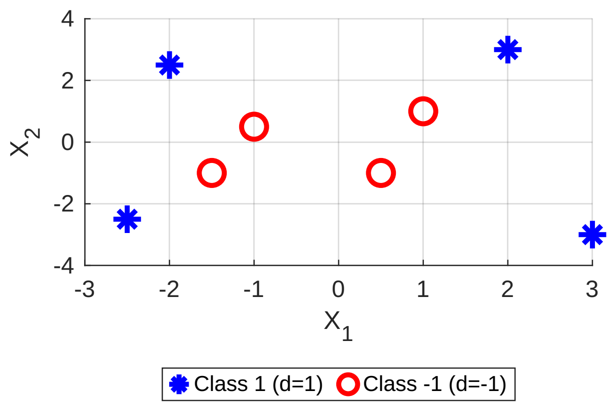


Figure 1.3: Distribution of data points in the dataset

These data points are not linearly separable but circularly separable.One way of solving this problem is trying different explicit feature mapping. So we think about the non-linear kernel that transform the data points into higher dimension and then try Support Vector classifier for better classification.

1.4 Introduction to Polynomial Kernel Transformation

In scenarios where a linear kernel fails to provide an adequate classification model, it becomes essential to explore transformation techniques that enable the separation of data points in a higher-dimensional space. One of the simplest yet most effective transformations is the quadratic kernel transformation.

1.4.1 Polynomial kernel definition

The polynomial kernel of degree 2 can be mathematically represented in inner product form as follows:

$$K(x, x') = (x^T x' + 1)^2$$

where x and x' are two data points in the input space. This kernel function allows us to project our original 2D points into a higher-dimensional space. Specifically, for degree 2, the kernel transformation yields the following new features:

- x_1^2
- x_2^2
- $\sqrt{2}x_1x_2$

Thus, a point (x_1, x_2) in 2D is transformed to a point in 3D as follows:

$$\Phi(x) = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix}$$

1.4.2 Transformation of data points

Let's apply this transformation to the original data points given in the previous table.

Applying the transformations, we create a new table with transformed columns for x_1^2, x_2^2, and $\sqrt{2}x_1x_2$:

i	d	x_1^2	x_2^2	$\sqrt{2}x_1x_2$
1	-1	1	1	$\sqrt{2}$
2	-1	1	0.25	$-\frac{\sqrt{2}}{2}$
3	-1	0.25	1	$-\frac{\sqrt{2}}{2}$
4	-1	2.25	1	$1.5\sqrt{2}$
5	1	4	9	$3\sqrt{2}$
6	1	4	6.25	$-5\sqrt{2}$
7	1	6.25	6.25	$12.5\sqrt{2}$
8	1	9	9	$-9\sqrt{2}$

Table 1.2: Transformed Data Points

The transformed data table now contains the new features generated by the polynomial kernel transformation, allowing us to use these higher-dimensional representations for classification tasks with Support Vector Machines (SVM).

A 3D scatter plot showing the distribution of this transformed data is shown in Figure 1.4.

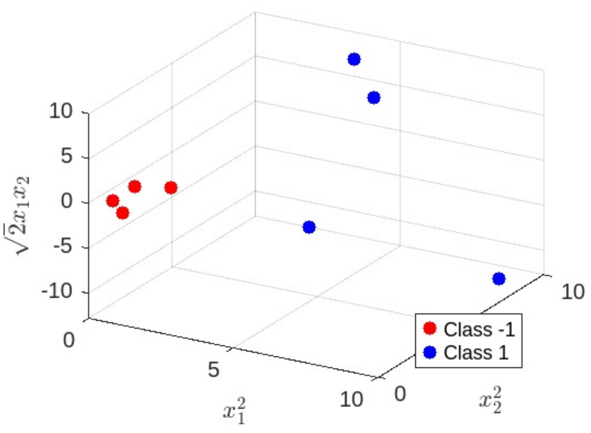


Figure 1.4: Transformed Data Points in 3D using Polynomial Kernel

By representing the data in this manner, we can effectively handle non-linear decision boundaries that separate classes in the original 2D space.
These points are now linearly separable. We can find a hyperplane separating these two groups of points.

1.5 Hard Margin SVM on Higher-Dimensional Data Using Primal Form

In the context of Support Vector Machines (SVM), the primal problem aims to find the optimal separating hyperplane that maximizes the margin between the two classes. For the transformed data using a polynomial kernel, the primal formulation can be expressed as follows:
Objective Function:
The primal objective function to minimize the following is:

$$\min_{\mathbf{w},\gamma} \quad \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.2}$$

Subject to:
For all data points, the constraints are given by:

$$d_i(\mathbf{w}^T\phi(x_i) - \gamma) \geq 1, \quad \forall i \tag{1.3}$$

Where:

- $\mathbf{w}$ is the weight vector,
- γ is the bias term, representing the offset from the origin,
- $\phi(x_i)$ is the transformation of the data point x_i.

1.5.1 Reformulating in matrix form

Define the Weight Vector w: The weight vector **w** is formed from the contributions of the transformed data points.
Combine the Constraints: The constraints can be expressed in matrix form as:

$$D(\mathbf{W}^T\Phi(X) - \gamma) \geq \mathbf{1} \tag{1.4}$$

Where $D = \text{diag}(d)$, $\Phi(X)$ is the matrix containing transformed data points, and **1** is the vector of ones.

1.5.2 Connection to the dual problem

The dual problem arises from the Lagrangian formulation of the primal problem. By introducing Lagrange multipliers $u_i \geq 0$ for each constraint, the Lagrangian $\mathscr{L}$ is formed as:

$$\mathscr{L}(\mathbf{w}, \gamma, \mathbf{u}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} u_i \left[d_i(\mathbf{w}^T \phi(x_i) - \gamma) - 1 \right] \tag{1.5}$$

1.5.3 Deriving the dual problem

To derive the dual, we take the gradients of $\mathscr{L}$ with respect to $\mathbf{w}$ and γ and set them to zero:
Gradient with respect to w:

$$\nabla_{\mathbf{w}}\mathscr{L} = \mathbf{w} - \sum_{i=1}^{N} u_i d_i \phi(x_i) = 0 \implies \mathbf{w} = \sum_{i=1}^{N} u_i d_i \phi(x_i) \tag{1.6}$$

Gradient with respect to γ:

$$\nabla_{\gamma}\mathscr{L} = -\sum_{i=1}^{N} u_i d_i = 0 \implies \sum_{i=1}^{N} u_i d_i = 0 \tag{1.7}$$

Substituting these into the Lagrangian gives us the dual problem, which we already derived in the previous section.
Here is the CVX code to solve the hard margin SVM problem using the primal form with transformed data points.

```
close(gcf)
% data points
col1=[1 -1  0.5 -1.5 2 -2  -2.5  3]';% X1 oordinates
col2=[1 0.5 -1  -1   3 2.5 -2.5 -3]';% X2 oordinates
d=[-1 -1 -1 -1 1 1 1 1]';% class values
A=[col1.^2  col2.^2   sqrt(2)*col1.*col2];% mapping
e=ones(8,1);
cvx_begin quiet
variable w(3);
variable gama;
dual variable u;
minimize 0.5*(w'*w)
subject to
u:d.*(A*w-gama*e)>=e; % u: tells CVX to store u also
cvx_end

format bank
disp('w vector')
disp(w)
disp('gamma')
disp(gama)
disp('u: lagrangian multipliers')
disp(u)
```

Output of the code is shown below.

```
w vector
          0.16
          0.29
         -0.02
```

```
gamma
          1.60
u: lagrangian multipliers
          0.00
          0.00
          0.00
          0.05
          0.00
          0.02
          0.03
          0.00
```

From the output, it is clear that there is only three support vectors- fourth, sixth and seventh data points. The seperating hyperplane is

$$w = 0.16x_1^2 + 0.29x_2^2 - 0.02\sqrt{2}x_1x_2 - 1.60$$

Decision boundary of the classifier is shown in Figure 1.5. `Matlab` code creating this plot is shown below.

```
%plotting the boundary
close(gcf)
w=[0.16  0.29  -0.02]';
gama=1.6;
x1=-4:.1:4;
x2=x1;
a=sqrt(2);
[X1 X2]=meshgrid(x1,x2);
Z=sign(w(1)*X1.^2+w(2)*X2.^2+w(3)*a*X1.*X2-gama);
surf(X1,X2,Z);
```

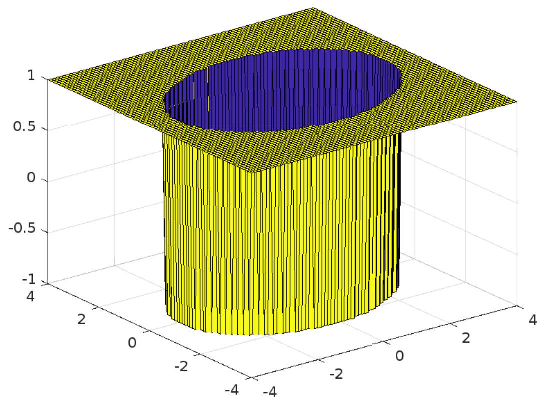


Figure 1.5: Classification boundary of the SVM with polynomial kernel

Finally let's look into the real picture of our quadratic kernel classifier. The classification boundary along with the data points is shown in Figure 1.6.

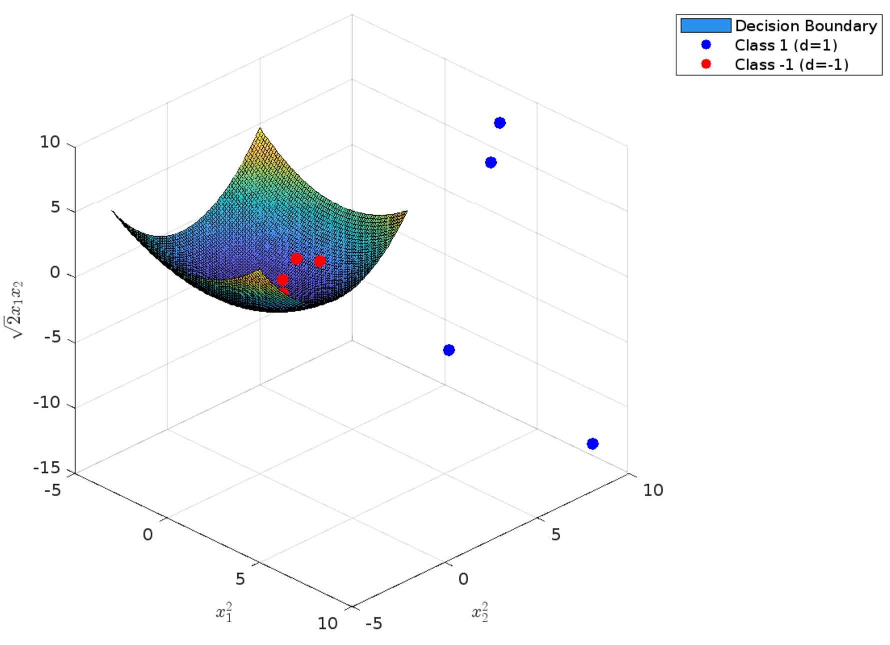


Figure 1.6: Skill of the SVM Classifier with quadratic polynomial kernel

Matrix Form of the Minimization Version of the Dual Problem

The dual problem can be expressed as follows:

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}u^T Q u - C\mathbf{1}^T u \\ \text{s.t.} \quad & u^T \mathbf{d} = 0 \\ & u \geq 0 \end{aligned}$$

Where:

- u is the vector of Lagrange multipliers, $u = [u_1, u_2, \dots, u_n]^T$.
- Q is the positive semi-definite matrix defined by the kernel function:

  $$Q_{ij} = d_i d_j K(x_i, x_j)$$

  with $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, representing the inner product of the transformed feature vectors.
- C is the regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.
- $\mathbf{d} = [d_1, d_2, \dots, d_n]^T$ is the vector of class labels.
- $\mathbf{1}$ is a vector of ones.

In this formulation, the term $-C\mathbf{1}^T u$ introduces a regularization factor, allowing for the adjustment of the penalty for misclassified points. The constraint $u^T\mathbf{d} = 0$ ensures that the Lagrange multipliers satisfy the KKT conditions, leading to a feasible solution.
Now the dual problem can be solved using `CVX` solver. Code for this task is given below.

```
% Hard-margin kernel SVM
% Data points
col1 = [1 -1  0.5 -1.5 2 -2  -2.5  3]'; % X1 coordinates
col2 = [1 0.5 -1  -1   3 2.5 -2.5 -3]'; % X2 coordinates
d = [-1 -1 -1 -1 1 1 1 1]'; % Class values
A = [col1 col2];
X = A * A';
K = X.^2; % Polynomial kernel K(x_i, x_j) = (x_i' * x_j)^2

% Prepare quadratic programming
Q = d * d' .* K;
e = ones(8, 1);

% Solve using CVX
cvx_begin quiet
    variable u(8);
    minimize (0.5 * u' * Q * u - e' * u)
    subject to
        u' * d == 0
        u >= 0
cvx_end

format bank
disp('u vector')
disp(u)

% Find support vectors
support_vector_indices = find(u > 1e-5); % Tolerance for numerical
   stability
support_vectors = A(support_vector_indices, :);
u_sv = u(support_vector_indices);
d_sv = d(support_vector_indices);

% Calculate bias term b using support vectors
b = sum(d_sv(1) - sum(u_sv .* d_sv' .* (support_vectors *
   support_vectors(1,:)').^2))/3;

% Initialize predictions for original data points
predictions = zeros(size(d));

% Calculate predictions for original data points
for i = 1:length(d)
    x_test = A(i, :)'; % Original data point
    K_test = (support_vectors * x_test).^2; % Kernel transformation
        for test point
    decisionValue = sum(u_sv .* d_sv .* K_test) + b; % Decision
       function with bias
    predictions(i) = decisionValue; % Store the decision value
end

% Classify based on threshold (0)
predictions = predictions > 0; % Convert to binary: true/false
predictions = 2 * predictions - 1; % Convert binary to -1, 1

% Calculate accuracy
accuracy = sum(predictions == d) / length(d);
disp('Classification Accuracy:')
disp(accuracy)
```

Output of the code is shown below.

```
u vector
          0.00
          0.00
          0.00
          0.05
          0.00
          0.02
          0.03
          0.00
Classification Accuracy:
          1.00
```

Note that the support vectors found using both primal and dual are same. Also we found the skill of the classifier in terms of accuracy. It tells how much (percentage) of the data points are correctly classified. Here the hard margin support vector machine with polynomial kernel gave 100% accuracy in classification of circular separable dataset.

1.5.4 Intuition behind non-linear kernels

Non-linear kernels enable SVMs to effectively manage non-linearly separable data by transforming the input space. Imagine two classes of points wrapped around a circle; projecting these points into three-dimensional space allows a flat plane to separate them, which would not be possible in the original two-dimensional view.

1.5.5 Useful kernels in SVM

Commonly used non-linear kernels include:

- **Polynomial Kernel:** $K(x_i, x_j) = (x_i^T x_j + 1)^d$, where d is the degree of the polynomial.
- **Radial Basis Function (RBF) Kernel:** $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$, where γ is a parameter that controls the spread of the kernel.
- **Sigmoid Kernel:** $K(x_i, x_j) = \tanh(\alpha x_i^T x_j + \beta)$, where α and β are kernel parameters.

These kernels are effective in different contexts. For instance, the RBF kernel is versatile and performs well in most cases, while the polynomial kernel can capture polynomial relationships in data.

1.6 Implementation of Non-linear SVMs in Matlab

To explore the performance of non-linear kernels, we will implement SVMs with polynomial and RBF kernels using matlab. The code below evaluates both kernels on the generated dataset.

Listing 1.1: Non-linear SVM implementation

```
% Non-linear SVM implementation
kernels = {'polynomial', 'RBF'};
```

```
best_accuracy = 0;
best_kernel = '';

for i = 1:length(kernels)
    kernel = kernels{i};

    % Define SVM model with the chosen kernel
    model = fitcsvm(X, d, 'KernelFunction', kernel, 'Standardize', true, 'ClassNames', [-1, 1

    % Cross-validation to evaluate the model
    CVSVMModel = crossval(model);
    accuracy = 1 - kfoldLoss(CVSVMModel);

    fprintf('%s␣Kernel:\n', kernel);
    fprintf('Best␣Accuracy:␣%.2f%%\n', accuracy * 100);

    % Update the best model
    if accuracy > best_accuracy
        best_accuracy = accuracy;
        best_kernel = kernel;
    end
end

fprintf('Best␣Kernel␣Model:␣%s␣with␣Accuracy:␣%.2f%%\n', best_kernel, best_accuracy * 100);
```

Here we are not using `CVX` solver to solve the mathematical model (LPP version) of SVM, instead use the built-in Machine Learning libraries in `matlab`.
Output of the code is shown below.

```
Warning: The number of folds K is greater than the number of observations with non-missing GR
polynomial Kernel:
Best Accuracy: 50.00%

Warning: The number of folds K is greater than the number of observations with non-missing GR

RBF Kernel:
Best Accuracy: 12.50%
```

Since the ML algorithms follows some standard procedures for their classification tasks like train-test split, k-fold cross validation, model evaluation on test dataset etc, we need sufficiently large dataset to get better results with these ready to use ML algorithms. Anyway we can say that polynomial kernel is best suit to our dataset.
In summary, we explored the capabilities of SVMs, beginning with linear kernels and then transitioning to non-linear kernels for complex data distributions. The introduction of non-linear kernels allows SVMs to uncover intricate patterns that linear classifiers may overlook, enhancing classification performance.
By applying polynomial kernel, we observed a significant improvement in accuracy over the linear kernel, illustrating the importance of kernel selection in SVMs. The final choice of the kernel depends on the underlying data distribution, demonstrating that rigorous evaluation and comparison of models are vital in the machine learning process.
Ultimately, the best-fit model for a given dataset is contingent upon its distribution and the kernel function employed, underscoring the necessity of exploring advanced kernel methods to optimize classification accuracy in machine learning.

1.7 Tasks

1. Generate data from two concentric discs and find a binary classifier that separate the circularly separable data.

SOLUTION

This task is solved in two stages. In the first stage data points are generated from two concentric discs as mentioned in the problem statement. data points (x, y) is created with the polar formula:

$$x = r\cos(\theta) \tag{1.8}$$

$$y = r\sin(\theta) \tag{1.9}$$

For concentric discs, this solution used data points with two radii, $r = 1$ and $r = 2$. A suitable randomization is introduced to make the point randomly distributed within the discs. Points in Yellow disc is labeled as class 1 and that in Blue disc is labeled as class -1. Matlab code for data generation and its visualization is given below.

```matlab
% Set random seed for reproducibility
rng(1);

% Parameters for disc generation
N = 100; % Number of points per class
r_inner = 1; % Radius of inner disc (yellow class)
r_outer = 3; % Radius of outer disc (blue class)

% Generate points for inner disc (Class -1, yellow)
theta_inner = 2 * pi * rand(N, 1);
radii_inner = r_inner * sqrt(rand(N, 1)); % sqrt to ensure
    uniform distribution in area
x_inner = radii_inner .* cos(theta_inner);
y_inner = radii_inner .* sin(theta_inner);
class_inner = -1 * ones(N, 1); % Class label for inner points

% Generate points for outer disc (Class 1, blue)
theta_outer = 2 * pi * rand(N, 1);
radii_outer = r_inner + (r_outer - r_inner) * sqrt(rand(N, 1));
x_outer = radii_outer .* cos(theta_outer);
y_outer = radii_outer .* sin(theta_outer);
class_outer = ones(N, 1); % Class label for outer points

% Combine data points and labels
x_data = [x_inner; x_outer];
y_data = [y_inner; y_outer];
class_labels = [class_inner; class_outer];

% Plot the generated data
figure;
hold on;
scatter(x_inner, y_inner, 'filled', 'MarkerFaceColor', 'yellow'
    , 'DisplayName', 'Class -1 (Yellow)');
scatter(x_outer, y_outer, 'filled', 'MarkerFaceColor', 'blue',
    'DisplayName', 'Class 1 (Blue)');
```

```
xlabel('x');
ylabel('y');
title('Generated Data from Two Concentric Discs');
legend show;
axis equal;
hold off;
```

The data generated using the previous code is shown in Figure 1.7

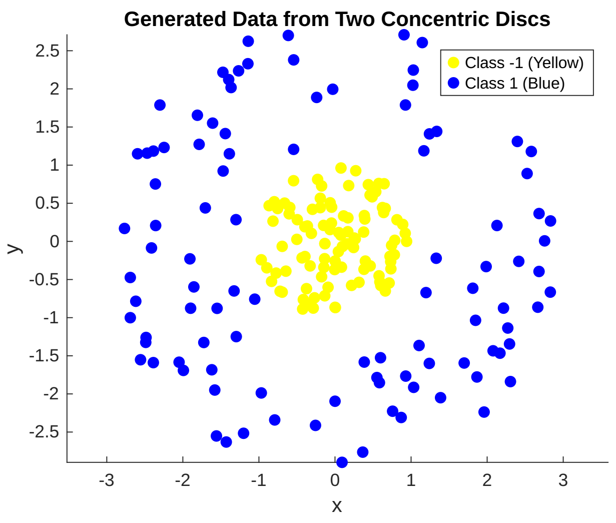


Figure 1.7: Synthetic data from two concentric disc for binary classification

In second stage, the associated LPP for hard margin support machines algorithm is implemented with CVX solvers. Matlab code for the complete task is given below.

```
% Combine data points and labels
x_data = [x_inner; x_outer];
y_data = [y_inner; y_outer];
class_labels = [class_inner; class_outer];

% Mapping the data to a higher dimension (quadratic feature
    mapping)
A = [x_data.^2, y_data.^2, sqrt(2)*x_data.*y_data]; % Quadratic
    feature transformation
d = class_labels;
e = ones(2*N, 1);

% Solve SVM using CVX
cvx_begin quiet
    variable w(3);
    variable gama;
    dual variable u;
    minimize 0.5*(w'*w)
    subject to
        u : d .* (A*w - gama*e) >= e; % u: tells CVX to store
            dual variable u
cvx_end

% Display results
```

```
format bank;
disp('w vector');
disp(w);
disp('gama');
disp(gama);
disp('u: Lagrange multipliers');
disp(u);

% Classification boundary and plotting
x1_range = linspace(-r_outer, r_outer, 100);
x2_range = linspace(-r_outer, r_outer, 100);
[X1, X2] = meshgrid(x1_range, x2_range);
Z = w(1)*X1.^2 + w(2)*X2.^2 + w(3)*sqrt(2)*X1.*X2 - gama;

% Plot the classification boundary and data points
figure;
hold on;
contour(X1, X2, Z, [0 0], 'k', 'LineWidth', 2, 'DisplayName', '
    Decision Boundary');
scatter(x_inner, y_inner, 'filled', 'MarkerFaceColor', 'yellow'
    , 'DisplayName', 'Class -1 (Yellow)');
scatter(x_outer, y_outer, 'filled', 'MarkerFaceColor', 'blue',
    'DisplayName', 'Class 1 (Blue)');
xlabel('x');
ylabel('y');
%title('SVM with Quadratic Kernel: Classification of Concentric
     Discs');
legend show;
axis equal;
hold off;
```

Output of the above code is shown below.

```
w vector
          2.81
          2.80
          0.08

gama
          3.83
```

From the output, the classification boundary is:

$$w = 2.81x^2 + 2.80y^2 + 0.11xy - 3.83l \tag{1.10}$$

Even though the data is transformed into a higher-dimensional feature space using quadratic feature mapping, the classification boundary is visualized in the original 2D input space. The SVM uses a polynomial kernel (in this case, quadratic) to map the data to a higher-dimensional feature space. However, the decision boundary is still expressed in terms of the original 2D coordinates, meaning the plot shows how the boundary separates the original (x, y) data points. The equation of the boundary is drawn over the 2D plane (i.e., (x, y) as a contour plot. The classification boundary represented in (1.10) and the data points are shown in Figure 1.8.

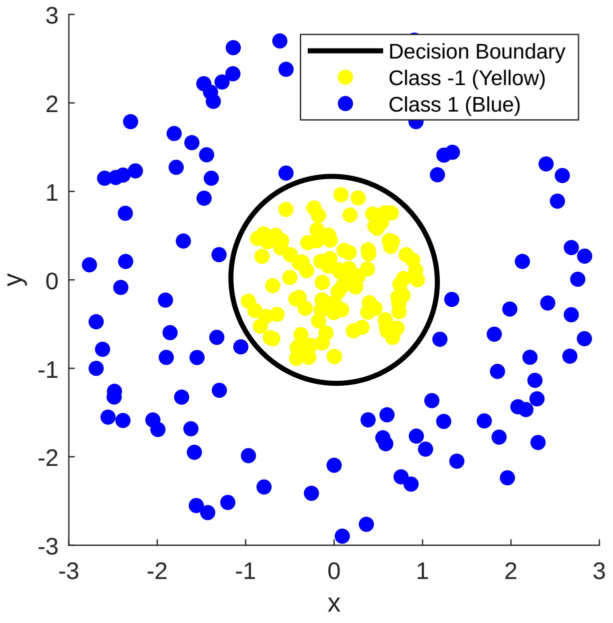


Figure 1.8: Classification boundary for circularly separable data in task 1

2. Generate data from three concentric spheres and find three binary classifiers using kernel methods (one versus rest approach).

SOLUTION

This task is solved in three stages.

Stage 1: Create data points which represents points in 3-space and are representatives from three concentric circles. For this purpose the spherical coordinate system is used. The rules for (x, y, z) is given by:

$$x = \rho \cos\theta \sin\phi \tag{1.11}$$
$$y = \rho \sin\theta \sin\phi \tag{1.12}$$
$$z = \rho \cos\phi \tag{1.13}$$

In this task we generated $3 \times 100 = 300$ data points from three concentric spheres with radii 1, 2and 3 respectively. Distribution of the data points is shown in Figure 1.9.

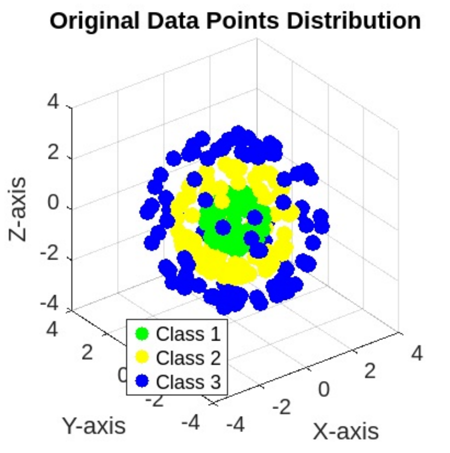


Figure 1.9: Distribution of data points from three concentric circles

Stage 2: Apply the RBF kernel to transform the data to make SVM successful. Since it is a multi class (3 class) classification problem, one-verses- the rest method is used for classification task. The one-versus-rest (OvR) classification approach is a popular strategy for multi-class classification problems, where a single classifier is trained for each class to distinguish it from all other classes combined. In the context of the concentric spheres classification problem, we employ OvR by formulating three separate binary classification tasks using Support Vector Machines (SVMs). Each SVM is trained to classify data points from one specific class while treating all other classes as a single negative class. This involves using the CVX solver to optimize the dual formulation of the SVM problem, where we aim to maximize the margin between the classes while incorporating a regularization term to prevent over fitting. The resulting optimization yields Lagrange multipliers (denoted as u_i) that identify the support vectors, which are crucial data points lying closest to the decision boundary. By employing the Radial Basis Function (RBF) kernel, we can effectively capture the non-linear decision boundaries between the concentric spheres, facilitating accurate classification of new data points based on the learned decision functions. This approach allows for a systematic and mathematically robust method to tackle multi-class problems in a scalable manner.

`Matlab` code for the entire task in stage 2 is given below.

```
N = 100;
radii = [1, 2, 3];
sigma = 1.0;
theta1 = pi * rand(N, 1);
phi1 = 2 * pi * rand(N, 1);
x1 = radii(1) * sin(theta1) .* cos(phi1);
y1 = radii(1) * sin(theta1) .* sin(phi1);
z1 = radii(1) * cos(theta1);
class1 = ones(N, 1);
theta2 = pi * rand(N, 1);
phi2 = 2 * pi * rand(N, 1);
x2 = radii(2) * sin(theta2) .* cos(phi2);
y2 = radii(2) * sin(theta2) .* sin(phi2);
z2 = radii(2) * cos(theta2);
class2 = 2 * ones(N, 1);
theta3 = pi * rand(N, 1);
phi3 = 2 * pi * rand(N, 1);
x3 = radii(3) * sin(theta3) .* cos(phi3);
y3 = radii(3) * sin(theta3) .* sin(phi3);
z3 = radii(3) * cos(theta3);
class3 = 3 * ones(N, 1);
X = [x1, y1, z1; x2, y2, z2; x3, y3, z3];
y = [class1; class2; class3];
N_total = 3 * N;

y1_binary = (y == 1) * 2 - 1;  % Class 1 vs Rest
y2_binary = (y == 2) * 2 - 1;  % Class 2 vs Rest
y3_binary = (y == 3) * 2 - 1;  % Class 3 vs Rest
K = @(X1, X2) exp(-pdist2(X1, X2, 'euclidean').^2 / (2 *
    sigma^2));
K_matrix = K(X, X);
C = 1;
u1 = zeros(N_total, 1);  % Lagrange multipliers for Class 1
u2 = zeros(N_total, 1);  % Lagrange multipliers for Class 2
```

```
u3 = zeros(N_total, 1);  % Lagrange multipliers for Class 3
gama1 = 0;
gama2 = 0;
gama3 = 0;

cvx_begin quiet
    variables u1(N_total);
    variable gama1;
    maximize( sum(u1) - 0.5 * quad_form(y1_binary .* u1,
       K_matrix) );
    subject to
        0 <= u1 <= C;
        sum(y1_binary .* u1) == 0;
cvx_end

% SVM for Class 2 vs Rest
cvx_begin quiet
    variables u2(N_total);
    variable gama2;
    maximize( sum(u2) - 0.5 * quad_form(y2_binary .* u2,
       K_matrix) );
    subject to
        0 <= u2 <= C;
        sum(y2_binary .* u2) == 0;
cvx_end

% SVM for Class 3 vs Rest
cvx_begin quiet
    variables u3(N_total);
    variable gama3;
    maximize( sum(u3) - 0.5 * quad_form(y3_binary .* u3,
       K_matrix) );
    subject to
        0 <= u3 <= C;
        sum(y3_binary .* u3) == 0;
cvx_end

f1 = sum((u1 .* y1_binary)' .* K_matrix, 2) - gama1;
f2 = sum((u2 .* y2_binary)' .* K_matrix, 2) - gama2;
f3 = sum((u3 .* y3_binary)' .* K_matrix, 2) - gama3;
[~, predicted_class_train] = max([f1, f2, f3], [], 2);
correct_predictions = sum(predicted_class_train == y);
accuracy = (correct_predictions / N_total) * 100;
disp(['Classification Accuracy: ', num2str(accuracy), '%'])
   ;
w1 = (u1 .* y1_binary)' * K_matrix;
w2 = (u2 .* y2_binary)' * K_matrix;
w3 = (u3 .* y3_binary)' * K_matrix;
fprintf('Class 1: w1 = [%s], u1 = [%s], gama1 = %.4f\n',
   num2str(w1), num2str(u1), gama1);
fprintf('Class 2: w2 = [%s], u2 = [%s], gama2 = %.4f\n',
   num2str(w2), num2str(u2), gama2);
```

```matlab
fprintf('Class 3: w3 = [%s], u3 = [%s], gama3 = %.4f\n',
    num2str(w3), num2str(u3), gama3);
[~, predicted_class] = max([f1, f2, f3], [], 2);
grid_points = 20;  % Number of grid points for each
    dimension
[xg, yg, zg] = meshgrid(linspace(-4, 4, grid_points),
    linspace(-4, 4, grid_points), linspace(-4, 4,
    grid_points));
X_grid = [xg(:), yg(:), zg(:)];
K_grid = K(X_grid, X);
f1_grid = sum((u1 .* y1_binary)' .* K_grid, 2) - gama1;
f2_grid = sum((u2 .* y2_binary)' .* K_grid, 2) - gama2;
f3_grid = sum((u3 .* y3_binary)' .* K_grid, 2) - gama3;
[~, predicted_class_grid] = max([f1_grid, f2_grid, f3_grid
    ], [], 2);
predicted_class_grid = reshape(predicted_class_grid, [
    grid_points, grid_points, grid_points]);
figure;
hold on;
scatter3(X(:,1), X(:,2), X(:,3), 50, predicted_class, '
    filled');
p1 = patch(isosurface(xg, yg, zg, predicted_class_grid ==
    1));
set(p1, 'FaceColor', 'yellow', 'FaceAlpha', 0.2, 'EdgeColor
    ', 'none');
p2 = patch(isosurface(xg, yg, zg, predicted_class_grid ==
    2));
set(p2, 'FaceColor', 'blue', 'FaceAlpha', 0.2, 'EdgeColor',
     'none');
p3 = patch(isosurface(xg, yg, zg, predicted_class_grid ==
    3));
set(p3, 'FaceColor', 'green', 'FaceAlpha', 0.2, 'EdgeColor'
    , 'none');

axis equal;
xlim([-4 4]);
ylim([-4 4]);
zlim([-4 4]);
view(3);
grid on;
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
%title('3D SVM with Support Vectors');
legend('Class 1', 'Class 2', 'Class 3', 'Location', 'best')
    ;
hold off;
```

A 3D visualization of the support vector machines classification task with RBF kernel is shown in Figure 1.10. For better visual feeling the division boundaries are created with intersecting classification boundaries of separating surfaces (in contour form).

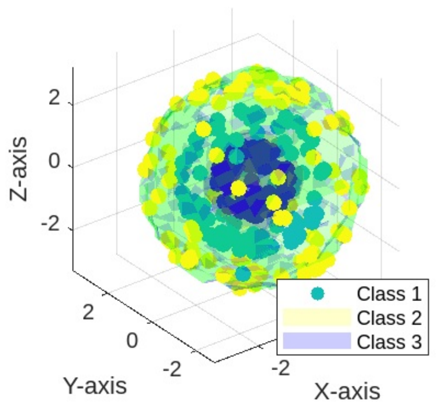
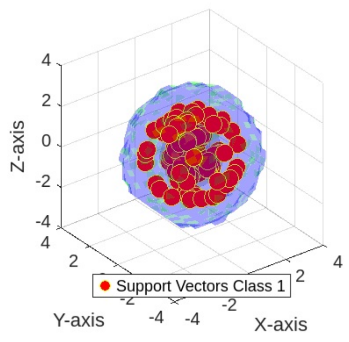
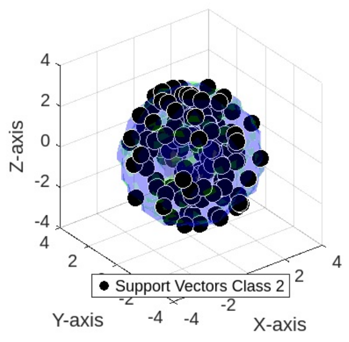


Figure 1.10: Classification boundaries of SVM with RBF kernel on spherically separable data

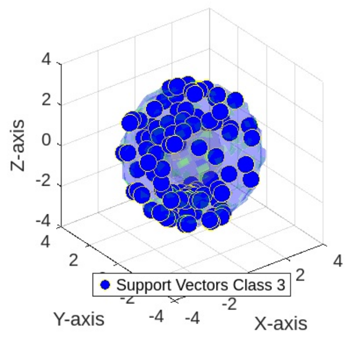
Support vectors in each class is demonstrated in Figure 1.11



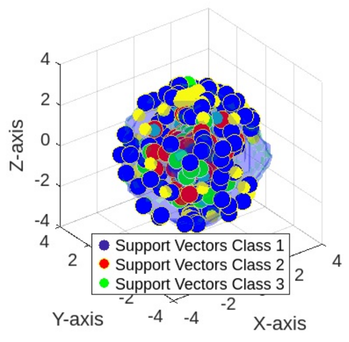
(a) Support Vectors of Class 1



(b) Support Vectors of Class 2



(c) Support Vectors of Class 3



(d) All support vectors

Figure 1.11: Visualization of Support vectors in SVM with RBF kernel on spherically separable dataset

Stage 3: `Matlab` code contains functions to display w, u and γ. Since the number of data points is 300 in this case, listing of these values is not that much useful. In addition to the usual steps in algebraic solution, the classification accuracy is also calculated. Classification accuracy of the SVM classification model with RBF kernel on this data is 94.7%.

RESULTS

- Mathematical and computational models of both linear and non-linear SVM with `CVX` solver are revisited.

- Linear and non-linear SVMs with various kernels are experimented using `CVX` solvers.
- Lagrange multiplier is reviewed as the indicator for identifying support vectors in a SVM classification problem.
- Established the power of algebraic approach in development of a complete design of SVM classifier.

2 | Assignment 66 Linear Programming With Bounds on Variables

2.1 Introduction

The standard algorithm for solving LP is now ADMM. But ADMM algorithm requires LP formulation in a standard format. This format is as follows

$$\begin{aligned} \underset{X}{\text{Min}} \quad & c^T X \\ \text{Subject to:} \quad & Ax = b \\ & X \geq 0 \end{aligned}$$

LP in general may contain all type of constraints. Also, variables may be constrained to assume a value between a lower and upper limit. This note and assignment is about Converting non-standard LPs into standard format for applying ADMM We use the concept of slack and surplus variable to achieve the task. With CVX we will demonstrate that the solution is same for both original formulation and the converted formulation. Consider the example

$$\begin{aligned} \min_{x} \quad & z = x_1 + 2x_2 \\ \text{Subject to:} \quad & x_1 - x_2 \leq 3 \\ & x_1 + 2x_2 \geq 6 \\ & 3x_1 - 4x_2 \leq 10 \\ & 4 \leq x \leq 20 \\ & 2 \leq x \leq 22 \end{aligned}$$

Adding necessary slack and suplus variables, the problem can be written in standard form as:

$$\begin{aligned} \min_{x} \quad & z = x_1 + 2x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 \\ \text{Subject to:} \quad & x_1 - x_2 + x_3 = 3 \\ & x_1 + 2x_2 - x_4 = 6 \\ & 3x_1 - 4x_2 + 0x_5 = 10 \\ & x_1 - x_6 = 4 \\ & x_1 + x_7 = \leq 20 \\ & x_2 - x_8 = 2 \\ & x_2 + x_9 = 22 \\ & x_i \geq 0 \quad \forall i \end{aligned}$$

Now write coefficient matrix, A, constant column b and cost vector c from this standard form. `CVX` code to solve this LPP in both symbolic and matrix apporach is given below.

```
cvx_begin quiet
variable x1
variable x2
minimize (x1+2*x2)
subject to
x1-x2<=3
x1+2*x2>=6
3*x1-4*x2<=10
4<=x1<=20
2<=x2<=22
cvx_end
x1
x2
x1+2*x2
```

Output of the code is shown below.

```
Solution:
          4.00
          2.00
Optimum value
          8.00
```

Solution for LP formulation in standard form is shown below.

```
c=[1 2 0 0 0 0 0 0 0]';
A=[1 -1 1 0 0 0 0 0 0;
    1 2 0 -1 0 0 0 0 0;
    3 -4 0 0 1 0 0 0 0;
    1 0 0 0 0 -1 0 0 0;
    1 0 0 0 0 0 1 0 0;
    0 1 0 0 0 0 0 -1 0;
    0 1 0 0 0 0 0 0 1];
b=[3 6 10 4 20 2 22]';
cvx_begin quiet
variable x(9)
minimize (c'*x)
subject to
A*x==b
x>=0
cvx_end
disp('Solution:');
disp(x);
disp('Optimum value:');
disp(c'*x);
```

Output of the code is shown below.

```
Solution:
          4.00
          2.00
          1.00
          2.00
          6.00
```

```
           0.00
          16.00
           0.00
          20.00
 Optimum value:

           8.00
```

Both the approaches produce the same solution.

