



# SCHOOL OF ARTIFICIAL INTELLIGENCE

---

## 24MA602 Computational Mathematics for Data Science

---

### Assignment Set-5

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

<b>1 Assignment 49</b>	
<b>Construction of Vectors in Row Space and Column Space of a Matrix</b>	<b>11</b>
1.1 Matrix Invertibility and Pseudo Inverse . . . . .	11
1.1.1 General representation of vectors in a vector space . . . . .	12
1.1.2 Role of pseudo inverse in solution of system of equations . . . . .	12
1.2 Tasks . . . . .	13
<b>2 Assignment 50</b>	
<b>Vector Space</b>	<b>15</b>
2.1 Introduction . . . . .	15
<b>3 Assignment 51</b>	
<b>Reflection Matrices Using Projections</b>	<b>21</b>
3.0.1 3D case- Reflection across Plane passing through origin . . . . .	22
3.0.2 Assignments . . . . .	23
<b>4 Assignment 52</b>	
<b>Mathematical Model for Support Vector Machine (SVM)</b>	<b>29</b>
<b>5 Assignment 53</b>	
<b>Formulation and Solution of Hard Margin SVM</b>	<b>37</b>
5.1 Introduction . . . . .	37
5.2 Primal and Dual Problem of Hard Margin SVM . . . . .	37
5.2.1 Primal Problem . . . . .	37
5.2.2 Matrix Form of the Primal Problem . . . . .	38
5.2.3 Dual Problem . . . . .	38
5.2.4 Matrix Form of the Dual Problem . . . . .	39
5.3 Tasks . . . . .	39
5.3.1 Formation of QPP for the dual . . . . .	41
<b>6 Assignment 55</b>	
<b>An Interesting Aspect of Lagrangian Function in Terms of Hessian Matrix at the Optimal Point</b>	<b>45</b>
6.1 Introduction . . . . .	45
6.1.1 Optimization Problem Setup . . . . .	45
6.1.2 Lagrange Function . . . . .	45
6.1.3 First-Order Optimality Conditions (Karush-Kuhn-Tucker Conditions) . . . . .	46
6.1.4 Second-Order Optimality Conditions . . . . .	46
6.1.5 Hessian Matrix with Inequality Constraints . . . . .	46
6.1.6 Example: Quadratic Minimization with Inequality Constraints . . . . .	47
6.2 Takeaway . . . . .	47
6.3 Task . . . . .	47

---

<b>7 Using CVX for Solving LP Problems</b>	<b>55</b>
7.1 Introduction	55
7.2 Historical Background of Linear Programming	55
7.3 Overview of CVX in MATLAB	55
7.3.1 Key Features of CVX	55
7.4 Example of Using CVX for LP Problems	56
7.4.1 Problem Statement	56
7.4.2 MATLAB Code Using CVX	56
<b>8 Assignment 56</b>	
<b>Loss and Penalty Function</b>	<b>63</b>
8.1 Introduction	63
8.1.1 Role of loss function in quality control- a real story	63
8.1.2 Loss function	64
8.1.3 Penalized function	64
8.1.4 Quadratic loss function	64
8.1.5 Task 1	64
8.1.6 Non-negative least square problem	66
8.1.7 Task 2	66
8.1.8 LASSO regression	67
8.1.9 Task 3	67
<b>9 Assignment 58-1</b>	
<b>Logistic Regression</b>	<b>73</b>
9.1 From Linear Regression to Logistic Regression	73
9.1.1 Recap- Linear regression	73
9.1.2 The Motivation for Logistic Regression	73
9.1.3 Introducing the Sigmoid Function	75
9.1.4 Interpreting Logistic Regression Output as Probability	75
9.1.5 Log-Odds and the Decision Boundary	75
9.1.6 Defining the Logistic Loss Function	75
9.1.7 Minimizing the Negative Log-Likelihood (Logistic Loss)	76
9.1.8 Adding Regularization	76
9.1.9 What does optimization tries to do?	76
9.2 Computational Part of Logistic Regression	77
9.2.1 Creating first logistic regression on the dummy dataset	77
9.2.2 Breast Cancer Prediction	78
9.2.3 Logistic regression model	79
<b>10 Assignment 59</b>	
<b><math>\epsilon</math> – Insensitive Loss Functions</b>	<b>81</b>
10.1 Introduction	81
10.1.1 Formulating the Optimization Problem	81
10.1.2 Optimization Objective	82
10.1.3 Constraints	82
10.2 Example	82
10.3 Task	85
<b>11 Assignment 60</b>	
<b>SVM Classifier With Soft Margin</b>	<b>89</b>
11.1 Introduction	89
11.1.1 Hard margin SVM	89
11.1.2 Soft margin SVM	89
11.1.3 Solution using Mat lab	90

11.1.4 Computational example . . . . .	91
11.2 Optimization in Soft Margin SVM . . . . .	93
11.3 LPP Formulation as an Advantage . . . . .	95
11.4 Kernel Trick in Soft Margin SVM . . . . .	96
11.5 Advantages of Soft Margin SVM with LPP Formulation . . . . .	96
11.6 Applications of Soft Margin SVM . . . . .	96
11.7 Task . . . . .	96
<b>12 Assignment 61</b>	
<b>Linear Programming for Signal Processing</b>	<b>101</b>
12.1 Introduction . . . . .	101
12.2 Overcomplete Dictionary: Combining DCT and Identity Matrix . . . . .	101
12.2.1 DCT Bases for Smooth Signal Representation . . . . .	101
12.2.2 Identity Matrix for Spiky Signal Representation . . . . .	101
12.2.3 Overcomplete Dictionary Construction . . . . .	102
12.3 Sparse Representation Using $\ell_1$ -Norm . . . . .	102
12.3.1 $\ell_1$ -Norm for Sparsity . . . . .	102
12.3.2 Interpretation of Coefficients . . . . .	102
12.4 Signal Generation . . . . .	102
12.5 Overcomplete Dictionary Construction . . . . .	103
12.5.1 Create the Overcomplete Dictionary . . . . .	103
12.6 Sparse Coefficient Representation . . . . .	103
12.6.1 Find Sparse Set of Coefficient Representation . . . . .	103
12.7 Interpretation of Results . . . . .	103
12.7.1 Extracting Smooth and Spiky Parts . . . . .	104
12.8 Computational Experiment . . . . .	104
12.9 Task . . . . .	106
<b>13 Assignment 62</b>	
<b>Verify Karush-Khun-Tucker Conditions for LP Problems Using CVX</b>	<b>111</b>
13.1 Introduction . . . . .	111
13.1.1 Example . . . . .	111
13.2 Tasks . . . . .	113
<b>14 Assignment 63</b>	
<b>Experiments in CVX with LP Problems to Understand Lagrangian Multipliers</b>	<b>119</b>
14.1 Introduction . . . . .	119
14.2 Linear Programming and Lagrangian Multipliers . . . . .	119
14.3 Geometric Interpretation of Lagrangian Multipliers . . . . .	120
14.4 Economic Interpretation of Lagrangian Multipliers . . . . .	120
14.5 Experiment 1: Solving an LP Problem Using CVX . . . . .	120
14.5.1 Problem Statement . . . . .	120
14.5.2 CVX Code . . . . .	120
14.5.3 Discussion . . . . .	121
14.6 Experiment 2: Economic Interpretation of Lagrangian Multipliers . . . . .	121
14.6.1 Problem Statement . . . . .	121
14.6.2 Discussion . . . . .	122
14.6.3 Solution using matrix approach . . . . .	122
14.7 Tasks . . . . .	123