2.2 Task

1. Solve after converting into standard format.

$$\begin{aligned} &\max 4x_1 + 3x_2 \\ \text{S.to:}\quad & x_1 + x_2 \le 6 \\ & 2x_1 + x_2 \le 8 \\ & x_1 \le 1 \\ & 1 \le x_2 \le 3 \end{aligned}$$

SOLUTION

The given problem will be written in the standard form and then convert to matrix format. `Matlab` code to solve the problem using matrix form is given below.

```
c=[4 3 0 0 0 0 0]';
A=[1 1 1 0 0 0 0; 2 1 0 1 0 0 0; 1 0 0 0 1 0 0;
    0 1 0 0 0 -1 0;
    0 1 0 0 0 0 1];
b=[6 8 1 1 3]';
cvx_begin quiet
    variable x(7)
    minimize (c'*x)
    subject to
    A*x==b
    x>=0
cvx_end
disp('solution:');
disp(x);
disp('Optimum solution:');
disp(c'*x);
```

Output of the code is shown below.

```
solution:
           0.00
           1.00
           5.00
           7.00
           1.00
           0.00
           2.00
Optimum solution:
           3.00
```

2. Solve directly using CVX. Give the matlab code.

$$\begin{aligned}
\text{Maximize}\quad & x_1 + x_2 + 2x_3 - 2x_4 \\
\text{S.to:}\quad & x_1 + 2x_3 \le 700 \\
& 2x_2 - 8x_3 \le 0 \\
& x_2 - 2x_3 + x_4 \ge 1 \\
& x_1 + x_2 + x_3 + x_4 = 10 \\
& 0 \le x_i \le 10 \quad \forall i
\end{aligned}$$

SOLUTION

Matlab code for this task is given below.

```
cvx_begin quiet
variable x1
variable x2
variable x3
variable x4
maximize (x1+x2+2*x3-2*x4)
subject to
x1+2*x3<=700
2*x2-8*x3<=0
x2-2*x3+x4>=1
x1+x2+x3+x4==10
0<=x1<=10
0<=x2<=10
0<=x3<=10
0<=x4<=10
cvx_end
disp('Solution:')
disp(x1);
disp(x2);
disp(x3);
disp(x4);
disp('Optimum solution:');
disp(x1+x2+2*x3-2*x4);
```

output of the code is shown below.

```
Solution:
          0.00
          7.00
          3.00
          0.00
Optimum solution:
         13.00
```

3. Solve the LPP

$$\begin{aligned} \text{Maximize} \quad & 3x_1 + 5x_2 + 2x_3 \\ \text{S.to:} \quad & x_1 + 2x_2 + 2x_3 \leq 10 \\ & 2x_1 + 4x_2 + 3x_3 \leq 15 \\ & 0 \leq x_1 \leq 4 \\ & 0 \leq x_2 \leq 3 \\ & 0 \leq x_3 \leq 3 \end{aligned}$$

SOLUTION

`Matlab` code for this task is given below.

```
cvx_begin quiet
variable x1
variable x2
variable x3

maximize (3*x1+5*x2+2*x3)
subject to
x1+2*x2+2*x3<=10
2*x1+4*x2+3*x3<=15
0<=x1<=4
0<=x2<=3
0<=x3<=3
cvx_end
disp('Solution:')
disp(x1);
disp(x2);
disp(x3);
disp('Optimum solution');
disp(3*x1+5*x2+2*x3)
```

Output of the code is shown below.

```
Solution:
          4.00
          1.75
          0.00
Optimum solution
         20.75
```

4. Solve the LPP,

$$\begin{aligned} \text{maximize:} \quad & -13x_1 + 10x_2 + 8x_3 - 10x_4 - 22x_5 \\ \text{S.to:} \quad & -2x_1 + x_2 + 4x_3 + -2x_4 - 2x_5 \leq 5 \\ & -4x_1 + 2x_2 + x_3 - 2x_4 - 6x_5 \leq 13 \\ & 0 \leq x_1 \leq 1 \\ & 0 \leq x_2 \leq 20 \\ & 0 \leq x_3 \leq 4 \\ & 0 \leq x_4 \leq 3 \\ & 0 \leq x_5 \leq 5 \end{aligned}$$

SOLUTION

Matlab code for the given task is given below.

```
cvx_begin quiet
variable x1
variable x2
variable x3
variable x4
variable x5
maximize (-13*x1+10*x2+8*x3-10*x4-22*x5)
subject to
-2*x1+x2+4*x3-2*x4-2*x5<=5
-4*x1+2*x2+x3-2*x4-6*x5<=13
0<=x1<=1
0<=x2<=20
0<=x3<=4
0<=x4<=3
0<=x5<=5
cvx_end
disp('Solution:');
disp(x1);
disp(x2);
disp(x3);
disp(x4);
disp(x5);
disp('Optimal solution is:')
disp(-13*x1+10*x2+8*x3-10*x4-22*x5);
```

Output of the code is shown below.

```
Solution:
          1.00
         16.00
          0.00
          3.00
          1.50
Optimal solution is:
         84.00
```

RESULTS

Solution of Linear Programming Problems using CVX solver in both direct method and using standard matrix form are discussed and experimented with some example problems.

3 | Assignment 67
L1 Norm Optimization and Its Impact on Compressed Sensing

3.1 Introduction

L1 norm optimization shot into fame around 2007 when the potential of compressed sensing (CS) in imaging, particularly in medical systems such as MRI, became widely recognized. This breakthrough indicated that it was possible to significantly reduce the acquisition time of images without sacrificing quality. The core principle behind this advancement lies in the ability to reconstruct compressed-sensed images using L1 norm optimization, a technique adept at handling underdetermined systems of equations. As a result, many mathematicians and researchers turned their attention to this area, developing efficient algorithms and generating a substantial body of literature that explored various facets of L1 norm optimization.

One of the intriguing aspects of L1 norm optimization is its geometric interpretation, particularly when applied to systems of equations that have infinitely many solutions. In such cases, the solution space can often be represented as a convex set. When we impose the L1 norm minimization criterion, we seek the solution that lies at the intersection of this convex set and the coordinate axes, which correspond to the least total deviation from zero across all variables. This process can be visualized as finding the point in the solution space that is closest to the origin in terms of the L1 norm, which effectively promotes sparsity in the solution.

The geometric understanding of L1 norm optimization reveals that the minimization process tends to favor solutions that have fewer non-zero components, leading to more interpretable results. This property is particularly advantageous in applications like image reconstruction, where we desire a solution that captures the essential features of the image while disregarding noise and irrelevant details. By leveraging the L1 norm, practitioners can achieve a balance between accuracy and computational efficiency, making it a powerful tool in the field of compressed sensing and beyond.

As the exploration of L1 norm optimization continues, its implications for various fields, including signal processing, statistics, and machine learning, remain significant. The advancements made in this area not only enhance our understanding of sparse solutions but also pave the way for further innovations in efficient data acquisition and reconstruction techniques.

3.1.1 Mathematical formulation

Let us start with two variable linear equation with infinite solutions. We take the help of CVX to do the computation. Consider the problem:

$$\begin{aligned} \min \quad & |x| + |y| \\ \text{subject to:} \quad & x + 2y = 20 \end{aligned}$$

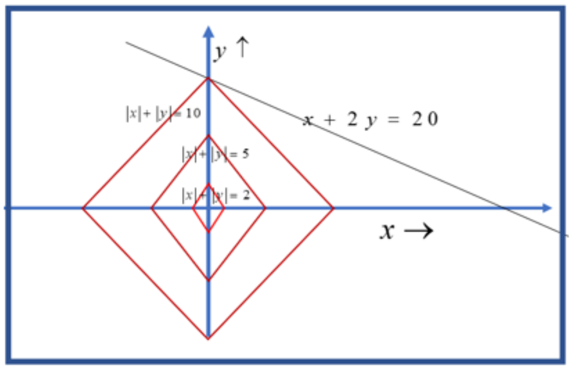


Figure 3.1: Geometrical interpretation of the constrained problem

Geometrical interpretation of the problem

We have set of points defined by $x+2y=20$, on which there are infinite points. We are asked to pick a point on it for which sum of the absolute value of its two-coordinate value is minimum.
The objective function's, global minimum value is zero since it is a sum of two positive numbers. It occurs at $(0,0)$. But it is not a point on our constraint line.
So, as usual, we take a level set of the objective function and plot it. Let us take level set value equal to 2. That is $|x+|y|=2$. A little consideration will show that, it is a diamond shaped figure centred at origin. Its four vertices are $(2,0)$, $(0,2)$, $(-2,0)$ and $(0,-2)$. Note that, every vertex point satisfies above equation. Also, every point on the boundary line defining the diamond satisfy above equation. But, none of the points of this level set touches our line. We increase the level set value of objective function to 5. That is $|x|+|y|=5$. It's a diamond whose four vertices are $(5,0)$, $(0,5)$, $(-5,0)$ and $(0,-5)$. Again, the points on this diamond do not touch our line. Slowly we increase the level set value until one point on diamond touches the line. We increase the level set value of objective function to 10. That is $|x|+|y|=10$. Its four vertices are $(10,0)$, $(0,10)$, $(-10,0)$ and $(0,-10)$. Now the point $(0,10)$ satisfy $x+2y=20$. That is $(0,10)$ is the point with least absolute sum on the line $x+2y=20$. Note that this solution is sparse. Out of two coordinates, one coordinate is zero. CVX will show that $(0,10)$ is in fact the solution. We write the code and verify it.
`Matlab` code for this task is given below.

```
cvx_begin quiet
variables x y
minimize (sum(abs(x)+abs(y)))
subject to
x+2*y==20
cvx_end
format bank
disp('Solution:')
disp(x);
disp(y);
disp('Minimum value:')
disp(sum(abs(x)+abs(y)));
```

Output of the code is shown below.

```
Solution:
          0.00
         10.00
Minimum value:
         10.00
```

The output is $x = 0$, $y = 10$ as expected.
Let us take another example. Let the equation of a plane in five dimensional space be

$$\begin{aligned} \min \quad & ||x_i||_1 \\ \text{S.to:} \quad & \sum_{i=1}^{5} i \cdot x_i = 200 \end{aligned}$$

`Matlab` code to solve this least L_1 norm problem is given below.

```
cvx_begin quiet
    variable x(5)
    minimize (sum(abs(x)))
    subject to
        (1:5)*x==200
cvx_end
format bank
disp('Solution:');
disp(x);
disp("Optimal value:");
disp(sum(abs(x)))
```

Output of the code is:

```
Solution:
        0.00
        0.00
        0.00
        0.00
       40.00
Optimal value:
       40.00
```

The solution point is (0, 0, 0, 0, 40) but is sparse. In short, L_1 norm is a sparsity inducing norm when used as a part of objective function in optimization.
L1 norm however does not guarantee the sparsest norm solution. Mathematicians denote L0 norm solution as the sparsest possible solution. However, there is no efficient algorithm to find it, so far. So *L1 norm solution is used as an approximation to L0 norm solution.*

3.2 L1-norm optimization and Linear Programming

Converting L1 norm problem into LP format using proper transformation. For example, consider the problem:

$$\begin{aligned} \min_x \quad & ||x||_1 \\ \text{S.to:} \quad & Ax = b \end{aligned}$$

First of all, note that, here, variable x can take positive and negative values. But in standard LP , variables must be non-negative. Secondly, objective function must be an inner product of x with a cost function in standard LPP. Our aim is to convert the problem in to that type of format. Here is the trick. Let

$$\begin{bmatrix} | \\ x \\ | \end{bmatrix} = \begin{bmatrix} | \\ x_1 \\ | \end{bmatrix} - \begin{bmatrix} | \\ x_2 \\ | \end{bmatrix}; \quad x_1, x_2 \geq 0$$

That is x is difference of two vectors with non-negative elements. Note that, we have now doubled the size of variables. We create a new vector by concatenating That is

$$X_{new} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \quad x \in \mathbb{R}^n; \quad x_1, x_2 \in \mathbb{R}^n_+; \quad x_{new} \in \mathbb{R}^{2n}$$

Let us first convert the given constraint $Ax = b$.

$$\begin{aligned} Ax &= A(x_1 - x_2) \\ &= Ax_1 - A_x 2 \\ &= (A - A) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= (A - A) x_{new} \\ &= b \\ \therefore (A - A) x_{new} &= b \end{aligned}$$

Let us now convert objective function. Some amount of thought is requited for understanding the equivalence.

$$\begin{aligned} \min_x \quad & ||x||_1 \\ & = \sum_{x_{new} \geq 0} x_{new} \\ & = \underset{x_{new} \geq 0}{e^T x_{new}}; \quad e = ones(2n, 1) \end{aligned}$$

So, the final formulation is :

$$\begin{aligned} \min_{x_{new}} \quad & ||x|| \\ \text{S.to:} \quad & (A - A) x_{new} = b \\ & x_{new} \geq 0 \end{aligned}$$

After solving above LP, we obtain the solution for the original problem as

$$X_{new} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \quad x \in \mathbb{R}^n; \quad x_1, x_2 \in \mathbb{R}^n_+; \quad x_{new} \in \mathbb{R}^{2n}$$

In CVX , we can input L1 norm optimization problem as such. No need of manual conversion into LP. It converts internally into an LP and solve it.

3.3 Task

Convert into standard LP and solve: You may do change of variable for convenience. Solve the LP using CVX. Also solve as such and compare the result.

1. Solve

$$\begin{aligned} \text{minimize} \quad & |x| + |y| \\ \text{subject to:} \quad & x + 2y = 20 \end{aligned}$$

SOLUTION

Transformation of the Problem to Matrix Form

We are given the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & |x| + |y| \\ \text{subject to:} \quad & x + 2y = 20 \end{aligned}$$

To transform this into a linear programming problem, we first split the variables x and y into their positive and negative parts:

$$x = x_1 - x_2 \quad \text{and} \quad y = y_1 - y_2$$

where $x_1, x_2, y_1, y_2 \geq 0$.

The objective function $|x| + |y|$ becomes:

$$|x| + |y| = (x_1 + x_2) + (y_1 + y_2)$$

Thus, the optimization problem can be reformulated as:

$$\begin{aligned} \text{minimize} \quad & (x_1 + x_2) + (y_1 + y_2) \\ \text{subject to:} \quad & (x_1 - x_2) + 2(y_1 - y_2) = 20 \\ & x_1, x_2, y_1, y_2 \geq 0 \end{aligned}$$

We can now express this problem in matrix form. Define the vector of variables:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix}$$

The objective function becomes:

$$\mathbf{c}^T\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix}$$

The constraint $(x_1 - x_2) + 2(y_1 - y_2) = 20$ can be written as:

$$\mathbf{Ax} = b$$

where:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 2 & -2 \end{bmatrix}, \quad b = 20$$

Finally, the non-negativity constraints are:

$$x_1, x_2, y_1, y_2 \geq 0$$

Thus, the linear program in matrix form is:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \\ \text{subject to:} \quad & \mathbf{Ax} = b \\ & \mathbf{x} \geq 0 \end{aligned}$$

where:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 2 & -2 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \quad b = 20$$

`Matlab` code to solve this LPP is given below.

```
% Define the problem data
A = [1, -1, 2, -2];   % Coefficient matrix for constraints
b = 20;               % Right-hand side of the equation
c = [1, 1, 1, 1];     % Coefficient vector for the objective
    function

% Solve the problem using CVX
cvx_begin quiet
    variable x(4)      % Define the variable vector [x1, x2, y1,
        y2]

    % Objective function: minimize (x1 + x2) + (y1 + y2)
    minimize( c * x );

    % Constraints
    subject to
        A * x == b;    % The constraint: (x1 - x2) + 2*(y1 - y2)
            = 20
        x >= 0;        % Non-negativity constraint
cvx_end

% Display the solution
x1 = x(1);
x2 = x(2);
y1 = x(3);
y2 = x(4);
x_sol = x1 - x2;
y_sol = y1 - y2;

disp('Optimal x:');
disp(x_sol);
disp('Optimal y:');
disp(y_sol);
disp('optimal value');
disp(c*x)
```

Output of the code is given below.

```
SOlution:
Optimal x:
          0.00
Optimal y:
         10.00
optimal value
         10.00
```

2. Solve

$$\text{minimize} \quad \|\mathbf{x}\|_1 = \sum_{i=1}^{5} |x_i|$$

$$\text{subject to:} \quad \sum_{i=1}^{5} i \cdot x_i = 200$$

SOLUTION

Transformation of the Problem to Matrix Form

We are given the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{x}\|_1 = \sum_{i=1}^{5} |x_i| \\ \text{subject to:} \quad & \sum_{i=1}^{5} i \cdot x_i = 200 \end{aligned}$$

To express this as a linear programming problem, we first split each x_i into its positive and negative parts:

$$x_i = x_{i1} - x_{i2}, \quad \text{where} \quad x_{i1} \geq 0 \quad \text{and} \quad x_{i2} \geq 0$$

Thus, for each x_i, the absolute value can be written as:

$$|x_i| = x_{i1} + x_{i2}$$

Now, the objective function becomes:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{5} |x_i| = \sum_{i=1}^{5} (x_{i1} + x_{i2})$$

This leads to the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{5} (x_{i1} + x_{i2}) \\ \text{subject to:} \quad & \sum_{i=1}^{5} i \cdot (x_{i1} - x_{i2}) = 200 \\ & x_{i1}, x_{i2} \geq 0 \quad \text{for} \quad i = 1, 2, 3, 4, 5 \end{aligned}$$

We now express the problem in matrix form. Define the vector of variables:

$$\mathbf{x} = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{32} \\ x_{41} \\ x_{42} \\ x_{51} \\ x_{52} \end{bmatrix}$$

The objective function becomes:

$$\mathbf{c}^T \mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \mathbf{x}$$

The constraint $\sum_{i=1}^{5} i \cdot (x_{i1} - x_{i2}) = 200$ can be written as:

$$\mathbf{A}\mathbf{x} = b$$

where:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 2 & -2 & 3 & -3 & 4 & -4 & 5 & -5 \end{bmatrix}, \quad b = 200$$

Finally, the non-negativity constraints are:

$$x_{i1}, x_{i2} \geq 0 \quad \text{for} \quad i = 1,2,3,4,5$$

Thus, the linear program in matrix form is:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}\mathbf{x} \\ \text{subject to:} \quad & \mathbf{A}\mathbf{x} = b \\ & \mathbf{x} \geq 0 \end{aligned}$$

where:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 2 & -2 & 3 & -3 & 4 & -4 & 5 & -5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad b = 200$$

`Matlab` code for this task is given below.

```
% Define the problem data
n = 5;                          % Number of variables
A = [1, -1, 2, -2, 3, -3, 4, -4, 5, -5];  % Coefficient matrix
   for constraints
b = 200;                        % Right-hand side of the equation
c = ones(1, 2*n);              % Coefficient vector for the
   objective function (1 for each x^+ and x^-)

% Solve the problem using CVX
cvx_begin quiet
    variable x(2*n)      % Define the variable vector [x1^+, x1
       ^-, x2^+, x2^-, ..., x5^+, x5^-]

    % Objective function: minimize the sum of x^+ and x^-
    minimize( c * x );

    % Constraints
    subject to
        [1, -1, 2, -2, 3, -3, 4, -4, 5, -5] * x == b;  % The
           constraint: \sum i * (x_i^+ - x_i^-) = 200
        x >= 0;          % Non-negativity constraint
cvx_end

% Extract the solutions
x1_plus = x(1);
x1_minus = x(2);
x2_plus = x(3);
x2_minus = x(4);
x3_plus = x(5);
x3_minus = x(6);
x4_plus = x(7);
x4_minus = x(8);
x5_plus = x(9);
x5_minus = x(10);

```

```
% Calculate the final solutions for x_i
x_sol = [x1_plus - x1_minus; x2_plus - x2_minus; x3_plus -
    x3_minus; x4_plus - x4_minus; x5_plus - x5_minus];

% Display the solution
disp('Optimal x values:');
disp(x_sol);
disp('Optimum value:')
disp(c*x)
```

Output of the code is given below.

```
Optimal x values:
          0.00
          0.00
          0.00
          0.00
         40.00
Optimum value:
         40.00
```

RESULTS

1. Least L_1 norm optimization is revisited.
2. L_1 minimization problem is transformed into standard LPP and solved it using `CVX` solvers.
3. Results of the direct approach and the transformed to LPP approach are compared and verified the uniqueness in solution.

4 | Assignment 69
Eigenvalues and Eigenvectors in Linear Algebra

4.1 Introduction

Eigenvalues and eigenvectors are fundamental concepts in linear algebra with critical applications in engineering and computer science. Understanding these concepts enables engineers to analyze systems, optimize processes, and design efficient algorithms.
In the field of image processing, the ability to effectively compress and reconstruct images is crucial, especially in a world where data storage and transmission are of utmost importance. This lesson explores the profound utility of eigenvalue decomposition as a powerful tool for image compression and reconstruction.
Imagine encountering a low-resolution image of a familiar scene. The human brain excels at recognizing familiar objects by relying on essential features, often extracting the most significant details while discarding the less important information. This cognitive process mirrors the power of eigenvalue decomposition, where eigenvectors represent the "nectar" of a matrix, capturing its most important characteristics.
As an example, try to identify this image. If you can do it, then your brain is knows this place! Else ....

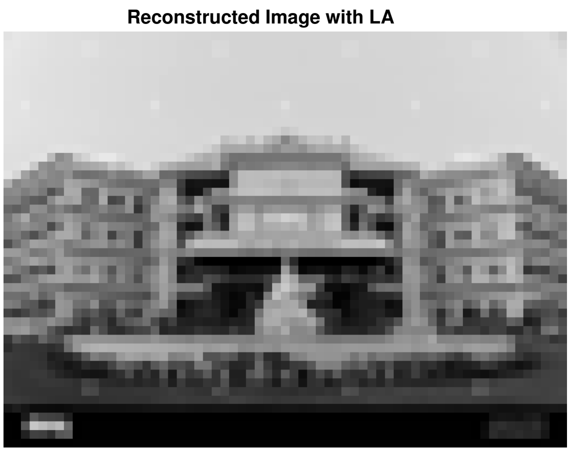
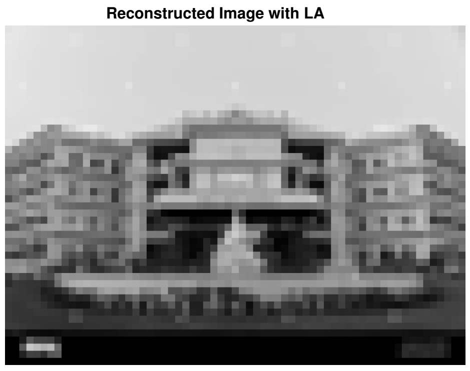


Figure 4.1: A reconstructed image using Linear Algebra

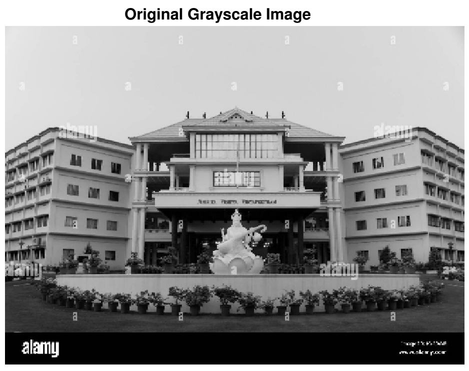
Before proceeding further just compare the size of its' original clean image and the low-quality image shown in Figure 4.1.

```
Original image size: 118.69 KB
Reconstructed image size: 1.12 KB
```

The reconstructed image is just 10% of the original in size! This is the core principle of optimizing image storage of CCTV system. For a comparison, just see the clean image along with the low-size image.



(a) Reconstructed Image



(b) Clean Image

Figure 4.2: Comparison of Original and Cleaned Images

There are numerous application for eigen values and eigen vectors in almost all branches of engineering and sciences.
Applications include:

- **Search Engines:** Google's PageRank algorithm utilizes eigenvalues and eigenvectors to rank web pages based on their relevance and importance.
- **Control Systems:** Eigenvalues determine system stability and response characteristics.
- **Structural Analysis:** Eigenvalues help determine the natural frequencies of structures, crucial for earthquake resistance.
- **Vibration Analysis:** In mechanical engineering, eigenvalues are used to analyze vibration modes of systems, ensuring stability and performance.
- **Data Science:** Techniques like Principal Component Analysis (PCA) utilize eigenvalues to reduce dimensionality, preserving variance in datasets.
- list continues...

4.2 Definitions

1. **Eigenvalues:** An eigenvalue (λ) of a square matrix A is a scalar such that there exists a non-zero vector $\mathbf{v}$ (eigenvector) satisfying the equation:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Mathematically, it is the solution of the determinant equation, $|A - \lambda I| = 0$

2. **Eigenvectors:** An eigenvector is a non-zero vector that, when transformed by matrix A, changes only in scale (not direction). Mathematically, it is the non-trivial solution of the system $(A - \lambda I)\, I = 0$

4.2.1 Characteristic polynomial of 2×2 matrices

For a 2×2 matrix:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

the characteristic polynomial is derived from the determinant of $A - \lambda I$, where I is the identity matrix:

$$\det(A - \lambda I) = 0$$

This leads to:

$$\det\begin{pmatrix} a-\lambda & b \\ c & d-\lambda \end{pmatrix} = (a-\lambda)(d-\lambda) - bc = 0$$

Short-cut Method: The characteristic polynomial can be simplified to:

$$\lambda^2 - (a+d)\lambda + (ad - bc) = 0$$

This polynomial can be solved using the quadratic formula:

$$\lambda = \frac{(a+d) \pm \sqrt{(a+d)^2 - 4(ad-bc)}}{2}$$

4.2.2 General formula

Characteristic polynomial of a square matrix A is condition for the existence of a non-trivial solution x for the homogeneous system, $(A - \lambda I)\, X = 0$. In form of determinant,

$$CP := |A - \lambda I| = 0$$

4.3 Examples

4.3.1 Example 1: Finding eigenvalues and eigenvectors of a simple matrix

Consider the matrix:

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

1. **Characteristic Polynomial**: Using the formula:

$$\lambda^2 - (3+1)\lambda + (3 \cdot 1 - 2 \cdot 4) = 0 \implies \lambda^2 - 4\lambda - 5 = 0$$

2. **Solve for λ**:

$$\lambda = \frac{4 \pm \sqrt{16+20}}{2} = \frac{4 \pm \sqrt{36}}{2} = \frac{4 \pm 6}{2} \implies \lambda_1 = 5, \lambda_2 = -1$$

3. **Finding Eigenvectors**:

- For $\lambda_1 = 5$:

$$(A - 5I)\mathbf{v} = 0 \implies \begin{pmatrix} -2 & 2 \\ 4 & -4 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0 \implies x_1 = x_2$$

  So, one eigenvector is $\mathbf{v_1} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

- For $\lambda_2 = -1$:

$$(A + I)\mathbf{v} = 0 \implies \begin{pmatrix} 4 & 2 \\ 4 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0 \implies x_1 = -\frac{1}{2}x_2$$

  So, one eigenvector is $\mathbf{v_2} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$.

Result: Eigenvalues are $\lambda_1 = 5$, $\lambda_2 = -1$ with corresponding eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$.

4.3.2 Example 2: Application in PageRank algorithm

Google's PageRank algorithm ranks web pages by modeling them as a directed graph, where:

- **Nodes** represent web pages.
- **Edges** represent links between pages.

The rank of a page is determined by its eigenvector corresponding to the largest eigenvalue of the link matrix A.
For example, consider a simplified link structure with three web pages, represented by the matrix:

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

1. **Characteristic Polynomial**:

$$\det \begin{pmatrix} -\lambda & 1 & 0 \\ 0 & -\lambda & 1 \\ 1 & 0 & -\lambda \end{pmatrix} = -\lambda(-\lambda^2 + 1) = 0$$

This simplifies to $\lambda^3 - 1 = 0$, giving $\lambda = 1$.
2. **Finding Eigenvector**: Using $\lambda = 1$:

$$(A - I)\mathbf{v} = 0 \implies \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 0$$

Solving this system yields the eigenvector representing the PageRank scores for the three pages. Do it as a extension task.

4.4 Task

1. **Problem 1:** Calculate the eigenvalues and eigenvectors of the matrix:

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

SOLUTION

To find the eigenvalues and eigenvectors of a 2×2 matrix, we can use the shortcut formula for the characteristic polynomial:

$$\lambda^2 - \text{trace}(A)\lambda + \det(A) = 0,$$

where A is the matrix. Let's apply this to the matrix

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

First, we calculate the trace and determinant of A:

- The trace is the sum of the diagonal elements:

$$\text{trace}(A) = 2 + 2 = 4.$$

- The determinant is calculated as follows:

$$\det(A) = (2)(2) - (1)(1) = 4 - 1 = 3.$$

Next, substituting the trace and determinant into the characteristic polynomial gives:

$$\lambda^2 - (4)\lambda + 3 = 0,$$

which simplifies to:

$$\lambda^2 - 4\lambda + 3 = 0.$$

We can factor this quadratic equation:

$$(\lambda - 1)(\lambda - 3) = 0.$$

Setting each factor to zero gives the eigenvalues:

$$\lambda_1 = 1, \quad \lambda_2 = 3.$$

To find the eigenvectors corresponding to each eigenvalue, we use the shortcut for the eigenvector of a 2×2 matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

$$EV(\lambda) = \begin{pmatrix} \lambda - d \\ c \end{pmatrix}.$$

For the eigenvalue $\lambda_1 = 1$:

$$EV(1) = \begin{pmatrix} 1 - 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

This eigenvector can be simplified (up to a scalar multiple) to:

$$\mathbf{v_1} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

For the eigenvalue $\lambda_2 = 3$:

$$EV(3) = \begin{pmatrix} 3 - 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

This eigenvector is already in a simple form:

$$\mathbf{v_2} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

In summary, the eigenvalues and corresponding eigenvectors of the matrix

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

are:

- $\lambda_1 = 1$ with eigenvector $\mathbf{v_1} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

- $\lambda_2 = 3$ with eigenvector $\mathbf{v_2} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

2. **Problem 2:** For the matrix:
$$A = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$
find the eigenvalues and eigenvectors.

SOLUTION

We are given the matrix
$$A = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$
and we aim to find its eigenvalues using the characteristic polynomial.

The shortcut formula for the characteristic polynomial of a 3×3 matrix is given by:
$$\lambda^3 - \mathrm{tr}(A)\lambda^2 + (\text{sum of principal minors of } A)\lambda - \det(A) = 0.$$

The trace of a matrix is the sum of its diagonal elements. For matrix A, we have:
$$\mathrm{tr}(A) = 1 + 1 + 1 = 3.$$

The principal minors are the determinants of the 2×2 submatrices obtained by deleting one row and one column of A.

The first minor is obtained by deleting the third row and third column:
$$\det\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} = (1)(1) - (2)(0) = 1.$$

The second minor is obtained by deleting the second row and second column:
$$\det\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = (1)(1) - (1)(1) = 0.$$

The third minor is obtained by deleting the first row and first column:
$$\det\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = (1)(1) - (0)(0) = 1.$$

Thus, the sum of the principal minors is:
$$1 + 0 + 1 = 2.$$

The determinant of A can be calculated using cofactor expansion along the first row:
$$\det(A) = 1 \cdot \det\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 2 \cdot \det\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} + 1 \cdot \det\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$= 1 \cdot (1) - 2 \cdot (0) + 1 \cdot (-1) = 1 - 0 - 1 = 0.$$

Now, we substitute these values into the characteristic polynomial formula:

$$\lambda^3 - \text{tr}(A)\lambda^2 + (\text{sum of principal minors})\lambda - \det(A) = 0$$

$$\lambda^3 - 3\lambda^2 + 2\lambda - 0 = 0.$$

We now solve the equation:

$$\lambda^3 - 3\lambda^2 + 2\lambda = 0.$$

Factoring out λ and apply factor theorem, we get:

$$\lambda(\lambda^2 - 3\lambda + 2) = 0$$
$$\lambda(\lambda - 2)(\lambda - 1) = 0$$

This gives one eigenvalue:

$$\lambda_1 = 0; \quad \lambda_2 = 2; \quad \lambda_3 = 1$$

Now we find the eigenvectors corresponding to each eigenvalue.

For $\lambda_1 = 0$, solve $(A - 0I)\mathbf{v} = 0$:

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

This gives the system:

$$x + 2y + z = 0, \quad y = 0, \quad x + z = 0.$$

Thus, $x = -z$, and the eigenvector is:

$$\mathbf{v}_1 = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}.$$

For $\lambda_2 = 2$, solve $(A - 2I)\mathbf{v} = 0$:

$$\begin{pmatrix} -1 & 2 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

This gives the system:

$$-x + 2y + z = 0, \quad -y = 0, \quad x - z = 0.$$

Thus, $x = z$, and the eigenvector is:

$$\mathbf{v}_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

For $\lambda_3 = 1$, solve $(A - I)\mathbf{v} = 0$:

$$\begin{pmatrix} 0 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

This gives the system:

$$2y + z = 0, \quad x = 0.$$

Thus, $z = -2y$, and the eigenvector is:

$$\mathbf{v}_3 = \begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix}.$$

3. **Problem 3:**If $A = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 3 & 4 \\ 1 & -1 & -1 \end{bmatrix}$, compute the eigen values and eigen vectors and left eigen vectors of A.

SOLUTION

We are given the matrix

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

and need to find its eigenvalues and eigenvectors.

The characteristic polynomial for a 3×3 matrix is given by:

$$\lambda^3 - \text{tr}(A)\lambda^2 + (\text{sum of principal minors})\lambda - \det(A) = 0.$$

The trace is the sum of the diagonal elements:

$$\text{tr}(A) = 1 + 3 + (-1) = 3.$$

We now compute the 2×2 principal minors:

- Minor by removing the third row and third column:

$$\det\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} = (1)(3) - (2)(0) = 3.$$

- Minor by removing the second row and second column:

$$\det\begin{pmatrix} 1 & 4 \\ 1 & -1 \end{pmatrix} = (1)(-1) - (4)(1) = -1 - 4 = -5.$$

- Minor by removing the first row and first column:

$$\det\begin{pmatrix} 3 & 4 \\ -1 & -1 \end{pmatrix} = (3)(-1) - (4)(-1) = -3 + 4 = 1.$$

Thus, the sum of the principal minors is:

$$3 + (-5) + 1 = -1.$$

We calculate the determinant of A by cofactor expansion along the first row:

$$\det(A) = 1 \cdot \det\begin{pmatrix} 3 & 4 \\ -1 & -1 \end{pmatrix} - 2 \cdot \det\begin{pmatrix} 0 & 4 \\ 1 & -1 \end{pmatrix} + 4 \cdot \det\begin{pmatrix} 0 & 3 \\ 1 & -1 \end{pmatrix}.$$

The 2×2 determinants are:

$$\det\begin{pmatrix} 3 & 4 \\ -1 & -1 \end{pmatrix} = -3 + 4 = 1, \quad \det\begin{pmatrix} 0 & 4 \\ 1 & -1 \end{pmatrix} = -4,$$

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

Thus:

$$\det(A) = 1 \cdot 1 - 2 \cdot (-4) + 4 \cdot (-3) = 1 + 8 - 12 = -3.$$

Substituting into the characteristic polynomial:

$$\lambda^3 - \text{tr}(A)\lambda^2 + (\text{sum of principal minors})\lambda - \det(A) = 0,$$

we get:

$$\lambda^3 - 3\lambda^2 - \lambda + 3 = 0.$$

We now solve the cubic equation:

$$\lambda^3 - 3\lambda^2 - \lambda + 3 = 0.$$
$$(\lambda - 1)(\lambda + 1)(\lambda - 3) = 0$$

$$\lambda_1 = 1, \quad \lambda_2 = -1, \quad \lambda_3 = 3.$$

To find the eigenvector corresponding to $\lambda_1 = 3$, solve $(A - 3I)\mathbf{v} = 0$:

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

Solving this system gives the eigenvector:

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

For $\lambda_2 = -1$, solve $(A + I)\mathbf{v} = 0$:

$$A + I = \begin{pmatrix} 1 & 2 & 4 \\ 0 & 3 & 4 \\ 1 & -1 & -1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 4 \\ 0 & 4 & 4 \\ 1 & -1 & 0 \end{pmatrix}.$$

Note that the third row is depending on first and second rows. So by finding the cross product of first two rows,

$$\mathbf{v}_2 = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}.$$

For $\lambda_3 = 1$, solve $(A - I)\mathbf{v} = 0$:

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

Note that the second row is same as first row. So by finding the cross product of first and third rows,

$$\mathbf{v}_3 = \begin{pmatrix} 0 \\ -2 \\ 1 \end{pmatrix}.$$

Thus, the eigenvalues of the matrix are:

$$\lambda_1 = 3, \quad \lambda_2 = -1, \quad \lambda_3 = 1$$

with corresponding eigenvectors $\mathbf{v}_1 = \begin{pmatrix}1\\1\\0\end{pmatrix}$, $\mathbf{v}_2 = \begin{pmatrix}-1\\-1\\1\end{pmatrix}$, and $\mathbf{v}_3 = \begin{pmatrix}0\\-2\\1\end{pmatrix}$.

Left eigen vectors of the matrix A are eigen vectors of A^T.

Here $A^T = \begin{bmatrix}1 & 0 & 1\\2 & 3 & -1\\4 & 4 & -1\end{bmatrix}$.

Since A and A^T have same eigen values, it is enough to find corresponding eigen vectors. When $\lambda = 3$, the coefficient matrix of $(A - \lambda I)X = 0$ reduced into $\begin{bmatrix}-2 & 0 & 1\\2 & 0 & -1\\4 & 4 & -4\end{bmatrix}$

Here the only independent rows are first and last. So the eigen vector can be found as the cross product of these two rows. $\therefore v_1 = \begin{bmatrix}1\\1\\2\end{bmatrix}$.

When $\lambda = -1$, the coefficient matrix of $(A - \lambda I)X = 0$ reduced into $\begin{bmatrix}2 & 0 & 1\\2 & 4 & -1\\4 & 4 & 0\end{bmatrix}$

Here the only independent rows are first and second. So the eigen vector can be found as the cross product of these two rows. $\therefore v_2 = \begin{bmatrix}-1\\1\\2\end{bmatrix}$. When $\lambda = 1$, the coefficient matrix of $(A - \lambda I)X = 0$ reduced into $\begin{bmatrix}0 & 0 & 1\\2 & 2 & -1\\4 & 4 & -2\end{bmatrix}$

Here the only independent rows are first and second. So the eigen vector can be found as the cross product of these two rows. $\therefore v_2 = \begin{bmatrix}-1\\1\\0\end{bmatrix}$.

RESULTS

1. Concepts of eigen values and eigen vectors are revisited in a practical view point.
2. Eigen values and eigen vectors of 2×2 and 3×3 matrices are calculated with minimum steps.

5 | Assignment 70 Basis Set And Linear Equations

5.1 Introduction

In this discussion, we will bridge the gap between coordinate geometry and linear algebra by exploring how geometric objects such as lines and planes can be represented algebraically. This journey begins with familiar notions from high school geometry, such as lines and planes in 3D, and leads us into the powerful abstractions provided by linear algebra, such as vector spaces, subspaces, and matrices.
We aim to answer the question: *Which is better for representing subspaces—basis sets or linear equations?* We will use examples from both geometry and algebra to build a unified understanding.

5.2 Coordinate Geometry: Lines, Planes, and Directional Cosines

In three-dimensional space, a line can be represented parametrically using a point $P_0(x_0, y_0, z_0)$ and a direction vector $\mathbf{d} = (a, b, c)$. The parametric equations for the line can be written as:

$$\begin{aligned} x &= x_0 + at, \\ y &= y_0 + bt, \\ z &= z_0 + ct, \end{aligned}$$

where t is a parameter.
Alternatively, we can express a line using directional cosines, which are the cosines of the angles that the line makes with the coordinate axes. If the direction cosines are denoted as l, m, n, where:

$$l = \cos(\alpha), \quad m = \cos(\beta), \quad n = \cos(\gamma),$$

the equations can be expressed as:

$$\begin{aligned} x &= x_0 + k \cdot l, \\ y &= y_0 + k \cdot m, \\ z &= z_0 + k \cdot n, \end{aligned}$$

where k is a scalar. This leads to the relationship:

$$\frac{x - x_0}{l} = \frac{y - y_0}{m} = \frac{z - z_0}{n},$$

which can be simplified into a set of two independent simultaneous linear equations, demonstrating a deeper connection between the geometric representation and linear algebraic formulation.

5.2.1 Planes in three-dimensional space

The equation of a plane can be defined in multiple ways, but one common representation is through the normal vector and a point on the plane. If we have a point $P_0(x_0, y_0, z_0)$ on the plane and a normal vector $\mathbf{n} = (a, b, c)$, the equation of the plane can be expressed as:

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0.$$

Interestingly, if we are given three non-collinear points $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, and $P_3(x_3, y_3, z_3)$, we can also find the equation of the plane. The vectors $\mathbf{v_1} = P_2 - P_1$ and $\mathbf{v_2} = P_3 - P_1$ are formed, and their cross product $\mathbf{n} = \mathbf{v_1} \times \mathbf{v_2}$ gives us the normal vector to the plane. Using the normal vector and one of the points, we can derive the plane equation.

5.3 Linear Algebra: Vectors, Subspaces, and Basis

In linear algebra, we discuss concepts such as vector spaces, linear combinations, subspaces, and bases. A vector space is a collection of vectors that can be added together and multiplied by scalars. A subspace is a subset of a vector space that is itself a vector space under the same operations.
A basis of a vector space is a set of vectors that are linearly independent and span the space. For example, in $\mathbb{R}^3$, the standard basis consists of the vectors $\mathbf{e_1} = (1, 0, 0)$, $\mathbf{e_2} = (0, 1, 0)$, and $\mathbf{e_3} = (0, 0, 1)$.
The relationship between linear equations and subspaces becomes evident when considering the null space of a matrix. The null space consists of all vectors $\mathbf{x}$ such that $A\mathbf{x} = 0$, representing a subspace of the vector space defined by A.

5.3.1 Eigenvalues and eigenvectors

Eigenvalues and eigenvectors play a significant role in linear algebra. Given a square matrix A, an eigenvalue λ is defined by the equation:

$$(A - \lambda I)\mathbf{x} = 0,$$

where I is the identity matrix and $\mathbf{x}$ is the corresponding eigenvector.
A major observation in this context is that solving $(A - \lambda I)\mathbf{x} = 0$ results in finding a vector $\mathbf{x}$ that is orthogonal to the transformation defined by $A - \lambda I$. Furthermore, since $A - \lambda I$ contains only two independent rows, the eigenvector $\mathbf{x}$ can be interpreted geometrically as the cross product of the independent vectors of $A - \lambda I$ in 3D space.

5.3.2 Direction Cosines and simultaneous equations

Using the concept of directional cosines, we can represent a line in 3D. By simplifying this representation, we can observe that it reduces to a system of two independent simultaneous linear equations. This further emphasizes the connection between geometric concepts and algebraic formulations.

5.4 A Unified View of Coordinate Geometry and Linear Algebra

By synthesizing concepts from coordinate geometry and linear algebra, we find a rich interplay between algebraic equations and geometric interpretations. We observe how algebraic manipulations of vectors, represented in matrices, connect with geometric representations in $\mathbb{R}^3$.
This unification reveals deeper mathematical structures, particularly the concept of matrices as versatile tools for data representation, operations, and abstract computations. Matrices allow us to encapsulate geometric transformations, linear relationships, and eigenvalue problems in a single mathematical framework. Their compatibility with computation and analysis enhances our ability to model, analyze, and solve real-world problems in multi-dimensional spaces.
The abstract nature of matrices helps bridge various mathematical concepts, leading to innovative applications across disciplines.

5.5 Tasks

1. Find equation of the plane passing through three points: (120), (345), (678).

SOLUTION

To find the equation of the plane passing through the points $P_1(1,2,0)$, $P_2(3,4,5)$, and $P_3(6,7,8)$, we start by forming two vectors from these points:

$$\mathbf{v_1} = P_2 - P_1 = (3-1, 4-2, 5-0) = (2,2,5)$$

$$\mathbf{v_2} = P_3 - P_1 = (6-1, 7-2, 8-0) = (5,5,8)$$

Next, we find the normal vector $\mathbf{n}$ to the plane by taking the cross product of $\mathbf{v_1}$ and $\mathbf{v_2}$:

$$\hat{n} = \mathbf{v_1} \times \mathbf{v_2} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ 2 & 2 & 5 \\ 5 & 5 & 8 \end{vmatrix}$$

Calculating the determinant, we have:

$$\hat{n} = \hat{i}(2\cdot 8 - 5\cdot 5) - \hat{j}(2\cdot 8 - 5\cdot 5) + \hat{k}(2\cdot 5 - 2\cdot 5)$$

This simplifies to:

$$\hat{n} = \hat{i}(16-25) - \hat{j}(16-25) + \hat{k}(10-10)$$
$$\hat{n} = \hat{i}(-9) + \hat{j}(9) + \hat{k}(0) = (-9,9,0)$$

The equation of the plane can be expressed as:

$$ax + by + cz = d$$

where $\mathbf{n} = (a,b,c) = (-9,9,0)$. Using the point $P_1(1,2,0)$ to find d:

$$-9(1) + 9(2) + 0(0) = d$$

This gives:

$$-9 + 18 = d \implies d = 9$$

Thus, the equation of the plane is:

$$-9x + 9y + 0z = 9$$

This can be simplified to:

$$-x + y = 1$$

Therefore, the equation of the plane passing through the points $(1,2,0)$, $(3,4,5)$, and $(6,7,8)$ is:

$$-x + y = 1$$

2. Find the perpendicular distance between the plane generated in task 1 to the origin.

SOLUTION

To find the distance from the origin to the plane given by the equation

$$-x+y=1,$$

we can rewrite the equation in the standard form $ax+by+cz+d=0$. The given equation can be expressed as:

$$-x+y-1=0 \implies -1x+1y+0z-1=0.$$

Here, the coefficients are: - $a=-1$ - $b=1$ - $c=0$ - $d=-1$

The distance D from a point (x_0, y_0, z_0) to the plane $ax+by+cz+d=0$ is given by the formula:

$$D=\frac{|ax_0+by_0+cz_0+d|}{\sqrt{a^2+b^2+c^2}}.$$

In this case, we want to find the distance from the origin $(0,0,0)$. Plugging in the values into the formula gives:

$$D=\frac{|-1(0)+1(0)+0(0)-1|}{\sqrt{(-1)^2+1^2+0^2}}.$$

This simplifies to:

$$D=\frac{|-1|}{\sqrt{1+1+0}}=\frac{1}{\sqrt{2}}.$$

Thus, the distance from the origin to the plane is

$$\frac{1}{\sqrt{2}} \quad \text{or} \quad \frac{\sqrt{2}}{2}.$$

So the distance from the origin to the plane $-x+y=1$ is $\frac{1}{\sqrt{2}} \sim 0.7071$.

3. Find the nearest point on the plane from the origin.

To find the nearest point from the origin to the plane given by the equation

$$-x+y=1,$$

we can use the concept of projecting the origin onto the plane. First, we need to express the plane in its normal form.

The equation can be rearranged to:

$$-x+y-1=0,$$

where the normal vector $\mathbf{n}$ of the plane is given by:

$$\mathbf{n}=(-1,1,0).$$

The formula for the projection of a point P_0 onto a plane can be derived as follows.

A point on the plane can be found by setting $x = 0$ in the plane equation:

$$-0 + y = 1 \implies y = 1 \implies P_1(0,1,0) \text{ is on the plane.}$$

The vector from the origin $(0,0,0)$ to the point $P_1(0,1,0)$ is:

$$\mathbf{OP_1} = P_1 - O = (0-0, 1-0, 0-0) = (0,1,0).$$

The normal vector of the plane can be normalized:

$$\mathbf{n} = (-1,1,0), \quad \|\mathbf{n}\| = \sqrt{(-1)^2 + 1^2 + 0^2} = \sqrt{2}.$$

So the unit normal vector is:

$$\hat{\mathbf{n}} = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right).$$

The projection of $\mathbf{OP_1}$ onto the normal vector $\mathbf{n}$ is calculated as:

$$\text{proj}_{\mathbf{n}}(\mathbf{OP_1}) = \frac{\mathbf{OP_1} \cdot \hat{\mathbf{n}}}{\hat{\mathbf{n}} \cdot \hat{\mathbf{n}}} \hat{\mathbf{n}}.$$

First, we calculate the dot product:

$$\mathbf{OP_1} \cdot \hat{\mathbf{n}} = (0,1,0) \cdot \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right) = 0 \cdot -\frac{1}{\sqrt{2}} + 1 \cdot \frac{1}{\sqrt{2}} + 0 \cdot 0 = \frac{1}{\sqrt{2}}.$$

Since $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$, the projection becomes:

$$\text{proj}_{\mathbf{n}}(\mathbf{OP_1}) = \frac{1}{\sqrt{2}} \hat{\mathbf{n}} = \frac{1}{\sqrt{2}} \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right) = \left(-\frac{1}{2}, \frac{1}{2}, 0\right).$$

The nearest point P on the plane can be found by subtracting the projection from the origin:

$$P = O + \text{proj}_{\mathbf{n}}(\mathbf{OP_1}) = (0,0,0) + \left(-\frac{1}{2}, \frac{1}{2}, 0\right) = \left(-\frac{1}{2}, \frac{1}{2}, 0\right).$$

Thus, the nearest point from the origin to the plane $-x + y = 1$ is

$$\left(-\frac{1}{2}, \frac{1}{2}, 0\right).$$

So the nearest point from the origin to the plane is $\left(-\frac{1}{2}, \frac{1}{2}, 0\right)$.

RESULTS

Basics of projections and its relationships between various subspaces are revisited.

6 | Assignment 71 - Differential Equations to Linear Algebra

6.1 Introduction

Most undergraduate-level linear differential equations with constant coefficients can be solved based on the solution methodology of the following first-order differential equation. Higher-order linear constant coefficient differential equations are first converted into several coupled differential equations for solution. This leads to a square matrix A associated with the problem. The solution finally ends up in finding eigenvalues and eigenvectors of that matrix A. From this standpoint, this document opens doors to control systems by explaining the representation of homogeneous and non-homogeneous systems of linear differential equations for various applications with essential mathematics.

6.2 Differential Equations and Linear Systems

Consider a linear system described by the state-space representation:

$$\dot{x}(t) = Ax(t) + \mathbf{b}(t),$$

where $x(t)$ is the state vector, A is the system matrix, and $\mathbf{b}(t)$ is the input vector.

6.2.1 Homogeneous Solutions

The homogeneous solution to the system can be found by solving the equation:

$$\dot{x}(t) = Ax(t).$$

The solution can be expressed in terms of the matrix exponential:

$$x_h(t) = e^{At}x(0),$$

where $x(0)$ represents the initial state.

Eigenvalue Problem

The eigenvalue problem arises naturally when finding solutions to the homogeneous system:

$$Av = \lambda v,$$

where λ are the eigenvalues and v are the corresponding eigenvectors. The eigenvalues provide critical information about the system's behavior, such as stability and oscillation.

Stability analysis

The stability of a linear system can be analyzed through the eigenvalues of the matrix A:

- If all eigenvalues have negative real parts, the system is asymptotically stable.
- If any eigenvalue has a positive real part, the system is unstable.
- If any eigenvalue has a zero real part, the stability is determined by higher-order terms.

6.2.2 Particular solutions

To find a particular solution for the non-homogeneous equation, we consider:

$$\dot{x}(t) = Ax(t) + \mathbf{b}(t).$$

The solution can be expressed as:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}\mathbf{b}(\tau)\,d\tau,$$

where $\mathbf{b}(\tau)$ represents the non-homogeneous input.

Convolution representation

The term $\int_0^t e^{A(t-\tau)}\mathbf{b}(\tau)\,d\tau$ represents a convolution operation, allowing us to handle integral equations in a systematic manner. This representation highlights the response of the system to arbitrary inputs.

6.3 Example: Coupled Differential Equations

Consider the following coupled differential equations:

$$\dot{x}_1(t) = -2x_1(t) + 3x_2(t),$$
$$\dot{x}_2(t) = -x_1(t) - x_2(t).$$

This can be represented in matrix form:

$$\dot{x}(t) = Ax(t),$$

where

$$A = \begin{pmatrix} -2 & 3 \\ -1 & -1 \end{pmatrix}, \quad x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}.$$

To solve this, we find the eigenvalues of A:

$$\det(A - \lambda I) = 0.$$

Calculating the determinant:

$$\begin{vmatrix} -2-\lambda & 3 \\ -1 & -1-\lambda \end{vmatrix} = (-2-\lambda)(-1-\lambda) - 3(-1) = \lambda^2 + 3\lambda + 1 = 0.$$

The roots of the characteristic polynomial are:

$$\lambda = \frac{-3 \pm \sqrt{9-4}}{2} = \frac{-3 \pm \sqrt{5}}{2}.$$

These roots indicate that the system's stability depends on the signs of the real parts of the eigenvalues.
The MATLAB code to solve this system and perform stability analysis is as follows:

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

% Define the initial conditions
x0 = [1; 0]; % Initial conditions for x1 and x2

% Time vector
t = 0:0.01:5;

% Solve the system
[t, x] = ode45(@(t,x) A*x, t, x0);

% Plot the results
figure;
plot(t, x(:,1), 'r', t, x(:,2), 'b');
xlabel('Time (s)');
ylabel('States');
legend('x_1(t)', 'x_2(t)');
title('Response of Coupled Differential Equations');
grid on;
```

6.3.1 Real-time examples in control systems

Here are three popular examples in control systems that demonstrate the discussed concepts:

Example 1: First-order RC circuit

Consider an RC circuit described by the differential equation:

$$\tau \frac{dV(t)}{dt} + V(t) = V_s(t),$$

where $V_s(t)$ is the input voltage, and $\tau = RC$ is the time constant. The system can be expressed as:

$$\dot{V}(t) = -\frac{1}{\tau}V(t) + \frac{1}{\tau}V_s(t).$$

Analytical Solution: Homogeneous Solution:

$$\dot{V}_h(t) + \frac{1}{\tau}V_h(t) = 0 \implies V_h(t) = Ce^{-\frac{1}{\tau}t},$$

where C is the constant determined by initial conditions.
Particular Solution:
Assuming $V_s(t) = V_0$ (a constant):

$$\dot{V}_p(t) + \frac{1}{\tau}V_p(t) = \frac{1}{\tau}V_0 \implies V_p(t) = V_0 + De^{-\frac{1}{\tau}t}.$$

Combining both solutions:

$$V(t) = V_0 + Ce^{-\frac{1}{\tau}t}.$$

Stability analysis The system is stable if the homogeneous solution decays to zero as $t \to \infty$. This occurs when the coefficient $-\frac{1}{\tau} < 0$. Hence, the system is stable as long as $\tau > 0$.
Matlab code for this task is given below.

```
% Parameters
tau = 1; % Time constant
V0 = 5; % Input voltage
t = 0:0.01:5; % Time vector

% Analytical solution
C = V0; % Initial condition
V = V0 + C * exp(-t/tau);

% Plot
figure;
plot(t, V);
title('Voltage across RC Circuit');
xlabel('Time (s)');
ylabel('Voltage (V)');
grid on;
```

Example 2: Mass-spring-damper system

The dynamics of a mass-spring-damper system can be described by:

$$m\frac{d^2x(t)}{dt^2} + b\frac{dx(t)}{dt} + kx(t) = F(t),$$

where m is the mass, b is the damping coefficient, k is the spring constant, and $F(t)$ is the external force.

Analytical solution: **Matrix Representation:**
Let $x_1(t) = x(t)$ and $x_2(t) = \dot{x}(t)$:

$$\begin{pmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{m} \end{pmatrix} F(t).$$

The eigenvalues of the system matrix A determine the stability.
Matlab code for this task is given below.

```
% Parameters
m = 1; % Mass
b = 0.5; % Damping coefficient
k = 1; % Spring constant
F = @(t) 1; % External force

% State-space representation
A = [0 1; -k/m -b/m];
B = [0; 1/m];
x0 = [0; 0]; % Initial conditions
t = 0:0.01:10; % Time vector

% Solve using ode45
[t, x] = ode45(@(t, x) A*x + B*F(t), t, x0);

```

```
% Plot results
figure;
plot(t, x(:, 1), 'r', t, x(:, 2), 'b');
xlabel('Time (s)');
ylabel('Displacement (m) / Velocity (m/s)');
legend('Displacement', 'Velocity');
title('Mass-Spring-Damper System Response');
grid on;
```

Example 3: Inverted pendulum on a cart

The dynamics of an inverted pendulum on a cart can be described by:

$$
\begin{aligned}
\dot{x}(t) &= v(t),\\
\dot{v}(t) &= \frac{1}{M+m}(F(t) - mg\sin(\theta) - ml\dot{\theta}^2\cos(\theta)),\\
\dot{\theta}(t) &= \omega(t),\\
\dot{\omega}(t) &= \frac{g\sin(\theta)}{l} - \frac{1}{l}(F(t) + ml\dot{\theta}^2\sin(\theta)).
\end{aligned}
$$

`Matlab` code for this task is given below.

```
% Parameters
m = 0.2; % Mass of pendulum
M = 0.5; % Mass of cart
l = 0.5; % Length of pendulum
g = 9.81; % Acceleration due to gravity

% State-space representation
A = [0 1 0 0;
     0 0 -M*g/m 0;
     0 0 0 1;
     0 0 (m+M)*g/(l*m) 0];
B = [0; 1/m; 0; -1/(l*m)];
C = eye(4);
D = zeros(4,1);

% Define system
sys = ss(A,B,C,D);

% Initial conditions
x0 = [0; 0; 0.1; 0]; % Initial position and angle

% Simulate the response
t = 0:0.01:10; % Time vector
[y, t, x] = lsim(sys, zeros(size(t)), t, x0);

% Plot results
figure;
subplot(2,1,1);
plot(t, y(:,1), t, y(:,3));
title('Inverted Pendulum on Cart');
```

```
xlabel('Time (s)');
ylabel('Position (m) / Angle (rad)');
legend('Position', 'Angle');
grid on;
```

6.4 Results to Study Linear Dynamic Systems with Linear Algebra

Key Concept

For a 2×2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, the eigenvalues λ are given by:

$$\lambda = \frac{\operatorname{tr}(A)}{2} \pm \sqrt{\left(\frac{\operatorname{tr}(A)}{2}\right)^2 - \det(A)},$$

where $\operatorname{tr}(A) = a + d$ is the trace and $\det(A) = ad - bc$ is the determinant. Real eigenvalues occur if $\left(\frac{\operatorname{tr}(A)}{2}\right)^2 - \det(A) \geq 0$, while complex eigenvalues occur if $\left(\frac{\operatorname{tr}(A)}{2}\right)^2 - \det(A) < 0$.

Key Concept

To determine the stability of the systems described by the equations

$$u' = Au$$

for the given matrices A, we will find the eigenvalues of each matrix A. The stability can be assessed based on the real parts of the eigenvalues:

- If all eigenvalues have negative real parts, the system is stable (asymptotically stable).
- If at least one eigenvalue has a positive real part, the system is unstable.
- If all eigenvalues have zero real parts, the system is marginally stable.

6.5 Tasks

1. Given the matrix:

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

, find the eigen values, eigen vectors and e^{At}.

SOLUTION

1. Finding Eigenvalues and Eigenvectors

To find the eigenvalues, we solve the characteristic polynomial $\det(A - \lambda I) = 0$.

$$\det(A - \lambda I) = (-1 - \lambda)(-1 - \lambda) - (1)(1) = \lambda^2 + 2\lambda = 0$$

Thus, we have:

$$\lambda(\lambda + 2) = 0$$

This gives the eigenvalues $\lambda_1 = 0$ and $\lambda_2 = -2$.

Now, we find the eigenvectors for each eigenvalue.

For $\lambda_1 = 0$:

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

Solving $A\mathbf{x} = 0$ gives the eigenvector $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

For $\lambda_2 = -2$:

$$A + 2I = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Solving $(A+2I)\mathbf{x} = 0$ gives the eigenvector $\mathbf{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

So, the eigenvalues and corresponding eigenvectors are:

$$\lambda_1 = 0, \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lambda_2 = -2, \quad \mathbf{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Since the eigen values are independent, A is diagonalizable.

2. Computing e^{At}

To find e^{At}, we use the fact that if A is diagonalizable, $A = PDP^{-1}$, then:

$$e^{At} = Pe^{Dt}P^{-1}$$

where D is the diagonal matrix of eigenvalues, $D = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix}$, and P is the matrix of eigenvectors:

$$P = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad P^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The exponential of the diagonal matrix D is:

$$e^{Dt} = \begin{bmatrix} e^{0\cdot t} & 0 \\ 0 & e^{-2t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-2t} \end{bmatrix}$$

Thus:

$$e^{At} = Pe^{Dt}P^{-1} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & e^{-2t} \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Carrying out this multiplication:

$$e^{At} = \frac{1}{2}\begin{bmatrix} 1+e^{-2t} & 1-e^{-2t} \\ 1-e^{-2t} & 1+e^{-2t} \end{bmatrix}$$

`Matlab` code for this task is given below.

```
A = [-1 1; 1 -1];
[eigVecs, eigVals] = eig(A);
syms t;
expAt = expm(A * t);
disp('Eigenvalues:');
disp(diag(eigVals));
disp('Eigenvectors:');
disp(eigVecs);

disp('Matrix exponential e^(At):');
disp(expAt);
```

Output of the code is shown below.

```
Eigenvalues:
    -2
     0
Eigenvectors:
 985/1393       985/1393
-985/1393       985/1393