<b>15 Assignment 64</b>	
<b>Hard Margin Support Vector Machine Formulation and Solution by CVX</b>	<b>127</b>
15.1 Introduction . . . . .	127
15.1.1 CVX Code Implementation . . . . .	128
15.1.2 Interpretation of Results . . . . .	129
15.1.3 Residuals and Complementary Conditions . . . . .	129
15.2 Task . . . . .	132

# List of Tables

9.1 Sample Dataset for Diabetes Prediction . . . . .	73
15.1 Dataset with support vectors . . . . .	129





# List of Figures

3.1	Reflection through projection along the normal to $y = x$ . . . . .	21
4.1	Visualization of constrained Optimization problem in Lagrange Multiplier Method . . .	30
4.2	Feasible set, level sets of $f(x)$ , directions of gradients with $g(x) \leq 0$ constraints . . . . .	32
4.3	Feasible set, level sets of $f(x)$ , directions of gradients of $g(x) \geq 0$ for a maximization problem . . . . .	33
10.1	$\epsilon$ tube generated with the solution of $\epsilon$ - insensitive loss function . . . . .	85
11.1	Hyper planes in SVM Classification problem . . . . .	89
11.2	Visualization of the SVM Classifier modelled through LPP . . . . .	93
12.1	Comparison of extraction of spiky components using optimization of LPP model . . . .	106
12.2	Comparison of extraction of spiky components with <i>sine</i> bases . . . . .	109
14.1	Feasible region of LPPs created with <code>plotregion</code> function . . . . .	123
15.1	Result of hard margin SVM classification through Lagrange Optimization Method . . .	135



# 1 | Assignment 49

## Construction of Vectors in Row Space and Column Space of a Matrix

### 1.1 Matrix Invertibility and Pseudo Inverse

A matrix  $A$  is said to be invertible if there exists another matrix  $B$  such that  $AB = BA = I$ , where  $I$  is the identity matrix. This matrix  $B$  is called the inverse of  $A$  and is denoted  $A^{-1}$ . Invertibility is a property that holds only for square matrices (matrices with equal numbers of rows and columns) and depends on certain conditions, primarily that the determinant of  $A$  is non-zero. When a matrix is invertible, it has a unique solution for systems of equations represented in the form  $Ax = b$ , and finding the inverse enables transformations and computations such as those involved in solving equations, transformations, and optimizations in multi-dimensional spaces.

The pseudo-inverse, often called the Moore-Penrose inverse, is a generalization of the inverse matrix that applies to both square and non-square matrices. Unlike the regular inverse, which exists only for square matrices that are invertible, the pseudo-inverse  $A^+$  can be defined for any matrix  $A$ , regardless of its shape or whether it is invertible. The pseudo-inverse provides a solution to systems of linear equations, particularly when the system has no exact solutions, or when there are more equations than unknowns (over-determined systems) or more unknowns than equations (under-determined systems).

The Moore-Penrose pseudo-inverse of a matrix  $A$  is denoted by  $A^+$  and is defined to satisfy the following conditions:

1.  $AA^+A = A$ ,
2.  $A^+AA^+ = A^+$ ,
3.  $(AA^+)^T = AA^+$ ,
4.  $(A^+A)^T = A^+A$ .

#### Condition of matrix invertibility

Consider a  $3 \times 3$  square matrix,  $A$ . If  $Ax = y$ , It maps from  $x \in \mathbb{R}^3$  to  $y \in \mathbb{R}^3$

If the matrix  $A$  is full rank, that is, if rank is three in our case, then

1. Row space = column space =  $\mathbb{R}^3$
2. Right null space = left null space = 0 vector
3. The mapping is a one to one mapping. That is, corresponding to a given vector  $x$ , there is a **unique**  $y$  vector in  $\mathbb{R}^3$ . Also given  $y$ , there is a **unique**  $x$  that maps to  $y$  under  $A$ . So, given  $y$  we can uniquely find corresponding  $x$ . This inverse mapping is achieved through a new matrix denoted as  $A^{-1}$ . Can you prove the uniqueness?.
4. What is  $A^{-1}$ ? It can be described in two ways.

a) using spectral theorem of matrices: It is a matrix with same eigenvectors as  $A$  but with eigenvalues that are reciprocal to that of  $A$ .

---

**1.1.1 General representation of vectors in a vector space**

b) In terms of orthogonality: Rows of  $A$  are bi-orthogonal to columns of  $A^{-1}$  and vice versa. This leads to  $AA^{-1} = A^{-1}A = I$

If the given square matrix is not having full rank, that is, if rank is less than three in our case, then

1. Row space  $\neq$  column space and both are different sub spaces of  $\mathbb{R}^3$  (unless  $A$  is symmetric)
2. Right null space  $\neq$  left null space  $\neq 0$  vector (we assume matrix is not symmetric)
3. The mapping is a **many to one** mapping. Hence there is no unique inverse mapping. So we say  $A^{-1}$  does not exist.
4. How to find several  $x'$ s that map to same  $C$ .

Take a vector from row-space and add different null space vector to that. Then all of them will be mapped to same vector  $y$  by  $A$ .

**1.1.2 Role of pseudo inverse in solution of system of equations**

5. What does pseudo inverse do? It just return row-space vector component of  $x$  which got mapped to  $y$ . This row-space vector component is unique.

Demonstrate this.

**Example 1**

For matrix  $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ , create an  $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$  such that  $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_R + x_N$ . Then map  $x$  using  $A$  to get  $y$ . Find  $\text{pinv}(A)^* y$ . Show that it is  $x_R$ .

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

$$\text{Let } x_R = 3 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}, x_N = 2 \begin{bmatrix} -2 \\ 1 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$$

$$x = x_R + x_N = \begin{bmatrix} 3 \\ 6 \end{bmatrix} + \begin{bmatrix} -4 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 8 \end{bmatrix}$$

$$y = Ax = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -1 \\ 8 \end{bmatrix} = \begin{bmatrix} 15 \\ 30 \end{bmatrix}.$$

$$\text{pinv}(A)^* y \text{ from matlab is } \begin{bmatrix} 3 \\ 6 \end{bmatrix} = x_R$$

Let us take another from null space and add to the same row space vector

$$\text{Let } x_N = 3 \begin{bmatrix} -2 \\ 1 \end{bmatrix} = \begin{bmatrix} -6 \\ 3 \end{bmatrix}$$

$$x = x_R + x_N = \begin{bmatrix} 3 \\ 6 \end{bmatrix} + \begin{bmatrix} -6 \\ 3 \end{bmatrix} = \begin{bmatrix} -3 \\ 9 \end{bmatrix}$$

$y = Ax = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -3 \\ 9 \end{bmatrix} = \begin{bmatrix} 15 \\ 30 \end{bmatrix}$ . Since we get same  $y$ ,  $\text{pinv}(A)^* y$  from matlab gives  $\begin{bmatrix} 3 \\ 6 \end{bmatrix} = x_R$ .

Note the points  $x = \begin{bmatrix} -3 \\ 9 \end{bmatrix}$  &  $x = \begin{bmatrix} -1 \\ 8 \end{bmatrix}$  map to the same point  $y = \begin{bmatrix} 15 \\ 30 \end{bmatrix}$  under the mapping by  $A$ .

The inverse mapping by pseudo inverse give  $\begin{bmatrix} 3 \\ 6 \end{bmatrix}$  which is the common row-space component of the vectors  $\begin{bmatrix} -3 \\ 9 \end{bmatrix}$  &  $\begin{bmatrix} -1 \\ 8 \end{bmatrix}$ .

## 1.2 Tasks

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

### SOLUTION

1. Generate a  $y$  (in column space) and then get five  $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$  that map to same  $y$ .