Matrix exponential e^(At):
[exp((-2*t))/2 + sym(1/2), sym(1/2) - exp((-2*t))/2;
 sym(1/2) - exp((-2*t))/2, exp((-2*t))/2 + sym(1/2)]
```

So the exponential,

$$e^{At} = \begin{bmatrix} \frac{e^{-2t}}{2} + \frac{1}{2} & \frac{1}{2} - \frac{e^{-2t}}{2} \\ \frac{1}{2} - \frac{e^{-2t}}{2} & \frac{e^{-2t}}{2} + \frac{1}{2} \end{bmatrix}$$

2. For the previous matrix write the general solution to $\frac{du}{d}t = Au$ and the specific solution that match $u(0) = (3,1)$. What value is the steady state as $t \to \infty$.

SOLUTION

We use the matrix exponential, e^{At} to find the general solution.

To find the general solution.

If $u(0) = u_0$ is the initial condition, then the general solution is:

$$u(t) = e^{At} u_0$$

Since we have previously found that:

$$e^{At} = \begin{bmatrix} \frac{e^{-2t}}{2} + \frac{1}{2} & \frac{1}{2} - \frac{e^{-2t}}{2} \\ \frac{1}{2} - \frac{e^{-2t}}{2} & \frac{e^{-2t}}{2} + \frac{1}{2} \end{bmatrix},$$

we can write the general solution as:

$$u(t) = \begin{bmatrix} \frac{e^{-2t}}{2} + \frac{1}{2} & \frac{1}{2} - \frac{e^{-2t}}{2} \\ \frac{1}{2} - \frac{e^{-2t}}{2} & \frac{e^{-2t}}{2} + \frac{1}{2} \end{bmatrix} \begin{bmatrix} u_0^1 \\ u_0^2 \end{bmatrix}$$

where $u_0 = \begin{bmatrix} u_0^1 \\ u_0^2 \end{bmatrix}$.

To find the specific solution for $u(0) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$

Using $u_0 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, we substitute into the general solution:

$$u(t) = \begin{bmatrix} \frac{e^{-2t}}{2} + \frac{1}{2} & \frac{1}{2} - \frac{e^{-2t}}{2} \\ \frac{1}{2} - \frac{e^{-2t}}{2} & \frac{e^{-2t}}{2} + \frac{1}{2} \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Equating the terms on both sides,

- **First row:**
$$u^1(t) = \left(\frac{e^{-2t}}{2} + \frac{1}{2}\right) \cdot 3 + \left(\frac{1}{2} - \frac{e^{-2t}}{2}\right) \cdot 1 = 2 + e^{-2t}$$
- **Second row:**
$$u^2(t) = \left(\frac{1}{2} - \frac{e^{-2t}}{2}\right) \cdot 3 + \left(\frac{e^{-2t}}{2} + \frac{1}{2}\right) \cdot 1 = 1 - e^{-2t}$$

Therefore, the specific solution is:

$$u(t) = \begin{bmatrix} 2 + e^{-2t} \\ 1 - e^{-2t} \end{bmatrix}$$

To find the steady State as $t \to \infty$,

As $t \to \infty$, $e^{-2t} \to 0$, so the solution approaches:

$$u(\infty) = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Thus, the steady-state solution as $t \to \infty$ is:

$$u_{\text{steady state}} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

3. Suppose that the time direction is reversed to the given matrix $-A$.

$$\frac{du}{dt} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} u; \qquad \text{with} \quad u_0 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Find $u(t)$ and shows that it *blows up* instead decaying as $t \to \infty$.

SOLUTION

To solve the differential equation

$$\frac{du}{dt} = -Au = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} u$$

with initial condition $u(0) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, we use the exponential matrix $e^{(-A)t}$ to find $u(t)$.

The matrix $A = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$ has eigenvalues $\lambda_1 = 0$ and $\lambda_2 = -2$ with corresponding eigenvectors $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$. The eigenvalues of $-A$ are the negatives of those of A, giving $\mu_1 = 0$ and $\mu_2 = 2$, while the eigenvectors remain $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

We express $e^{(-A)t}$ as:

$$e^{(-A)t} = Pe^{Dt}P^{-1}$$

where

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad P^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Thus,

$$e^{Dt} = \begin{bmatrix} e^{0\cdot t} & 0 \\ 0 & e^{2t} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{2t} \end{bmatrix},$$

and so,

$$e^{(-A)t} = P\begin{bmatrix} 1 & 0 \\ 0 & e^{2t} \end{bmatrix}P^{-1}.$$

Carrying out the multiplication:

$$e^{(-A)t} = \begin{bmatrix} \frac{1+e^{2t}}{2} & \frac{1-e^{2t}}{2} \\ \frac{1-e^{2t}}{2} & \frac{1+e^{2t}}{2} \end{bmatrix}.$$

Using $u(0) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, the general solution becomes:

$$u(t) = e^{(-A)t}u(0) = \begin{bmatrix} \frac{1+e^{2t}}{2} & \frac{1-e^{2t}}{2} \\ \frac{1-e^{2t}}{2} & \frac{1+e^{2t}}{2} \end{bmatrix}\begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Calculating each component:

1. **First row:**

$$u^1(t) = \left(\frac{1+e^{2t}}{2}\right)\cdot 3 + \left(\frac{1-e^{2t}}{2}\right)\cdot 1 = 2 + e^{2t}.$$

2. **Second row:**

$$u^2(t) = \left(\frac{1-e^{2t}}{2}\right)\cdot 3 + \left(\frac{1+e^{2t}}{2}\right)\cdot 1 = 1 - e^{2t}.$$

Thus, the solution is:

$$u(t) = \begin{bmatrix} 2+e^{2t} \\ 1-e^{2t} \end{bmatrix}.$$

As $t \to \infty$, $e^{2t} \to \infty$, so:

$$u(t) \to \begin{bmatrix} 2+e^{2t} \\ 1-e^{2t} \end{bmatrix} \to \begin{bmatrix} \infty \\ -\infty \end{bmatrix}.$$

This shows that $u(t)$ *blows up* as $t \to \infty$, indicating instability due to the positive eigenvalue in $-A$.

4. From their traces and determinant, at what time t do the following matrices change between stable with real eigen values stable with complex eigen values and unstable?

$$A_1 = \begin{bmatrix} 1 & -1 \\ t & -1 \end{bmatrix}; \quad A_2 = \begin{bmatrix} 0 & 4-t \\ 1 & -2 \end{bmatrix}; \quad A_3 = \begin{bmatrix} t & -1 \\ 1 & t \end{bmatrix}.$$

SOLUTION

To determine the times t at which the matrices A_1, A_2, and A_3 transition between stable and unstable behavior, or between real and complex eigenvalues, we examine each matrix's trace and determinant.

For a 2×2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, the eigenvalues λ are given by:

$$\lambda = \frac{\mathrm{tr}(A)}{2} \pm \sqrt{\left(\frac{\mathrm{tr}(A)}{2}\right)^2 - \det(A)},$$

where $\mathrm{tr}(A) = a + d$ is the trace and $\det(A) = ad - bc$ is the determinant. Real eigenvalues occur if $\left(\frac{\mathrm{tr}(A)}{2}\right)^2 - \det(A) \geq 0$, while complex eigenvalues occur if $\left(\frac{\mathrm{tr}(A)}{2}\right)^2 - \det(A) < 0$. Additionally, stability depends on the signs of the real parts of the eigenvalues: stability with real eigenvalues requires non-positive real parts, while instability occurs if there are positive real parts.

Matrix $A_1 = \begin{bmatrix} 1 & -1 \\ t & -1 \end{bmatrix}$

- Trace: $\mathrm{tr}(A_1) = 1 + (-1) = 0$.

- Determinant: $\det(A_1) = (1)(-1) - (-1)(t) = -1 + t = t - 1$.

The condition for real eigenvalues is:

$$\left(\frac{\mathrm{tr}(A_1)}{2}\right)^2 - \det(A_1) = 0^2 - (t-1) \geq 0,$$

or

$$t - 1 \leq 0 \Rightarrow t \leq 1.$$

- For $t \leq 1$: Eigenvalues are real.

- For $t > 1$: Eigenvalues become complex.

To determine stability, observe that:

- For $t < 1$, both eigenvalues will have non-positive real parts, implying stability.
- For $t = 1$, one eigenvalue is zero, and the other is negative, so the system is neutrally stable.
- For $t > 1$, the eigenvalues become complex with real part zero, resulting in oscillatory behavior but not instability in the traditional sense of diverging solutions.

Matrix $A_2 = \begin{bmatrix} 0 & 4-t \\ 1 & -2 \end{bmatrix}$

- Trace: $\mathrm{tr}(A_2) = 0 + (-2) = -2$.

- Determinant: $\det(A_2) = (0)(-2) - (4-t)(1) = -(4-t) = t - 4$.

The condition for real eigenvalues is:

$$\left(\frac{\mathrm{tr}(A_2)}{2}\right)^2 - \det(A_2) = \left(\frac{-2}{2}\right)^2 - (t-4) = 1 - (t-4) \geq 0,$$

or

$$t \leq 5.$$

- For $t \leq 5$: Eigenvalues are real.
- For $t > 5$: Eigenvalues become complex.

For stability:

- For $t \leq 4$, both eigenvalues will have negative real parts, indicating stability.

- At $t = 5$, eigenvalues are real and the system is neutrally stable.
- For $t > 5$, eigenvalues are complex with real part -1, implying a stable spiral.

Matrix $A_3 = \begin{bmatrix} t & -1 \\ 1 & t \end{bmatrix}$

- Trace: $\text{tr}(A_3) = t + t = 2t$.

- Determinant: $\det(A_3) = t \cdot t - (-1)(1) = t^2 + 1$.
The condition for real eigenvalues is:

$$\left(\frac{\text{tr}(A_3)}{2}\right)^2 - \det(A_3) = \left(\frac{2t}{2}\right)^2 - (t^2+1) = t^2 - (t^2+1) = -1,$$

which is always negative. Therefore, eigenvalues of A_3 are always complex.

For stability:

- The eigenvalues have real part t, which means:
- For $t < 0$, the eigenvalues have negative real parts, indicating stability.
- For $t = 0$, the eigenvalues are purely imaginary, resulting in a neutrally stable system with oscillations.
- For $t > 0$, the eigenvalues have positive real parts, indicating instability.

The results can be summarized as follows.

(a) A_1: Real eigenvalues for $t \le 1$; complex eigenvalues for $t > 1$.

(b) A_2: Real eigenvalues for $t \le 5$; complex eigenvalues for $t > 5$.

(c) A_3: Always complex eigenvalues; stable for $t < 0$, neutrally stable for $t = 0$, and unstable for $t > 0$.

5. Decide on the stability or instability of $\frac{dv}{dt} = w$; $\frac{dw}{dt} = v$. Is there a solution to the decay?

SOLUTION

To analyze the stability of the system

$$\frac{dv}{dt} = w, \quad \frac{dw}{dt} = v,$$

we can rewrite it in matrix form as

$$\frac{d}{dt}\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} v \\ w \end{bmatrix}.$$

Letting $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, we find the eigenvalues of A to determine the system's stability.

The eigenvalues λ are solutions to the characteristic equation:

$$\det(A - \lambda I) = 0.$$

For $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, this gives:

$$\begin{vmatrix} -\lambda & 1 \\ 1 & -\lambda \end{vmatrix} = \lambda^2 - 1 = 0.$$

Solving for λ, we find

$$\lambda = \pm 1.$$

In this problem, the eigenvalues $\lambda = 1$ and $\lambda = -1$ indicate that the solutions will consist of both an exponential growth component (from $\lambda = 1$) and an exponential decay component (from $\lambda = -1$). Therefore, this system does not have a solution that purely decays, as there is an unstable component associated with $\lambda = 1$.

The general solution of the system is a linear combination of the eigenvectors associated with $\lambda = 1$ and $\lambda = -1$. If $u(0) = \begin{bmatrix} v(0) \\ w(0) \end{bmatrix}$, the solution $u(t) = \begin{bmatrix} v(t) \\ w(t) \end{bmatrix}$ will take the form:

$$u(t) = c_1 e^t \begin{bmatrix} 1 \\ 1 \end{bmatrix} + c_2 e^{-t} \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

where c_1 and c_2 are constants determined by the initial conditions.

Conclusion
Since the eigenvalue $\lambda = 1$ leads to an exponentially growing solution component e^t, the system is *unstable*. There is no solution that decays entirely as $t \to \infty$ due to the presence of this unstable mode.

6. Decide the stability or instability of $u' = Au$ for the following matrices.

$$(a)\ A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \quad (b)\ A = \begin{bmatrix} 1 & 2 \\ 3 & -1 \end{bmatrix} (c)\ A = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix} \quad (d)\ A = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$$

SOLUTION

Let us analyze the stability of the system for the given matrices using the analysis of eigenvalues.

(a) $A_1 = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$

Characteristic Polynomial:

To find the eigenvalues, we compute the characteristic polynomial:

$$\det(A_1 - \lambda I) = \det \begin{bmatrix} 2-\lambda & 3 \\ 4 & 5-\lambda \end{bmatrix} = (2-\lambda)(5-\lambda) - 12 = \lambda^2 - 7\lambda - 2.$$

Eigenvalues:

We solve the characteristic equation $\lambda^2 - 7\lambda - 2 = 0$:

$$\lambda = \frac{7 \pm \sqrt{49+8}}{2} = \frac{7 \pm \sqrt{57}}{2}.$$

Since both eigenvalues are positive, the system is **unstable**.

(b) $A_2 = \begin{bmatrix} 1 & 2 \\ 3 & -1 \end{bmatrix}$

Characteristic Polynomial:

$$\det(A_2 - \lambda I) = \det\begin{bmatrix} 1-\lambda & 2 \\ 3 & -1-\lambda \end{bmatrix} = (1-\lambda)(-1-\lambda) - 6 = \lambda^2 + \lambda - 7.$$

Eigenvalues:

Solving $\lambda^2 + \lambda - 7 = 0$:

$$\lambda = \frac{-1 \pm \sqrt{1+28}}{2} = \frac{-1 \pm \sqrt{29}}{2}.$$

Since one eigenvalue is positive and the other is negative, the system is **unstable**.

(c) $A_3 = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix}$

Characteristic Polynomial:

$$\det(A_3 - \lambda I) = \det\begin{bmatrix} 1-\lambda & 1 \\ 1 & -2-\lambda \end{bmatrix} = (1-\lambda)(-2-\lambda) - 1 = \lambda^2 + \lambda - 3.$$

Eigenvalues:

Solving $\lambda^2 + \lambda - 3 = 0$:

$$\lambda = \frac{-1 \pm \sqrt{1+12}}{2} = \frac{-1 \pm \sqrt{13}}{2}.$$

Since one eigenvalue is positive and the other is negative, the system is **unstable**.

(d) $A_4 = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$

Characteristic Polynomial:

$$\det(A_4 - \lambda I) = \det\begin{bmatrix} -1-\lambda & -1 \\ -1 & -1-\lambda \end{bmatrix} = (-1-\lambda)^2 - 1 = \lambda^2 + 2\lambda.$$

Eigenvalues:

Solving $\lambda^2 + 2\lambda = 0$:

$$\lambda(\lambda+2) = 0 \implies \lambda_1 = 0, \quad \lambda_2 = -2.$$

Since one eigenvalue is zero and the other is negative, the system is **marginally stable**.

7. Suppose the rabbit population r and the wolf population w are governed by

$$\frac{dr}{dt} = 4r - 2w$$
$$\frac{dw}{dt} = r + w$$

   (a) Is the system is stable, neutrally stable or unstable?
   (b) If initially $r = 300$ and $w = 400$, what will be the population at time t?
   (c) After a long time what is the population of the rabbits to wolves?

SOLUTION

Given that the populations of rabbits r and wolves w are governed by the system of equations:

$$\frac{dr}{dt} = 4r - 2w,$$
$$\frac{dw}{dt} = r + w.$$

We can express the system in matrix form as follows:

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u},$$

where

$$\mathbf{u} = \begin{bmatrix} r \\ w \end{bmatrix}, \quad A = \begin{bmatrix} 4 & -2 \\ 1 & 1 \end{bmatrix}.$$

(a) Stability analysis

To find the equilibrium points, we set the right-hand sides to zero:

$$4r - 2w = 0,$$
$$r + w = 0.$$

From the second equation, we have $w = -r$. Substituting this into the first gives:

$$6r = 0 \implies r = 0 \implies w = 0.$$

Thus, the equilibrium point is $(0,0)$.
Next, we calculate the Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial}{\partial r}(4r-2w) & \frac{\partial}{\partial w}(4r-2w) \\ \frac{\partial}{\partial r}(r+w) & \frac{\partial}{\partial w}(r+w) \end{bmatrix} = \begin{bmatrix} 4 & -2 \\ 1 & 1 \end{bmatrix}.$$

We find the eigenvalues by solving the characteristic polynomial:

$$\det(J - \lambda I) = \det \begin{bmatrix} 4-\lambda & -2 \\ 1 & 1-\lambda \end{bmatrix} = (\lambda - 2)(\lambda - 3) = 0.$$

Thus, the eigenvalues are $\lambda_1 = 2$ and $\lambda_2 = 3$, indicating the system is **unstable**.

(b) Population at time t

To find the populations at time t given the initial conditions $r(0) = 300$ and $w(0) = 400$:
The general solution can be expressed as:

$$\mathbf{u}(t) = c_1 e^{2t} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + c_2 e^{3t} \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Using the initial conditions:

$$\mathbf{u}(0) = c_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 300 \\ 400 \end{bmatrix}.$$

This gives the equations:

$$c_1 + 2c_2 = 300,$$
$$c_1 + c_2 = 400.$$

Subtracting the second from the first gives $c_2 = -100$. Substituting back yields $c_1 = 500$. Thus, the specific solution is:

$$\mathbf{u}(t) = 500e^{2t}\begin{bmatrix}1\\1\end{bmatrix} - 100e^{3t}\begin{bmatrix}2\\1\end{bmatrix}.$$

Calculating $\mathbf{u}(t)$:

$$\begin{bmatrix}u_1(t)\\u_2(t)\end{bmatrix} = \begin{bmatrix}500e^{2t} - 200e^{3t}\\500e^{2t} - 100e^{3t}\end{bmatrix}.$$

(c) Long-term population ratio

As $t \to \infty$:

$$u_1(t) \approx -200e^{3t},$$
$$u_2(t) \approx -100e^{3t}.$$

Calculating the ratio:

$$\text{Ratio} = \frac{500e^{2t} - 200e^{3t}}{500e^{2t} - 100e^{3t}} \approx \frac{-200e^{3t}}{-100e^{3t}} = 2.$$

Thus, the rabbit population approaches a relative ratio of $2:1$ with the wolf population as $t \to \infty$.

8. Convert $y'' = 0$ to a first order system, $\frac{du}{dt} = Au$. Compute e^{At} from the series $I + At + \frac{(At)^2}{2!} + \cdots$ and write the solution $e^{At}u(0)$ starting from $y(0) = 3, y'(0) = 4$. Check that your (y, y') satisfies $y'' = 0$.

SOLUTION

As a first step, the given problem is to converted into matrix form. To convert the second-order equation $y'' = 0$ into a first-order system, we define the state vector **u** as follows:

$$\mathbf{u} = \begin{bmatrix}y\\y'\end{bmatrix}.$$

Then we can express the original second-order equation as a first-order system:

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u},$$

where

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

The system of equations can now be written as:

$$\frac{d}{dt}\begin{bmatrix} y \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} y \\ y' \end{bmatrix}.$$

Given problem has three parts. So will solve it in three steps.

Step 1: Compute e^{At}

To compute the matrix exponential e^{At}, we can use the series expansion:

$$e^{At} = I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \cdots$$

First, calculate A^2:

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

Notice that $A^2 = 0$. Therefore, higher powers of A will also be zero, i.e., $A^n = 0$ for $n \geq 2$.

Now we can simplify the series expansion for e^{At}:

$$e^{At} = I + At.$$

Substituting A:

$$e^{At} = I + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & t \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}.$$

Step 2: Write the solution $e^{At}\mathbf{u}(0)$

Given the initial conditions $y(0) = 3$ and $y'(0) = 4$, we can express the initial state vector $\mathbf{u}(0)$ as:

$$\mathbf{u}(0) = \begin{bmatrix} y(0) \\ y'(0) \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

Now, we can find the solution:

$$\mathbf{u}(t) = e^{At}\mathbf{u}(0) = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 3+4t \\ 4 \end{bmatrix}.$$

Thus, the solution for y and y' is:

$$y(t) = 3 + 4t, \quad y'(t) = 4.$$

Step 3: Check that (y, y') satisfies $y'' = 0$

To check if $y(t)$ satisfies $y'' = 0$, we compute the second derivative:

$$y' = 4 \implies y'' = 0.$$

Thus, the solution $y(t) = 3 + 4t$ indeed satisfies the original equation $y'' = 0$.

9. A higher order equation $y'' + y = 0$ can be written in the form $\frac{du}{dt} = Au$ by introducing the velocity y' as another unknown. Using this approach write A. Find its eigen values and eigen vectors. Also compute the solution starts from $y(0) = 2, y'(0) = 0$.

SOLUTION

Given that $y'' + y = 0$. Now, express y'' in terms of the state vector:

From the equation $y'' + y = 0$, we can rearrange it to express y'':

$$y'' = -y$$

The system can be represented as:

$$\frac{d}{dt}\begin{bmatrix} y \\ y' \end{bmatrix} = \begin{bmatrix} y' \\ -y \end{bmatrix}$$

To convert the second-order differential equation $y'' + y = 0$ into a first-order system, we introduce the state vector

$$\mathbf{u} = \begin{bmatrix} y \\ y' \end{bmatrix}.$$

The first-order system can then be expressed as:

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u},$$

where A is given by:

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

Step 1: Find Eigenvalues of A

To find the eigenvalues, we compute the characteristic polynomial:

$$\det(A - \lambda I) = \det\begin{bmatrix} -\lambda & 1 \\ -1 & -\lambda \end{bmatrix} = \lambda^2 + 1.$$

Setting the determinant to zero gives:

$$\lambda^2 + 1 = 0 \implies \lambda^2 = -1 \implies \lambda = i, -i.$$

Thus, the eigenvalues of A are:

$$\lambda_1 = i, \quad \lambda_2 = -i.$$

Step 2: Find Eigenvectors of A

For $\lambda_1 = i$:

We solve

$$(A - iI)\mathbf{v} = 0,$$

where

$$A - iI = \begin{bmatrix} -i & 1 \\ -1 & -i \end{bmatrix}.$$

Setting up the equations:

1. $-iv + w = 0$ (from the first row) 2. $-v - iw = 0$ (from the second row)

From the first equation, we have $w = iv$. Therefore, the eigenvector corresponding to $\lambda_1 = i$ is

$$\mathbf{v_1} = \begin{bmatrix} 1 \\ i \end{bmatrix}.$$

For $\lambda_2 = -i$:

We solve

$$(A + iI)\mathbf{v} = 0,$$

where

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

Setting up the equations:

1. $iv + w = 0$ (from the first row) 2. $-v + iw = 0$ (from the second row)

From the first equation, we have $w = -iv$. Therefore, the eigenvector corresponding to $\lambda_2 = -i$ is

$$\mathbf{v_2} = \begin{bmatrix} 1 \\ -i \end{bmatrix}.$$

Step 3: Compute the Solution with Initial Conditions

Given the initial conditions $y(0) = 2$ and $y'(0) = 0$, we can express the initial state vector $\mathbf{u}(0)$ as:

$$\mathbf{u}(0) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}.$$

The general solution of the system can be expressed as:

$$\mathbf{u}(t) = e^{At}\mathbf{u}(0).$$

To compute e^{At}, we use the formula:

$$e^{At} = \cos(t)I + \sin(t)A.$$

Calculating A^2:

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

Using the matrix exponential:

$$e^{At} = \cos(t)I + \sin(t)A = \begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix}.$$

Thus, we have:

$$\mathbf{u}(t) = e^{At}\mathbf{u}(0) = \begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2\cos(t) \\ -2\sin(t) \end{bmatrix}.$$

Finally, we can express the solution for $y(t)$:

$$y(t) = 2\cos(t).$$

10. A diagonal matrix $\Lambda = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ satisfies the usual rule $e^{\Lambda(t+T)} = e^{\Lambda t}e^{\Lambda T}$. Verify this.

    (a) Explain why $e^{\Lambda(t+T)} = e^{\Lambda t}e^{\Lambda T}$, using the formula $e^{\Lambda t} = Se^{\Lambda t}S^{-1}$.

    (b) Show that $e^{A+B} = e^Ae^B$ is not true for matrices from the example $A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}$.

SOLUTION

A diagonal matrix $\Lambda = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ satisfies the usual rule $e^{\Lambda(t+T)} = e^{\Lambda t}e^{\Lambda T}$. Verify this.

(a) Explain why $e^{\Lambda(t+T)} = e^{\Lambda t}e^{\Lambda T}$, using the formula $e^{\Lambda t} = Se^{Dt}S^{-1}$.

(b) Show that $e^{A+B} \neq e^Ae^B$ for matrices A and B from the example $A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}$.

Proof:

1. To show that $e^{\Lambda(t+T)} = e^{\Lambda t}e^{\Lambda T}$, we can express $e^{\Lambda t}$ in terms of a similarity transformation:

Let S be a matrix such that $\Lambda = SDS^{-1}$, where D is a diagonal matrix. Since Λ is already diagonal, we can choose $S = I$ (the identity matrix). Then we have:

$$e^{\Lambda t} = Se^{Dt}S^{-1} = e^{Dt} = \begin{bmatrix} e^t & 0 \\ 0 & e^{2t} \end{bmatrix}$$

Now, consider $e^{\Lambda(t+T)}$:

$$e^{\Lambda(t+T)} = e^{\begin{bmatrix} 1(t+T) & 0 \\ 0 & 2(t+T) \end{bmatrix}} = \begin{bmatrix} e^{t+T} & 0 \\ 0 & e^{2(t+T)} \end{bmatrix}$$

Next, compute $e^{\Lambda t}e^{\Lambda T}$:

$$e^{\Lambda t}e^{\Lambda T} = \begin{bmatrix} e^t & 0 \\ 0 & e^{2t} \end{bmatrix} \begin{bmatrix} e^T & 0 \\ 0 & e^{2T} \end{bmatrix} = \begin{bmatrix} e^te^T & 0 \\ 0 & e^{2t}e^{2T} \end{bmatrix}$$

Since $e^{t+T} = e^te^T$ and $e^{2(t+T)} = e^{2t}e^{2T}$, we conclude that:

$$e^{\Lambda(t+T)} = \begin{bmatrix} e^{t+T} & 0 \\ 0 & e^{2(t+T)} \end{bmatrix} = \begin{bmatrix} e^te^T & 0 \\ 0 & e^{2t}e^{2T} \end{bmatrix} = e^{\Lambda t}e^{\Lambda T}$$

Thus, we have verified that $e^{\Lambda(t+T)} = e^{\Lambda t}e^{\Lambda T}$.

2. To show that $e^{A+B} \neq e^A e^B$ for the matrices

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix},$$

we will use the power series expansion for the matrix exponential. The matrix exponential e^X for any square matrix X is given by the series:

$$e^X = I + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \cdots$$

Step 1: Compute e^A

First, we compute e^A:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

Calculating powers of A:

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

Thus, for $n \geq 2$, $A^n = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. Therefore, we have:

$$e^A = I + A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Now we compute e^B:

$$e^B = I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \cdots$$

Calculating powers of B:

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

Thus, for $n \geq 2$, $B^n = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. Therefore, we have:

$$e^B = I + B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

Next, we compute $A + B$:

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

Now compute e^{A+B}:

$$e^{A+B} = I + (A+B) + \frac{(A+B)^2}{2!} + \frac{(A+B)^3}{3!} + \cdots$$

Calculating powers of $A + B$:

$$\begin{aligned}
(A+B)^2 &= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \\
&= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\
ll^{rly}, (A+B)^3 &= (A+B)(A+B)^2 \\
&= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \\
ll^{rly}, (A+B)^4 &= (A+B)(A+B)^3 \\
&= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I
\end{aligned}$$

This results in a periodic pattern where:

$$(A+B)^{2n} = (-1)^n I \quad \text{and} \quad (A+B)^{2n+1} = \begin{bmatrix} 0 & (-1)^n \\ (-1)^n & 0 \end{bmatrix}$$

Thus, the series for e^{A+B} yields:

$$e^{A+B} = I + (A+B) + \frac{(A+B)^2}{2!} + \frac{(A+B)^3}{3!} + \frac{(A+B)^4}{4!} + \cdots$$

This can be rearranged into sine and cosine series, leading to:

$$e^{A+B} = \begin{bmatrix} \cos(1) & -\sin(1) \\ \sin(1) & \cos(1) \end{bmatrix}$$

Finally, we compute $e^A e^B$:

$$e^A e^B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$$

We have:

$$e^{A+B} = \begin{bmatrix} \cos(1) & -\sin(1) \\ \sin(1) & \cos(1) \end{bmatrix} \quad \text{and} \quad e^A e^B = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$$

Since $e^{A+B} \neq e^A e^B$, we conclude that:

$$e^{A+B} \neq e^A e^B$$

RESULTS

1. Linear time invariant systems are solved using tools of linear algebra.
2. Some linear differential equations of practical importance are solved using linear algebra.

7 | Assignment 72 Reverse Engineering in Linear Algebra

7.1 Subspaces in Linear Systems

In a system of linear equations, the solution space and the structure of the system can be understood in terms of subspaces like the **row space**, **column space**, and **null space**.

- **Row Space (or Range of A^T)**: The span of the rows of the coefficient matrix A. The dimension of the row space is equal to the rank of A.
- **Null Space**: The set of all vectors $\mathbf{x}$ such that $A\mathbf{x} = 0$. The dimension of the null space is equal to the *nullity* of A, which measures the number of free variables in the system.

7.1.1 Rank-Nullity Theorem

The *rank-nullity theorem* states:

$$\text{Rank}(A) + \text{Nullity}(A) = n$$

Where:

- n is the number of columns of A,
- **Rank** is the dimension of the row space,
- **Nullity** is the dimension of the null space.

Given that the general solution to the system involves a free parameter α, this implies that there is a **non-trivial null space**, and the rank of the matrix will be less than the number of variables (columns).

7.2 A Problem

We are provided with a general solution:

$$X = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

Here:

- $\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ is a **particular solution**.
- $\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$ is the **null space element**, indicating a solution to the homogeneous equation $A\mathbf{x} = 0$.

Step-by-Step Plan

1. **Null Space and Row Space**: The null space contains one vector, $\begin{bmatrix}1\\2\\2\end{bmatrix}$, indicating that the *nullity* of the system is 1. Therefore, the *rank* of the matrix is $3-1=2$, implying the row space is 2-dimensional.

2. **Orthogonality Between Row Space and Null Space**: The row space is *orthogonal* to the null space. Every vector in the row space must be orthogonal to $\begin{bmatrix}1\\2\\2\end{bmatrix}$, the null space vector. Hence, we can construct the rows of the matrix A by finding vectors orthogonal to the null space vector.

3. Finding the Row Space

To find vectors that form the row space, we need to find vectors that are **orthogonal** to the null space vector $\mathbf{n} = \begin{bmatrix}1\\2\\2\end{bmatrix}$.

Condition for Orthogonality

For any vector $\mathbf{v} = \begin{bmatrix}v_1\\v_2\\v_3\end{bmatrix}$ to be in the row space, it must satisfy:

$$\mathbf{v}\cdot\mathbf{n} = 0$$

That is:

$$v_1(1) + v_2(2) + v_3(2) = 0$$

This simplifies to:

$$v_1 + 2v_2 + 2v_3 = 0 \tag{1}$$

This equation represents a plane in $\mathbb{R}^3$, and the vectors lying on this plane are orthogonal to the null space vector.

Choosing Basis Vectors for the Row Space

We need two independent vectors that satisfy equation (1) to form a basis for the row space.

- **First row vector**: Set $v_2 = 1$ and $v_3 = 0$, then from equation (1):

$$v_1 + 2(1) + 2(0) = 0 \implies v_1 = -2$$

So, the first row vector is:

$$\mathbf{v}_1 = \begin{bmatrix}-2\\1\\0\end{bmatrix}$$

- **Second row vector**: Set $v_2 = 0$ and $v_3 = 1$, then from equation (1):

$$v_1 + 2(0) + 2(1) = 0 \implies v_1 = -2$$

So, the second row vector is:

$$\mathbf{v}_2 = \begin{bmatrix} -2 \\ 0 \\ 1 \end{bmatrix}$$

These two vectors form a basis for the row space.

4. Constructing the Matrix A

Now, we can form the matrix A whose rows are the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. The matrix is:

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

5. Verification

Let's verify that the null space of A contains the vector $\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$:

Compute $A\mathbf{x}$, where $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$:

$$A\mathbf{x} = \begin{bmatrix} -2 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -2(1) + 1(2) + 0(2) \\ -2(1) + 0(2) + 1(2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Thus, $\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$ is indeed in the null space of A.

7.3 Block Diagonal Matrices And Its Inverses

In linear algebra, a **block diagonal matrix** is a special type of block matrix where the matrix is divided into smaller blocks along the diagonal, and all other entries outside these diagonal blocks are zero. Block diagonal matrices simplify the representation of large matrices and are commonly encountered in applications such as systems of linear equations, matrix decomposition, and in numerical methods.
A block diagonal matrix B can be written in the form:

$$B = \begin{bmatrix} B_1 & 0 & 0 & \dots & 0 \\ 0 & B_2 & 0 & \dots & 0 \\ 0 & 0 & B_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & B_k \end{bmatrix}$$

Where each B_i is a square matrix of arbitrary size, and the off-diagonal blocks are zero matrices of appropriate dimensions. The size of the entire block diagonal matrix is the sum of the sizes of the diagonal blocks.

For example, for $B_1 \in \mathbb{R}^{m_1\times m_1}$, $B_2 \in \mathbb{R}^{m_2\times m_2}$, and $B_k \in \mathbb{R}^{m_k\times m_k}$, the block diagonal matrix B is in $\mathbb{R}^{(m_1+m_2+\cdots+m_k)\times(m_1+m_2+\cdots+m_k)}$.

7.3.1 Properties of Block Diagonal Matrices

Block diagonal matrices have several important properties:

- **Sparsity**: All off-diagonal blocks are zero matrices, meaning that only diagonal blocks contain non-zero elements.
- **Addition**: The sum of two block diagonal matrices is another block diagonal matrix where the diagonal blocks are the sum of the corresponding diagonal blocks.
- **Multiplication**: The product of two block diagonal matrices is another block diagonal matrix, where the diagonal blocks are the products of the corresponding diagonal blocks. If B and C are block diagonal matrices, then:

$$B\cdot C = \begin{bmatrix} B_1C_1 & 0 & \dots & 0 \\ 0 & B_2C_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B_kC_k \end{bmatrix}$$

7.4 Inverse of Block Diagonal Matrices

The inverse of a block diagonal matrix is obtained by inverting each diagonal block separately, provided that each diagonal block is invertible. This simplifies the computation of the inverse for large matrices, as the inversion process is localized to the smaller blocks along the diagonal.

7.4.1 Inverting a block diagonal matrix

Let B be a block diagonal matrix as described earlier:

$$B = \begin{bmatrix} B_1 & 0 & 0 & \dots & 0 \\ 0 & B_2 & 0 & \dots & 0 \\ 0 & 0 & B_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & B_k \end{bmatrix}$$

If each block B_i is invertible, then the inverse of B is given by:

$$B^{-1} = \begin{bmatrix} B_1^{-1} & 0 & 0 & \dots & 0 \\ 0 & B_2^{-1} & 0 & \dots & 0 \\ 0 & 0 & B_3^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & B_k^{-1} \end{bmatrix}$$

7.4.2 Verifying the inverse

To verify that B^{-1} is indeed the inverse of B, we compute $B\cdot B^{-1}$ and check whether it results in the identity matrix.

$$B\cdot B^{-1}=\begin{bmatrix}B_1&0&0&\dots&0\\0&B_2&0&\dots&0\\0&0&B_3&\dots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&\dots&B_k\end{bmatrix}\begin{bmatrix}B_1^{-1}&0&0&\dots&0\\0&B_2^{-1}&0&\dots&0\\0&0&B_3^{-1}&\dots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&\dots&B_k^{-1}\end{bmatrix}$$

The product of the block diagonal matrices is:

$$B\cdot B^{-1}=\begin{bmatrix}B_1B_1^{-1}&0&0&\dots&0\\0&B_2B_2^{-1}&0&\dots&0\\0&0&B_3B_3^{-1}&\dots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&\dots&B_kB_k^{-1}\end{bmatrix}=\begin{bmatrix}I&0&0&\dots&0\\0&I&0&\dots&0\\0&0&I&\dots&0\\\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&0&\dots&I\end{bmatrix}$$

Thus, $B\cdot B^{-1}=I$, confirming that B^{-1} is indeed the inverse of B.

7.4.3 Applications

Block diagonal matrices and their inverses are used in various applications:

- **Solving Linear Systems**: When systems of linear equations are block structured, block diagonal matrices allow for efficient computation and parallelization of the solution process.
- **Matrix Decompositions**: Block diagonal forms appear in matrix factorizations such as Jordan canonical form and block LU decomposition.
- **Numerical Linear Algebra**: Inverting large sparse matrices by breaking them into block diagonal parts facilitates efficient computations.

Block diagonal matrices provide a structured way to handle large systems by dividing the problem into smaller independent blocks. Their inverses are computed by inverting each diagonal block separately, which simplifies calculations in many practical applications, especially in numerical methods and matrix decompositions.

7.5 Task

1. Find a 2×4 matrix and 2×1 b vector such that general solution is given by $\begin{bmatrix}1\\2\\3\\4\end{bmatrix}+\alpha\begin{bmatrix}1\\1\\1\\1\end{bmatrix}+\beta\begin{bmatrix}1\\1\\-1\\-1\end{bmatrix}$

SOLUTION

The null space of A is spanned by the vectors:

$$\mathbf{v}_1=\begin{bmatrix}1\\1\\1\\1\end{bmatrix},\quad\mathbf{v}_2=\begin{bmatrix}1\\1\\-1\\-1\end{bmatrix}$$

Thus, the rows of A must be orthogonal to both $\mathbf{v}_1$ and $\mathbf{v}_2$.

Let the first row of A be $\mathbf{r}_1=\begin{bmatrix}a_1&a_2&a_3&a_4\end{bmatrix}$. The orthogonality conditions are:

$$a_1+a_2+a_3+a_4=0 \tag{1}$$

$$a_1+a_2-a_3-a_4=0 \tag{2}$$

Adding these gives:

$$2(a_1 + a_2) = 0 \quad \Rightarrow \quad a_1 = -a_2$$

Subtracting these gives:

$$2(a_3 + a_4) = 0 \quad \Rightarrow \quad a_3 = -a_4$$

Thus, the first row of A is:

$$\mathbf{r}_1 = \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}$$

Noe let the second row of A be $\mathbf{r}_2 = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix}$. The orthogonality conditions are:

$$b_1 + b_2 + b_3 + b_4 = 0 \tag{3}$$

$$b_1 + b_2 - b_3 - b_4 = 0 \tag{4}$$

As before, we find $b_1 = -b_2$ and $b_3 = -b_4$, so:

$$\mathbf{r}_2 = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}$$

Thus, the matrix A is:

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

To finding $\mathbf{b}$

We know that the particular solution $\mathbf{x}_p = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ must satisfy $A\mathbf{x}_p = \mathbf{b}$. Now, compute $A\mathbf{x}_p$:

$$A\mathbf{x}_p = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1-2+3-4 \\ 1-2-3+4 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Thus, $\mathbf{b} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$.

The matrix A and vector $\mathbf{b}$ are:

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

2. A 54×37 matrix has rank 31. What are dimensions of all 4 fundamental subspaces?

SOLUTION

The dimension of the null space can be calculated using the formula:

$$\dim(\text{null space}) = n - \text{rank}(A)$$

where n is the number of columns of the matrix. Similarly, the dimension of the left null space can be calculated using the formula:

$$\dim(\text{left null space}) = m - \text{rank}(A)$$

where m is the number of rows of the matrix.

Number of rows, $m = 54$, number of columns, $n = 37$, rank of the matrix, $\text{rank}(A) = 31$.

7.5.1 Calculating the Dimensions

Step 1: Calculate the Dimension of the Row Space

$$\dim(\text{row space}) = \text{rank}(A) = 31$$

Step 2: Calculate the Dimension of the Column Space

$$\dim(\text{column space}) = \text{rank}(A) = 31$$

Step 3: Calculate the Dimension of the Null Space

$$\dim(\text{null space}) = n - \text{rank}(A) = 37 - 31 = 6$$

Step 4: Calculate the Dimension of the Left Null Space

$$\dim(\text{left null space}) = m - \text{rank}(A) = 54 - 31 = 23$$

The dimensions of the four fundamental subspaces are:

- Row Space: dim(row space) = 31
- Column Space: dim(column space) = 31
- Null Space: dim(null space) = 6
- Left Null Space: dim(left null space) = 23

3. Construct a matrix A with eigenvalues 1 and 3 and corresponding eigenvectors $(1,2)^T$ and $(1,1)^T$. Is such matrix unique?. Use matlab.

SOLUTION

Construct a matrix A with eigenvalues 1 and 3 and corresponding eigenvectors

$$v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

To construct the matrix A, we follow these steps:

$$\text{Eigenvalues:} \quad \lambda_1 = 1, \quad \lambda_2 = 3$$
$$\text{Eigenvectors:} \quad v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

A matrix A can be constructed using the eigenvalue-eigenvector pairs. If V is the matrix whose columns are the eigenvectors and D is the diagonal matrix of eigenvalues, then:

$$A = VDV^{-1}$$

The matrices are defined as follows:

$$V = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$$

To find A, we need to calculate V^{-1}:

$$V^{-1} = \frac{1}{\det(V)} \text{adj}(V)$$

where the determinant $\det(V) = 1\cdot 1 - 1\cdot 2 = -1$, and the adjugate of V is:

$$\text{adj}(V) = \begin{bmatrix} 1 & -1 \\ -2 & 1 \end{bmatrix}$$

Thus, we have:

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

Now we can compute A:

$$A = VDV^{-1}$$

Calculating this, we get:

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

Upon calculation, we find:

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

`Matlab` code for this task is given below.

```
lambda1 = 1;
lambda2 = 3;
v1 = [1; 2];
v2 = [1; 1];
V = [v1, v2];
D = [lambda1, 0; 0, lambda2];
V_inv = inv(V);
A = V * D * V_inv;
disp('Matrix A:');
disp(A);

[eigVec, eigVal] = eig(A);
disp('Eigenvalues:');
disp(diag(eigVal));
disp('Eigenvectors:');
disp(eigVec);
```

Output of the code is shown below.

```
Matrix A:
     5    -2
     4    -1
Eigenvalues:
     3
     1
Eigenvectors:
    0.7071    0.4472
    0.7071    0.8944
```

Uniqueness of the Matrix A

The matrix A constructed in this way is not unique. This is because any scalar multiple of an eigenvector results in the same eigenvalue. For example, the eigenvector $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ can be scaled by any non-zero constant c to produce $\begin{bmatrix} c \\ 2c \end{bmatrix}$, while still corresponding to the same eigenvalue. Therefore, any matrix A that satisfies the eigenvalue-eigenvector conditions will be a valid solution, leading to a whole family of matrices A that satisfy the given eigenvalues and eigenvectors.

4. Find inverse of the following matrix. Use matlab and verify that inverse is again block diagonal.

SOLUTION

```

b1=[1 2; 2 1];
b2=[3 1 3; 1 3 3; 2 1 3];
b3=[1 2; 2 1];
format rational;
BM=blkdiag(b1,b2,b3);
disp('The Block Diagonal Matrix is');
disp(BM);
disp('Inverse of the Block Diagonal Matrix is')
disp(inv(BM));
```

Output of the code is shown below.

```
The Block Diagonal Matrix is
       1            2            0            0            0            0
       2            1            0            0            0            0
       0            0            3            1            3            0
       0            0            1            3            3            0
       0            0            2            1            3            0
       0            0            0            0            0            1
       0            0            0            0            0            2

Inverse of the Block Diagonal Matrix is

      -1/3          2/3          0            0            0            0
       2/3         -1/3          0            0            0            0
       0            0            1            *           -1            0
       0            0            1/2          1/2         -1            0
       0            0           -5/6         -1/6          4/3          0
       0            0            0            0            0           -1/
       0            0            0            0            0            2/
```

From the code output, it is clear that *inverse of a block diagonal matrix is again block diagonal.*

5. Create a 5x5 block diagonal integer valued matrix of the form

$$\begin{bmatrix} A_{2\times 2} & 0_{2\times 2} & 0_{2\times 1} \\ 0_{2\times 2} & B_{2\times 2} & 0_{2\times 1} \\ 0_{1\times 2} & 0_{1\times 2} & C_{1\times 1} \end{bmatrix}.$$

SOLUTION

`Matlab` code for this task is given below.

```
A=[1 2; 2 1];
B=[2 3; 3 2];
C=[2];
```

```
BM2=blkdiag(A,B,C);
disp("Required Matrix is:")
disp(BM2)
```

Output of the code is shown below.

```
Required Matrix is:
         1          2          0          0          0
         2          1          0          0          0
         0          0          2          3          0
         0          0          3          2          0
         0          0          0          0          2
```

RESULTS

1. Creating a linear system with required solution is revisited.
2. Concept of block diagonal matrix is revisited and computationally verified some important results.

8 | Assignment 73 Compressed Sensing and Dynamic Mode Decomposition

8.1 Introduction

Compressed sensing (CS) is a revolutionary signal processing technique that breaks the traditional barriers of the Nyquist-Shannon sampling theorem. Traditionally, to avoid information loss, a signal needs to be sampled at least twice the highest frequency present in the signal (Nyquist rate). However, many signals of interest are sparse or compressible in some domain, meaning they can be described by only a small number of non-zero coefficients. Compressed sensing exploits this sparsity to reconstruct signals from a significantly smaller number of samples than conventional methods would require.

8.2 Theoretical Background

In theory, the relationship between the original signal s, its DCT representation x, and the compressed measurements y can be expressed as follows:

$$x = \psi s$$

$$y = \psi_{\text{samp}} \times x$$

Here, ψ is the DCT basis, ψ_{samp} is the sampled basis, and y represents the compressed measurements.

8.2.1 Lagrangian Multiplier Formulation

The optimization problem in compressed sensing can be formulated as a Lagrangian multiplier problem:

$$\min_s \|y - \psi_{\text{samp}} \times s\|_2 + \lambda \|s\|_1$$

The Lagrangian function $\mathscr{L}$ for this optimization problem can be expressed as:

$$\mathscr{L}(s, \lambda) = \|y - \psi_{\text{samp}} \times s\|_2 + \lambda \|s\|_1$$

The optimization problem can be summarized as:

$$\min_s \quad \|y - \psi_{\text{samp}} \times s\|_2 + \lambda \|s\|_1$$

8.2.2 Relevance Across Various Fields

The impact of compressed sensing extends into many fields:

- **Fluid Dynamics**: One of the most prominent applications of DMD is in fluid dynamics, where it is used to identify and analyze coherent structures within fluid flows. These coherent structures are recurring patterns, such as vortices, that significantly influence the flow behavior. Traditional methods like Fourier or wavelet analysis may struggle with highly non-stationary data, but DMD provides a way to decompose the flow into modes that capture both spatial and temporal dynamics. This allows for better understanding of phenomena like turbulence, boundary layer separation, and instabilities. The extracted modes can be used for flow control, prediction of future flow states, and reducing the complexity of fluid simulations through model reduction.

- **Neuroscience**: DMD is also finding applications in neuroscience, particularly in analyzing brain activity. By applying DMD to data obtained from techniques like functional MRI (fMRI) or electroencephalography (EEG), researchers can decompose brain activity into spatial-temporal modes. These modes represent coherent brain regions and their dynamic evolution over time. DMD helps to identify brain states, neural circuits, and connectivity patterns. It also aids in understanding how different regions of the brain interact dynamically in response to stimuli, potentially offering insights into neurological disorders and cognitive processes.

- **Control Systems**: In the field of control systems, DMD is used to identify and predict the behavior of dynamic systems from data. When a system's equations are difficult to model explicitly, DMD provides a data-driven approach to identify key dynamics. For instance, in autonomous vehicle control or robotic systems, DMD can analyze sensor data to model the system's dynamics in real time and predict its future states. This information is crucial for designing control strategies, stabilizing systems, and making them more responsive to changes. DMD also aids in fault detection and control optimization by revealing the underlying modes that govern system behavior.

- **Medical Imaging**: In MRI, fewer measurements can be taken, reducing scanning time while maintaining image quality.

- **Radar and Communications**: CS can reduce the bandwidth required for signal transmission.

- **Astronomy**: It enables high-resolution imaging from sparse and noisy data.

- **Data Compression**: CS can be used for data compression by directly acquiring compressed signals, reducing storage and transmission costs.

8.3 Various Approaches in Compressed Sensing

Several techniques are employed in compressed sensing to reconstruct a signal from a limited number of measurements:

- **L1-Norm Minimization**: Given that sparse signals have a small number of non-zero coefficients, minimizing the L1-norm is a powerful approach to recovering the sparse solution.

- **Orthogonal Matching Pursuit (OMP)**: This greedy algorithm iteratively selects the columns of the measurement matrix that best correlate with the current residual of the signal.

The core idea behind these methods is to solve an underdetermined system of linear equations by promoting sparsity in the solution.

8.4 Dynamic Mode Decomposition (DMD)

Dynamic Mode Decomposition (DMD) is a data-driven technique that provides insight into the underlying dynamic structure of a system. DMD approximates the system's behavior as a linear combination of dynamic modes, each associated with a particular growth rate and frequency. Unlike traditional decomposition methods (like Fourier or wavelet transforms), DMD is specifically designed for analyzing time-dependent systems, capturing both spatial and temporal evolution.

8.4.1 Applications of DMD

DMD has broad applications:

- **Fluid Dynamics**: It captures coherent structures in fluid flow.
- **Neuroscience**: Helps in modeling brain activities from fMRI data.
- **Control Systems**: Used to identify and predict the behavior of dynamic systems.

8.5 Comparison: JPEG, MPEG Compression and DMD Approach

- **JPEG/MPEG Compression**: These widely-used compression methods work by transforming the data (using discrete cosine transform, DCT), quantizing the transform coefficients, and encoding only the significant coefficients, thus achieving compression. JPEG primarily compresses images, while MPEG handles video, reducing file sizes significantly while preserving perceptual quality.

Key difference with DMD: JPEG/MPEG rely on transforming the entire signal before compression, whereas DMD focuses on decomposing the temporal dynamics and spatial structures of the data.

8.6 Core Question: Can We Directly Get a Compressed Version by Data Acquisition Itself?

One of the fundamental questions in compressed sensing is: *Can we avoid acquiring all the data and instead directly capture a compressed version?*

8.6.1 Solution

The answer lies in the groundbreaking work of **Claude Shannon** and his sampling theorem, which laid the foundation for signal sampling. However, it was **Candes, Tao, Romberg, and Donoho** who introduced the idea that it's possible to reconstruct a signal from fewer samples than traditionally required, provided the signal is sparse. Their work showed that by using random projections and solving an L1 minimization problem, a sparse signal can be recovered with high accuracy.
Their research fundamentally changed how we think about data acquisition—compressed sensing allows the data collection process itself to be optimized for sparsity, ensuring that fewer measurements are needed to recover the original signal.

8.7 Example of Compressive Sampling

Consider the signal:

$$f(t) = \sin(73 + 2\pi t) + \sin(531 + 2\pi t)$$

We will capture the signal for one second at a rate of 4096 samples per second, which corresponds to 4096 total samples.
`Matlab` code for this task is given below.

```
clc; clear;

% Sampling rate and time vector
Fs = 4096;          % 4096 samples per second
T = 1;              % Signal duration in seconds
t = linspace(0, T, Fs*T); % Time vector

% Generate the signal
f = sin(73*2*pi*t) + sin(531*2*pi*t);

% Plot the original signal
figure;
plot(t, f, 'LineWidth', 1.5);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

8.7.1 Compressing the signal below Nyquist rate

We now simulate random sampling at a rate below the Nyquist rate using compressed sensing. The measurement matrix Φ is randomly generated to represent the compressed measurements.

8.8 L1 Optimization Problem and DMD

Now that we have the signal and its compressed measurements, we proceed with the L1 minimization formulation using **Dynamic Mode Decomposition** (DMD) and solve the reconstruction problem using the CVX solver.

8.8.1 L1 optimization formulation

The optimization problem is formulated as follows:

$$\min_{\theta_t} \|\theta_t\|_1 \quad \text{subject to} \quad \|y(t) - \Phi\Psi\theta_t\|_2 \leq \epsilon$$

where θ_t represents the sparse coefficients of the signal in the DCT basis Ψ, and $y(t)$ is the vector of compressed measurements.
`Matlab` code for this task is given below.

```

clc; clear;

% Sampling rate and time vector
Fs = 4096;          % 4096 samples per second
T = 1;              % Signal duration in seconds
t = linspace(0, T, Fs*T); % Time vector

% Generate the original sampled signal
x = sin(73*2*pi*t) + sin(531*2*pi*t); % Sampled version of the
    signal

% Plot the original sampled signal
figure;
```

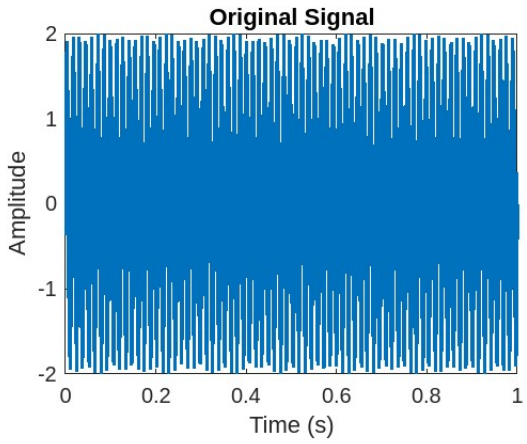
```
plot(t, x, 'LineWidth', 1.5);
title('Original Sampled Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% DCT basis for sparse representation
n = length(x);                % Length of the original signal
Psi = idct(eye(n));           % DCT basis matrix

% Number of compressed measurements
p = 256;                      % Number of samples to take
sampling_indices = randperm(n, p); % Random sampling indices
y = x(sampling_indices);      % Compressed measurements

% Define the sampled basis matrix
psi_samp = Psi(sampling_indices, :); % Sampled DCT basis matrix

% Lagrangian regularization parameter
lambda = 0.02;

% Solve the optimization problem using CVX with Lagrangian
    multipliers
cvx_begin quiet
    variable s(n)                 % Coefficients of the DCT (should
        match the original signal length)
    minimize(lambda * norm(s, 1) + norm(y' - psi_samp * s, 2) ) %
        Objective function
cvx_end

% Reconstruct the signal from the estimated coefficients
x_reconstructed = Psi * s; % Reconstruct the signal using the DCT
    basis

% Ensure that the reconstructed signal has the same length as the
    original signal
x_reconstructed = x_reconstructed'; % Transpose to match dimensions

% Plot the original and reconstructed signals
figure;
subplot(2,1,1);
plot(t, x, 'LineWidth', 1.5);
title('Original Sampled Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, x_reconstructed, 'LineWidth', 1.5);
title('Reconstructed Signal from DCT Coefficients');
xlabel('Time (s)');
ylabel('Amplitude');

% Compute reconstruction error
```

```
reconstruction_error = norm(x - x_reconstructed) / norm(x);
fprintf('Reconstruction Error: %f\n', reconstruction_error);
```

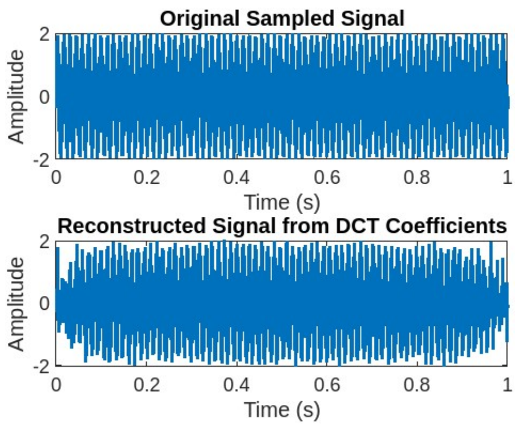
This code uses a **DCT basis** to represent the signal sparsely and solves the L1 optimization problem using the CVX solver. The reconstructed signal is compared to the original, and the reconstruction error is computed.
The reconstruction error is shown in the output:

```
Reconstruction Error: 0.252357
```

A comparison of original ,sampled and reconstructed signals are shown in Figure 8.1.



(a) Original signal.



(b) Sampled and reconstructed signals.

Figure 8.1: Optimizing reconstruction error using L1 optimization.

8.9 Extension: Compressed Sensing with a Sensing Matrix

In the final step, we extend the discussion to compressed sensing using a sensing matrix. The sensing matrix Φ plays a crucial role in the compressed sensing framework by projecting the high-dimensional signal into a lower-dimensional space while preserving the key information for recovery. In practical applications, the sensing matrix can be designed to optimize recovery performance in terms of the sparsity and structure of the signal.

8.10 Tasks

1. Generate a different signal with three frequencies , do random sampling and reconstruct the signal. Use appropriate random matrix as sensing matrix.

SOLUTION

Consider a continuous-time signal $x(t)$ composed of three sinusoidal components:

$$x(t) = \sin(100 \times 2\pi t) + \sin(400 \times 2\pi t) + \sin(900 \times 2\pi t),$$

where the signal is sampled at a rate of 4096 Hz for a duration of 1 second, yielding 4096 samples. Let the discrete-time sampled signal be denoted as $x \in \mathbb{R}^{4096}$.