### SOLUTION

Any vector  $x$  in row space of  $A$  can be written as the linear combination of basis vectors of row space and the right null space as follows.

$$x = ax_R + bx_N$$

where  $x_R \in \text{basis}(\text{row space})(A)$  and  $x_N \in \text{basis}(N(A))$ .

The formula to create  $x$  having same  $y = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$  under  $Ax = y$  is

$$X = A^{-1}y + bX_N$$

The  $x$  values created are:

$$x_1 = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$$

$$x_2 = \begin{bmatrix} -2 \\ 6 \end{bmatrix}$$

$$x_3 = \begin{bmatrix} -4 \\ 7 \end{bmatrix}$$

$$x_4 = \begin{bmatrix} -6 \\ 8 \end{bmatrix}$$

$$x_5 = \begin{bmatrix} -8 \\ 9 \end{bmatrix}$$

Matlab code producing this result is shown below.

```
1 A=[1 2; 2 4];
2 y=[10;20];
3 x_N=[-2;1];
4 x_r=pinv(A)*y;
5 for a =1:5
```

## 1. Assignment 49

### Construction of Vectors in Row Space and Column Space of a Matrix

---

```
6 X=x_r+(a*x_N);  
7 disp(X);  
8 end
```

2. Generate five  $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$  that give same  $y$

To create vectors  $x$  which produce same  $y$ , we will use the following linear combination

$$x = x_R + bx_N$$

So five  $x$  vectors that produce same  $y$  under the linear transformation  $Ax = y$  are:

$$x_1 = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

$$x_2 = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$$

$$x_3 = \begin{bmatrix} -5 \\ 5 \end{bmatrix}$$

$$x_4 = \begin{bmatrix} -7 \\ 6 \end{bmatrix}$$

$$x_5 = \begin{bmatrix} -9 \\ 7 \end{bmatrix}$$

## RESULTS

1. Reinforce the core concept related to general representation of vectors in row space , column space and matrix inevitability
2. Create vectors in row space and column space of a given matrix

## 2 | Assignment 50

# Vector Space

### 2.1 Introduction



CONCEPT of vector space is the corner stone of advanced linear algebra. In the first section, basic concepts related to vector space is revisited.

Let  $V$  be a set of vectors. It is a vector space if  $x_1, x_2 \in V$  implies  $c_1 x_1 + c_2 x_2 \in V$ ;  $c_1, c_2 \in R$ . This means  $V$  is closed under vector addition.

In mathematics, it is called closure property of elements of set  $V$ .

**Corollary :**

0 vector is necessarily an element of vector space  $V$ .

**2. Prove the following constitute a vector space (associated with matrix A)**

- *Right null space*

Let  $x_1, x_2 \in N(A)$

$$x_1, x_2 \in N(A) \Rightarrow Ax_1 = 0; Ax_2 = 0;$$

We have to show that  $c_1 x_1 + c_2 x_2 \in N(A)$

$$\text{It is true since } A(c_1 x_1 + c_2 x_2) = c_1 Ax_1 + c_2 Ax_2 = 0 + 0 = 0$$

This proves  $c_1 x_1 + c_2 x_2 \in N(A)$

Therefore, right null space is a vector space.

- *Row space*

Usually a vector is always assumed as column vector in any derivations.

Let  $x_1, x_2 \in \text{Rowspace}(A)$ . This imply,

$$x_1 = A^T y_1 \equiv \text{linear combination of rows of } A$$

Elements of  $y_1$  vector are coefficients of linear combination. Similarly,

$$\begin{pmatrix} | \\ x_2 = A^T y_2 \equiv \text{linear combination of rows of } A \\ | \end{pmatrix}$$

We have to show that given  $c_1, c_2 \in R$ .

vector  $c_1 x_1 + c_2 x_2$  is linear combination of rows of A

It can be readily proved as follows.

$$c_1 x_1 + c_2 x_2 = c_1 A^T y_1 + c_2 A^T y_2 = A^T c_1 y_1 + A^T c_2 y_2 = A^T \underbrace{(c_1 y_1 + c_2 y_2)}_{\text{coefficient vector}}$$

$\Rightarrow c_1 x_1 + c_2 x_2$  is linear combination of rows of A

$\Rightarrow \text{row space is a vector space}$

- Column space

Let  $x_1, x_2 \in \text{columnspace}(A)$ . This imply,

$$\begin{pmatrix} | \\ x_1 = A y_1 \equiv \text{linear combination of columns of } A \\ | \end{pmatrix}$$

Elements of  $y_1$  vector are coefficients of linear combination. Similarly,

$$\begin{pmatrix} | \\ x_2 = A y_2 \equiv \text{linear combination of columns of } A \\ | \end{pmatrix}$$

We have to show that given  $c_1, c_2 \in R$ ,

$c_1 x_1 + c_2 x_2$  vector is linear combination of columns of A

It can be readily proved as follows.

$$c_1 x_1 + c_2 x_2 = c_1 A y_1 + c_2 A y_2 = A c_1 y_1 + A c_2 y_2 = A \underbrace{(c_1 y_1 + c_2 y_2)}_{\text{coefficient vector}}$$

$\Rightarrow c_1 x_1 + c_2 x_2$  is linear combination of columns of A

$\Rightarrow \text{columnspace is a vector space}$



**3. Prove that the following constitute a vector space**

**1. Set of 2x2 symmetric matrices**

$$\text{Let } A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}; B = \begin{pmatrix} e & f \\ f & g \end{pmatrix};$$

Hint : proceed to prove

**2. set of 2x2 Upper triangular matrices**

$$\text{Let } A = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix}; B = \begin{pmatrix} e & f \\ 0 & g \end{pmatrix};$$

Hint: proceed to prove

**4. If A and B are two nxn positive definite matrices then A+B is positive definite. Proof**

Let  $A$  and  $B$  be two positive definite matrices  
it imply  $x^T A x > 0$  &  $x^T B x > 0 \forall x$

We need to show  $A + B$  is positive definite

$$x^T (A + B) x = x^T A x + x^T B x > 0$$

Hence positive definite

**5. What are the required properties of a 'basis set' of vector spaces.**

- The set of Vectors in the basis set should be independent
- It should completely span the space. **No more, no less.**

**6. Prove that mapping of a non-singular matrix A is unique.**

We have to show that there is no

$$x_2 \neq x_1 \text{ such that } Ax_1 = y \text{ \& } Ax_2 = y$$

(this is answer to a question raised in last assignment)

Let  $A$  be the matrix. Since matrix  $A$  is non-singular, null space is 0 vector

$$\text{Let } Ax_1 = y$$

We have to show that there is no  $x_2 \neq x_1$  such that  $Ax_2 = y$

Suppose  $Ax_2 = y$

Then  $Ax_1 - Ax_2 = y - y = \underline{0} \Rightarrow A(x_1 - x_2) = \underline{0}$

But null space of A is a 0 vector

This imply  $x_1 - x_2 = \underline{0} \Rightarrow x_1 = x_2$

This contradict our assumption.

Hence if Square matrix is full rank(non-singular), then no two vectors will be mapped to a same vector.

This allows us to have unique inverse mapping from y to x.

### Assignment Questions from Strang's book.

**7. Find a basis for the plane  $x-2y+3z=0$  in  $R^3$ . Also find basis set for the line perpendicular to the plane.**

Hint:  $x-2y+3z=0$  define a plane passing through origin and hence is a vector space. So we have find two independent (x,y,z) vectors that satisfy the relation.

$$\begin{pmatrix} 1 & -2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0$$

Required vectors are the right null-space basis set of the matrix

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

Vector  $\begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}$  is perpendicular to the plane.