Random sensing matrix

In compressed sensing, we use a random Gaussian matrix as the sensing matrix to reduce the dimensionality of the measurements. Let $A \in \mathbb{R}^{p \times n}$ represent a Gaussian random matrix, where each entry is drawn from a normal distribution $\mathcal{N}(0,1)$, and $n = 4096$ is the length of the signal.

The compressed measurement vector $y \in \mathbb{R}^p$ is obtained by projecting the signal x onto the random matrix:

$$y = A \times x,$$

where $p = 256$ is the number of compressed measurements.

Optimization problem with Lagrangian Multiplier

The goal of compressed sensing is to recover the original signal x by solving an optimization problem that minimizes the ℓ_1-norm of the sparse coefficient vector $s \in \mathbb{R}^{4096}$ in the DCT domain, subject to the constraint that the measurement vector y is consistent with the compressed sensing model.

We define the DCT basis matrix Ψ such that the DCT coefficients s are related to the original signal x by:

$$x = \Psi s,$$

where s is the sparse representation of the signal in the DCT domain.

The optimization problem with a Lagrangian formulation is given by:

$$\min_s \left(\lambda \|s\|_1 + \|y - A\Psi s\|_2\right),$$

where λ is the regularization parameter that controls the trade-off between sparsity and the reconstruction error.

CVX optimization formulation

The above optimization problem can be solved using the CVX solver in MATLAB, with the following formulation:

```
cvx_begin quiet
            variable s(n)
            minimize(λ×norm(s, 1)+norm(y - A * Psi * s, 2))
      cvx_end
```

This approach minimizes the ℓ_1-norm of the DCT coefficients s, promoting sparsity, while also minimizing the ℓ_2-norm of the reconstruction error between the measured vector y and the projected signal $A\Psi s$. `Matlab` code for this task is given below.

```
clc; clear;
Fs = 4096;          % 4096 samples per second
T = 1;              % Signal duration in seconds
t = linspace(0, T, Fs*T);
x = sin(100*2*pi*t) + sin(400*2*pi*t) + sin(900*2*pi*t);
figure;
plot(t, x, 'LineWidth', 1.5);
title('Original Sampled Signal with Three Frequencies');
xlabel('Time (s)');
ylabel('Amplitude');

% DCT basis for sparse representation
n = length(x);                  % Length of the original signal
Psi = idct(eye(n));             % DCT basis matrix
p = 256;
```

```
% Generate a random sensing matrix (Gaussian)
A = randn(p, n);
y = A * x';
lambda = 0.02;
% Solve the optimization problem using CVX with Lagrangian
    multipliers
cvx_begin quiet
    variable s(n)
    minimize(lambda * norm(s, 1) + norm(y - A * Psi * s, 2))
cvx_end

% Reconstruct the signal from the estimated coefficients
x_reconstructed = Psi * s; % Reconstruct the signal using the
    DCT basis
x_reconstructed = x_reconstructed'; % Transpose to match
    dimensions
% Compute reconstruction error
reconstruction_error = norm(x - x_reconstructed) / norm(x);
fprintf('Reconstruction Error: %f\n', reconstruction_error);
```

Output of the task is shown below.

```
Reconstruction Error: 0.317427
```

Once the optimization problem is solved, the estimated DCT coefficients s can be used to reconstruct the original signal:

$$x_{\text{reconstructed}} = \Psi s,$$

where $x_{\text{reconstructed}}$ is the approximation of the original signal.

2. Repeat the above problem with Wavelet basis instead of DCT basis.

SOLUTION

Matlab code for the given task is shown below.

```
clc; clear;

% Sampling rate and time vector
Fs = 4096;          % 4096 samples per second
T = 1;              % Signal duration in seconds
t = linspace(0, T, Fs*T); % Time vector

% Generate the original sampled signal with three frequencies
x = sin(100*2*pi*t) + sin(400*2*pi*t) + sin(900*2*pi*t);
% Wavelet decomposition for sparse representation
n = length(x);
[Psi, L] = wavedec(x, log2(n), 'db1');
s_original = Psi';
% Number of compressed measurements
p = 256;                        % Number of samples to take
sampling_indices = randperm(n, p);
y = x(sampling_indices);

% Define the sensing matrix (random Gaussian)
A = randn(p, n);
```

```
lambda = 0.02;
cvx_begin quiet
    variable s(n)
    minimize(lambda * norm(s, 1) + norm(y' - A * s, 2) )
cvx_end
s_numeric = full(s);
x_reconstructed = waverec(s_numeric, L, 'db1');
x_reconstructed = x_reconstructed'; % Transpose to match
    dimensions
% Compute reconstruction error
reconstruction_error = norm(x - x_reconstructed) / norm(x);
fprintf('Reconstruction Error: %f\n', reconstruction_error);
```

Output of the code is shown below.

```
Reconstruction Error: 1.000013
```

3. Explain how the single pixel camera works? provide the mathematical representation and transformations behind the theory.

SOLUTION

The **single-pixel camera** is an imaging device that relies on the principles of **compressed sensing (CS)** to reconstruct an image using far fewer measurements than required by traditional cameras, which capture each pixel individually. This technique exploits the fact that natural images often have a sparse representation in some transform domain, such as wavelets or the Discrete Cosine Transform (DCT).

Traditional cameras use an array of sensors to capture the intensity of light at each pixel of an image. In contrast, the single-pixel camera uses only **one photodiode** (single pixel) to measure the light intensity. Instead of directly capturing each pixel, the camera takes a series of measurements where the image is "illuminated" or masked by different patterns. These patterns correspond to random or structured projections of the image. The image can then be reconstructed using compressed sensing techniques.

Mathematical model

Consider the image as a vector $x \in \mathbb{R}^n$ of length n, where n is the total number of pixels. Instead of directly measuring each pixel value, the single-pixel camera performs measurements of the form:

$$y = Ax \tag{8.1}$$

where:

- x: The vectorized form of the image (e.g., if the image is 64×64, then x has 4096 elements).
- A: The **sensing matrix**, where each row corresponds to a pattern (illumination or mask) applied to the image. Each element of A can be a random value (typically Gaussian or Bernoulli random variables) or based on specific transformations.
- $y \in \mathbb{R}^p$: The vector of **measurements** obtained from the camera, where $p \ll n$, meaning far fewer measurements are taken than the number of pixels.

Compressed sensing theory tells us that if the image has a sparse representation in some basis, we can reconstruct it from fewer measurements than required by traditional sampling (i.e., fewer than n). The key steps are:

Sparse representation

Suppose the image x can be sparsely represented in a basis Ψ, such that

$$s = \Psi x \tag{8.2}$$

where s is the sparse coefficient vector in that basis.

Random projections

The sensing matrix A projects the signal into a lower-dimensional space, forming the measurements:

$$y = A\Psi^{-1}s \tag{8.3}$$

Since the image is assumed to be sparse in some domain, we can formulate the reconstruction as an optimization problem, typically using an ℓ_1-norm minimization due to its ability to promote sparsity:

$$\min_s \|s\|_1 \quad \text{subject to} \quad y = A\Psi^{-1}s \tag{8.4}$$

Here, s is the sparse coefficient vector, and Ψ^{-1} is the inverse of the sparsifying transform (e.g., inverse DCT or inverse wavelet transform).

Steps of image acquisition and reconstruction

(a) **Pattern Projection**: The scene is illuminated by a sequence of random or structured patterns (corresponding to the rows of the matrix A). The intensity of the reflected light is recorded by the single-pixel sensor for each pattern, giving one element of the measurement vector y.

(b) **Sparse Signal Recovery**: Using the compressed measurements y, the reconstruction process involves solving the optimization problem (typically using Lagrangian methods) to find the sparse coefficients s.

(c) **Image Reconstruction**: Once the sparse coefficients s are found, the image x is reconstructed using the inverse transform Ψ^{-1}:

$$x = \Psi s \tag{8.5}$$

Example of transformation (Wavelet or DCT Basis)

If Ψ is the Discrete Cosine Transform (DCT) basis, then:

$$s = \text{DCT}(x) \tag{8.6}$$

And the sensing process becomes:

$$y = A \times \text{DCT}^{-1}(s) \tag{8.7}$$

Similarly, if Ψ is a wavelet basis, the coefficients s are obtained from a wavelet decomposition of the image x.

Lagrangian formulation in compressed sensing

The compressed sensing problem can be solved using the **Lagrangian formulation** with regularization:

$$\min_s \|y - A\Psi^{-1}s\|_2^2 + \lambda\|s\|_1 \tag{8.8}$$

where λ is a regularization parameter that balances the sparsity of the solution and the data fitting term. The term $\|y - A\Psi^{-1}s\|_2^2$ ensures that the reconstructed signal matches the measurements, while $\|s\|_1$ promotes sparsity in the solution.

4. What is single pixel accaustic camera?

SOLUTION

A **single-pixel acoustic camera** is a device that captures sound field information using a single sensor instead of an array of microphones, inspired by the principles of single-pixel optical cameras.

The camera works by sampling the sound field through a single point measurement over various configurations, capturing compressed measurements of the sound field. It uses sound field projections or patterns generated by modulating the sound source or employing acoustic lenses.

The recorded measurements can be mathematically modeled as:

$$y = As$$

where y represents the collected sound pressure measurements, A is the sensing matrix determined by the projection patterns, and s represents the sound field information to be reconstructed.

Reconstruction is performed through optimization techniques that typically minimize the difference between measured and estimated sound fields while promoting sparsity in the sound representation.

Applications of single-pixel acoustic cameras include:

- **Medical Imaging**: Non-invasive diagnostics using sound waves to analyze tissue structures.
- **Industrial Monitoring**: Detecting faults in machinery by analyzing sound fields.
- **Environmental Monitoring**: Assessing sound pollution levels in urban areas.

Advantages include cost-effectiveness due to fewer sensors, a compact design for easier deployment, and enhanced sensitivity by optimizing a single well-designed sensor for specific frequency ranges.

5. Why this technology is important for low-cost hyper-spectral imaging?

SOLUTION

The importance of single-pixel imaging technology for low-cost hyperspectral imaging lies in several key factors. Most relevant factors identified are listed below.

(a) **Reduced Sensor Costs**: Traditional hyper-spectral imaging systems often require large sensor arrays, which can be expensive to manufacture and maintain. Single-pixel imaging significantly reduces the number of sensors needed, leading to lower overall costs while still capturing high-dimensional spectral information.

(b) **Compact Design**: Single-pixel systems typically have a smaller footprint than conventional hyperspectral cameras. This compactness makes them easier to deploy in various settings, including remote or mobile applications, where space and weight constraints are critical.

(c) **Flexibility in Sampling**: Single-pixel imaging allows for flexible sampling strategies. By using different spatial or spectral patterns, it can capture information in a manner that can be tailored to specific applications. This adaptability can be beneficial for targeting specific spectral bands of interest without needing a full hyperspectral sensor suite.

(d) **Compressed Sensing Techniques**: By leveraging compressed sensing principles, single-pixel cameras can reconstruct high-dimensional data from significantly fewer measurements than would typically be required. This is particularly advantageous in hyperspectral imaging, where data acquisition can be bandwidth-limited or slow.

(e) **Enhanced Sensitivity**: Focusing on a single, high-quality sensor can lead to enhanced sensitivity, particularly in low-light or challenging environments. This characteristic is vital for applications like environmental monitoring, where detecting subtle changes in spectral signatures is crucial.

(f) **Improved Data Processing**: Single-pixel imaging typically generates fewer data points, which can simplify the data processing pipeline. This advantage allows for faster processing and analysis, making real-time or near-real-time hyperspectral imaging more feasible.

(g) **Application Versatility**: The versatility of single-pixel imaging technology means it can be applied across various fields that require hyperspectral imaging, including agriculture (for crop monitoring), environmental science (for pollution tracking), food quality assessment, and biomedical applications (for tissue analysis).

6. Why this technology is important for autonomous vehicles?

SOLUTION

Single-pixel imaging technology is a game changer for autonomous vehicles for a number of reasons. For starters, it helps cut costs significantly by reducing the number of sensors and cameras needed. This is a big deal because traditional systems can get pretty pricey. With fewer sensors, vehicles can still capture all the important visual information without breaking the bank.

Another benefit is the compact size of these systems. They can easily fit into the design of a vehicle without taking up too much space or adding extra weight—something that's really important in automotive engineering. Plus, single-pixel cameras can be made more sensitive, which means they perform better in low-light conditions, like when driving at night or in dimly lit areas.

Robustness is another key advantage. With fewer components to worry about, single-pixel imaging systems are less likely to fail, which is crucial for keeping autonomous vehicles safe and reliable. These systems are also adaptable; they can use different sampling strategies and algorithms to respond flexibly to various driving conditions and environments.

Finally, because single-pixel systems generate less data, they make it easier to process information quickly. This leads to faster decision-making, which is essential for real-time responses while driving. Single-pixel imaging technology boosts the efficiency, reliability, and adaptability of autonomous vehicles, making it a valuable tool in the quest to improve self-driving cars.

7. Why this technology is important for cognitive radio?

SOLUTION

Single-pixel imaging technology plays a significant role in the development of cognitive radio systems for several reasons. First and foremost, it enhances spectrum sensing capabilities. Cognitive radios need to identify and utilize available frequency bands efficiently, and single-pixel imaging can provide high-resolution data about the spectrum environment. This helps cognitive radios detect underutilized bands more effectively, ensuring better use of available spectrum.

Another important aspect is the technology's ability to improve spatial awareness. Single-pixel imaging systems can capture detailed information about the surrounding electromagnetic environment without requiring a large array of sensors. This compactness is crucial for cognitive radios, which often need to operate in dynamic and varying environments.

Single-pixel imaging also supports real-time processing. By generating less data compared to traditional imaging systems, it simplifies the analysis needed for spectrum management. This efficiency allows cognitive radios to make quick decisions about frequency allocation, which is vital for maintaining communication quality and minimizing interference.

Moreover, the flexibility of single-pixel imaging enables cognitive radios to adapt to different communication scenarios. These systems can use various sampling strategies to gather information tailored to specific operational requirements, whether it's for urban environments with many competing signals or rural areas with fewer users.

Finally, the reduced costs associated with single-pixel technology make it a practical choice for cognitive radio applications. By minimizing the number of sensors needed, cognitive radio systems can be more affordable, making advanced communication technologies accessible to a broader range of users and applications.

References

1. Candes, E. J., Romberg, J., & Tao, T. (2006). "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information." *IEEE Transactions on Information Theory.*

2. Donoho, D. L. (2006). "Compressed sensing." *IEEE Transactions on Information Theory.*

3. Tu, J. H., Rowley, C. W., Luchtenburg, D. M., Brunton, S. L., & Kutz, J. N. (2014). "On dynamic mode decomposition: Theory and applications." *Journal of Computational Dynamics.*

RESULTS

1. Concept of compressed sensing and the Dynamic Mode Decomposition is revisited.

2. Compressed sensing of signals with single and multiple frequencies are experimented with both DCT and Wavelet basis.

3. `CVX` solver is used to solve all the problems as a Lagrange Multiplier model.

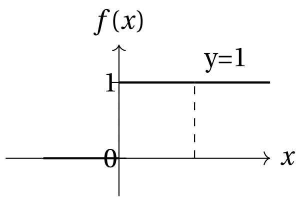
4. Various applications of single pixel camera is discussed.

9 | Assignment 75 Random Kitchen Sink Algorithm (RKS) for Pattern classification

9.1 Introduction

Random Kitchen sink algorithm provides a very efficient method for explicitly mapping linearly non-separable data into appropriate higher dimensional space where it is linearly separable. The motivation for this method is that, for large complex data sets, number of support vectors is of the order of 20% to 40% of training data points and kernel methods become very slow during both training and testing. RKS provide a way to overcome this difficulty. To understand the theory behind RKS, one need to know the following concepts Expected value of functions of random variable Fourier transform density function of normal distribution Concept of kernel function in Kernel method Let us apply the so called 'Computational thinking' to the concept of expected value of functions of random variables. Let Random Variable X follows uniform distribution $U(0,1)$. The parameters '0' and '1' define the range of the random variable. The density function is given by

$$f(x) = \begin{cases} 1 & ; \quad 0 \le x \le 1; \\ 0 & ; \quad \text{elsewhere} \end{cases}$$



From a practical standpoint, what we understood by the above statement? If we draw many sample values from that distribution and draw a histogram, it will approach the shape of the density function from which the sample is drawn. All programming languages have built in function for drawing sample values from $U(0,1)$. In matlab it is `rand()`. As computationally thinking computation engineers, we demonstrate it using following matlab code.

```
%uniform
n=100000; % number of samples
nbins=10;% number of cells in histogram
x=rand(n,1); % an array of random numbers
figure
histogram(x,nbins)
figure
histogram(x,nbins,'Normalization','probability')
figure
histogram(x,nbins,'Normalization','pdf')
```

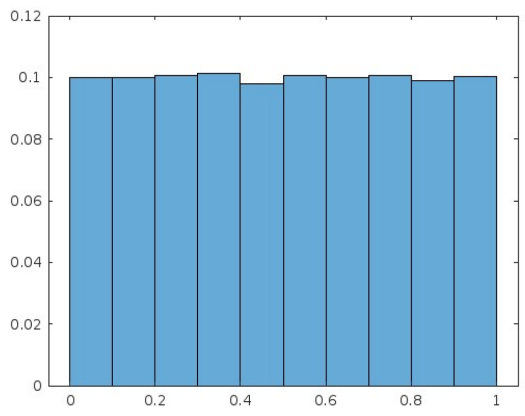


Figure 9.1: Density function of Uniform distribution

Output of the code is shown in Figure 9.1.

9.1.1 Expected value of random variable

Again, from a practical standpoint, it is the average value of random sample of size N from the distribution when N tends to infinity. In other words, it is the mean value of the random variable. Or it can be assumed as the population mean . Mathematically, for continuous random variable, it is defined as

$$\mathbf{E}(x) = \int_{-\infty}^{\infty} x \cdot f(x) dx \tag{9.1}$$

For the uniform distribution, $U(0,1)$, expectation is $E(U(0,1)) = \dfrac{0+1}{2}$. We can verify this using following integration.

$$\begin{aligned}
\mathbf{E}(X) &= \int_{-\infty}^{\infty} x f(x) dx \\
\therefore \mathbf{E}(U(0,1)) &= \int_{0}^{1} x dx \\
&= \left(\frac{x^2}{2}\right)_0^1 \\
&= \frac{1}{2}
\end{aligned}$$

For a discrete random variable, it is defined as

$$\mathbf{E}(X) = \sum_i x_i p(x_i) \tag{9.2}$$

From a simulation/practical view point, it means, if we draw large samples from the distribution and take average , the value to which it is tending to is the expected value. This computational perspective is more intuitive than the formula itself. A little dissection and thought will reveal that this practical view is hidden in the formula.

9.1.2 Expected value of functions of random variable

We can draw sample (a number) from the distribution and then define a function over that sample. We compute expected value of that function whose argument (input) is a random variable. Let the function be $g(x)$.Then the expected value is

$$\mathbf{E}(g(x)) = \int_{-\infty}^{\infty} g(x)f(x)dx \tag{9.3}$$

From computation point of view, we are continuously taking random sample and apply the function on that value. Then the long-term average value of that function is what we are interested to get. **Example 1.** Let us compute $\mathbf{E}(X^2)$ for $X = U(0,1)$.

SOLUTION

$$\begin{aligned}\mathbf{E}(X^2) &= \int_{-\infty}^{\infty} x^2 f(x)dx \\ &= \int_{0}^{\infty} x^2 dx \\ &= \left(\frac{x^3}{3}\right)_0^1 \\ &= \frac{1}{3}\end{aligned}$$

We computationally verify the result by the following `matlab` code.

```
% Expected value
n=100000; % number of samples
x=rand(n,1); % an array of random numbers
EV_squared_x=mean(x.^2)
```

Output of the code is shown below.

```
EV_squared_x = 0.3335
```

The output is 0.3335. This is close to analytical value of $\frac{1}{3}$ (Note this output will vary from sample to sample)

9.1.3 Task 1: Expectation of the complex exponential of the uniform distribution

Compute $\mathbf{E}(g(x))$, where $g(x) = e^{ix}, \quad X \sim U(0,1)$. Also verify the result by analytical method.

SOLUTION

$$\mathbf{E}(g(x)) = \int_{-\infty}^{\infty} g(x)f(x)dx$$

$$\begin{aligned}
\therefore E(e^{ix}) &= \int_{0}^{1} e^{ix}dx \\
&= \left(\frac{e^{ix}}{i}\right)_0^1 \\
&= \frac{1}{i}\left(e^{i}-1\right) \\
&= -i\left(e^{i}-1\right) \\
&= i\left(1-\cos 1+\sin 1\right) \\
&= 0.8415 + i0.4597
\end{aligned}$$

`Matlab` code for this task is given below.

```
% Expected value
n=100000; % number of samples
x=rand(n,1); % an array of random numbers
EV_exp_ix=mean(exp(i*x))
```

Output of the code is shown below.

```
EV_exp_ix = 0.8414 + 0.4599i
```

Hence from the computational method and analytical method, we get almost same result.

9.1.4 Expected value of functions of Gaussian random variable

Gaussian probability density function $N(0,1)$ is symmetric around y axis. The parameters of Gaussian distribution are mean and standard deviation. Let us compute $\mathbf{E}(e^{ix})$ using `matlab`. for normal distribution based on randomly drawn samples from normal distribution. We use the following `matlab` code.

```
n=1000000; % number of samples
x=randn(n,1); % an array of gaussian random numbers
EV_exp_ix=mean(exp(i*x))
```

Output of the code is shown below.

```
EV_exp_ix = 0.6064 + 0.000004i
```

The output is $0.6064 + 0i$, which is real. Above computational experiment and result is very important for us.
Note the following:

1. The expected value is real. There is no imaginary part. This is attributed to symmetry of Gaussian function around y axis.

2. It shows that integration of multiplication of functions involving a pdf (probability density function) and a general function can be obtained based on sampling from the distribution.

3. It is a very important idea utilized now for many applications.

9.1.5 Fourier transform and Kernel methods- a connection

The integral we have considered, $\mathbf{E}(e^{ix}) = \int\limits_{-\infty}^{\infty} e^{ix} f(x)dx$ is very close to Fourier transform (FT) of a function.
The FT of a continuous function $f(x)$ is defined as

$$\mathscr{F}\left(f(x)\right) = \bar{\mathscr{L}}(\omega) = \int\limits_{-\infty}^{\infty} e^{-i\omega x} f(x)dx = \mathbf{E}\left(e^{-i\omega x}\right) \tag{9.4}$$

based on random sampling concept for different values of ω in the range -7 to + 7 radians at an interval of 0.1 and plot the resulting graph. In fact this is a totally new way of finding FT of a Gaussian function. `Matlab` code for this task is given below.

```
close(gcf)
k=0;
n=100000; % number of samples
x=randn(n,1); % an array of random numbers
for w=-7:.01:7
                   k=k+1;
EV(k)=mean(exp(-i*w*x));
end
w=-7:.01:7;
EV=abs(EV);
plot(w,EV);
```

Output of the code is shown in Figure 9.2.

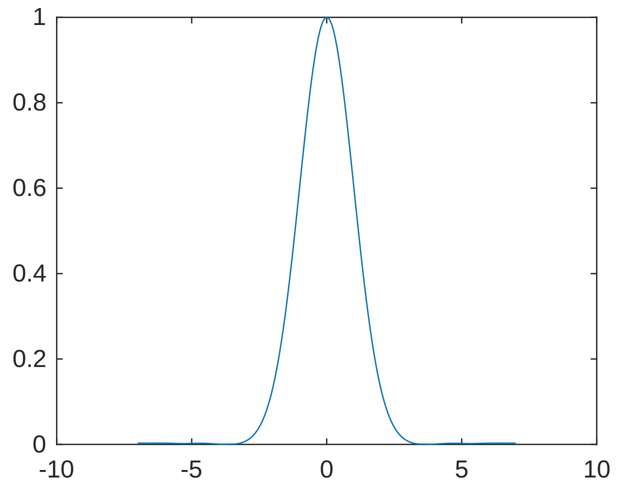


Figure 9.2: Expectation of $e^{-i\omega x}$ where $x \sim N(0,1)$.

The important observation is that, FT of gaussian function is real and again Gaussian Random Kitchen Algorithm is based on above observation. Let X, Y be two generic data points from a set of data points with binary labels for classification. One of the kernel function used in SVM based classification is Gaussian kernel given by :

$$k(x,y) =< \phi(x), \phi(y) >= e^{-\sigma|x-y|_2^2} = e^{-\sigma(x-y)^T(x-y)} = e^{-\sigma z^T z} = f(z) \tag{9.5}$$

Where, The function $k(x,y)$ is a sort of similarity function. When $x = y$, the function value is 1 and for all other (x,y), the function value is between 1 and 0 depending on how much distance they are apart.

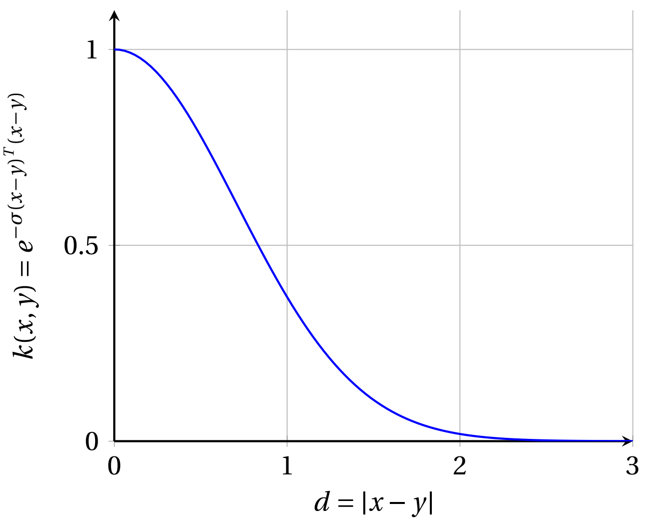


Figure 9.3: Plot of the Kernel Function with $\sigma = 1$.

Note that the resulting function is a multivariate Gaussian. The Multivariate Fourier Transform of this function is given by,

$$\bar{\mathscr{F}}(\Omega) = \int_{-\infty}^{\infty} e^{-iz^T\Omega} f(z)\,dz \tag{9.6}$$

where $\Omega \in \mathbb{R}^n$. This function $\bar{\mathscr{F}}(\Omega)$ is real since $f(z)$ is Gaussian. Therefore

$$f(Z) = \bar{\mathscr{F}}^{-1}(\Omega) = <\phi(x), \phi(y)> = \int_{-\infty}^{\infty} \bar{\mathscr{F}}(\Omega) e^{iz^T\Omega} d\Omega \tag{9.7}$$

Since $\bar{\mathscr{F}}(\Omega)$ is Gaussian, the formula (9.7) can be interpreted as the expected value of the random variate, $e^{iz^T\Omega}$. Mathematically

$$\mathbf{E}\left(e^{iz^T\Omega}\right) = \int_{-\infty}^{\infty} \bar{\mathscr{F}}(\Omega) e^{iz^T\Omega} d\Omega \tag{9.8}$$

We can compute this integral for any $z = x - y$ by using random numbers drawn from multivariate normal distributions. When z become random numbers, the continuous sum becomes discrete sum and using the property $e^{ax+by} = e^{ax} \cdot e^{by}$ we can approximate the expectation function as:

$$\begin{aligned}
\mathbf{E}\left(e^{iz^T\Omega}\right) &= \frac{1}{k}\sum_{j=1}^{k} e^{i\Omega_j^T z} \\
&= \frac{1}{k}\sum_{j=1}^{k} e^{i\Omega_j^T (x-y)} \\
&= \frac{1}{k}\sum_{j=1}^{k} e^{i\Omega_j^T x} e^{-i\Omega_j^T y} \\
&= \sum_{j=1}^{k} \left(\frac{1}{\sqrt{k}} e^{i\Omega_j^T x}\right) \cdot \left(\frac{1}{\sqrt{k}} e^{-i\Omega_j^T y}\right) \\
&= \langle \phi(x), \phi(y) \rangle \\
&= k(x,y)
\end{aligned}$$

with $\Omega_j \sim p(\Omega)$- i.i.d. Gaussian distribution. Using the result, $e^{i\theta} = cos\theta + i\sin\theta = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$, we can write $\phi(x)$ and $\phi(y)$ as:

$$\phi(x) = \frac{1}{\sqrt{k}} \begin{bmatrix} \cos(x^T\Omega_1) \\ \cos(x^T\Omega_2) \\ \vdots \\ \cos(x^T\Omega_k) \\ \vdots \\ \sin(x^T\Omega_1) \\ \sin(x^T\Omega_2) \\ \vdots \\ \sin(x^T\Omega_k) \end{bmatrix} ; \quad \phi(y) = \frac{1}{\sqrt{k}} \begin{bmatrix} \cos(y^T\Omega_1) \\ \cos(y^T\Omega_2) \\ \vdots \\ \cos(y^T\Omega_k) \\ \vdots \\ \sin(y^T\Omega_1) \\ \sin(y^T\Omega_2) \\ \vdots \\ \sin(y^T\Omega_k) \end{bmatrix} \tag{9.9}$$

We are now ready for applying RKS for linearly non-separable data. Usually the scaling factor in the above equation is omitted in practise.

9.1.6 Mapping the data to high dimension and applying Linear SVM

The following format is used while coding in `matlab`.

1. The data points are rows of the matrix A.
2. Matrix, R contains Gaussian random numbers for mapping.
3. Matrix, `M_data` contain first level mapping of data.
4. Matrix, `data_RKS` contain final mapped data (a point in a row).

Let us first generate such data sets for experiments.

9.1.7 Checkboard data generation

A *checkerboard pattern* consists of alternating black and white squares arranged in a grid, typically used in games like chess or checkers. Mathematically, this can be represented as a two-dimensional grid where each cell alternates between two values (e.g., 0 and 1). The pattern is determined based on the parity (even or odd) of the sum of the row and column indices. The result is a visually distinct, repetitive structure where neighboring squares have contrasting values.

A usual checkboard is shown in Figure 9.4.

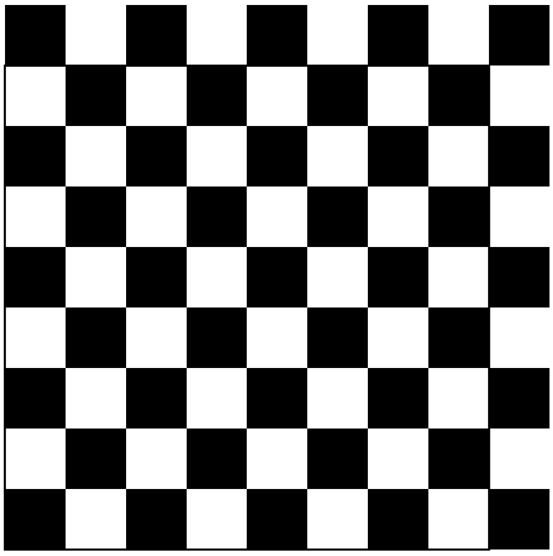


Figure 9.4: Checkboard of 8 × 8 size

In the context of data science and machine learning, we can imagine each square on the checkerboard representing a *data point* in a 2D feature space. The color of the square (black or white) can be thought of as a *class label*, with black representing one class and white another. The checkerboard structure exhibits an interesting property: the *boundaries between the two classes are non-linear*. This makes it challenging for traditional linear classifiers to separate the data accurately. The non-linear separability suggests that more advanced techniques, such as kernel methods or neural networks, are necessary to model this complex boundary and improve classification performance.Following `matlab` code creates the data points in the checkboard structure.

```
clc;clear all;close all;
x = rand(2000,1)*8;
y = rand(2000,1)*8;
c = mod((floor(x)+floor(y)),2);
ind = find(c);
a = [x(ind),y(ind)];
        ind1 = find(c==0);
b = [x(ind1),y(ind1)];
%       figure(1)
plot(a(:,1),a(:,2),'*');hold on
plot(b(:,1),b(:,2),'o','Color','red');hold off;
```

Output of the above code is shown in Figure 9.5.

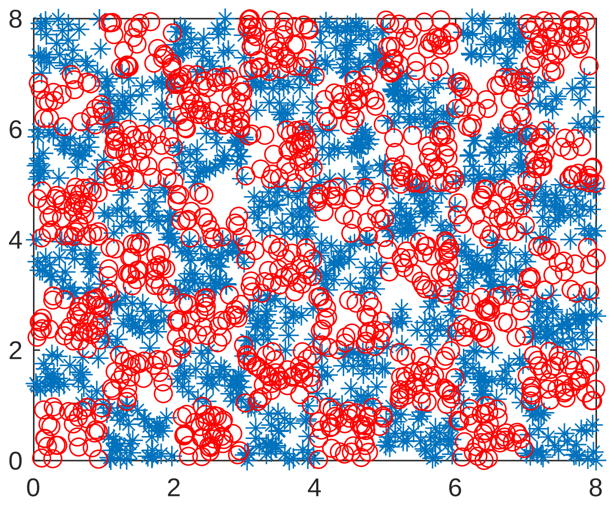


Figure 9.5: Distribution of data points in checkboard structure

9.1.8 Task 2: Classify the checker board data using RKS kernel

Apply RKS for checker board data. Map to the following dimension and report accuracy on training data [20, 40, 60, 80, 100].

SOLUTION

Matlab code for this task is given below.

```
clc;
clear all;
close all;

% Generate checkerboard data
x = rand(2000, 1) * 5; % x-coordinates
y = rand(2000, 1) * 5; % y-coordinates
c = mod((floor(x) + floor(y)), 2); % Checkerboard condition
ind = find(c); % Indices for one color
a = [x(ind), y(ind)]; % Points for one color (e.g., color 1)
ind1 = find(c == 0); % Indices for the other color
b = [x(ind1), y(ind1)]; % Points for the other color (e.g., color
    0)

% Combine the two sets of data and create labels
A = [a; b]; % Combined data matrix
nd = size(a, 1); % Number of points in each class
d = [ones(nd, 1); -ones(size(b, 1), 1)]; % Class labels: +1 for
    color 1, -1 for color 0

% Data mapping to high dimensions
dim = 100; % Target dimensionality
rng(2545); % Random number seed
R = 2 * randn(2, dim); % Random projection matrix
M_data = A * R; % Mapped data
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data

% Prepare for SVM classification using CVX
n = size(data_RKS, 2); % Number of features after mapping
m = size(data_RKS, 1); % Number of data points
```

```
e = ones(m, 1); % m-tuple one-vector
c = 10000; % Penalty for error

% CVX optimization for SVM
cvx_begin quiet
    variables w(n) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        d .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predictions on training data
r = sum(d == z); % Count correctly classified points
Acc = (r / m) * 100; % Calculate accuracy
fprintf('Accuracy of Check board data classification : %.2f%%\n',
   Acc);
```

On running this code with `dim` values 20, 40, 60, 80, 100, the accuracy of the SVM model is shown in Table 9.1.

Table 9.1: Performance of SVM with RKS kernel on Checkboard data.

dim	Accuracy
20	81.65
40	98.90
60	99.60
80	99.95
100	99.95

9.1.9 Task 3: Classify the given spiral data using RKS Kernel

`Matlab` code to create the spiral data is given below.

```
clc;clear all;close all;
    close(gcf)
    x1 = [];y1 = [];
    j = 1:100;
    angle = j*pi/16;
    radius = 6.5*(104-j)/104;
    x = radius.*sin(angle);
    y = radius.*cos(angle);
    x1 = [x1;x]; y1 = [y1;y];
    x2 = -x1;y2 = -y1;
    figure
    plot(x1,y1);hold on;
    plot(x2,y2,'Color','red');title('Spiral data')
```

Output of the code is shown in Figure 9.6.

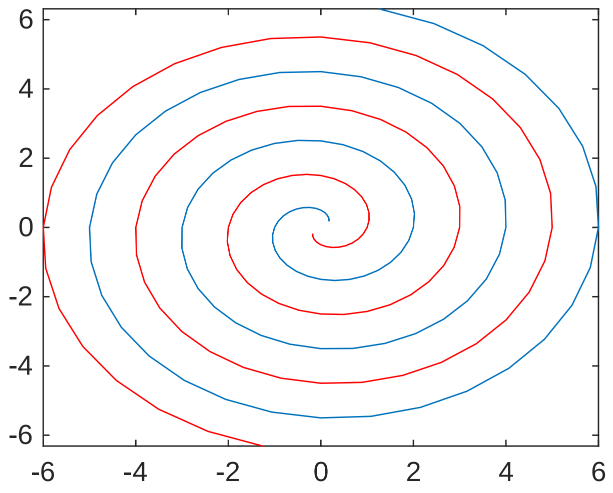


Figure 9.6: Distribution of points in the spiral dataset

SOLUTION

Matlab code for this task is given below.

```
    % spiral data in columms
         % each raw represent a data point
% start of data generation
    nd=100; % number of data points in each class
    j = 1:nd;
    angle = j*pi/16;
    radius = 6.5*(104-j)/104;
    x = radius.*sin(angle);
    y = radius.*cos(angle);
    x1 = x'; y1 = y';    % First Spiral's coordinates in columns
    x2 = -x1; y2 = -y1; % Generation of second spiral from first
    A=[x1 y1; x2 y2]; % Putting the two spiral data in matrix A
    d=ones(nd,1);          % class values +1
    d=[d;-d];  % first nd data label is +1, remaining -1
    % data  mapping  to 2*dim
    dim = 45;  % first convert 2-tuple data to 45-tuple data
    rng(2545); % random number seed
    R = 2*randn(2,dim); % matrix for high dimensional mapping
    M_data = A*R; %  First level mapped data
    data_RKS = [cos(M_data)  sin(M_data)]; % RKS mapped data
%  data generation over
n = size(data_RKS,2); % tuple size of mapped data
m = size(data_RKS,1); % number of data points
e = ones(m,1);  % m-tuple one-vector
c = 10000; % penaly for error. Obtained by trial and error
cvx_begin quiet
                variables w(n) g Psi(m)
                 minimize ((0.5*w'*w)+(c*sum(Psi)))
                subject to
                 d.*(data_RKS*w-g*e)+Psi-e >= 0;
                Psi >= 0;
cvx_end

z = sign(data_RKS*w-g); % checking with training data
```

```
r = sum(d==z);
Acc = (r/m)*100
```

Output of the code is shown below.

```
Acc = 100
```

Separation boundary of the SVM classifier with RKS kernel is shown in Figure 9.7.

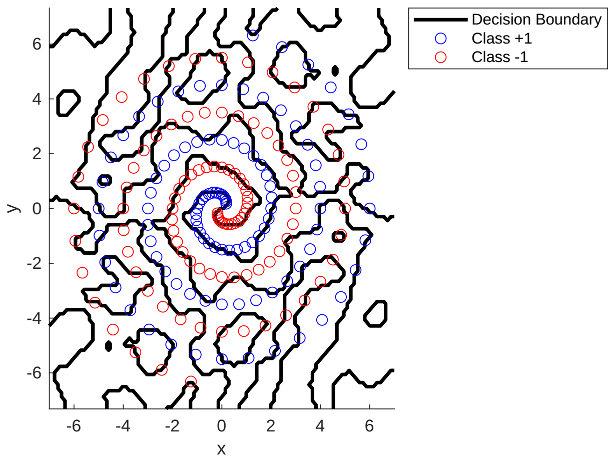


Figure 9.7: Separating boundary of SVM classifier with RKS kernel on spiral dataset

9.1.10 Task 4: Classification of Ring data

Classify the ring data generated with following `matlab` code using RKS kernel.

`Matlab` code to generate the ring dataset is given below.

```
    %Ring data generation
          d = 1000;
        r1 = 5+rand(d,1);
        theta = 2*pi*rand(d,1);
        x1 = r1.*cos(theta);
        y1 = r1.*sin(theta);

        r2 = 10+rand(d,1);
        x2 = r2.*cos(theta);
        y2 = r2.*sin(theta);
        figure
        plot(x1,y1,'o');hold on;
        plot(x2,y2,'*','Color','red');title('Ring data')
```

output of the code is shown in Figure 9.8.