**8. Find a basis set for each of the subspaces of  $\mathbb{R}^4$**

- All vectors whose components are equal
- All vectors whose components add to zero
- All vectors that are perpendicular to  $(1\ 1\ 0\ 0)$  and  $(1\ 0\ 1\ 1)$

**SOLUTION**

**1. All vectors whose components are equal**

We are asked to find the basis for the subspace of  $\mathbb{R}^4$  where all components of the vectors are equal. Let the vector be  $\mathbf{v} = (x, x, x, x)$ , where  $x \in \mathbb{R}$ . This vector can be factored as:

$$\mathbf{v} = x(1, 1, 1, 1)$$

Thus, the subspace is spanned by the vector  $\mathbf{v}_1 = (1, 1, 1, 1)$ .

$$\text{Basis} = \{(1, 1, 1, 1)\}$$

This is a 1-dimensional subspace.

**2. All vectors whose components add to zero**

Let  $\mathbf{v} = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$  and suppose the components of  $\mathbf{v}$  must add to zero:

$$x_1 + x_2 + x_3 + x_4 = 0$$

This implies  $x_4 = -(x_1 + x_2 + x_3)$ , so we can express  $\mathbf{v}$  as:

$$\mathbf{v} = (x_1, x_2, x_3, -(x_1 + x_2 + x_3))$$

We can split this into a linear combination of independent vectors:

$$\mathbf{v} = x_1(1, 0, 0, -1) + x_2(0, 1, 0, -1) + x_3(0, 0, 1, -1)$$

Thus, the subspace is spanned by the following three vectors:

$$\text{Basis} = \{(1, 0, 0, -1), (0, 1, 0, -1), (0, 0, 1, -1)\}$$

This is a 3-dimensional subspace.

**3. All vectors that are perpendicular to  $(1, 1, 0, 0)$  and  $(1, 0, 1, 1)$**

Let  $\mathbf{u}_1 = (1, 1, 0, 0)$  and  $\mathbf{u}_2 = (1, 0, 1, 1)$ . We are asked to find the subspace of  $\mathbb{R}^4$  consisting of vectors  $\mathbf{v} = (x_1, x_2, x_3, x_4)$  that are perpendicular to both  $\mathbf{u}_1$  and  $\mathbf{u}_2$ . The conditions are:

$$\mathbf{v} \cdot \mathbf{u}_1 = 0 \quad \text{and} \quad \mathbf{v} \cdot \mathbf{u}_2 = 0$$

These give the following system of equations:

$$x_1 + x_2 = 0 \quad (\text{from } \mathbf{v} \cdot \mathbf{u}_1 = 0)$$

$$x_1 + x_3 + x_4 = 0 \quad (\text{from } \mathbf{v} \cdot \mathbf{u}_2 = 0)$$

From the first equation, we have  $x_1 = -x_2$ . Substituting this into the second equation:

$$-x_2 + x_3 + x_4 = 0 \quad \Rightarrow \quad x_3 + x_4 = x_2$$

Thus,  $\mathbf{v}$  can be written as:

$$\mathbf{v} = (-x_2, x_2, x_2 - x_4, x_4)$$

This gives two free variables,  $x_2$  and  $x_4$ , and we can express  $\mathbf{v}$  as a linear combination of two independent vectors:

$$\mathbf{v} = x_2(-1, 1, 1, 0) + x_4(0, 0, -1, 1)$$

Thus, the subspace is spanned by the following two vectors:

$$\text{Basis} = \{(-1, 1, 1, 0), (0, 0, -1, 1)\}$$

This is a 2-dimensional subspace.

**If A is 64 by 17 matrix of rank 11, how many independent vectors satisfy  $Ax=0$ ?. How many independent vectors satisfy  $A^T y=0$**

#### **SOLUTION**

Since  $\rho(A) = 11$ , using the rank nullity theorem,

$$\begin{aligned}\rho(A) + \text{Nullity}(A) &= \text{Number of columns}(A) \\ 11 + \text{Nullity}(A) &= 17 \\ \therefore \text{Nullity}(A) &= 17 - 11 \\ &= 6\end{aligned}$$

Number of independent vectors that satisfies  $Ax = 0$  is the cardinality of the basis of the right null space. Hence it is equal to  $\text{Nullity}(A) = 6$

#### **RESULTS**

1. Basic theorems in vector space are revisited.
2. Basis of some special vector spaces are found.

### 3 | Assignment 51

## Reflection Matrices Using Projections

**R**EFLECTION matrices are important in computational geometry and graphics. But less known fact is that, it is very important for finding solution to certain classes of  $Ax = b$  (arising in solution of partial differential equation using iterative method) and also for finding eigen values of symmetric matrices (House holder transformation matrices). Again, finding eigen values of symmetric matrices are important for computing SVD, the most power tool of LA. A simple case of reflection across the line  $y = x$  is shown in Figure 3.1.

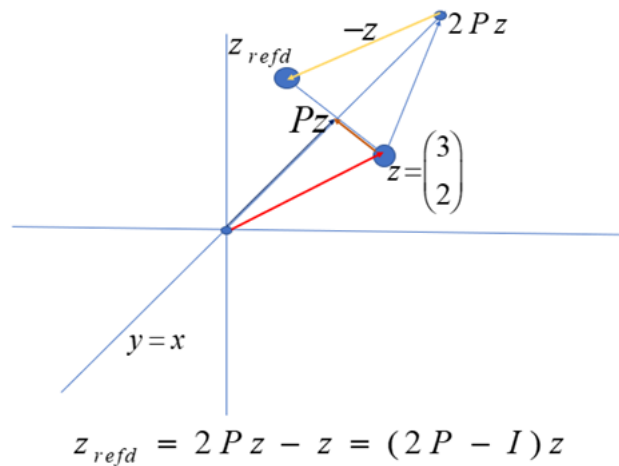


Figure 3.1: Reflection through projection along the normal to  $y = x$

Suppose we want to reflect point  $z = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$  across the line given by  $y = x$ .  $y = x$  is a line passing through origin.

Our aim is to use the concept of projection of a vector on to some vector space using appropriate projection matrix. We know the formula for projection of a vector on to column space of a matrix  $A$ . The projection formula using the coefficient matrix is of the system,  $Ax = b$  is  $P = A(A^T A)^{-1} A^T$ . Projected vector of  $b$  on to column space of  $A$  is  $Ax$ .

In our case the reflection plane is  $y = x$ . A vector on the line is  $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ . This spans entire points on the line. So we take  $A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  and project the given point  $z = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$  on to column space of  $A$ . The projection matrix  $P$  is

$$P = A(A^T A)^{-1} A^T = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

From the above diagram, we observe that

$$Z_{refd} = 2Pz - Iz = (2P - I)z = 2 \times \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Matlab code for the above task is given below.

```
1 A=null([1 -1]);  
2 z=[3 2]';  
3 P=A*inv(A'*A)*A';  
4 zrefd=2*P*z-z
```

Output of the code is shown below.

```
zrefd = 2x1  
    2.0000  
    3.0000
```

Reflection of the vector is generated using the following code.

```
1 %checking by reflecting back  
2 z1=2*P*zrefd-zrefd
```

```
z1 = 2x1  
    3.0000  
    2.0000
```

### 3.0.1 3D case- Reflection across Plane passing through origin

Let the plane be given by  $x - 2y + z = 0$  and we want to project  $z = \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}$ .

We need to create two independent vectors (basis set for the plane through origin) on the plane and put as columns of a matrix A. This is readily obtained in matlab as null space of matrix

$$\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

```
1 A=null([1 2 -1])
```

The required projection matrix is

$$P = A \operatorname{inv}(A^T A) A^T$$

The reflected vector is

$$Z_{refd} = 2Pz - z$$

From  $Z_{refd}$  you can get back  $z$  by reflecting  $Z_{refd}$  across the same plane.