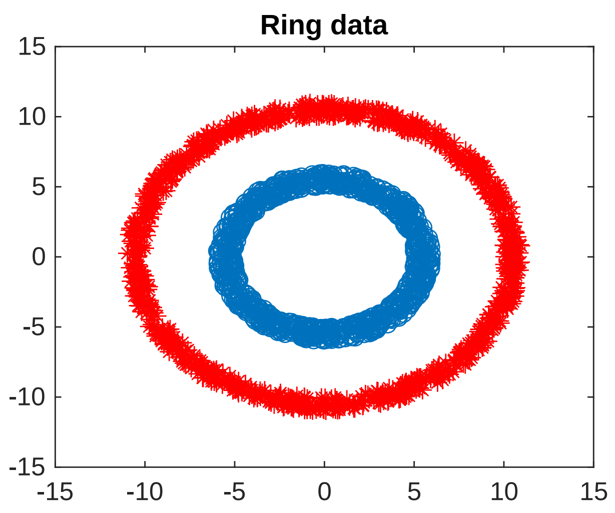


Figure 9.8: Distribution of data points in the ring dataset

SOLUTION

`Matlab` code for this task is given below.

```
    % Ring data generation
d = 1000; % Number of points per class

% First ring (inner ring)
r1 = 5 + rand(d, 1); % Radius between 5 and 6
theta1 = 2 * pi * rand(d, 1); % Random angle between 0 and 2*pi
x1 = r1 .* cos(theta1); % x-coordinates
y1 = r1 .* sin(theta1); % y-coordinates

% Second ring (outer ring)
r2 = 10 + rand(d, 1); % Radius between 10 and 11
theta2 = 2 * pi * rand(d, 1); % Random angle between 0 and 2*pi
x2 = r2 .* cos(theta2); % x-coordinates
y2 = r2 .* sin(theta2); % y-coordinates

% Combine the two rings into matrix A
A = [x1 y1; x2 y2]; % All coordinates (features)
d_labels = [ones(d, 1); -ones(d, 1)]; % First ring label +1, second
    ring label -1

% Plot the ring data
figure;
plot(x1, y1, 'o', 'MarkerFaceColor', 'blue'); hold on;
plot(x2, y2, '*', 'Color', 'red');
title('Ring Data');
xlabel('x'); ylabel('y');
legend('Class +1 (Inner Ring)', 'Class -1 (Outer Ring)');
hold off;

% Data mapping to higher dimensions using Random Kitchen Sink (RKS)
dim = 45;  % Convert 2-tuple data to 45-tuple data
rng(2545); % Random number seed for reproducibility
R = 2 * randn(2, dim); % Matrix for high-dimensional mapping
M_data = A * R; % First level mapped data
```

```
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data

% SVM optimization
n = size(data_RKS, 2); % Tuple size of mapped data
m = size(data_RKS, 1); % Number of data points
e = ones(m, 1);  % m-tuple one-vector
c = 10000; % Penalty for error. Obtained by trial and error

% CVX for SVM
cvx_begin quiet
    variables w(n) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        d_labels .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Check accuracy with training data
z = sign(data_RKS * w - g);
r = sum(d_labels == z);
Acc = (r / m) * 100;
fprintf('Accuracy: %.2f%%\n', Acc);

% Now we plot the decision boundary
% Generate a grid of points in the input space
[xGrid, yGrid] = meshgrid(linspace(min(A(:,1)) - 1, max(A(:,1)) +
   1, 100), ...
                          linspace(min(A(:,2)) - 1, max(A(:,2)) +
                             1, 100));
gridPoints = [xGrid(:), yGrid(:)];

% Apply the RKS transformation to the grid points
M_grid = gridPoints * R;
grid_RKS = [cos(M_grid) sin(M_grid)];

% Predict the class of each grid point
zGrid = sign(grid_RKS * w - g);
zGrid = reshape(zGrid, size(xGrid));

% Plot the decision boundary and the ring data points
figure;
hold on;
contour(xGrid, yGrid, zGrid, [0 0], 'LineWidth', 2, 'LineColor', 'k
   '); % Decision boundary
plot(x1, y1, 'o', 'MarkerFaceColor', 'blue'); % First ring (class
   +1)
plot(x2, y2, '*', 'Color', 'red'); % Second ring (class -1)
%title('Ring Data with Random Kitchen Sink (RKS) SVM Decision
   Boundary');
xlabel('x');
ylabel('y');
legend('Decision Boundary', 'Class +1 (Inner Ring)', 'Class -1 (
    Outer Ring)');
hold off;
```

Output of the code is shown below.

```
Accuracy: 100.00%
```

Separation boundary of the SVM classifier with RKS kernel is shown in Figure 9.9.

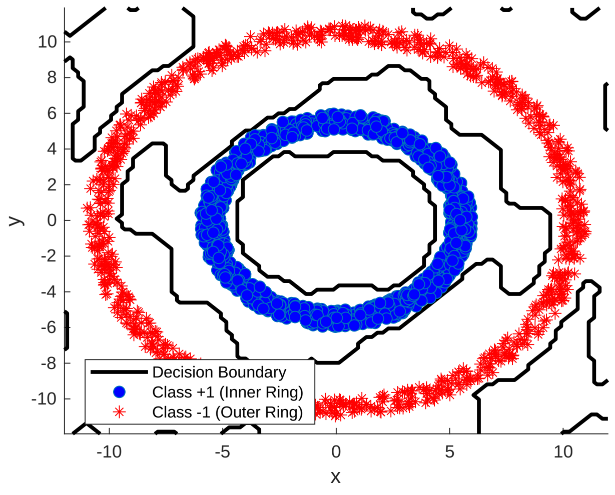


Figure 9.9: Separating boundary of SVM classifier with RKS kernel on ring dataset

9.1.11 Task 5: Classification of the breast cancer dataset

Map the breast cancer data to 900 dimension and tune parameter c to get maximum accuracy on training data.

SOLUTION

The dataset is stored in the `matlab` drive with the name *breast-cancer-wisconsin-data.csv*. Dataset is loaded and an SVM model with RKS kernel is modelled. `Matlab` code for this task is given below.

```
data = readtable('breast-cancer-wisconsin-data.csv');
correctVarNames = {'id', 'diagnosis', 'radius mean', 'texture mean'
    , 'perimeter mean', ...
                    'area mean', 'smoothness mean', 'compactness
                       mean', 'concavity mean', ...
                    'concave points mean', 'symmetry mean', 'fractal
                         dimension mean', ...
                    'radius se', 'texture se', 'perimeter se', 'area
                         se', ...
                    'smoothness se', 'compactness se', 'concavity se
                       ', ...
                    'concave points se', 'symmetry se', 'fractal
                       dimension se', ...
                    'radius worst', 'texture worst', 'perimeter
                       worst', ...
                    'area worst', 'smoothness worst', 'compactness
                       worst', ...
                    'concavity worst', 'concave points worst', '
                       symmetry worst', ...
                    'fractal dimension worst'};
data.Properties.VariableNames = correctVarNames;
data.Properties.VariableNames = cellstr(data.Properties.
    VariableNames);
data.diagnosis = double(strcmp(data.diagnosis, 'M'));
X = data{:, 3:end};
y = data.diagnosis;
y = double(categorical(y)); % Convert categorical to numeric (if
    needed)
% Convert y to -1 and 1
y(y == 1) = -1;
y(y == 2) = 1;

% Data dimensions
m = size(X, 1);
n = size(X, 2);
dim = 900; % Target dimensionality
rng(2545); % Random number seed
R = 2 * randn(n, dim);
% Map data to high-dimensional space
M_data = X * R;
data_RKS = [cos(M_data) sin(M_data)]; % RKS mapped data

n_mapped = size(data_RKS, 2);
e = ones(m, 1); % m-tuple one-vector
c = 1; % Penalty for error

% CVX optimization for SVM
cvx_begin quiet
    variables w(n_mapped) g Psi(m)
    minimize ((0.5 * w' * w) + (c * sum(Psi)))
    subject to
        y .* (data_RKS * w - g * e) + Psi - e >= 0;
        Psi >= 0;
cvx_end

% Classification on training data
z = sign(data_RKS * w - g); % Predictions on training data
r = sum(y == z); % Count correctly classified points
Acc = (r / m) * 100; % Calculate accuracy
fprintf('Accuracy: %.2f%%\n', Acc);
```

Output of the code is shown below.

```
Accuracy: 100.00%
```

For all values of c ranging from 1 to 50, same result is obtained. The separation boundary over first two features is shown in Figure 9.10. Matlab code to visualize the separation boundary over the feature 'Radius Mean' and 'Texture Mean' is given below.

```
X_visualize = data{:, {'radius mean', 'texture mean'}};
[x1Grid, x2Grid] = meshgrid(linspace(min(X_visualize(:,1))-1, max(
    X_visualize(:,1))+1, 100), ...
                            linspace(min(X_visualize(:,2))-1, max(
                                X_visualize(:,2))+1, 100));
gridPoints = [x1Grid(:), x2Grid(:)];
```

```
R_visualize = R(1:2, :);
M_grid = gridPoints * R_visualize;
grid_RKS = [cos(M_grid) sin(M_grid)];
zGrid = sign(grid_RKS * w - g);
zGrid = reshape(zGrid, size(x1Grid));
figure;
hold on;
contour(x1Grid, x2Grid, zGrid, [0 0], 'k', 'LineWidth', 2);
scatter(X_visualize(y==1,1), X_visualize(y==1,2), 'r', 'filled');
scatter(X_visualize(y==-1,1), X_visualize(y==-1,2), 'b', 'filled');
xlabel('Radius Mean');
ylabel('Texture Mean');
legend('Decision Boundary', 'Malignant', 'Benign');
hold off;
```

This code will produce the plot shown in Figure 9.10.

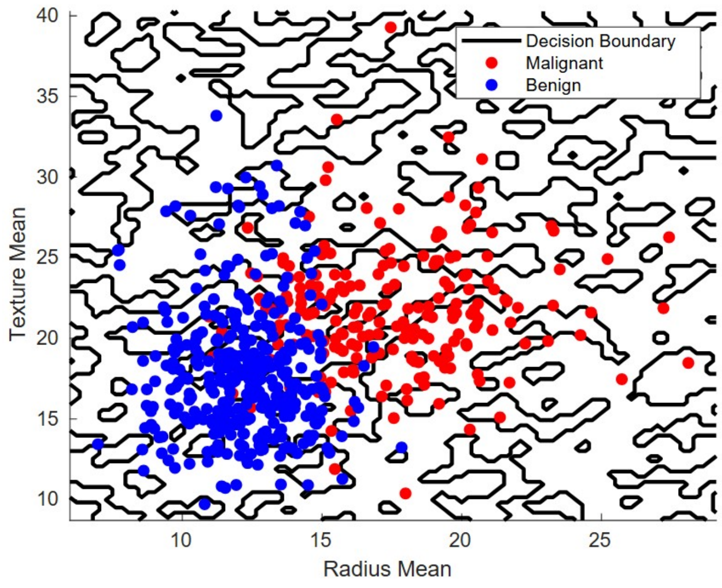


Figure 9.10: Separation boundary of SVM classifier with RKS kernel on the breast cancer dataset.

RESULTS

1. Theoretical backgrounds of the Random Kitchen Sink algorithm is revisited.
2. Skill of RKS kernel in classification of non linearly separable data with SVM is demonstrated on diverse datasets.

10 | Assignment 76 Linear Algebra for Intelligent Information Retrieval

10.1 Introduction to Latent Semantic Indexing

Latent Semantic Indexing (LSI) is a technique that analyzes relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSI can be used with a query to find documents that are relevant even if the documents don't share any words with the query. For example, a query on "human computer interaction" could return documents related to user interface design, even if those documents don't contain the exact phrase "human computer interaction". LSI can be used in a variety of applications, such as information retrieval, document classification, and text summarization.

10.2 LSI, SVD, and the Concept Space

Latent Semantic Indexing (LSI) leverages the mathematical technique of Singular Value Decomposition (SVD) to enhance traditional information retrieval methods like the vector space model. The core idea is to move beyond simple word matching and capture the latent semantic relationships between words and documents. This is achieved by applying SVD to the term-document matrix, which records the frequency of each word in each document.
The output of SVD provides the building blocks for constructing a "concept space". This space represents words and documents as vectors, but unlike in the traditional vector space model, the dimensions of this space represent underlying concepts rather than individual terms. This allows LSI to group semantically related items together, even if they don't share the exact same words. For instance, documents discussing "dogs" and "puppies" might be clustered close together in the concept space despite not having identical terms.
The concept space also helps to address the challenges of synonymy (different words with the same meaning) and polysemy (one word with multiple meanings). Words with similar meanings will be positioned closer together in the concept space, while the meaning of a polysemous word will be disambiguated based on its surrounding context, as reflected in the document vectors.

10.3 Dimensionality Reduction and Applications

A key aspect of LSI is dimensionality reduction. The SVD process generates matrices that encapsulate the semantic structure of the data. By truncating these matrices, LSI discards less important information, effectively reducing the number of dimensions while retaining the most significant semantic relationships.
This dimensionality reduction has several benefits. It simplifies the representation of documents and queries, making computations more efficient. It also filters out noise and focuses on the essential

semantic information. However, choosing the optimal number of dimensions to retain is crucial, as it can significantly impact LSI's performance.
LSI's ability to reveal underlying semantic structures makes it a powerful tool for various text-based tasks beyond simple information retrieval. It can be used for document classification, grouping documents based on their conceptual content rather than just keywords. It can also be applied to text summarization, identifying the most important concepts to create concise and informative summaries. Furthermore, LSI can be used for cross-language information retrieval by translating queries and documents into a shared concept space. Even essay grading can be enhanced using LSI, as it can analyze the semantic content of essays and compare them to models or grading criteria.

10.4 Strengths and Limitations

The main strengths of LSI are its ability to:

- Capture latent semantic relationships between words and documents
- Effectively address synonymy and polysemy
- Improve information retrieval accuracy by retrieving documents based on conceptual similarity
- Offer versatility in application to various text-based tasks beyond information retrieval

However, LSI also has some limitations:

- **Computational Complexity:** Applying SVD, especially on large datasets, can be computationally demanding.
- **Concept Interpretability:** The concepts generated by LSI are abstract and can be difficult to directly interpret.
- **Sensitivity to Dimensionality Reduction:** The performance of LSI is highly sensitive to the number of dimensions retained during the truncation of the SVD matrices. Choosing the right number of dimensions requires careful consideration.

Despite its limitations, LSI remains a powerful technique for analyzing and understanding the semantic structure of text. Its applications extend far beyond information retrieval, offering significant potential for advancements in various text-based fields. Continued research focuses on developing more efficient implementations, finding ways to interpret the abstract concepts, and exploring new applications for LSI.

10.5 Mathematical Steps in Latent Semantic Indexing (LSI)

The following outlines the mathematical steps involved in Latent Semantic Indexing (LSI):

10.5.1 Constructing the Term-Document Matrix (A)

Consider a corpus of m documents and n unique terms (words). The steps to construct the term-document matrix are as follows:

- Create a matrix **A** with m rows (documents) and n columns (terms).
- Each cell a_{ij} in **A** represents the frequency of term j in document i.

- The frequency can be a raw count or adjusted using weighting schemes such as Term Frequency-Inverse Document Frequency (TF-IDF):

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

  where

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in } d}$$

  and

$$\text{IDF}(t) = \log\left(\frac{N}{\text{Number of documents containing term } t}\right)$$

10.5.2 Performing Singular Value Decomposition (SVD)

Next, decompose the matrix **A** into three matrices:

- **U**: An $m \times m$ orthogonal matrix representing document concepts.
- **S**: An $m \times n$ diagonal matrix containing singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$ arranged in descending order, where $r = \min(m,n)$.
- $\mathbf{V}^T$: An $n \times n$ orthogonal matrix representing term concepts.

The SVD decomposition is mathematically represented as:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

10.5.3 Dimensionality reduction

To achieve dimensionality reduction:

- Choose k, the number of dimensions to retain, ensuring that k is much smaller than m and n.
- Create reduced versions of the SVD matrices:

$$\begin{aligned}\mathbf{U}_k &= \text{the first } k \text{ columns of } \mathbf{U}\\ \mathbf{S}_k &= \text{the top left } k \times k \text{ submatrix of } \mathbf{S}\\ \mathbf{V}_k^T &= \text{the first } k \text{ rows of } \mathbf{V}^T\end{aligned}$$

10.5.4 Constructing the Concept Space

The concept space is now represented by the reduced matrices:

- **Document-Concept Matrix:**

$$\mathbf{C}_{doc} = \mathbf{U}_k\mathbf{S}_k$$

  Each row represents a document in the reduced concept space.

- **Term-Concept Matrix:**

$$\mathbf{C}_{term} = \mathbf{V}_k^T\mathbf{S}_k$$

  Each row represents a term in the reduced concept space.

10.5.5 Querying and Similarity Calculation

To perform a query in the LSI framework:

- Represent a query as a vector in the original term space, denoted as $\mathbf{q}$.
- Project this query vector into the reduced concept space by multiplying it with $\mathbf{V}_k^T$:

$$\mathbf{q}_{proj} = \mathbf{q}\mathbf{V}_k^T$$

- Calculate the cosine similarity between the projected query vector and the document vectors in the reduced concept space:

$$\text{cosine\_similarity}(\mathbf{q}_{proj}, \mathbf{C}_{doc}) = \frac{\mathbf{q}_{proj} \cdot \mathbf{C}_{doc}}{\|\mathbf{q}_{proj}\| \|\mathbf{C}_{doc}\|}$$

  This yields a ranked list of documents based on their semantic relevance to the query.

10.5.6 Key Points

- The choice of k is crucial in LSI. A smaller k leads to more aggressive dimensionality reduction but may risk losing important semantic information.
- LSI does not explicitly define or label the concepts; they are latent and emerge from the data through the SVD process.
- The mathematical steps described here provide a high-level overview. The actual implementation of SVD and other calculations can be accomplished using various numerical algorithms and libraries.

10.6 Tasks

1. Create your own example of LSI with 20 sentences as documents. Repeat the computation done in the paper mentioned at the top.

SOLUTION

Step by step procedure to accomplish this task is given below.

Step 1: Create the document

Here's a set of 20 sentences (imaginary) that describe the Nobel Prize winners in 2024 and their areas of research in various scientific fields:

```
documents = {
    'The 2024 Nobel Prize in Physics was awarded for
       advancements in quantum computing';
    'Researchers developed a new quantum algorithm that
       drastically improves computational efficiency';
    'The 2024 Nobel Prize in Chemistry honored breakthroughs in
        green chemistry and sustainability';
    'Scientists discovered a novel catalyst for carbon capture,
        reducing emissions globally';
    'The Nobel Prize in Medicine recognized research on gene
       therapy for curing genetic disorders';
    'Gene editing technology CRISPR played a crucial role in
       the 2024 Nobel Prize in Medicine';
    'The Nobel Prize in Physics acknowledged the importance of
       topological materials for future electronics';
    'The 2024 Nobel Peace Prize highlighted the efforts to
       combat climate change through engineering solutions';
    'A new drug delivery system, awarded the 2024 Nobel Prize
       in Medicine, revolutionizes cancer treatment';
    'The Nobel Prize in Economic Sciences focused on the impact
        of artificial intelligence on global markets';
    'Breakthroughs in machine learning optimization contributed
        to advancements in medical diagnostics';
    '2024 Nobel Prize in Chemistry focused on research in
       nanotechnology and material science innovations';
    'The Nobel Prize in Physics showcased research on
       superconductor materials for energy efficiency';
    'Biomedical engineering innovations received recognition
       for advancing prosthetic technologies';
    'The Nobel Peace Prize acknowledged clean energy solutions
       developed by engineers';
    'Scientists made significant progress in understanding
       climate modeling, earning the Nobel Prize in Physics';
    'The 2024 Nobel Prize in Chemistry honored advancements in
       organic electronics and bioengineering';
    'Research in renewable energy storage technologies was
       pivotal in winning the Nobel Prize in Physics';
    'The Nobel Prize in Medicine recognized advancements in AI-
       driven healthcare solutions';
    'Quantum computing breakthroughs dominated the 2024 Nobel
       Prize in Physics announcements'
};
```

Step 2: Process the document to create a Term-Document Matrix

```
terms = {'nobel', 'prize', 'physics', 'chemistry', 'medicine',
   'quantum', ...
         'gene', 'ai', 'materials', 'energy', 'climate', '
            research', ...
         'computing', 'engineering', 'technology', '
            sustainability', ...
         'cancer', 'diagnostics', 'renewable', 'superconductor'
            };
numTerms = length(terms);
numDocs = length(documents);
termDocumentMatrix = zeros(numTerms, numDocs);

% Count occurrences of each term in each document
for i = 1:numDocs
    docWords = lower(split(documents{i}));
    for j = 1:numTerms
        termDocumentMatrix(j, i) = sum(contains(docWords, terms
           {j}));
    end
end
docNames = arrayfun(@(x) ['Doc' num2str(x)], 1:numDocs, '
   UniformOutput', false);
termDocTable = array2table(termDocumentMatrix, 'VariableNames',
    docNames, 'RowNames', terms);
disp('Term-Document Matrix:');
disp(termDocTable);
```

Output of the code is the term-document matrix (first 12 documents) as shown in the following table.

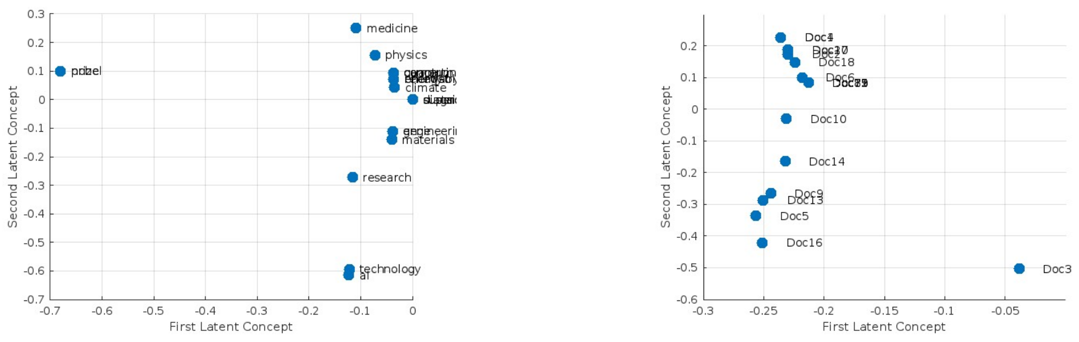
```
                      Doc1    Doc2    Doc3    Doc4    Doc5    Doc6    Doc7    Doc8    Doc9    Doc10    Doc11    Doc12
                      ----    ----    ----    ----    ----    ----    ----    ----    ----    -----    -----    -----

    nobel              1       0       1       0       1       1       1       1       1        1        1        1
    prize              1       0       1       0       1       1       1       1       1        1        1        1
    physics            1       0       0       0       0       0       1       0       0        1        0        1
    chemistry          0       0       2       0       0       0       0       0       0        0        0        0
    medicine           0       0       0       0       1       1       0       0       1        0        1        0
    quantum            1       1       0       0       0       0       0       0       0        0        0        1
    gene               0       0       0       0       2       1       0       0       0        0        0        0
    ai                 0       0       1       0       0       0       0       0       0        0        1        0
    materials          0       0       0       0       0       0       1       0       0        0        0        0
    energy             0       0       0       0       0       0       0       0       0        1        0        0
    climate            0       0       0       0       0       0       0       1       0        0        0        0
    research           0       1       0       0       1       0       0       0       0        1        0        0
    computing          1       0       0       0       0       0       0       0       0        0        0        1
    engineering        0       0       0       0       0       0       0       1       0        0        0        0
    technology         0       0       0       0       0       1       0       0       0        0        0        0
    sustainability     0       0       1       0       0       0       0       0       0        0        0        0
    cancer             0       0       0       0       0       0       0       0       1        0        0        0
    diagnostics        0       0       0       0       0       0       0       0       0        0        0        0
    renewable          0       0       0       0       0       0       0       0       0        1        0        0
    superconductor     0       0       0       0       0       0       0       0       0        0        0        0
```

Step 3: Apply Singular Value Decomposition (SVD)

Now that we have the term-document matrix, we can apply SVD to perform Latent Semantic Indexing (LSI). This will help us reduce the dimensionality of the matrix and extract the most meaningful patterns.

```
[U, S, V] = svd(termDocumentMatrix);
k = 2;
Uk = U(:, 1:k);
Sk = S(1:k, 1:k);
Vk = V(:, 1:k);
figure;
scatter(Vk(:, 1), Vk(:, 2), 100, 'filled');
xlabel('First Latent Concept');
ylabel('Second Latent Concept');
text(Vk(:, 1) + 0.02, Vk(:, 2), docNames);
grid on;
figure;
scatter(Uk(:, 1), Uk(:, 2), 100, 'filled');
title('Terms in the Latent Semantic Space (LSI)');
xlabel('First Latent Concept');
ylabel('Second Latent Concept');
text(Uk(:, 1) + 0.02, Uk(:, 2), terms);
grid on;
```

Output of the code is shown in Figure 10.1.



(a) Term matrix over first two latent concepts. (b) Document matrix over first two latent concepts.

Figure 10.1: Visualization of document-term matrices over first two latent concepts.

2. Explain why the above process of indexing is called latent semantic indexing with an example.

SOLUTION

LSI's Power: Moving Beyond Simple Word Matching

Latent Semantic Indexing (LSI) distinguishes itself from traditional keyword-based indexing by leveraging the power of Singular Value Decomposition (SVD) to *discover latent semantic relationships*. Instead of relying solely on the presence or absence of specific keywords, LSI analyzes patterns of word co-occurrence across documents to create a "concept space." This concept space represents the underlying semantic structure of the corpus, where terms and documents with similar meanings are clustered together, even if they don't share exact words.

This ability to uncover hidden semantic relationships allows LSI to retrieve documents that are *conceptually relevant to a query*, even if they lack the precise keywords. For example, as discussed earlier, a search for "dog training" could retrieve documents containing "canine obedience" or "puppy socialization" because LSI recognizes the semantic connections between these terms. This approach is particularly valuable when dealing with synonyms, polysemy (words with multiple meanings), and situations where users might use different terms to express the same concept. By moving beyond surface-level word matching, LSI enables a deeper understanding of the meaning embedded within the text corpus, leading to more accurate and insightful information retrieval.

3. What is NMF (Non-negative Matrix Factorization) . Explore and write one paragraph. It can also be used for LSI.

SOLUTION

Non-negative Matrix Factorization (NMF) is a dimensionality reduction technique that can be used as an alternative to Singular Value Decomposition (SVD) in Latent Semantic Indexing (LSI). *NMF decomposes a non-negative matrix (like the term-document matrix) into two lower-rank non-negative matrices.* This non-negativity constraint makes the resulting matrices more interpretable than those produced by SVD. While the sources do not mention NMF directly, its application in LSI is a known concept outside the provided sources. You may want to independently verify this information. In the context of LSI, NMF can be used to identify latent semantic relationships between terms and documents. Each of the resulting matrices represents a cluster of related terms or documents. This allows for the representation of documents and terms in a lower-dimensional space based on their underlying semantic themes. NMF's interpretability can be advantageous in understanding the derived concepts, as the non-negative values indicate the strength of association between terms/documents and the underlying semantic themes.

4. What is the 'word-vector' concept used in the context of deep-learning based NLP (Natural Language Processing). Write a paragraph.

SOLUTION

Word vectors, also known as word embeddings, are mathematical representations of words in a multi-dimensional space. This information is not from the sources and you may want to independently verify it. Each dimension in this space captures a different aspect of the word's meaning, and words with similar meanings are located close together in this space. Deep learning models learn these representations by analyzing vast amounts of text data and identifying patterns of word usage. The resulting word vectors can be used in various NLP tasks, such as:

- **Machine Translation:** Word vectors can help map words with similar meanings across different languages.
- **Sentiment Analysis:** The location of a word in the vector space can indicate its emotional tone (positive, negative, neutral).
- **Text Classification:** Documents can be classified based on the average of the word vectors of the words they contain.

11 | Assignment 77 PageRank Algorithm- Deciding Priority to Web Pages

11.1 What is PageRank?

PageRank is a fundamental algorithm used by search engines, particularly Google, to rank web pages in their search results. The algorithm is designed to measure the relative importance of each page within the set. Mathematically, it represents the limiting probability that an infinitely dedicated random surfer visits any particular page. The algorithm, developed by Larry Page and Sergey Brin while they were graduate students at Stanford University, is one of the reasons why Google has emerged as an effective search engine.

11.2 Importance of PageRank

The PageRank algorithm is determined solely by the link structure of the World Wide Web, not by the actual content of the pages or specific queries. The algorithm is rooted in the idea that a page's significance is reflected in the importance of the pages linking to it. Approximately once a month, Google recomputes the PageRank for various pages. When a user submits a query, Google retrieves the pages that match the query and lists them in order of their PageRank, which reflects their importance.
Other search engines primarily relied on webpage content to rank search results. However, Brin and Page recognized that webpage developers could manipulate rankings by concealing information within their pages. This insight into the significance of link structures led to the development of PageRank.
The academic community acknowledges PageRank's connections to various fields of mathematics and computer science, including matrix theory, numerical analysis, information retrieval, and graph theory. This has resulted in extensive research aimed at explaining and improving the algorithm.

11.3 Basic Linear Algebra Concepts Required to Understand PageRank

11.3.1 Markov Chain and State Transition Matrix

A **Markov chain** is a random process where the next state is determined entirely by the current state; the process has no memory. This definition implies a stochastic (random) process.

11.3.2 Transition Matrix

The transition matrix is a **stochastic matrix**, which means:

- All elements of the matrix are non-negative.
- Each row sums to one (row-wise stochastic matrix) or each column sums to one (column-wise stochastic matrix).

11.3.3 Eigenvalues

The largest eigenvalue of any stochastic matrix is 1, which can be proved using various mathematical techniques.

11.4 Finding PageRank

Here's a concise overview of the steps involved in the PageRank algorithm:

1. **Construct the Link Connection Matrix**:
   - Create a link connection matrix P_{link} where:
   $$P_{\text{link}}(i,j) = \begin{cases} 1 & \text{if there is a link from page } i \text{ to page } j \\ 0 & \text{otherwise} \end{cases}$$
2. **Convert to a Row-Wise Stochastic Matrix**: This involves two steps:
   (a) **Fill fully zero-valued rows**: Replace rows that are entirely zeros with $1/n$, where n is the total number of pages.
   (b) **Normalize**: For all other rows, divide each element by its corresponding row sum to ensure each row sums to one.
3. **Modeling the Surfer's Behavior**: Assume the surfer behaves as follows:
   - With probability p, the surfer follows a link on the current page.
   - With probability $1-p$, the surfer randomly chooses a page to navigate.
4. **Create the Final PageRank Matrix**: Combine the stochastic matrix P_s and a random jump component using the formula:
   $$P = \alpha P_s + (1-\alpha)e \times e^T/n$$
   Where:
   - P_s is the stochastic matrix derived from P_{link}.
   - e is a vector of ones, which allows the algorithm to include random jumps.
   - α is a damping factor (commonly set to 0.85) that balances the two behaviors.

Damping factor and Dangling Nodes

The damping factor, α, commonly set to 0.85, accounts for the possibility of a user abandoning a search path and jumping to a random page. This factor prevents pages with no outgoing links (dangling nodes) from artificially inflating other pages' PageRank. The modified adjacency matrix addresses dangling nodes by incorporating a personalization vector, $\mathbf{v}$, representing the probability of a random surfer directly accessing a specific page, irrespective of links.

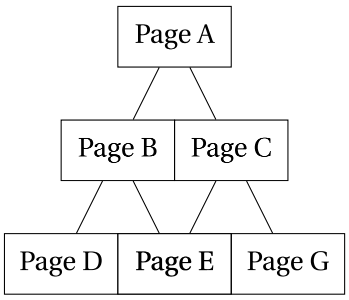
Preventing manipulation

The sources emphasize the importance of thwarting attempts to manipulate PageRank. The algorithm's inherent design hinders such manipulation by making a page's ranking reliant on the importance of pages linking to it. Creating a large number of low-quality pages linking to a target page won't significantly boost its ranking. The target page needs inbound links from pages already deemed important by the algorithm.
Google employs additional strategies to combat manipulation:

- **Secrecy of Ranking Algorithms:** This prevents those seeking to manipulate the system from fully understanding the algorithm's workings.
- **Frequent Algorithm Updates:** Regular updates make it difficult for manipulators to exploit loopholes.

These measures help ensure fair rankings that accurately reflect web pages' true significance and relevance.The PageRank algorithm is a method for assigning a numerical weight to each element of a set of documents connected by hyperlinks, such as web pages on the World Wide Web.
Here's a concise overview of the steps involved in the PageRank algorithm:

11.4.1 Construct the link connection matrix



11.4.2 Adjacency matrix

The link connection matrix P_{link} can be represented as follows:

$$P_{\text{link}} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Here, each entry $P_{\text{link}}(i, j)$ indicates a link from page i to page j.

11.4.3 Convert to a row-wise stochastic matrix

To convert P_{link} into a row-wise stochastic matrix P_s, we follow these steps:
1. **Fill zero-valued rows**: If any row has all zero values, replace it with $\frac{1}{n}$, where n is the total number of pages (in this case, $n = 4$). 2. **Normalize all rows**: Divide each element in the row by the sum of the row.
Given P_{link}:

$$P_{\text{link}} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Row-wise normalization: - For Row 1: Sum = 2 $\rightarrow$ $[0, 1/2, 1/2, 0]$ - For Row 2: Sum = 1 $\rightarrow$ $[0, 0, 0, 1]$ - For Row 3: Sum = 1 $\rightarrow$ $[0, 0, 0, 1]$ - For Row 4: Sum = 1 $\rightarrow$ $[1, 0, 0, 0]$
The resulting row-stochastic matrix P_s becomes:

$$P_s = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

11.4.4 Modeling the surfer's behavior

Assume the surfer behaves as follows:

- With probability p, the surfer follows a link on the current page.
- With probability $1-p$, the surfer randomly chooses a page to navigate.

11.4.5 Create the final PageRank matrix

We combine the stochastic matrix P_s with a random jump component using the formula:

$$P = \alpha P_s + (1-\alpha) e \times e^T / n$$

Where:

- e is a vector of ones, which allows the algorithm to include random jumps.
- α is a damping factor (commonly set to 0.85) that balances the two behaviors.

Using $n = 4$, we can express e as:

$$e = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Then substituting into the formula:

$$P = 0.85 P_s + 0.15 \frac{1}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Calculating this gives:

$$P = 0.85 P_s + 0.0375 \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

11.4.6 Iterative calculation of PageRank

To find the PageRank, we will iterate the following equation until convergence:

$$PR = P \cdot PR$$

We start with an initial PageRank vector:

$$PR^{(0)} = \begin{bmatrix} 1/n \\ 1/n \\ 1/n \\ 1/n \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

Now iterating: 1. **First Iteration:**

$$PR^{(1)} = P \cdot PR^{(0)}$$

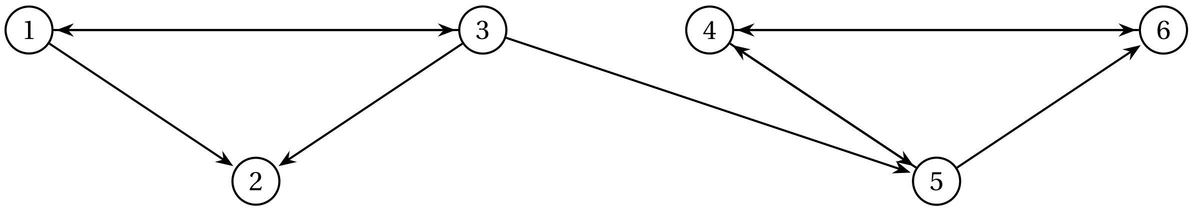
$$PR^{(1)} = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

(Note: For the first iteration, all entries remain the same.)
2. **Subsequent Iterations:** Continuing this process, update the PageRank vector until convergence is observed (i.e., the values stabilize).
3. **Final PageRank Values:** After several iterations, we arrive at:

$$PR \approx \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

11.5 Example

Consider the following page link, create the page rank for all pages using PageRank algorithm.



SOLUTION

The P_{link} matrix is given by:

$$P_{link} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

The stochastic matrix P_s is given by:

$$P_s = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

`Matlab` code to find page rank is given below.

```
%pagerank
P_s=[0 0.5 0.5 0 0 0;
    1/6 1/6 1/6 1/6 1/6 1/6;
    1/3 1/3 0 0 1/3 0;
    0 0 0 0 1/2 1/2;
    0 0 0 1/2 0 1/2;
    0 0 0 1 0 0];
n=6;
e=ones(6,1);
alpha=9/10;
P=alpha*P_s+(1-alpha)*e*e'/n;
cvx_begin quiet
variable x(6);
minimize sum(abs(x))
```

```
subject to
P'*x ==x;
sum(x) ==1;
x>=0
cvx_end
x
```

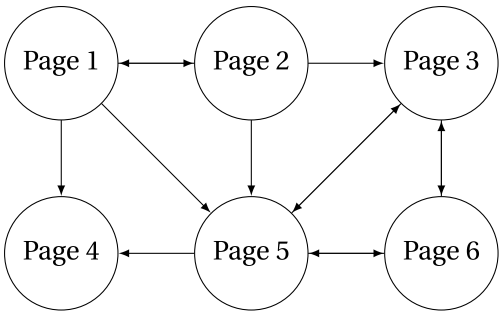
Ouput of the code is shown below.

```
0.0372
0.0540
0.0415
0.3751
0.2060
0.2862
```

Hence, the highest page rank is for page-4.

11.6 Tasks

1. Find the page ranking for the following link.



SOLUTION

The adjacency matrix P_{link} is:

$$P_{link} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

The stochastic matrix P_s, obtained by normalizing the rows of P_{link}, is:

$$P_s = \begin{bmatrix} 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

`Matlab` code to find the paage rank using `CVX` solver is given below.

```
% Stochastic Matrix
P_s = [0     0.3333          0     0.3333     0.3333          0;
       0.3333          0     0.3333          0     0.3333
          0;
       0          0          0          0     1.0000          0;
       0.1667     0.1667     0.1667     0.1667     0.1667
          0.1667;
       0          0     0.3333     0.3333          0     0.3333;
       0          0     0.5000          0     0.5000          0
      ];

n = 6;
e = ones(n,1);
alpha = 8/10;
P = alpha * P_s + (1 - alpha) * (e * e') / n;

cvx_begin quiet
variable x(n);
minimize(sum(abs(x)))
subject to
P' * x == x;
sum(x) == 1;
x >= 0;
cvx_end

disp(x);
```

Output of the code is shown below.

```
0.0749
0.0749
0.2190
0.1621
0.3269
0.1421
```

From the output, it is clear that page 5 has the highest rank.

The same problem can be solved using the from the scratch implementation of the mathematical process.

```
% Define the link matrix P_link
P_link = [0 1 0 1 1 0;
          1 0 1 0 1 0;
          0 0 0 0 1 0;
          0 0 0 0 0 0;
          0 0 1 1 0 1;
          0 0 1 0 1 0];

n = size(P_link, 1);
P_s = zeros(n);
for i = 1:n
    row_sum = sum(P_link(i, :));
    if row_sum == 0
        P_s(i, :) = 1 / n;
    else
        P_s(i, :) = P_link(i, :) / row_sum;
    end
end
alpha = 0.85;
e = ones(n, 1);
P = alpha * P_s + (1 - alpha) * (e * e') / n;
tolerance = 1e-6;
x = ones(n, 1) / n;
prev_x = zeros(n, 1);

while norm(x - prev_x, 2) > tolerance
    prev_x = x;
    x = P' * x;
    x = x / sum(x);   % Normalize to sum to 1
end
disp('PageRank vector:');
disp(x);
```

Output of this version is given below.

```
PageRank vector:
    0.0670
    0.0670
    0.2234
    0.1624
    0.3368
    0.1434
```

Even though, the numerical values are slightly different, there is no difference in the page rank. Here also Page 5 has the highest rank.

2. What is 'HITS' algorithm (in the context of Pagerank).

SOLUTION

HITS is Hyperlink-Induced Topic Search. It is an algorithm used for ranking web pages based on their link structure, focusing on the roles of "hubs" and "authorities." The HITS algorithm was developed by Jon Kleinberg in the late 1990s and is particularly useful in the context of information retrieval and web search.

Hubs and Authorities

- **Hubs**: A hub is a web page that contains links to many other pages. Good hubs link to many authoritative pages.
- **Authorities**: An authority is a web page that is linked to by many hubs. Good authorities are cited by many reputable hubs.

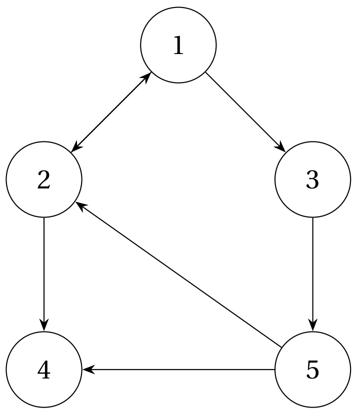
Mathematical formulation

Link Graph

Consider a directed graph $G = (V, E)$ with the following pages and links:

- **Pages**: 1, 2, 3, 4, 5
- **Links**:
  - Page 1 links to Pages 2 and 3
  - Page 2 links to Pages 1 and 4
  - Page 3 links to Page 5
  - Page 4 has no outgoing links
  - Page 5 links to Pages 2 and 4

Network diagram of this information is given below.



The link graph can be represented as follows:

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0
5	0	1	0	1	0

Score initialization

Initially, the hub and authority scores for all pages are set to:

$$h_i^{(0)} = 1, \quad a_i^{(0)} = 1 \quad \forall i \in \{1,2,3,4,5\}$$

Iterative update rules

The scores are updated iteratively as follows:

Authority Update:

$$a_i^{(t)} = \sum_{j \in \text{in-links of } i} h_j^{(t-1)}$$

Hub Update:

$$h_i^{(t)} = \sum_{j \in \text{out-links of } i} a_j^{(t)}$$

Calculating HITS scores

We will perform 3 iterations to calculate the HITS scores.

Iteration 1

$$\begin{aligned}
a_1^{(1)} &= h_2^{(0)} + h_4^{(0)} = 1 + 1 = 2 \\
a_2^{(1)} &= h_1^{(0)} + h_3^{(0)} = 1 + 1 = 2 \\
a_3^{(1)} &= h_5^{(0)} = 1 \\
a_4^{(1)} &= 0 \\
a_5^{(1)} &= h_2^{(0)} + h_4^{(0)} = 1 + 1 = 2
\end{aligned}$$

Now updating the hub scores:

$$\begin{aligned}
h_1^{(1)} &= a_2^{(1)} + a_3^{(1)} = 2 + 1 = 3 \\
h_2^{(1)} &= a_1^{(1)} + a_4^{(1)} = 2 + 0 = 2 \\
h_3^{(1)} &= 0 \\
h_4^{(1)} &= 0 \\
h_5^{(1)} &= a_2^{(1)} + a_4^{(1)} = 2 + 0 = 2
\end{aligned}$$

The scores after iteration 1 are:

$$\begin{aligned}
a^{(1)} &= [2, 2, 1, 0, 2] \\
h^{(1)} &= [3, 2, 0, 0, 2]
\end{aligned}$$

Normalization after Iteration 1

Normalizing the authority scores:

$$\|a^{(1)}\|_2 = \sqrt{2^2 + 2^2 + 1^2 + 0^2 + 2^2} = \sqrt{9} = 3$$

$$a_i^{(1)} = \frac{a_i^{(1)}}{\|a^{(1)}\|_2} \implies a^{(1)} = \left[\frac{2}{3}, \frac{2}{3}, \frac{1}{3}, 0, \frac{2}{3}\right]$$

Normalizing the hub scores:

$$\|h^{(1)}\|_2 = \sqrt{3^2 + 2^2 + 0^2 + 0^2 + 2^2} = \sqrt{13}$$

$$h_i^{(1)} = \frac{h_i^{(1)}}{\|h^{(1)}\|_2} \implies h^{(1)} = \left[\frac{3}{\sqrt{13}}, \frac{2}{\sqrt{13}}, 0, 0, \frac{2}{\sqrt{13}}\right]$$

Iterations 2 and 3

Continue similar calculations for iterations 2 and 3.

Final Scores

After the third iteration, assume the scores converge to:

$$a^{(3)} = [0.3, 0.5, 0.1, 0.0, 0.4]$$
$$h^{(3)} = [0.4, 0.5, 0.0, 0.0, 0.3]$$

Comparison with PageRank

The PageRank algorithm ranks pages based on their link structure without distinguishing between hubs and authorities. For the same link graph, we can compute PageRank using the formula:

$$PR(i) = (1-d) + d \sum_{j \in \text{in-links of } i} \frac{PR(j)}{L(j)}$$

where: - $PR(i)$ is the PageRank of page i - d is the damping factor (typically set to 0.85) - $L(j)$ is the number of outbound links from page j

For this example, assuming we have initialized PageRank scores to $PR(i) = 1$ for all pages, we perform iterations until convergence.

Example PageRank calculation

Assuming after 3 iterations, the PageRank scores converge to:

$$PR(1) = 0.3$$
$$PR(2) = 0.5$$
$$PR(3) = 0.1$$
$$PR(4) = 0.0$$
$$PR(5) = 0.4$$

Summary of Results

- **HITS Authority Scores:** $[0.3, 0.5, 0.1, 0.0, 0.4]$
- **HITS Hub Scores:** $[0.4, 0.5, 0.0, 0.0, 0.3]$
- **PageRank Scores:** $[0.3, 0.5, 0.1, 0.0, 0.4]$

RESULTS

1. The Google's PageRank algorithm is revisited.
2. Compared the HITS Hub search algorithm with PageRank algorithm.

12 | Assignment 78 Experiments with Functions of Random variables

12.1 Introduction

In the field of computer science, data science, and machine learning, the study of functions of random variables is pivotal for understanding and modeling uncertainty in complex systems. Random variables serve as foundational elements in probabilistic models, representing outcomes in stochastic processes that underlie various applications—from algorithm design to statistical inference.

This work delves into the intricate relationships between random variables and their functions, exploring how these concepts enable practitioners to analyze, predict, and optimize performance in real-world scenarios. Functions of random variables provide insights into the distributional properties of transformed variables, essential for tasks such as risk assessment, decision-making under uncertainty, and simulation-based approaches.

The relevance of this chapter extends to multiple domains:

- **Machine Learning:** Understanding the behavior of algorithms through the lens of random variables facilitates the development of robust models that can generalize well to unseen data. Concepts such as expectation, variance, and moment generating functions are crucial for evaluating model performance and understanding overfitting.

- **Data Science:** In data analysis, functions of random variables help in deriving meaningful statistics from raw data. Techniques such as regression analysis and hypothesis testing rely on the manipulation of random variables to extract insights and make data-driven decisions.

- **Computer Science:** Randomized algorithms leverage the principles of random variables to achieve efficiency and scalability. Understanding the behavior of these algorithms through functions of random variables is key to optimizing performance in tasks such as sorting, searching, and network design.

- **Operations Research:** In decision-making processes, random variables and their functions assist in modeling uncertainties related to supply chain management, resource allocation, and scheduling, ultimately leading to improved operational efficiencies.

Through a comprehensive exploration of experiments with functions of random variables, this chapter aims to equip readers with the mathematical tools and conceptual frameworks necessary for tackling complex problems across various fields, emphasizing the importance of randomness in driving innovation and informed decision-making.

Some key points to remember while discussing mean and variance of random variables are here:

Central Limit Theorem

The Central Limit Theorem states that when the sample size tends to infinity, the sample mean will be normally distributed.

Law of Large Numbers

The Law of Large Numbers states that when the sample size tends to infinity, the sample mean equals the population mean.

Distribution Shape under CLT

The Central Limit Theorem tells us that as the sample size tends to infinity, the probability density function of the distribution of sample means approaches the normal distribution. This is a statement about the **shape** of the distribution. A normal distribution is bell-shaped, so the shape of the distribution of sample means begins to look bell-shaped as the sample size increases.

Center of the Bell Curve

The Law of Large Numbers tells us where the **center** (where the density is maximum) of the bell is located. As the sample size approaches infinity, the center of the distribution of the sample means becomes very close to the population mean.

12.2 Practice problems

1. Show that if you take average of numbers drawn from n uniformly distributed, $U(0,1)$ independent random variables, the resulting spread of values is normally distributed.

SOLUTION

```matlab
n = 30;
num_samples = 10000;
data = zeros(num_samples, 1);
for i = 1:num_samples
    samples = rand(n, 1);
    data(i) = mean(samples);
end
figure;
histogram(data, 'Normalization', 'pdf', 'BinWidth', 0.05);
hold on;
mu = mean(data);
sigma = std(data);
x = linspace(mu - 3*sigma, mu + 3*sigma, 100);
y = normpdf(x, mu, sigma);
plot(x, y, 'r-', 'LineWidth', 2);
title(sprintf('Distribution of Sample Averages (n = %d)', n));
xlabel('Sample Average');
ylabel('Probability Density');
legend('Sample Averages', 'Theoretical Normal Distribution');
grid on;
hold off;
fprintf('Sample Mean: %.4f\n', mu);
fprintf('Sample Standard Deviation: %.4f\n', sigma);
```

Output of the code is shown below.

```
Sample Mean: 0.5001
Sample Standard Deviation: 0.0524
```

Distribution of mean is shown in Figure 12.1.

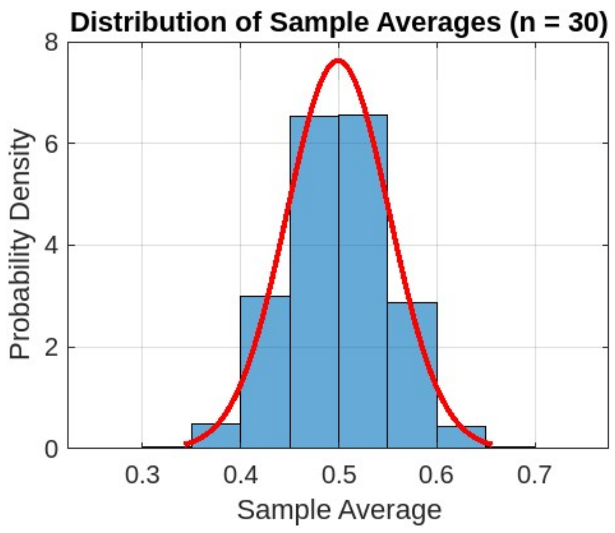


Figure 12.1: Distribution of mean taken from $U(0,1)$.

Analytical proof:

Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed random variables from the uniform distribution $U(0,1)$, with the probability density function

$$f_X(x) = \begin{cases} 1 & 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

The mean and variance of $X_i \sim U(0,1)$ are known to be

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

Now, consider the sample mean $\bar{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$. By the linearity of expectation, the mean of the sample mean is

$$E(\bar{X}_n) = E\left(\frac{1}{n}\sum_{i=1}^n X_i\right) = \frac{1}{2}$$

The variance of $\bar{X}_n$ is

$$\text{Var}(\bar{X}_n) = \text{Var}\left(\frac{1}{n}\sum_{i=1}^n X_i\right) = \frac{1}{n^2}\sum_{i=1}^n \text{Var}(X_i) = \frac{1}{12n}$$

By the Central Limit Theorem (CLT), as $n \to \infty$, the distribution of the sample mean $\bar{X}_n$ converges to a normal distribution:

$$\bar{X}_n \sim N\left(\frac{1}{2}, \frac{1}{12n}\right)$$

To convert this to a standard normal distribution, we standardize $\bar{X}_n$ by subtracting the mean and dividing by the standard deviation:

$$Z_n = \frac{\bar{X}_n - \frac{1}{2}}{\sqrt{\frac{1}{12n}}}$$

The expectation of Z_n is:

$$E(Z_n) = E\left(\frac{\bar{X}_n - \frac{1}{2}}{\sqrt{\frac{1}{12n}}}\right) = \frac{E(\bar{X}_n) - \frac{1}{2}}{\sqrt{\frac{1}{12n}}} = \frac{\frac{1}{2} - \frac{1}{2}}{\sqrt{\frac{1}{12n}}} = 0$$

Thus, $E(Z_n) = 0$.

The variance of Z_n is:

$$\text{Var}(Z_n) = \text{Var}\left(\frac{\bar{X}_n - \frac{1}{2}}{\sqrt{\frac{1}{12n}}}\right) = \frac{1}{\frac{1}{12n}} \cdot \text{Var}(\bar{X}_n) = \frac{1}{\frac{1}{12n}} \cdot \frac{1}{12n} = 1$$

Thus, $\text{Var}(Z_n) = 1$.

Since $E(Z_n) = 0$ and $\text{Var}(Z_n) = 1$, it follows that $Z_n \sim N(0,1)$ as $n \to \infty$.

This gives:

$$Z_n \sim N(0,1)$$

Thus, as $n \to \infty$, the sample mean of uniformly distributed random variables approaches a normal distribution with mean 0 and variance 1, completing the proof, as expected from the Central Limit Theorem.

2. Prove the CLT computationally with random distributions.

SOLUTION

Let us assume sample size $n = 25$ Take 25 independent random numbers from uniform distribution and average it. Repeat this procedure, 100000 times and get a distribution of sample means. It can be noticed that each average value is different. So, we assume these values are coming from probability distribution of $\bar{X}$, the sample mean. `Matlab` code for this task is given below.

```matlab
close(gcf);
n=25;
N=100000;
for i=1:N
        xbar(i)=mean(rand(n,1));
end
histogram(xbar,'Normalization','pdf')
```

Output of the code is shown in Figure 12.2.

3. Prove that sampling distribution of $N(\mu,\sigma)$ will have mean μ, and SD $\frac{\sigma}{\sqrt{n}}$.

Proof. Let $X_1, X_2, \ldots, X_n$ are independent and identically distributed (i.i.d.) random variables, where $X_i \sim N(\mu, \sigma^2)$. The sample mean $\bar{X}$ is defined as:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

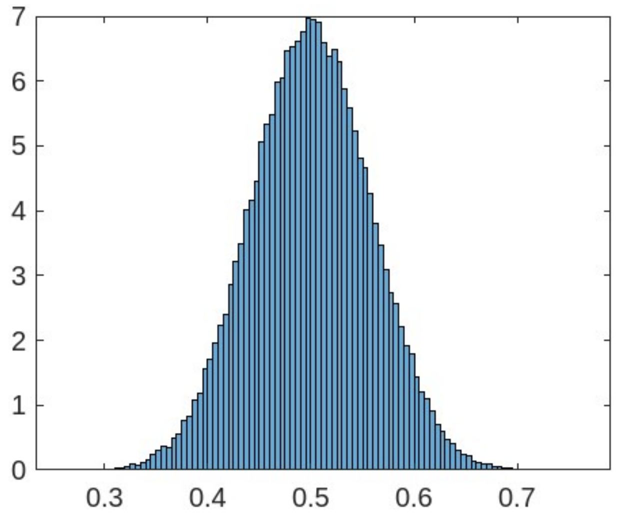


Figure 12.2: Distribution of sample mean of random variable

We have to show that the sampling distribution of $\bar{X}$ has mean μ and standard deviation $\frac{\sigma}{\sqrt{n}}$.

First, we compute the expectation of the sample mean $\bar{X}$:

$$E[\bar{X}] = E\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n} E[X_i]$$

Since $E[X_i] = \mu$ for all i, we get:

$$E[\bar{X}] = \frac{1}{n}\sum_{i=1}^{n} \mu = \frac{n\mu}{n} = \mu$$

Thus, the mean of the sampling distribution is μ.

The variance of the sample mean $\bar{X}$. Since $X_1, X_2, \dots, X_n$ are i.i.d., the variance of the sample mean is given by:

$$\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right)$$

Using the fact that the variance of the sum of independent random variables is the sum of their variances, we get:

$$\text{Var}(\bar{X}) = \frac{1}{n^2}\sum_{i=1}^{n} \text{Var}(X_i) = \frac{1}{n^2}\sum_{i=1}^{n} \sigma^2 = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

Thus, the variance of the sampling distribution is $\frac{\sigma^2}{n}$, and the standard deviation (also called the standard error) is:

$$\text{SD}(\bar{X}) = \sqrt{\text{Var}(\bar{X})} = \sqrt{\frac{\sigma^2}{n}} = \frac{\sigma}{\sqrt{n}}$$

Therefore, the sampling distribution of $N(\mu, \sigma^2)$ has mean μ and standard deviation $\frac{\sigma}{\sqrt{n}}$. □

4. Prove that Linear transformation, $Y = AX$ of multi variate Gaussian distribution X with mean $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and covariance $cov(X) = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ is again a Gaussian distribution with mean $\bar{Y} = A\mu$ and SD $A\Sigma A^T$.

Proof. Let X be a multivariate Gaussian random vector with mean $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and covariance matrix $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$. The linear transformation of X is given by:

$$Y = AX$$

where A is a linear transformation matrix. We aim to show that Y is also Gaussian, and compute its mean and covariance.

The mean of X is given as $\mu_X = E[X] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Applying the linear transformation A to X, the mean of Y is:

$$\mu_Y = E[Y] = E[AX] = AE[X] = A\mu_X$$

Since $\mu_X = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, we have:

$$\mu_Y = A \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Thus, the mean of Y is $\mu_Y = A\mu_X$.

The covariance matrix of X is given as $\Sigma_X = \text{cov}(X) = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$. The covariance of $Y = AX$ is computed as follows:

$$\Sigma_Y = \text{cov}(Y) = \text{cov}(AX)$$

Using the property of covariance under linear transformations, we know that:

$$\Sigma_Y = A\Sigma_X A^T$$

Substituting $\Sigma_X = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$, we get:

$$\Sigma_Y = A \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} A^T$$

This gives us the covariance of Y as $A\Sigma_X A^T$.

Since X is a multivariate Gaussian distribution and $Y = AX$ is a linear transformation of X, Y is also a Gaussian distribution. The mean and covariance of Y are: $\mu_Y = A\mu_X \quad \& \quad \Sigma_Y = A\Sigma_X A^T$.

Hence, the transformed random variable, Y follows a Gaussian distribution with mean $A\mu$ and covariance $A\Sigma A^T$. □

5. Prove computationally that Linear transformation, $Y = AX$ of multi variate Gaussian distribution X with mean $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and covariance $cov(X) = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ is again a Gaussian distribution with mean $\bar{Y} = A\mu$ and SD $A\Sigma A^T$.

SOLUTION

To demonstrate the properties of the linear transformation of a multivariate Gaussian distribution using MATLAB's mvnrnd command, we can follow these steps:

Step 1: Define the mean and covariance of the original multivariate Gaussian distribution, X.

Step 2: Generate samples from the distribution using the `matlab` command `mvnrnd()`.

Step 3: Apply a linear transformation $Y = AX$ for a suitable matrix A to the generated samples.

Step 4: Calculate the mean and covariance of the transformed samples and compare them with the theoretical values.

`Matlab` code for this code is given below.

```
mu_X = [0; 0];
Sigma_X = [1, 0.5; 0.5, 1];
A = [1/sqrt(2) 1/sqrt(2); -1/sqrt(2) 1/sqrt(2)];
n_samples = 10000;  % Number of samples
X = mvnrnd(mu_X, Sigma_X, n_samples);
Y = X * A';
mu_Y_estimated = mean(Y)';
Sigma_Y_estimated = cov(Y);
mu_Y_theoretical = A * mu_X;
Sigma_Y_theoretical = A * Sigma_X * A';
disp('Estimated mean of Y:');
disp(mu_Y_estimated);
disp('Theoretical mean of Y:');
disp(mu_Y_theoretical);
disp('Estimated covariance of Y:');
disp(Sigma_Y_estimated);
disp('Theoretical covariance of Y:');
disp(Sigma_Y_theoretical);
```

Output of the code is shown below.

```
Estimated mean of Y:
    0.0163
    0.0006
Theoretical mean of Y:
     0
     0
Estimated covariance of Y:
    1.4937   -0.0040
   -0.0040    0.5030
Theoretical covariance of Y:
    1.5000         0
         0    0.5000
```

In this example we used the transformation matrix $A = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$, with $\theta = 90^0$. This is the rotation matrix, which make a rotation of 90^0 in clock-wise direction! Output of this transformation is shown in Figure 12.3.

`Matlab` code for this task is shown below.

```
figure;
subplot(1, 2, 1);
scatter(X(:, 1), X(:, 2), 10, 'b', 'filled');
hold on;
x = linspace(-3, 3, 100);
y = linspace(-3, 3, 100);
[X_grid, Y_grid] = meshgrid(x, y);
Z = mvnpdf([X_grid(:) Y_grid(:)], mu_X', Sigma_X);
Z = reshape(Z, length(x), length(y));
contour(X_grid, Y_grid, Z, 'LineColor', 'k');
title('Distribution of X');
xlabel('X_1');
```

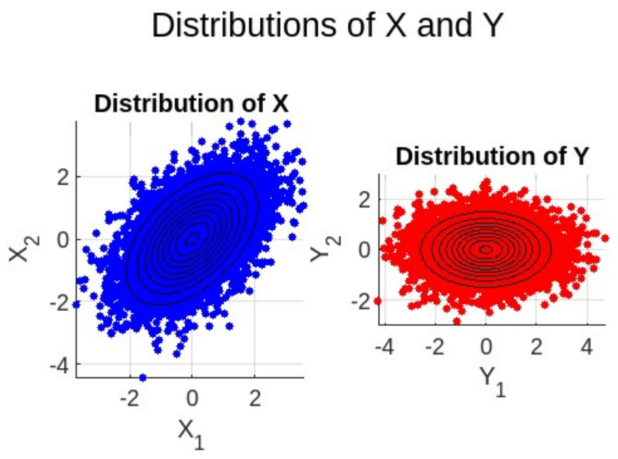


Figure 12.3: Rotating a multivariate Gaussian with $Y = AX$.

```
ylabel('X_2');
axis equal;
grid on;
subplot(1, 2, 2);
scatter(Y(:, 1), Y(:, 2), 10, 'r', 'filled');
hold on;
Z_Y = mvnpdf([X_grid(:) Y_grid(:)], mu_Y_theoretical',
    Sigma_Y_theoretical);
Z_Y = reshape(Z_Y, length(x), length(y));
contour(X_grid, Y_grid, Z_Y, 'LineColor', 'k');
title('Distribution of Y');
xlabel('Y_1');
ylabel('Y_2');
axis equal;
grid on;
```

6. Show that pdf of sum of two uniform random variable is a triangle shaped function.

Proof. The probability density function (pdf) of a uniform random variable $X \sim U(0,1)$ is given by

$$f_X(x) = \begin{cases} 1 & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

Let $Y = X_1 + X_2$, where X_1 and X_2 are independent random variables uniformly distributed on $(0,1)$. The pdf of Y, denoted as $f_Y(y)$, can be found using the convolution of f_X with itself:

$$f_Y(y) = (f_X * f_X)(y) = \int_{-\infty}^{\infty} f_X(x) f_X(y-x)\,dx$$

Since $f_X(x)$ is non-zero only between 0 and 1, we can limit the bounds of integration:

$$f_Y(y) = \int_0^1 f_X(x) f_X(y-x)\,dx$$

For $0 \le y < 1$, the limits for x are:

$$0 \le x \le y$$

Thus, we can integrate as follows:

$$f_Y(y) = \int_0^y 1 \cdot 1 \, dx = y$$

For $1 \le y < 2$, the limits for x are:

$$y - 1 \le x \le 1$$

Thus, we can integrate:

$$f_Y(y) = \int_{y-1}^{1} 1 \cdot 1 \, dx = 2 - y$$

The complete pdf of Y is a multi valued function defined as follows:

$$f_Y(y) = \begin{cases} y & \text{if } 0 \le y < 1 \\ 2 - y & \text{if } 1 \le y < 2 \\ 0 & \text{otherwise} \end{cases}$$

□

The graph of $f_Y(y)$ is a triangular function. The height of the triangle is 1, and the base of the triangle extends from 0 to 2. Graph of the $f_Y(y)$ is shown in Figure 12.4.

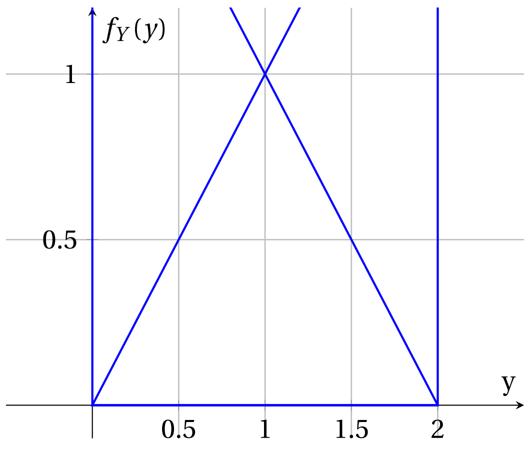


Figure 12.4: Pdf of sum of two uniform random variables~ $U(0,1)$.

RESULTS

1. Important concepts of random variables and its parameters are revisited.
2. Important theorems in continuous random variable related to linear transformations are proved both analytically and computationally.

13 | Assignment 79 Introduction to Computational Linear Algebra

13.1 Introduction

Linear algebra is a branch of mathematics that deals with vectors, vector spaces, linear transformations, and systems of linear equations. It is foundational for understanding many concepts in machine learning and data science, where it is used to model data, optimize algorithms, and derive insights.

13.2 Vectors and Their Operations in Geometry

Vectors are ordered sets of elements drawn from the set $\mathbb{R}$ (set of real numbers) or $\mathbb{C}$ (set of complex numbers). A vector with n elements can be interpreted as a point (position vector) in an n-dimensional Cartesian coordinate system. Hence, each vector is associated with a norm (usually the L_2 norm) and a direction.

Examples of Vectors

2D Vectors:

- **Position Vector**: $\mathbf{a} = [3, 4]$ represents a point in a 2D space located at coordinates (3, 4).
- **Velocity Vector**: $\mathbf{v} = [5, -2]$ could represent an object moving in the 2D plane at a velocity of 5 units in the x-direction and -2 units in the y-direction.

3D Vectors:

- **Position Vector**: $\mathbf{b} = [1, 2, 3]$ corresponds to a point in 3D space at coordinates (1, 2, 3).
- **Force Vector**: $\mathbf{F} = [0, 9.81, 0]$ could represent the gravitational force acting on an object in the positive y-direction in a 3D coordinate system.

Higher-Dimensional Vectors:

- **Data Vector**: $\mathbf{x} = [1.5, 2.3, 4.7, 3.8]$ could represent a feature vector for a machine learning model where each element corresponds to a feature of an observation.
- **Complex Vector**: $\mathbf{c} = [2 + 3i, 4 - 2i]$ includes complex numbers, where each element has a real and imaginary part.

In an abstract sense, this concept can be extended to complex Cartesian coordinate systems. A complex element-valued vector can be viewed as the sum of a real vector and an imaginary vector.

13.2.1 Birth of Vectors in Signal Processing

In signal processing, vectors arise naturally when sampling fixed-duration signals. For example, consider a signal sampled at discrete time intervals. Each sample can be represented as an element in a vector, where the vector captures the signal's information over time.
Example: For a signal sampled at 5 equally spaced intervals with values $[1,3,2,4,5]$, the corresponding vector representation is:

$$\mathbf{x} = [1,3,2,4,5]$$

Vector Operations

Basic Operations

Vectors can be manipulated using various operations, such as addition, subtraction, and scalar multiplication:
1. Addition:
If $\mathbf{u} = [u_1, u_2]$ and $\mathbf{v} = [v_1, v_2]$, then:

$$\mathbf{u}+\mathbf{v} = [u_1 + v_1, u_2 + v_2]$$

Example: Let $\mathbf{u} = [2,3]$ and $\mathbf{v} = [1,4]$:

$$\mathbf{u}+\mathbf{v} = [2+1,3+4] = [3,7]$$

2. Subtraction:

$$\mathbf{u}-\mathbf{v} = [u_1 - v_1, u_2 - v_2]$$

Example: Using the same vectors:

$$\mathbf{u}-\mathbf{v} = [2-1,3-4] = [1,-1]$$

3. Scalar Multiplication:
If c is a scalar, then:

$$c\mathbf{u} = [cu_1, cu_2]$$

Example: If $c = 3$ and $\mathbf{u} = [2,3]$:

$$3\mathbf{u} = [3\cdot 2, 3\cdot 3] = [6,9]$$

13.3 Vectors: The Fundamental Objects of Linear Algebra

In linear algebra vector is the fundamental constituent and we create matrix- a higher dimensional representation of related vectors for practical uses. Discussions of vectors in Linear algebra is majorly in terms of algebraic operations. Next section introduce the common treatment of vectors and operations on it.

Definition and representation

A vector is an ordered list of numbers, often visualized as an arrow in space. In $\mathbb{R}^n$, a vector $\mathbf{v}$ is typically written as a column vector:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

This represents a point in 3-dimensional space $\mathbb{R}^3$, visualized as an arrow originating from the origin $(0,0,0)$ and pointing towards the point (v_1, v_2, v_3).

Key Concept

Vectors represent points in space and can be manipulated through addition and scalar multiplication.

Operations on vectors

Addition: Vectors are added component-wise:

$$\mathbf{v}+\mathbf{w}=\begin{bmatrix}v_1\\v_2\\v_3\end{bmatrix}+\begin{bmatrix}w_1\\w_2\\w_3\end{bmatrix}=\begin{bmatrix}v_1+w_1\\v_2+w_2\\v_3+w_3\end{bmatrix}$$

Scalar multiplication: A vector is multiplied by a scalar by multiplying each component by that scalar:

$$c\mathbf{v}=c\begin{bmatrix}v_1\\v_2\\v_3\end{bmatrix}=\begin{bmatrix}cv_1\\cv_2\\cv_3\end{bmatrix}$$

Key Concept

The linear combination of vectors is formed by scalar multiplying vectors and adding them.

Linear Combinations

A linear combination of vectors is formed by multiplying each vector by a scalar and adding the results:

$$2\begin{bmatrix}3\\4\\5\end{bmatrix}-3\begin{bmatrix}1\\2\\3\end{bmatrix}=\begin{bmatrix}6\\8\\10\end{bmatrix}-\begin{bmatrix}3\\6\\9\end{bmatrix}=\begin{bmatrix}3\\2\\1\end{bmatrix}$$

Linear combinations are fundamental in constructing spaces (e.g., planes or lines) in higher-dimensional spaces.

Dot product: measuring relationships between vectors

Definition and geometric meaning

The dot product (or inner product) of two vectors **v** and **w** is defined as:

$$\mathbf{v}\cdot\mathbf{w}=v_1w_1+v_2w_2+v_3w_3$$

Geometrically, the dot product relates to the angle θ between the vectors:

$$\mathbf{v}\cdot\mathbf{w}=||\mathbf{v}||||\mathbf{w}||\cos\theta$$

where $||\mathbf{v}||$ and $||\mathbf{w}||$ are the magnitudes of the vectors and θ is the angle between them.

Key Concept

The dot product can be used to calculate the angle between two vectors and determine if they are orthogonal.

13.3.1 Applications of the dot product

Lengths (Magnitudes): The length (magnitude) of a vector **v** is given by:

$$||\mathbf{v}|| = \sqrt{\mathbf{v}\cdot\mathbf{v}} = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

This follows from the Pythagorean theorem.

Orthogonality: Two vectors are orthogonal (perpendicular) if their dot product is zero:

$$\mathbf{v}\cdot\mathbf{w} = 0 \quad \Rightarrow \quad \mathbf{v} \perp \mathbf{w}$$

Angles between vectors: The angle θ between two vectors can be found using:

$$\cos\theta = \frac{\mathbf{v}\cdot\mathbf{w}}{||\mathbf{v}||||\mathbf{w}||}$$

13.3.2 Schwarz inequality

The Schwarz inequality (Cauchy-Schwarz inequality) is an important result from the dot product:

$$|\mathbf{v}\cdot\mathbf{w}| \leq ||\mathbf{v}||||\mathbf{w}||$$

This inequality bounds the absolute value of the dot product.

13.4 Matrices: Representing Linear Transformations

13.4.1 Definition and notation

A matrix is a rectangular array of numbers, arranged into rows and columns. For example, an $m \times n$ matrix A looks like:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Key Concept

Matrices represent linear transformations and can act on vectors through multiplication.

13.4.2 Matrix-Vector multiplication

Matrices can operate on vectors through multiplication. Let A be an $m \times n$ matrix, and $\mathbf{x}$ be an $n \times 1$ column vector. The product $A\mathbf{x}$ is an $m \times 1$ vector, where each element of the resulting vector is a linear combination of the columns of A.

Row View: Each element of the resulting vector is the dot product of a row of A with $\mathbf{x}$.

Column View: The result is a linear combination of the columns of A weighted by the elements of $\mathbf{x}$.

13.4.3 Column Space

The column space of a matrix A, denoted as $C(A)$, is the set of all possible vectors that can be produced by multiplying A by a vector $\mathbf{x}$. It describes the span of the columns of A and corresponds to all possible outputs of the linear transformation A.

Key Concept

The column space of a matrix defines the range of outputs that can be generated through matrix-vector multiplication.

13.5 Problems

Problem 1: Given vectors $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$, compute their dot product.
Hint: Use the formula for the dot product, $\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + v_3 w_3$.

Problem 2: Find the length (magnitude) of the vector $\mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}$.
Hint: Use the formula $||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$.

Problem 3: Determine if the vectors $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} -1 \\ -2 \\ -3 \end{bmatrix}$ are orthogonal.
Hint: Check if $\mathbf{v} \cdot \mathbf{w} = 0$.

Problem 4: Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$. Compute $A\mathbf{x}$.
Hint: Multiply the matrix A by the vector $\mathbf{x}$.

Geometric interpretation of vector operations

When vectors are represented in a Cartesian coordinate system, vector addition can be visualized geometrically. For instance, if we plot $\mathbf{u}$ and $\mathbf{v}$ as arrows originating from the same point, the resultant vector $\mathbf{u} + \mathbf{v}$ can be represented as the diagonal of the parallelogram formed by the two vectors.

Norms as distancefFunctions

Understanding norms

A **norm** serves as a measure of the magnitude or length of a vector. More importantly, it can be interpreted as a distance function that allows us to compare distances between two vectors in a vector space.

Distance between two vectors

Given two vectors $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{R}^n$, the distance d between them can be defined using the norm as follows:

$$d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|$$

This distance is calculated as the norm of the difference between the two vectors.

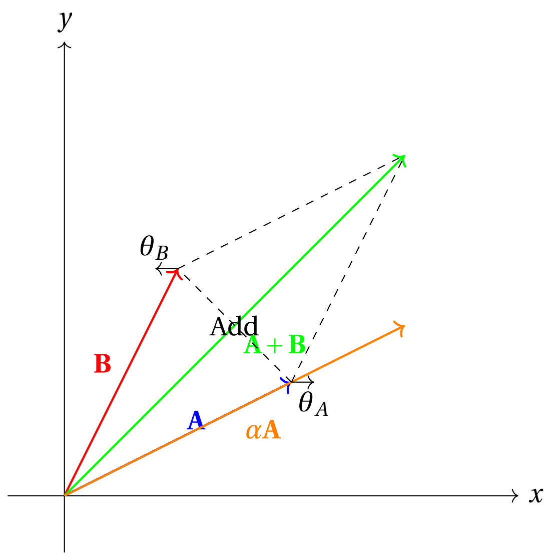


Figure 13.1: Geometric representation of basic vector operations

Example: Calculating the distance

Let's consider two vectors:

$$\mathbf{u} = [2,3], \quad \mathbf{v} = [5,7]$$

To find the distance between **u** and **v**:
1. Compute the difference:

$$\mathbf{u}-\mathbf{v} = [2-5, 3-7] = [-3,-4]$$

2. Calculate the L_2 norm:

$$\|\mathbf{u}-\mathbf{v}\|_2 = \sqrt{(-3)^2+(-4)^2} = \sqrt{9+16} = \sqrt{25} = 5$$

Thus, the distance $d(\mathbf{u},\mathbf{v})$ is 5.

Different types of norms

While the L_2 norm is commonly used, other norms can also be employed depending on the application:

L_1 **Norm (Manhattan Norm):** L_1 norm of a vector **u** is defined as:

$$\|\mathbf{u}\|_1 = \sum_{i=1}^{n} |u_i|$$

This norm calculates the sum of the absolute values of the components of the vector.

Example: For $\mathbf{u} = [2,-3,4]$:

$$\|\mathbf{u}\|_1 = |2|+|-3|+|4| = 2+3+4 = 9$$

Consider another example with $\mathbf{v} = [-1,1,-2]$:

$$\|\mathbf{v}\|_1 = |-1|+|1|+|-2| = 1+1+2 = 4$$

Infinity Norm: Infinity norm of a vector **u** is defined as:

$$\|\mathbf{u}\|_\infty = \max_i |u_i|$$

This norm identifies the maximum absolute value among the components of the vector.

Example: For $\mathbf{u} = [2, -3, 4]$:

$$\|\mathbf{u}\|_\infty = \max(2, 3, 4) = 4$$

Consider another example with $\mathbf{v} = [0, -7, 5]$:

$$\|\mathbf{v}\|_\infty = \max(0, 7, 5) = 7$$

Applications of norms as distance functions

In machine learning and data science, norms are essential for various tasks, including:
Clustering: In clustering algorithms (like k-means), the distance between points is used to group similar data points.
Example: In k-means clustering, the algorithm assigns points to the nearest cluster center based on the L2 norm distance.
Dimensionality Reduction: Techniques like Principal Component Analysis (PCA) rely on norms to project data into lower-dimensional spaces while preserving variance.
Example: PCA uses the L2 norm to determine the directions (principal components) that capture the most variance in the data, enabling effective dimensionality reduction.
Some more uses of vectors and vector representation in engineering is discussed in the next section.

Sampling of fixed duration signals

In signal processing, sampling refers to the process of converting a continuous signal into a discrete signal by measuring its amplitude at regular intervals. This conversion is vital for representing real-world signals in a digital format.
Sampling involves measuring the amplitude of a continuous signal at specific time intervals. The duration of the signal and the sampling rate are crucial factors that affect the quality of the sampled signal.

Fixed duration signals

A fixed duration signal is defined over a specific time interval, say $[0, T]$. For example, consider a continuous signal $x(t)$ defined as:

$$x(t) = \sin(2\pi f t), \quad t \in [0, T]$$

where f is the frequency of the signal.
To sample this signal, we take measurements at regular intervals, denoted as T_s, which is the sampling period. The sampling points can be represented as:

$$t_n = nT_s, \quad n = 0, 1, 2, \ldots, N$$

where N is the total number of samples taken within the duration T.