Matlab code for this task is given below.

```
1 A=null([1 -2 1]);  
2 z=[2 3 5]';  
3 P=A*inv(A'*A)*A';  
4 zrefd=2*P*z-z
```

### 3. Assignment 51

#### Reflection Matrices Using Projections

---

```
zrefd = 3x1
    1.6667
    3.6667
    4.6667
```

```
1 %checking by reflecting back
2 z1=2*P*zrefd-zrefd
```

```
z1 = 3x1
    2.0000
    3.0000
    5.0000
```

```
1 % you should get z1 as z
```

### 3.0.2 Assignments

1. Reflect  $z = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$  across the line  $x + 2y = 0$ .

#### SOLUTION

Here  $A = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$ . So the projection,

$$\begin{aligned} P &= A(AA^T)^{-1}A^T \\ &= \begin{bmatrix} \frac{4}{5} & -\frac{2}{5} \\ -\frac{2}{5} & \frac{1}{5} \end{bmatrix} \\ \therefore Z_{refd} &= (2P - I)z \\ &= 2\frac{1}{5} \begin{bmatrix} 4 & 5 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} \\ &= \begin{bmatrix} -\frac{6}{5} \\ \frac{17}{5} \end{bmatrix} \end{aligned}$$

Matlab code for this task is given below.

```
1 A=null([1 2]);
2 z=[2 3]';
3 P=A*inv(A'*A)*A';
4 zrefd=2*P*z-z
```

```
zrefd = 2x1
    -1.2000
    -3.4000
```

```
1 %checking by reflecting back
2 z1=2*P*zrefd-zrefd
```

### 3. Assignment 51

#### Reflection Matrices Using Projections

---

$$z1 = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$$

2. Reflect  $z = \begin{bmatrix} 2 \\ 2 \\ 5 \end{bmatrix}$  across the plane  $2x + 3y + 2z = 0$ .

#### SOLUTION

We can solve this problem in the same approach. Only difference is that, here the span of null space of  $A$  contains two independent vectors. Matlab code for this task is given below.

```
1 A=null([2 3 2]);
2 z=[2 2 -5]';
3 P=A*inv(A'*A)*A';
4 zrefd=2*P*z-z
```

$$zrefd = \begin{bmatrix} 2.0000 \\ 2.0000 \\ -5.0000 \end{bmatrix}$$

```
1 %checking by reflecting back
2 z1=2*P*zrefd-zrefd
```

$$z1 = \begin{bmatrix} 2.0000 \\ 2.0000 \\ -5.0000 \end{bmatrix}$$

3. Reflect  $z = \begin{bmatrix} 2 \\ 2 \\ -5 \end{bmatrix}$  across the plane  $2x + 3y + 2z = 10$ .

#### SOLUTION

Here there is a small twist in the formation of the matrix  $A$ . The plane  $2x + 3y + 2z = 10$  does not pass through the origin. So  $A$  is not the null space instead  $A = \begin{bmatrix} 2 & 3 & 2 \end{bmatrix}^{-1} 10$ .

Matlab code for this task is given below.

```
1 CM=[2 3 2];
2 A=pinv(CM)*[10];
3 z=[2 2 -5]';
4 P=A*inv(A'*A)*A';
5 zrefd=2*P*z-z
```

$$zrefd = \begin{bmatrix} -2.0000 \\ -2.0000 \\ 5.0000 \end{bmatrix}$$



```
1 %checking by reflecting back
2 z1=2*P*zrefd-zrefd
```

```
z1 = 3x1
      2.0000
      2.0000
     -5.0000
```

4. Write down the 2x2 matrices that:

(a) Reverse the direction of the vector  $v$ .

**SOLUTION**

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

(b) Project every vector on to  $x$  axis

**SOLUTION**

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

(c) Turn every vector counter clockwise 90 degree

**SOLUTION**

$$R_{90} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

(d) Reflect every vector through the 45 degree line  $y = x$

**SOLUTION**

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

Let  $\mathbf{v} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$ .

The reflection matrix through the line  $y = x$  is:

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

To find the reflected vector  $\mathbf{v}'$ :

$$\mathbf{v}' = M \cdot \mathbf{v} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Calculating the result:

$$\mathbf{v}' = \begin{pmatrix} 0 \cdot 3 + 1 \cdot 2 \\ 1 \cdot 3 + 0 \cdot 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

The reflected vector is:

$$\mathbf{v}' = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

5. Find  $3 \times 3$  matrices  $B$  such that

- (a)  $BA = 2A$  for every  $A$  ( $3 \times n$  matrix)

**SOLUTION**

We seek a  $3 \times 3$  matrix  $B$  such that for any  $3 \times n$  matrix  $A$ , the relation

$$BA = 2A$$

holds true.

Assuming  $B$  can be expressed as:

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

we require that

$$B \begin{pmatrix} a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix} = 2 \begin{pmatrix} a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix}$$

for every column vector  $A_j$  of  $A$ .

This implies that for each component, the following equations must be satisfied:

$$b_{11}a_{1j} + b_{12}a_{2j} + b_{13}a_{3j} = 2a_{1j}$$

$$b_{21}a_{1j} + b_{22}a_{2j} + b_{23}a_{3j} = 2a_{2j}$$

$$b_{31}a_{1j} + b_{32}a_{2j} + b_{33}a_{3j} = 2a_{3j}$$

Since this must hold for all  $a_{ij}$ , we conclude:

$$b_{11} = 2, \quad b_{12} = 0, \quad b_{13} = 0$$

$$b_{21} = 0, \quad b_{22} = 2, \quad b_{23} = 0$$

$$b_{31} = 0, \quad b_{32} = 0, \quad b_{33} = 2$$

Thus, the matrix  $B$  that satisfies the equation for any  $A$  is:

$$B = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} = 2I$$

where  $I$  is the  $3 \times 3$  identity matrix.

- (b)  $BA$  has the first and last rows of  $A$  reversed.

**SOLUTION**

Let  $A$  be a  $3 \times n$  matrix defined as:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \end{pmatrix}$$

We aim to find a  $3 \times 3$  matrix  $B$  such that the product  $BA$  results in a matrix where the first and last rows of  $A$  are reversed:

$$BA = \begin{pmatrix} a_{31} & a_{32} & \cdots & a_{3n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{11} & a_{12} & \cdots & a_{1n} \end{pmatrix}$$

To achieve this, we propose the following matrix  $B$ :

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

Now, we will compute the product  $BA$ :

$$\begin{aligned} BA = B \cdot A &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \end{pmatrix} \\ &= \begin{pmatrix} a_{31} & a_{32} & \cdots & a_{3n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{11} & a_{12} & \cdots & a_{1n} \end{pmatrix} \end{aligned}$$

Thus, we have shown that the matrix  $B$  successfully reverses the first and last rows of the matrix  $A$ :

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


## RESULTS

Reflection of a vector is interpreted as a projection across the given line/ plane. In 2 space , the reflection matrix is the right reflection of  $I_2$ .



## 4 | Assignment 52

# Mathematical Model for Support Vector Machine (SVM)

 LAGRANGIAN multiplier is the most important concept in optimization theory. It forms the basis for the modern distributed computing optimization algorithms that is getting utilized in many data science and machine learning algorithms.

Indian Engineers lack in two things

- Mathematics required for AI and Data Science
- Ability to code for mathematical concepts

China has overcome this problem, by “mathematizing and digitalizing computer science and engineering departments” & algorithmizing mathematics department”. It is tomorrow's super power in AI and robotics.

**India may take long time to adapt to the situation.**

As we proceed we will find that (like SVD in LA) Lagrangian multiplier pervades all of modern optimization algorithms. Entire Signal and image processing algorithms now use optimization theory. Entire machine learning is formulated as optimization problems. Entire control theory (adaptive, model predictive control and data driven control theory) is slowly becoming optimization based. So learn this concept properly to make ourselves ‘future proof’.

**Lagrangian multiplier with inequality constraints**

Unlike the case with equality constraints, where Lagrangian multiplier can take any sign, here the Lagrangian multiplier is restricted to taking only one particular sign if the constraint is **active** (read on to understand what it is). This peculiarity should be understood geometrically.

**Zero Lagrangian multiplier** happens when optimum value of the objective function is not on the boundary of the feasible region. If optimum is not on boundary, then we say the **constraint is inactive**

**Non-zero Lagrangian multiplier** happens when optimum value of objective function is on the boundary of the feasible region. In this case we say the **constraint is active. This means, if we relax the constraint, we will get better optimum.**

Let us demonstrate with an obvious example. Let our problem is:

$$\underset{(x_1, x_2) \in \mathbb{R}^2}{\text{minimize}} f(x_1, x_2) = x_1^2 + x_2^2$$

Subject to:

$$g(x_1, x_2) = 5x_1 + 6x_2$$

This Optimization problem has a simple meaning. In the feasible region defined by the inequality constraint, which point has **minimum square norm**. Obviously, it is the point closest to origin.

A lay person (say, a high school student) can find this point by drawing concentric circles around origin. For some radius, the circle will just graze the boundary of the feasible region at one point.

That point is the required solution point,  $X^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix}$ .

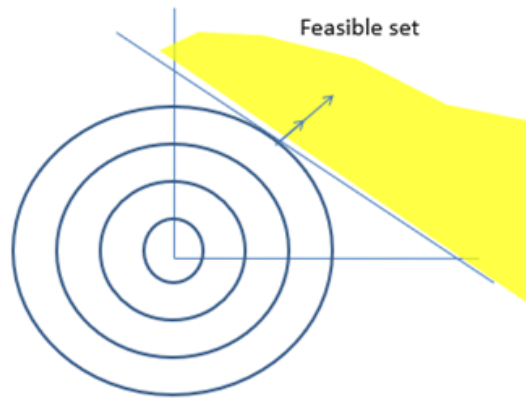


Figure 4.1: Visualization of constrained Optimization problem in Lagrange Multiplier Method

This geometrical view point is very essential for your scaling new heights in Optimization theory and is shown in Figure 4.1.

Here the optimum (min value) is on the boundary and **gradient of objective function and gradient of constraint (as a function) are in the same direction.**

This imply that

$$\nabla f(x) = \lambda \nabla g(x) \quad ; \lambda \geq 0$$

So while writing Lagrangian function, we should take care so that above condition is obtained after applying first order optimality condition.

So the **Lagrangian must** be written as

$$\mathcal{L}(X, \lambda) = f(X) - \lambda g(X) \quad \lambda \geq 0$$

so that

$$\nabla f(x) = \lambda \nabla g(x) \quad ; \lambda \geq 0$$

as required.

**In case we write**

$$\mathcal{L}(X, \lambda) = f(X) + \lambda g(X) \quad \lambda \geq 0$$

We obtain

$$\nabla f(x) = -\lambda \nabla g(x) \quad ; \lambda \geq 0$$

This is wrong.

So for our problem, Lagrangian is

$$\mathcal{L}(X, \lambda) = x_1^2 + x_2^2 - \lambda (5x_1 + 6x_2 - 30); \quad \lambda \geq 0$$

And first order optimality conditions are given by

$$\begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} = \lambda \begin{bmatrix} 5 \\ 6 \end{bmatrix} \quad (4.1)$$

$$\text{and } \lambda (5x_1 + 6x_2) = 30 \quad (4.2)$$

Equation (4.2), provides two choices for  $\lambda$ .

$$\lambda = 0 \text{ or } \lambda \neq 0$$

Putting  $\lambda = 0$  in (4.1) gives

$$X^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

But this solution won't satisfy our constraint inequality  $5x_1 + 6x_2 \geq 0$  Hence is nonzero and positive and from (4.2), we obtain

$$5x_1 + 6x_2 = 0 \quad (4.3)$$

It says solution is on boundary of the constraint region. From (4.1) we obtain ,

$$X^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} \frac{5\lambda}{2} \\ \frac{6\lambda}{2} \end{bmatrix} \quad (4.4)$$

Substituting this in (4.3) we obtain,

$$25\frac{\lambda}{2} + 36\frac{\lambda}{2} - 36 = 0 \quad (4.5)$$

Put this in (4.4) to obtain the required point.

#### Extension to multiple constraints

$$\text{Min}_X f(X)$$

Subject to:

$$g_1(X) \geq 0$$

$$g_2(X) \geq 0$$

The Lagrangian function is

$$\mathcal{L}(X, \lambda_1, \lambda_2) = f(X) - \lambda_1 g_1(X) - \lambda_2 g_2(X); \quad \lambda_1, \lambda_2 \geq 0$$

#### First order Optimality conditions are

$$\nabla f(X) = \lambda_1 \nabla g_1(X) + \lambda_2 \nabla g_2(X); \quad \lambda_1, \lambda_2 \geq 0$$

$$\left. \begin{array}{l} \lambda_1 g_1(X) = 0 \\ \lambda_2 g_2(x) = 0 \end{array} \right\} \text{Complementary Conditions}$$

In literature , these conditions together are called KKT(Karush-Kuhn-Tucker) conditions for optimality.

#### 1. Write KKT conditions for

$$\min_x f(x); \text{ Subject to: } g(x) \geq 0$$

For example

$$\min_x f(X) = x_1^2 + x_2^2$$

$$\text{Subject to: } g(x) = 5x_1 + 6x_2 + 30 \leq 0$$

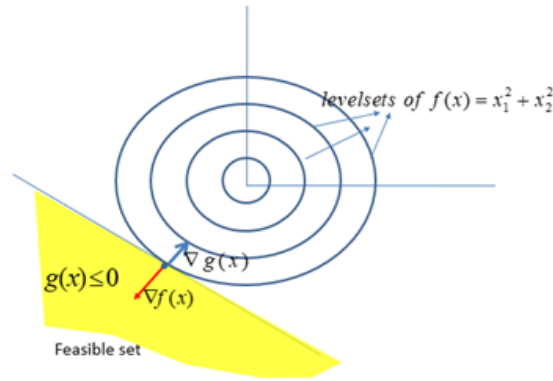


Figure 4.2: Feasible set, level sets of  $f(x)$ , directions of gradients with  $g(x) \leq 0$  constraints

In this case of **minimization** problem with “less than or equal to zero” ( $\leq 0$ ) constraint, **if the constraint is active, gradients will be always in opposite direction**. So we write Lagrangian as

$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x); \quad \lambda \geq 0$$

So that KKT condition for optimality are

$$\nabla f(x) = -\lambda \nabla g(x); \quad \lambda \geq 0$$

## 2. Write KKT conditions for

$$\begin{aligned} & \text{Min}_X \quad f(X) \\ & \text{Subject to:} \\ & \quad g_1(X) \leq 0 \\ & \quad g_2(X) \leq 0 \end{aligned}$$

The Lagrangian function is

$$\mathcal{L}(X, \lambda_1, \lambda_2) = f(X) + \lambda_1 g_1(X) + \lambda_2 g_2(X); \quad \lambda_1, \lambda_2 \geq 0$$

**First order Optimality conditions are**

$$\begin{aligned} & \nabla f(X) = -\lambda_1 \nabla g_1(X) - \lambda_2 \nabla g_2(X); \quad \lambda_1, \lambda_2 \geq 0 \\ & \left. \begin{aligned} \lambda_1 g_1(X) &= 0 \\ \lambda_2 g_2(X) &= 0 \end{aligned} \right\} \text{Complementary Conditions} \end{aligned}$$