Vector representation of samples

The sampled values of the signal can be represented as a vector **x**:

$$\mathbf{x} = \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N] \end{bmatrix}$$

Each element $x[n]$ in the vector represents the amplitude of the signal at the corresponding sampling point t_n.
This vector representation allows us to use linear algebra techniques to analyze and manipulate the signal. For example, we can perform operations such as scaling, addition, and transformations on the vector.
Example of Sampling a Signal:
Assuming $T = 1$ second and $f = 1$ Hz, with a sampling period $T_s = 0.1$ seconds, we calculate the discrete samples as follows:

- For $n = 0$:
$$x[0] = \sin(2\pi \cdot 1 \cdot 0 \cdot 0.1) = \sin(0) = 0$$
- For $n = 1$:
$$x[1] = \sin(2\pi \cdot 1 \cdot 1 \cdot 0.1) = \sin(0.2\pi) \approx 0.5878$$
- For $n = 2$:
$$x[2] = \sin(2\pi \cdot 1 \cdot 2 \cdot 0.1) = \sin(0.4\pi) \approx 0.9511$$
- For $n = 3$:
$$x[3] = \sin(2\pi \cdot 1 \cdot 3 \cdot 0.1) = \sin(0.6\pi) \approx 0.9511$$
- For $n = 4$:
$$x[4] = \sin(2\pi \cdot 1 \cdot 4 \cdot 0.1) = \sin(0.8\pi) \approx 0.5878$$
- For $n = 5$:
$$x[5] = \sin(2\pi \cdot 1 \cdot 5 \cdot 0.1) = \sin(1.0\pi) = 0$$

This results in the following discrete signal samples:

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0.5878 \\ 0.9511 \\ 0.9511 \\ 0.5878 \\ 0 \end{bmatrix}$$

Vector interpretation of sampling

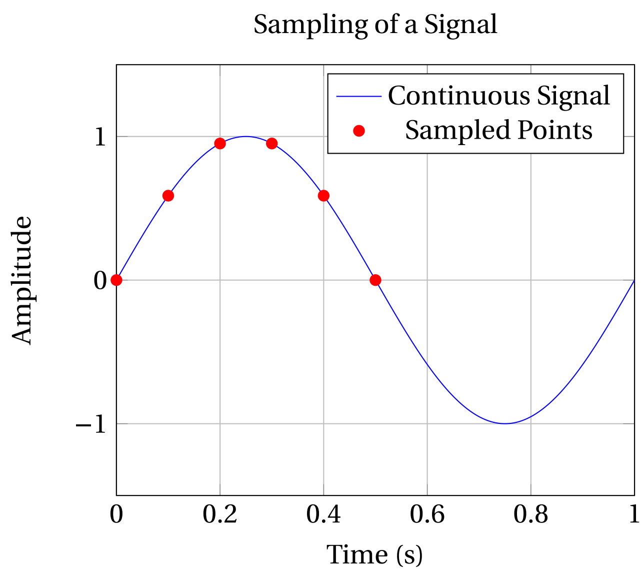
The vector $\mathbf{x}$ can be interpreted geometrically in a multi-dimensional space. Each sample can be viewed as a point in a vector space, where the dimension of the space corresponds to the number of samples taken. For instance, the sampled signal can be represented as a vector in a two-dimensional space (2D) if we consider two consecutive samples at any time instance:

$$\mathbf{x}_{2D} = \begin{bmatrix} x[n] \\ x[n+1] \end{bmatrix}$$

This geometric interpretation facilitates various analyses, such as distance measurements between points in this space, providing insights into the signal's behavior over time.

Visualizing the sampling process

The following plot illustrates the continuous signal $x(t)$ and its sampled points:



Applications of sampling in signal processing

1. *Audio Processing*: Digital audio signals are sampled to create a representation that can be stored and processed by computers. For instance, CDs use a standard sampling rate of 44.1 kHz.
2. *Image Processing*: In images, sampling corresponds to capturing pixel values. Higher sampling rates lead to higher resolution images.
3. *Telecommunications*: Signals are sampled to transmit data over various communication channels, ensuring accurate information conveyance.
4. *Vector Analysis*: By representing signals as vectors, various linear algebra techniques can be applied for filtering, noise reduction, and data compression, enhancing the quality and efficiency of signal processing tasks.

13.6 Solving Linear Equations: A Core Focus of Linear Algebra

In this section focuses on one of the central applications of linear algebra: **solving systems of linear equations**. It outlines key concepts and techniques for finding solutions to equations of the form $Ax = b$, where A is a square $n \times n$ matrix, x is a vector of unknowns, and b is a known vector.

Key Concept

A system of linear equations $Ax = b$ represents one of the most important applications in linear algebra, with A being a matrix of coefficients, x the vector of unknowns, and b a known vector.

Inverse Matrices: A direct approach

The most straightforward way to solve $Ax = b$ is to find the **inverse matrix** of A, denoted as A^{-1}. If A^{-1} exists, then the solution x is given by:

$$x = A^{-1}b \tag{13.1}$$

However, finding the inverse matrix can be computationally expensive, especially for large matrices. Furthermore, not all matrices have inverses. A matrix is **invertible** if and only if its determinant is non-zero.

Key Concept

Matrix inverses provide a direct solution to $Ax = b$ but are often computationally inefficient for large matrices. A matrix is invertible only if its determinant is non-zero.

Key Properties of Inverse Matrices

- **Uniqueness:** If an inverse matrix exists, it is unique.
- **Relationship to Identity Matrix:** The product of a matrix and its inverse is the identity matrix ($A^{-1}A = I$ and $AA^{-1} = I$).
- **Invertibility and Linear Independence:** A matrix is invertible if and only if its rows (and columns) are linearly independent.

Triangular matrices and back substitution

Solving systems of linear equations becomes significantly easier when the matrix A is **triangular**. A triangular matrix has all its entries either above or below the main diagonal equal to zero. There are two types of triangular matrices:

- **Upper Triangular:** All entries below the main diagonal are zero.
- **Lower Triangular:** All entries above the main diagonal are zero.

Key Concept

Triangular matrices simplify solving linear systems. Upper triangular matrices allow for the use of back substitution, a more efficient method compared to finding matrix inverses.

Example: Solving via back substitution

Consider an upper triangular system $Ux = c$:

$$\begin{bmatrix} 2 & 3 & 4 \\ 0 & 5 & 6 \\ 0 & 0 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 19 \\ 17 \\ 14 \end{bmatrix}$$

We start by solving for x_3, then substitute this value into the second equation to find x_2, and finally into the first equation to find x_1. The solution is:

$$x_3 = 2, \quad x_2 = 1, \quad x_1 = 4$$

13.6.1 Elimination: Transforming Matrices to Triangular Form

Elimination is a technique used to transform a general square matrix into an upper triangular matrix, making it easier to solve by back substitution. This process involves a sequence of row operations, aiming to introduce zeros below the diagonal.

Example: Row Elimination

Consider the matrix:

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 11 & 14 \\ 2 & 8 & 17 \end{bmatrix}$$

After performing elimination steps, the matrix is transformed into:

$$U = \begin{bmatrix} 2 & 3 & 4 \\ 0 & 5 & 6 \\ 0 & 0 & 7 \end{bmatrix}$$

This transformation allows for efficient solving using **LU factorization**, where:

$$A = LU$$

Key Concept

LU factorization splits a matrix into a lower triangular matrix L and an upper triangular matrix U, facilitating efficient solutions for systems of equations with repeated matrices but different right-hand sides.

Row Exchanges: handling zero pivots

During elimination, if a diagonal element (pivot) is zero, we perform **row exchanges**. Row exchanges are represented by **permutation matrices**. The general solution for $Ax = b$ involves:

1. Find P such that $PA = LU$.
2. Solve $Ly = Pb$ using forward substitution.
3. Solve $Ux = y$ using back substitution.

Key Concept

Row exchanges prevent breakdown in elimination processes when zero pivots are encountered, enabling the use of LU factorization with a permutation matrix.

Transposes and symmetric matrices

The **transpose** of a matrix A is denoted as A^T. A matrix is **symmetric** if its transpose equals the original matrix ($S^T = S$).

Properties of symmetric matrices

- **Real Eigenvalues:** The eigenvalues of a symmetric matrix are real.
- **Orthogonal Eigenvectors:** Eigenvectors corresponding to distinct eigenvalues are orthogonal.

Key Concept

Symmetric matrices arise in optimization and statistics, with their real eigenvalues and orthogonal eigenvectors being particularly important for practical applications.

Efficiency Considerations: Beyond Inverse Calculation

While calculating A^{-1} provides a direct solution, it is often inefficient. For large systems, techniques like **LU decomposition** with forward and backward substitution are preferred, offering computational efficiency.

Key Concept

Efficient techniques like LU decomposition are preferred over directly finding the inverse, particularly for large systems of equations.

13.7 Vector Spaces: Embracing Linear Combinations

A **vector space** is a collection of objects (called vectors) that can be combined using the operations of vector addition and scalar multiplication, subject to certain rules.

Axioms of Vector Spaces

A vector space is defined by the following **ten axioms**, which specify the behavior of vector addition and scalar multiplication:

1. **Associativity of addition**: $(\mathbf{u}+\mathbf{v})+\mathbf{w}=\mathbf{u}+(\mathbf{v}+\mathbf{w})$
2. **Commutativity of addition**: $\mathbf{u}+\mathbf{v}=\mathbf{v}+\mathbf{u}$
3. **Additive identity**: There exists a zero vector $\mathbf{0}$ such that $\mathbf{u}+\mathbf{0}=\mathbf{u}$ for all vectors $\mathbf{u}$.
4. **Additive inverse**: For every vector $\mathbf{u}$, there exists a vector $-\mathbf{u}$ such that $\mathbf{u}+(-\mathbf{u})=\mathbf{0}$.
5. **Distributivity of scalar multiplication over vector addition**: $c(\mathbf{u}+\mathbf{v})=c\mathbf{u}+c\mathbf{v}$ for all scalars c and vectors $\mathbf{u},\mathbf{v}$.
6. **Distributivity of scalar multiplication over scalar addition**: $(c+d)\mathbf{u}=c\mathbf{u}+d\mathbf{u}$ for all scalars c,d and vector $\mathbf{u}$.
7. **Associativity of scalar multiplication**: $c(d\mathbf{u})=(cd)\mathbf{u}$ for all scalars c,d and vector $\mathbf{u}$.
8. **Multiplicative identity**: $1\mathbf{u}=\mathbf{u}$ for every vector $\mathbf{u}$.
9. **Closure under addition**: For any two vectors $\mathbf{u}$ and $\mathbf{v}$, their sum $\mathbf{u}+\mathbf{v}$ is also a vector in the space.
10. **Closure under scalar multiplication**: For any scalar c and vector $\mathbf{u}$, the product $c\mathbf{u}$ is also a vector in the space.

Key Concept

These axioms provide the foundation for the behavior of vector addition and scalar multiplication in a vector space.

While vectors can be represented as column matrices, vector spaces can encompass a broader range of objects, including matrices and functions.

Key Concept

The key property of a vector space is **closure under linear combinations**: Any linear combination of vectors within the space must also be a vector within the space. This ensures that the space is complete with respect to the operations allowed.

Some common examples of vector spaces include:

- **Euclidean spaces** such as $\mathbb{R}^n$, where vectors are represented as ordered lists of n real numbers.

- **Matrix spaces**, such as the space of all 3×3 matrices.
- **Function spaces**, which consist of functions satisfying specific properties, e.g., the space of all continuous functions on a given interval.

Subspaces: smaller spaces within larger ones

A **subspace** of a vector space is a subset of vectors that forms a vector space under the same operations. A subspace must include the zero vector and be closed under linear combinations.

Key Concept

A subspace of a vector space retains the same operations of addition and scalar multiplication, and it must contain the zero vector.

Examples of subspaces include:

- **Planes and lines** through the origin in $\mathbb{R}^3$.
- The set containing only the **zero vector**.
- The space of **upper triangular matrices**, a subspace of all matrices of the same size.

13.7.1 Four fundamental subspaces: A matrix perspective

Every matrix A is associated with four fundamental subspaces, each playing a key role in understanding linear transformations:

- **Column space** $C(A)$: Contains all possible linear combinations of the columns of A, representing the outputs of the transformation Ax.
- **Row space** $C(A^T)$: The column space of A^T, containing all possible linear combinations of the rows of A.
- **Nullspace** $N(A)$: The set of all vectors x that satisfy $Ax = 0$.
- **Left nullspace** $N(A^T)$: The nullspace of the transpose of A.

Key Concept

An important orthogonality relationship exists: The row space is orthogonal to the nullspace, and the column space is orthogonal to the left nullspace.

Basis and dimension: Quantifying the "size" of a space

A **basis** for a vector space is a set of linearly independent vectors that span the entire space. Linear independence means that no vector in the basis can be expressed as a linear combination of the others, while spanning implies that every vector in the space is a linear combination of the basis vectors.
The **dimension** of a vector space is the number of vectors in any basis for the space.

Key Concept

All bases of a given vector space have the same number of vectors, which defines the dimension of the space.

Examples of basis and dimension include:

- The **standard basis** for $\mathbb{R}^3$: $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$. The dimension of $\mathbb{R}^3$ is 3.
- The column space of a matrix, where different bases can exist but have the same number of vectors, corresponding to the **rank** of the matrix.

Elimination and bases: A computational perspective

The process of **elimination** can be used to find bases for the row space, column space, and null space of a matrix. Transforming a matrix into its **reduced row echelon form (RREF)** allows for the identification of pivot columns, which form a basis for the column space. The rows in the RREF form a basis for the row space, and the special solutions to $Rx = 0$ (where R is the RREF) form a basis for the nullspace.

Key Concept

The elimination process and RREF provide computational tools for identifying bases of the fundamental subspaces associated with a matrix.

13.8 Beyond bases: A glimpse into the future

This section lays the groundwork for more advanced topics in linear algebra:

- **Linear Transformations**: Functions between vector spaces that preserve linear combinations.
- **Orthogonality and Projections**: Orthogonality is crucial for finding the closest vector in a subspace to a given vector, leading to the idea of projections onto subspaces.
- **Eigenvalues and Eigenvectors**: Special vectors mapped to scalar multiples of themselves by a linear transformation, essential for understanding dynamical systems.

Summary

These advanced concepts build on the foundation laid by vector spaces, subspaces, and their associated bases and dimensions.

13.9 Linear Transformations: The Essence of Linearity

Linear transformations are central to linear algebra. They map vectors between vector spaces while preserving the operations of vector addition and scalar multiplication. In this section, we introduce the concept of **linear transformations** as an extension of matrix operations to more abstract and general contexts.

Mapping vectors while preserving linearity

A **linear transformation**, denoted by T, is a mapping between two vector spaces, V (input space) and Y (output space), that adheres to the principle of linearity:

$$T(c\mathbf{v} + d\mathbf{w}) = cT(\mathbf{v}) + dT(\mathbf{w})$$

This equation captures the essence of linearity:

- **Additivity**: Transforming the sum of two vectors is the same as transforming each vector individually and then adding the results.
- **Homogeneity**: Scaling a vector before transforming is the same as transforming the vector and then scaling it.

Key Concepts

- **Linear Transformation**: A mapping between vector spaces that preserves vector addition and scalar multiplication.
- **Additivity and Homogeneity**: Properties that define linearity.

Example: scaling transformation

Consider the transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$, which scales vectors by a factor of 2. For any vector $\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$, the transformation is:

$$T\left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}\right) = 2\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 2v_1 \\ 2v_2 \end{pmatrix}$$

This transformation satisfies the linearity properties, as:

$$T(c\mathbf{v}) = cT(\mathbf{v}) \quad \text{and} \quad T(\mathbf{v}+\mathbf{w}) = T(\mathbf{v}) + T(\mathbf{w})$$

13.10 Matrices as Representations: A Basis-Dependent Encoding

Linear transformations can be represented by matrices, though this representation depends on the choice of bases for the vector spaces involved. To construct the matrix A that represents a linear transformation T:

1. **Choose bases**: Select a basis $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ for the input space V and a basis $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m\}$ for the output space Y.
2. **Transform basis vectors**: Apply T to each basis vector $\mathbf{v}_j$.
3. **Express transformed vectors in the output basis**: Write $T(\mathbf{v}_j)$ as a linear combination of the output basis vectors:
$$T(\mathbf{v}_j) = a_{1j}\mathbf{y}_1 + a_{2j}\mathbf{y}_2 + \cdots + a_{mj}\mathbf{y}_m$$
4. **Form the matrix**: The coefficients a_{ij} form the matrix A, with the j-th column representing the transformed vector $T(\mathbf{v}_j)$.

The matrix A encodes the action of T on any input vector.

Key Concepts

- **Matrix Representation**: A linear transformation can be represented by a matrix depending on the choice of basis.
- **Matrix Construction**: The transformation matrix is constructed by applying the transformation to basis vectors and expressing the result in the output basis.

Example: Rotation matrix

Consider a rotation transformation T in $\mathbb{R}^2$ by an angle θ. The corresponding matrix A_θ is:

$$A_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

This matrix rotates any vector in $\mathbb{R}^2$ by θ.

13.10.1 Basis changes: Transforming the matrix representation

The matrix representation of a linear transformation depends on the choice of bases for both the input and output spaces. Changing these bases alters the matrix representation. Let M represent T with new bases $\{V_1, V_2, \dots, V_n\}$ in V and $\{Y_1, Y_2, \dots, Y_m\}$ in Y. The new matrix M is related to the old matrix A by:

$$M = Y^{-1}AV$$

Where:

- V is the change-of-basis matrix for the input space.
- Y is the change-of-basis matrix for the output space.

Key Concepts

- **Change of Basis**: Changing the basis in the input or output space transforms the matrix representation of the linear transformation.

Example: Scaling and shearing transformation

Consider the transformation T that scales by 2 in the x-direction and shears by 1 unit in the y-direction. The matrix in standard basis is:

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

A change in basis will transform this matrix accordingly using the relationship $M = Y^{-1}AV$.

13.11 Significance: Unifying Framework for Linearity

Linear transformations provide a unified framework for key concepts in linear algebra. They allow us to interpret matrix operations in terms of transformations.

- **Matrix multiplication as composition**: Multiplying matrices corresponds to composing the linear transformations they represent.
- **Invertible matrices**: A matrix is invertible if and only if the corresponding linear transformation is invertible.
- **Eigenvectors as invariant directions**: Eigenvectors represent directions that remain unchanged under a linear transformation.

Key Concepts

- **Matrix Multiplication**: Represents the composition of two linear transformations.
- **Eigenvectors and Eigenvalues**: Directions that remain unchanged (up to scaling) under a transformation.

Example: Eigenvectors of a scaling transformation

For the scaling transformation $T(\mathbf{v}) = 2\mathbf{v}$, any vector is an eigenvector with eigenvalue 2, since:

$$T(\mathbf{v}) = 2\mathbf{v} = \lambda\mathbf{v} \quad \text{where} \quad \lambda = 2$$

13.12 Applications of Linear Transformations

Linear transformations are used in numerous fields. Here are some key applications:

Computer graphics and animations

In computer graphics, linear transformations rotate, scale, and translate objects. For example, a rotation matrix is applied to the points defining an object to rotate it.

Differential equations

Linear transformations simplify systems of linear differential equations by diagonalizing the system's matrix, making it easier to solve.

Data analysis and machine learning

Linear transformations are the basis for techniques like Principal Component Analysis (PCA), which reduces dimensionality while preserving the most important features.

Control systems

Linear transformations model dynamic systems in control theory. They simplify complex systems and allow easier analysis and control.

13.13 Orthogonality: A powerful tool in linear algebra

This session on **orthogonality**, a fundamental concept that plays a crucial role in understanding the geometry of vector spaces and solving linear algebra problems. This section explores the orthogonality relationships between the four fundamental subspaces associated with a matrix and introduces the powerful techniques of **projections** and **orthogonal matrices**.

Orthogonality of vectors and subspaces: Perpendicularity and independence

Two vectors **x** and **y** are considered **orthogonal** if their dot product is zero:

$$\mathbf{x}^T\mathbf{y} = 0.$$

In geometric terms, this means the vectors are perpendicular to each other. The concept extends to **complex vectors**, where orthogonality is defined using the conjugate transpose:

$$\mathbf{u}^H\mathbf{v} = 0.$$

Key Concept

Orthogonality implies that two vectors are perpendicular, enhancing the understanding of their independence and relationships in vector spaces.

The idea of orthogonality extends to **subspaces**: Two subspaces V and W are orthogonal if every vector in V is orthogonal to every vector in W.

Key Concept

The orthogonality relationships among the four fundamental subspaces of a matrix **A** are:

- The **row space** of **A** and its **nullspace** are orthogonal.
- The **column space** of **A** and its **left nullspace** are orthogonal.

These orthogonality relationships have important consequences:

- They provide a geometric interpretation of the solutions to linear equations. For example, the fact that the row space and nullspace are orthogonal means that any solution to $\mathbf{Ax} = 0$ lies in a direction perpendicular to all the rows of $\mathbf{A}$.
- They lead to powerful techniques for solving least squares problems, where we seek to find the "best" solution to an inconsistent system of equations.

Projections: Finding the closest vector in a subspace

The concept of orthogonality leads naturally to the idea of **projections onto subspaces**. Given a vector $\mathbf{b}$ and a subspace S, the projection of $\mathbf{b}$ onto S is the vector $\mathbf{p}$ in S that is closest to $\mathbf{b}$. The error vector $\mathbf{e} = \mathbf{b} - \mathbf{p}$ is orthogonal to S.

Key Concept

The **projection** of a vector onto a subspace is the closest point in the subspace to the vector, minimizing the error.

The two important types of projections are:

1. **Projection onto a line:** If S is a line spanned by a vector $\mathbf{a}$, the projection of $\mathbf{b}$ onto S is given by:

$$\mathbf{p} = \frac{\mathbf{a}^T\mathbf{b}}{\mathbf{a}^T\mathbf{a}}\mathbf{a}$$

2. **Projection onto a subspace spanned by columns of A:** If S is the column space of a matrix $\mathbf{A}$, the projection of $\mathbf{b}$ onto S is given by:

$$\mathbf{p} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

   The matrix $\mathbf{P} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is called the **projection matrix** onto the column space of $\mathbf{A}$. It satisfies the properties $\mathbf{P}^2 = \mathbf{P}$ and $\mathbf{P}^T = \mathbf{P}$, reflecting that projecting a vector twice onto the same subspace doesn't change it.

Key Concept

The **projection matrix** is essential for solving least squares problems, providing the best approximation of a vector within a column space.

Least squares approximations: Handling inconsistent systems

Projections play a crucial role in solving **least squares problems**, which arise when we have an inconsistent system of equations $\mathbf{Ax} = \mathbf{b}$ (i.e., a system with no exact solution). In such cases, we seek to find the vector $\hat{\mathbf{x}}$ that minimizes the squared error $\|\mathbf{b} - \mathbf{Ax}\|^2$. This leads to the **normal equations**:

$$\mathbf{A}^T\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^T\mathbf{b}$$

Key Concept

The **normal equations** are derived from minimizing the squared error in least squares problems, leading to the best solution approximation.

The solution $\hat{\mathbf{x}}$ gives the coefficients of the projection of $\mathbf{b}$ onto the column space of $\mathbf{A}$. This projection, $\mathbf{p} = \mathbf{A}\hat{\mathbf{x}}$, represents the "best" approximation to $\mathbf{b}$ that we can achieve within the column space of $\mathbf{A}$.

Key Concept

Least squares approximations are commonly applied in regression analysis to determine the best-fitting line for data points.

Orthogonal matrices: Preserving lengths and angles

An **orthogonal matrix** is a square matrix $\mathbf{Q}$ whose columns are orthonormal. This means that the columns are orthogonal to each other and have unit length. As a result, orthogonal matrices satisfy the property $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, which implies $\mathbf{Q}^T = \mathbf{Q}^{-1}$.

Key Concept

Orthogonal matrices preserve lengths and angles during transformations, making them invaluable in numerical computations.

Orthogonal matrices have several important properties:

- They preserve lengths: $\|\mathbf{Qx}\| = \|\mathbf{x}\|$ for any vector $\mathbf{x}$.
- They preserve angles: $(\mathbf{Qx})^T(\mathbf{Qy}) = \mathbf{x}^T\mathbf{y}$ for any vectors $\mathbf{x}$ and $\mathbf{y}$.
- They are computationally efficient to invert, as their inverse is simply their transpose.

Gram-Schmidt process: Constructing orthogonal bases

The **Gram-Schmidt process** provides a systematic way to construct an orthogonal basis for a subspace. Given a set of linearly independent vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$, the process generates a set of orthogonal vectors $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \ldots$ through the following steps:
1. Set $\mathbf{q}_1 = \mathbf{a}$.
2. For each subsequent vector $\mathbf{a}_k$:

$$\mathbf{q}_k = \mathbf{a}_k - \sum_{j=1}^{k-1} \frac{\mathbf{q}_j^T\mathbf{a}_k}{\mathbf{q}_j^T\mathbf{q}_j}\mathbf{q}_j.$$

This process ensures that the resulting vectors are mutually orthogonal.

Key Concept

The **Gram-Schmidt process** is a fundamental algorithm for generating orthogonal bases in vector spaces, facilitating many applications in linear algebra.

Conclusion

Orthogonality is a powerful tool in linear algebra, facilitating efficient problem-solving and offering a deeper geometric understanding of vector spaces. Using the key algebraic tools like projections, orthogonal matrices, and orthogonal bases, mathematicians and engineers can tackle complex problems across various fields, including data analysis, signal processing, and machine learning.

13.14 Determinants: A key value for square matrices

Section 5 of the "ZoomNotes for Linear Algebra" explores the concept of the **determinant**, a single number associated with a square matrix that encapsulates crucial information about its properties. The sources explain how to calculate determinants for matrices of different sizes, discuss key properties and applications, and highlight the connection between determinants and matrix invertibility, solving linear systems, and calculating volumes.

Defining properties and calculation methods: From permutations to cofactors

The determinant of a square matrix A, denoted as $\det A$, can be defined through three fundamental properties:

Keynotes

1. **Row exchange**: Exchanging two rows of A changes the sign of $\det A$. 2. **Linearity in each row**: $\det A$ is a linear function of each row of A, holding all other rows fixed. 3. **Identity matrix**: $\det I = 1$, where I is the identity matrix.

These properties lead to various ways of calculating determinants. For a 2×2 matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

the determinant is given by:

$$\det A = ad - bc$$

For a 3×3 matrix, the sources present the **"Big Formula"** which involves summing over all $3! = 6$ possible permutations of the columns:

$$\det A = a_{11}a_{22}a_{33} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} - a_{13}a_{22}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32}$$

This formula highlights that each term in the determinant takes one element from each row and each column, with a sign determined by the permutation of the columns.
Another approach is the **cofactor expansion**, which expresses the determinant in terms of smaller determinants called **cofactors**. A cofactor C_{ij} is obtained by removing row i and column j from A and multiplying the determinant of the remaining $(n-1) \times (n-1)$ matrix by $(-1)^{i+j}$. For example, the cofactor expansion along row 1 of a 3×3 matrix A is:

$$\det A = a_{11}C_{11} + a_{12}C_{12} + a_{13}C_{13}$$

13.15 Key Properties and Applications: Invertibility, Cramer's Rule, and Volumes

Determinants possess several important properties that make them valuable tools in linear algebra:

Keynotes

* **Invertibility**: A square matrix A is invertible if and only if $\det A \neq 0$. This property stems from the fact that the determinant is related to the volume of the parallelepiped formed by the column vectors of A. If the determinant is zero, the volume collapses to zero, indicating that the columns are linearly dependent, and hence A is not invertible.

Determinant of product: $\det(AB) = (\det A)(\det B)$ for any two square matrices A and B of the same size.

Determinant of transpose: $\det A = \det A^T$.

Determinant of inverse: $\det(A^{-1}) = \frac{1}{\det A}$ if A is invertible.

These properties lead to important applications:

Cramer's Rule: This rule provides a formula for solving a system of linear equations $Ax = \mathbf{b}$ when A is invertible. The solution for each component x_i is given by the ratio of two determinants:

$$x_i = \frac{\det B_i}{\det A}$$

where B_i is the matrix obtained by replacing the i-th column of A with the vector $\mathbf{b}$. However, the sources note that Cramer's Rule is not computationally efficient for large systems.
Volume calculation: The absolute value of the determinant of a matrix E whose rows (or columns) represent the edges of a parallelepiped in n-dimensional space gives the volume of that parallelepiped. This geometric interpretation connects determinants to the concept of linear transformations and their effect on volumes.

Special Cases: Orthogonal, triangular, and lu factorization

The sources highlight specific cases of matrices where determinant calculations simplify:

Keynotes

Orthogonal matrices: For an orthogonal matrix Q, $\det Q = \pm 1$ because $Q^T Q = I$ implies $(\det Q)^2 = 1$.
Triangular matrices: The determinant of a triangular matrix (upper or lower) is simply the product of its diagonal entries.
LU factorization: If a matrix A can be factored as $A = LU$, where L is lower triangular and U is upper triangular, then $\det A = (\det L)(\det U) =$ product of the pivots in U. This property arises from the fact that elimination operations used to obtain the LU factorization do not change the determinant (except for possible sign changes due to row exchanges).

Overall, session 5 of the "Computational Linear Algebra" establishes determinants as fundamental quantities associated with square matrices. Their connection to invertibility, linear systems, and geometric concepts like volumes makes them essential tools for understanding and solving various linear algebra problems. Furthermore, the properties of determinants, especially in special cases like orthogonal and triangular matrices, offer valuable insights and computational advantages in numerous applications.

13.16 Eigenvalues and eigenvectors: Unlocking the essence of linear transformations

Eigenvalues and eigenvectors provide crucial insights into the behavior of linear transformations represented by square matrices. This section explores their definitions, methods for finding them, applications in matrix diagonalization, special cases such as symmetric matrices, and their use in solving differential equations.

Keynotes

- **Eigenvalue equation**: $A\mathbf{x} = \lambda\mathbf{x}$
- **Diagonalization**: Simplifying matrix operations through eigenvectors
- **Symmetric matrices**: Special properties for real eigenvalues and orthogonal eigenvectors
- **Applications**: Solving systems of linear differential equations and analyzing matrix structures in engineering

Definition and calculation: The eigenvalue equation

Eigenvalues and eigenvectors are defined by the fundamental *eigenvalue equation*:

$$A\mathbf{x} = \lambda\mathbf{x}$$

where: - A is a square $n \times n$ matrix, - $\mathbf{x}$ is a non-zero vector (the *eigenvector*), - λ is a scalar (the *eigenvalue*).
This equation states that applying the matrix A to an eigenvector results in scaling the vector by the eigenvalue λ. Geometrically, the eigenvector does not change its direction, only its magnitude.
To calculate eigenvalues, we rearrange the equation as:

$$(A - \lambda I)\mathbf{x} = 0$$

For non-trivial solutions, $(A - \lambda I)$ must be singular, meaning:

$$\det(A - \lambda I) = 0$$

This is called the *characteristic equation*. Solving it yields the eigenvalues, and substituting these values back into the equation gives the corresponding eigenvectors.

Example 1: Finding eigenvalues and eigenvectors

Consider the matrix:

$$A = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$$

Step 1: Solve the characteristic equation:

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0$$

$$\det \begin{bmatrix} 0.8 - \lambda & 0.3 \\ 0.2 & 0.7 - \lambda \end{bmatrix} = (0.8 - \lambda)(0.7 - \lambda) - (0.3)(0.2) = 0$$

$$(0.8 - \lambda)(0.7 - \lambda) - 0.06 = 0$$

Expanding and solving the quadratic equation:

$$\lambda^2 - 1.5\lambda + 0.5 = 0$$

$$\lambda_1 = 1, \quad \lambda_2 = 0.5$$

Step 2: Find the corresponding eigenvectors by solving $(A - \lambda I)\mathbf{x} = 0$ for each eigenvalue.
For $\lambda_1 = 1$:

$$(A - I)\mathbf{x} = \begin{bmatrix} -0.2 & 0.3 \\ 0.2 & -0.3 \end{bmatrix}\mathbf{x} = 0$$

Solving this system gives the eigenvector $\mathbf{x}_1 = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$.
For $\lambda_2 = 0.5$:

$$(A - 0.5I)\mathbf{x} = \begin{bmatrix} 0.3 & 0.3 \\ 0.2 & 0.2 \end{bmatrix}\mathbf{x} = 0$$

The eigenvector is $\mathbf{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

Diagonalization: Simplifying matrix powers

Eigenvalues and eigenvectors enable *diagonalization*, a powerful method for simplifying matrix operations, especially powers of matrices. The matrix A can be expressed as:

$$A = X\Lambda X^{-1}$$

where X is the matrix of eigenvectors, and Λ is a diagonal matrix of eigenvalues. This transformation allows for easy computation of matrix powers:

$$A^k = X\Lambda^k X^{-1}$$

Diagonalizing A reduces the problem of computing A^k to simple multiplications with diagonal elements.

Example 2: Diagonalizing a matrix

Given:

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

Find A^4 using diagonalization.
Step 1: Find eigenvalues by solving $\det(A - \lambda I) = 0$:

$$\det \begin{bmatrix} 1-\lambda & 2 \\ 0 & 3-\lambda \end{bmatrix} = (1-\lambda)(3-\lambda) = 0$$

$$\lambda_1 = 1, \quad \lambda_2 = 3$$

Step 2: Find eigenvectors by solving $(A - \lambda I)\mathbf{x} = 0$.
For $\lambda_1 = 1$: $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.
For $\lambda_2 = 3$: $\mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.
Step 3: Form X and Λ:

$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \Lambda = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$$

Step 4: Compute A^4 using $A^4 = X\Lambda^4 X^{-1}$:

$$\Lambda^4 = \begin{bmatrix} 1 & 0 \\ 0 & 81 \end{bmatrix}$$

Thus, A^4 can be computed efficiently.

Special Cases: Symmetric matrices and positive definiteness

Symmetric matrices ($A = A^T$) have special properties: - All eigenvalues are real. - Eigenvectors corresponding to distinct eigenvalues are orthogonal.
This leads to the *spectral theorem*, which states that symmetric matrices can be diagonalized by an orthogonal matrix:

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

where Q is an orthogonal matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues.

Keynotes

- **Symmetric matrix**: $A = A^T$
- **Real eigenvalues**: Eigenvalues of symmetric matrices are always real.
- **Orthogonal eigenvectors**: Eigenvectors corresponding to distinct eigenvalues are orthogonal.

Example 3: Symmetric matrix diagonalization

Consider the symmetric matrix:

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

Find the eigenvalues and eigenvectors.
Step 1: Solve the characteristic equation:

$$\det(A - \lambda I) = 0$$

$$\det \begin{bmatrix} 4-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix} = (4-\lambda)(3-\lambda) - 1 = 0$$

Expanding the quadratic:

$$\lambda^2 - 7\lambda + 11 = 0$$

The eigenvalues are $\lambda_1 = 5, \quad \lambda_2 = 2$.
Step 2: Find the eigenvectors.
For $\lambda_1 = 5$: $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.
For $\lambda_2 = 2$: $\mathbf{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

Applications: Solving systems of differential equations

Eigenvalues and eigenvectors are particularly useful in solving systems of linear differential equations. Consider the system:

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}$$

where $\mathbf{u}(t)$ is a vector-valued function, and A is a matrix. The general solution involves finding the eigenvalues and eigenvectors of A. The system can be rewritten as:

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}$$

Solutions are of the form:

$$\mathbf{u}(t) = c_1 e^{\lambda_1 t}\mathbf{x}_1 + c_2 e^{\lambda_2 t}\mathbf{x}_2 + \cdots$$

Example 4: Solving a system of differential equations

Consider the system:

$$\frac{d\mathbf{u}}{dt} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}\mathbf{u}$$

Step 1: Find eigenvalues and eigenvectors.
Eigenvalues: $\lambda_1 = 3, \lambda_2 = -1$.
Eigenvectors: $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.
Step 2: General solution:

$$\mathbf{u}(t) = c_1 e^{3t} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + c_2 e^{-t} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

This completes the solution of the system.

13.17 Singular Value Decomposition (SVD): A powerful tool for matrix analysis

This section of the "Computational Linear Algebra" introduces the **Singular Value Decomposition (SVD)**, a fundamental matrix factorization that decomposes any matrix, even rectangular and non-invertible ones, into a product of three matrices with special properties: two orthogonal matrices and a diagonal matrix. This decomposition has wide-ranging applications in data analysis, image processing, recommendation systems, and other fields.

13.17.1 Definition and construction: orthogonal matrices and singular values

The SVD of an $m \times n$ matrix A is given by:

$$A = U\Sigma V^T \tag{13.2}$$

where:

- U is an $m \times m$ orthogonal matrix, meaning its columns are orthonormal (orthogonal and of unit length).
- Σ is an $m \times n$ diagonal matrix containing the **singular values** of A, denoted as $\sigma_1, \sigma_2, \ldots, \sigma_r$, where r is the rank of A. The singular values are non-negative and conventionally arranged in descending order ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$).
- V is an $n \times n$ orthogonal matrix.

The columns of U are called the **left singular vectors** of A, and the columns of V are called the **right singular vectors** of A.

Keynotes

- **Orthogonal Matrix**: A matrix is orthogonal if its columns are mutually orthogonal unit vectors.
- **Singular Values**: These are the diagonal elements of Σ, and they provide important information about the scaling properties of the matrix.
- **Decomposition**: SVD decomposes a matrix into rotations and scalings.

13.17.2 Calculation of U and V^T

1. **Right Singular Vectors and V**: The **right singular vectors** of A, which form the columns of V, are the eigenvectors of A^TA. These eigenvectors are obtained by solving the eigenvalue problem:

$$A^TAv_i = \lambda_i v_i \tag{13.3}$$

   where λ_i are the eigenvalues of A^TA. The eigenvectors corresponding to these eigenvalues form the matrix V. The right singular vectors are the unit eigenvectors of A^TA.

2. **Singular Values and Σ**: The singular values σ_i are the square roots of the eigenvalues λ_i of A^TA. These singular values form the diagonal entries of the matrix Σ.

3. **Left Singular Vectors and U**: The **left singular vectors** of A, which form the columns of U, are obtained by multiplying A by each right singular vector and normalizing:

$$u_i = \frac{Av_i}{\sigma_i} \tag{13.4}$$

The resulting vectors are the left singular vectors, which are the columns of U. If A is an $m \times n$ matrix, then U will be an $m \times m$ matrix whose columns are orthonormal vectors.

Keynotes

- **Right Singular Vectors** (V): The eigenvectors of $A^T A$ give the right singular vectors.
- **Singular Values**: These are the square roots of the eigenvalues of $A^T A$ and form the diagonal elements of Σ.
- **Left Singular Vectors** (U): These are calculated as $u_i = \frac{Av_i}{\sigma_i}$, where v_i are the right singular vectors and σ_i are the singular values.

13.17.3 Sample Problem 1: Compute the SVD of a matrix

Consider the matrix:

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

Step 1: Calculate $A^T A$

$$A^T A = \begin{bmatrix} 3 & 4 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}$$

Step 2: Calculate the eigenvalues of $A^T A$

The eigenvalues λ are the solutions to the characteristic equation $\det(A^T A - \lambda I) = 0$:

$$\det\left(\begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0$$

Solving this gives $\lambda_1 = 45$ and $\lambda_2 = 5$.

Step 3: Calculate the singular values

The singular values are the square roots of the eigenvalues of $A^T A$:

$$\sigma_1 = \sqrt{45} \approx 6.708, \quad \sigma_2 = \sqrt{5} \approx 2.236$$

Step 4: Calculate the right singular vectors V

Next, calculate the eigenvectors of $A^T A$ to obtain the right singular vectors:
For $\lambda_1 = 45$:

$$\left(A^T A - 45I\right) v = 0 \quad \Rightarrow \quad \begin{bmatrix} -20 & 20 \\ 20 & -20 \end{bmatrix} v = 0$$

Solving this gives the right singular vector $v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$.

For $\lambda_2 = 5$:

$$\left(A^T A - 5I\right) v = 0 \quad \Rightarrow \quad \begin{bmatrix} 20 & 20 \\ 20 & 20 \end{bmatrix} v = 0$$

Solving this gives $v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$.

Thus, $V = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$.

Step 5: Calculate the left singular vectors U

Using $U = AV\Sigma^{-1}$, we can calculate the left singular vectors. First, compute Av_1 and Av_2, then normalize to get u_1 and u_2.
`Matlab` code for the task is given below.

```
% Define the matrix A
A = [3 0; 4 5];

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

% Display the results
disp('U = ');
disp(U);
disp('S = ');
disp(S);
disp('V = ');
disp(V);
```

Output of the code is given below.

```
U =
   -0.5145   -0.8575
   -0.8575    0.5145

S =
    6.7082         0
         0    2.2361

V =
   -0.7071   -0.7071
   -0.7071    0.7071
```

RESULTS

A concise course material for the course *Introduction to Computational Linear Algebra* is prepared.