Instead of remembering how to write Lagrangian properly for all distinct cases, we can convert every inequality constraint into one type (say  $\geq 0$ ) and write the Lagrangian. So we need to remember only one case. For example we convert the above  $\geq 0$  constraints into as follows:

$$\begin{aligned} & \text{Min}_X \quad f(X) \\ & \text{Subject to:} \\ & \quad -g_1(X) \geq 0 \\ & \quad -g_2(X) \geq 0 \end{aligned}$$



And write the Lagrangian as

$$\begin{aligned}\mathcal{L}(x, \lambda) &= f(x) + \lambda g_1(x) + \lambda_2 g_2(x); \quad \lambda_1, \lambda_2 \geq 0 \\ \nabla f(X) &= -\lambda_1 \nabla g_1(X) - \lambda_2 \nabla g_2(X); \quad \lambda_1, \lambda_2 \geq 0 \\ \left. \begin{aligned} \lambda_1 g_1(X) &= 0 \\ \lambda_2 g_2(x) &= 0 \end{aligned} \right\} &\text{Complementary Conditions}\end{aligned}$$

Note that we get the required optimality condition as required.

**One more complexity if the given objective function is of maximization type.**

For example

$$\begin{aligned}\max_x f(x) \\ g(x) \geq 0\end{aligned}$$

For example

$$\begin{aligned}\text{Max}_X \quad & e^{-(x_1^2 + x_2^2)} \\ \text{Subject to:} \\ & 5x_1 + 6x_2 + 30 \geq 0\end{aligned}$$

The unconstrained maximum is at origin. For the constrained problem, the maximum is on the boundary where gradients are in opposite direction.

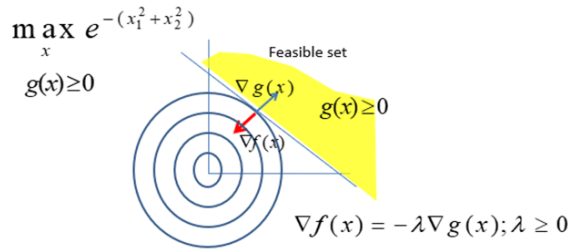


Figure 4.3: Feasible set, level sets of  $f(x)$ , directions of gradients of  $g(x) \geq 0$  for a maximization problem

We have to find geometrically what will be the relationship between direction of gradients. To avoid the confusion, we convert maximization problem into minimization problem as follows.

$$\begin{aligned}\text{Min}_x \quad & (-f(x)) \\ \text{Subject to:} \\ & g(x) \geq 0\end{aligned}$$

So the Lagrangian function is (one must remember rule for “min”,  $\geq 0$  combination)

$$\mathcal{L}(x, \lambda) = -f(x) - \lambda g(x); \quad \lambda \geq 0$$

So the KKT conditions are:

$$\begin{aligned}\nabla f(x) &= -\lambda \nabla g(x); \quad \lambda \geq 0 \\ \lambda g(x) &= 0\end{aligned}$$

Note that we are getting the required condition with this technique.

So always remember just one rule : the rule for “min”, “ $\geq 0$ ” combination.

**Now we are ready for writing KKT condition for a general problem.**

$$\begin{aligned} \min_x f(x) \quad & \left\{ \begin{array}{l} h(x) = 0 \\ g_1(x) \leq 0 \\ g_2(x) \geq 0 \end{array} \right\} \Rightarrow \min_x f(x) \quad \left\{ \begin{array}{l} h(x) = 0 \\ -g_1(x) \geq 0 \\ g_2(x) \geq 0 \end{array} \right\} \\ \Rightarrow \mathcal{L}(x, \lambda_0, \lambda_1, \lambda_2) = f(x) - \lambda_0 h(x) + \lambda_1 g_1(x) - \lambda_2 g_2(x); & \lambda_1, \lambda_2 \geq 0 \\ \nabla f(x) = \lambda_0 \nabla h(x) - \lambda_1 \nabla g_1(x) + \lambda_2 \nabla g_2(x); & \lambda_1, \lambda_2 \geq 0 \\ \lambda_1 g_1(x) = 0 & \\ \lambda_2 g_2(x) = 0 & \end{aligned}$$

**KKT condition for a general problem**

$$\begin{aligned} \min_x f(x) \\ h_i(x) = 0; \quad i = 1, 2, \dots, n \\ g_j(x) \geq 0; \quad j = 1, 2, \dots, m \end{aligned}$$

Lagrangian in vector notation

$$\mathcal{L}(X, \mu, \lambda) = f(X) - \mu^T \mathbf{h}(X) - \lambda^T \mathbf{g}(X)$$

KKT conditions in matrix form is

$$\nabla f(x) = \mu^T \nabla \mathbf{h}(x) - \lambda^T \nabla \mathbf{g}(x); \quad \lambda \geq 0$$

### Question

Write KKT condition for hard margin Support vector machine problem given by. Take lagrangian multipliers as  $\mu_i$ ,  $i=1$  to  $m$

$$\min_{w, \gamma} \frac{1}{2} \mathbf{w}^T \mathbf{w}$$

Subject to:

$$d_i(W^T x_i - \gamma) \geq 1 \quad \forall i = 1, \dots, m$$

### SOLUTION

Given optimization problem is:

$$\min_{W, \gamma} \frac{1}{2} W^T W$$

subject to:

$$d_i(W^T x_i - \gamma) \geq 1 \quad \forall i = 1, \dots, m$$

### 1. Lagrangian

The Lagrangian for this problem is:

$$\mathcal{L}(W, \gamma, \mu) = \frac{1}{2} W^T W - \sum_{i=1}^m \mu_i (d_i(W^T x_i - \gamma) - 1)$$

where  $\mu_i \geq 0$  are the Lagrange multipliers associated with each constraint.

## 2. KKT Conditions

The Karush-Kuhn-Tucker (KKT) conditions include the following:

### Stationarity

Taking the partial derivatives of the Lagrangian with respect to  $W$  and  $\gamma$ :

$$\frac{\partial \mathcal{L}}{\partial W} = W - \sum_{i=1}^m \mu_i d_i x_i = 0$$

which gives:

$$W = \sum_{i=1}^m \mu_i d_i x_i$$

Similarly, the derivative with respect to  $\gamma$  gives:

$$\frac{\partial \mathcal{L}}{\partial \gamma} = - \sum_{i=1}^m \mu_i d_i = 0$$

which simplifies to:

$$\sum_{i=1}^m \mu_i d_i = 0$$

### Primal Feasibility

The primal feasibility condition is simply the original constraint:

$$d_i(W^T x_i - \gamma) \geq 1 \quad \forall i = 1, \dots, m$$

### Dual Feasibility

The dual feasibility condition requires that the Lagrange multipliers are non-negative:

$$\mu_i \geq 0 \quad \forall i = 1, \dots, m$$

### Complementary Slackness

The complementary slackness condition ensures that either the Lagrange multiplier is zero or the corresponding constraint is active:

$$\mu_i (d_i(W^T x_i - \gamma) - 1) = 0 \quad \forall i = 1, \dots, m$$

## RESULTS

1. Concept behind the Lagrange multiplier method is revisited.
2. Solution of linear system is compared using algebraic and Lagrange methods, and identified the major differences.
3. Karush-Kuhn-Tucker conditions for various type of non-linear constrained optimization problem is revisited.
4. Matrix form of KKT conditions is formulated.
5. KKT conditions for the Lagrange multiplier model of hard margin Support Vector Machine problem is formulated.



# 5 | Assignment 53

## Formulation and Solution of Hard Margin SVM

### 5.1 Introduction

The best way to learn the art of applying Lagrangian multiplier is trying to solve SVM problem in the dual variables.

Consider 4 points as given below. Two points belong to class -1 and another two point belongs to class +1. Aim is to find a linear classifier (equation of a plane) that maximally separate the two classes of objects.

Point ID	$x_1$	$x_2$	$d$
1	1	1	-1
2	0.5	0.5	-1
3	3	3	1
4	4	5	1

### 5.2 Primal and Dual Problem of Hard Margin SVM

Given the dataset:

Point ID	$x_1$	$x_2$	$d$
1	1	1	-1
2	0.5	0.5	-1
3	3	3	1
4	4	5	1

we will derive both the primal and dual problem formulations for the hard-margin SVM.

#### 5.2.1 Primal Problem

The primal optimization problem for the hard-margin SVM is to find the separating hyperplane  $W^T x - \gamma = 0$ , subject to the constraints that each data point is correctly classified with a margin of at least 1. The primal problem can be formulated as:

$$\min_{W, \gamma} \quad \frac{1}{2} W^T W$$

subject to:

$$d_i(W^T x_i - \gamma) \geq 1 \quad \forall i = 1, 2, 3, 4$$

This can be written explicitly for each data point as:

---

$$\begin{aligned}
d_1(W_1 \cdot 1 + W_2 \cdot 1 - \gamma) &\geq 1 &\Rightarrow -W_1 - W_2 + \gamma &\geq 1, \\
d_2(W_1 \cdot 0.5 + W_2 \cdot 0.5 - \gamma) &\geq 1 &\Rightarrow -0.5W_1 - 0.5W_2 + \gamma &\geq 1, \\
d_3(W_1 \cdot 3 + W_2 \cdot 3 - \gamma) &\geq 1 &\Rightarrow 3W_1 + 3W_2 - \gamma &\geq 1, \\
d_4(W_1 \cdot 4 + W_2 \cdot 5 - \gamma) &\geq 1 &\Rightarrow 4W_1 + 5W_2 - \gamma &\geq 1.
\end{aligned}$$

### 5.2.2 Matrix Form of the Primal Problem

The primal problem can be written in matrix form as:

$$\min_{W, \gamma} \quad \frac{1}{2} W^T W$$

subject to:

$$D(XW - \gamma \mathbf{1}) \geq \mathbf{1}$$

where:

$$X = \begin{bmatrix} 1 & 1 \\ 0.5 & 0.5 \\ 3 & 3 \\ 4 & 5 \end{bmatrix}, \quad D = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

### 5.2.3 Dual Problem

To derive the dual problem, we introduce Lagrange multipliers  $\mu_i$  for each constraint  $d_i(W^T x_i - \gamma) \geq 1$ . The Lagrangian for the primal problem is:

$$L(W, \gamma, \mu) = \frac{1}{2} W^T W - \sum_{i=1}^m \mu_i (d_i(W^T x_i - \gamma) - 1)$$

To derive the dual, we take the partial derivatives of the Lagrangian with respect to  $W$  and  $\gamma$  and set them to 0:

$$\frac{\partial L}{\partial W} = W - \sum_{i=1}^m \mu_i d_i x_i = 0$$

$$W = \sum_{i=1}^m \mu_i d_i x_i$$

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^m \mu_i d_i = 0$$

Substitute  $W = \sum_{i=1}^m \mu_i d_i x_i$  into the Lagrangian to get the dual problem:

$$\max_{\mu} \quad \sum_{i=1}^m \mu_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \mu_i \mu_j d_i d_j x_i^T x_j$$

subject to:

$$\sum_{i=1}^m \mu_i d_i = 0$$

$$\mu_i \geq 0 \quad \forall i$$

### 5.2.4 Matrix Form of the Dual Problem

The dual problem can also be written in matrix form as:

$$\max_{\mu} \quad \mathbf{1}^T \mu - \frac{1}{2} \mu^T (D X X^T D) \mu$$

subject to:

$$\mu^T D \mathbf{1} = 0, \quad \mu \geq 0$$

where  $X$ ,  $D$ , and  $\mathbf{1}$  are as defined above. Those data points for which the Lagrangian multiplier is zero are called *support vectors*.

## 5.3 Tasks

1. Solve the above primal and dual problem using CVX

### SOLUTION

As the first step of solution, let's formulate the problem using given data point mathematically.

The points given in the sample problem can be vectorize as follows:

$$x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \quad x_4 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

And their corresponding labels  $d_1 = -1$ ,  $d_2 = -1$ ,  $d_3 = 1$ ,  $d_4 = 1$ .

### Step 1: Kernel Matrix Calculation

The kernel matrix  $K$  is computed as the inner products between all pairs of data points  $x_i^T x_j$ . Specifically:

$$K_{ij} = x_i^T x_j$$

Let's calculate the entries of the kernel matrix  $K$ :

$$K = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & x_1^T x_3 & x_1^T x_4 \\ x_2^T x_1 & x_2^T x_2 & x_2^T x_3 & x_2^T x_4 \\ x_3^T x_1 & x_3^T x_2 & x_3^T x_3 & x_3^T x_4 \\ x_4^T x_1 & x_4^T x_2 & x_4^T x_3 & x_4^T x_4 \end{bmatrix}$$

Substitute the values of the points:

- (a)  $x_1^T x_1 = 1 \times 1 + 1 \times 1 = 2$
- (b)  $x_1^T x_2 = 1 \times 0.5 + 1 \times 0.5 = 1$
- (c)  $x_1^T x_3 = 1 \times 3 + 1 \times 3 = 6$
- (d)  $x_1^T x_4 = 1 \times 4 + 1 \times 5 = 9$
- (e)  $x_2^T x_2 = 0.5 \times 0.5 + 0.5 \times 0.5 = 0.5$
- (f)  $x_2^T x_3 = 0.5 \times 3 + 0.5 \times 3 = 3$
- (g)  $x_2^T x_4 = 0.5 \times 4 + 0.5 \times 5 = 4.5$
- (h)  $x_3^T x_3 = 3 \times 3 + 3 \times 3 = 18$

$$(i) \quad x_3^T x_4 = 3 \times 4 + 3 \times 5 = 27$$

$$(j) \quad x_4^T x_4 = 4 \times 4 + 5 \times 5 = 41$$

Thus, the kernel matrix  $K$  is:

$$K = \begin{bmatrix} 2 & 1 & 6 & 9 \\ 1 & 0.5 & 3 & 4.5 \\ 6 & 3 & 18 & 27 \\ 9 & 4.5 & 27 & 41 \end{bmatrix}$$

## Step 2: Dual Problem Formulation

The dual problem for the hard-margin SVM is formulated as:

$$\max_{\mu} \left( \sum_{i=1}^4 \mu_i - \frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 \mu_i \mu_j d_i d_j K_{ij} \right)$$

where  $\mu$  are the Lagrange multipliers, and  $d_i$  are the labels. The kernel matrix  $K$  has been computed in Step 1.

The dual problem is subject to the constraints:

$$\sum_{i=1}^4 \mu_i d_i = 0, \quad \mu_i \geq 0 \quad \forall i$$

## Step 3: Write the Dual Problem Explicitly

We can now explicitly write the dual problem. Let the labels be:

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

Then, the dual problem becomes:

$$\begin{aligned} \text{Max}_{\mu} \left( \mu_1 + \mu_2 + \mu_3 + \mu_4 - \frac{1}{2} [2\mu_1^2 + 0.5\mu_2^2 + 18\mu_3^2 + 41\mu_4^2 \right. \\ \left. + 2\mu_1\mu_2 + 6\mu_1\mu_3 + 9\mu_1\mu_4 + 3\mu_2\mu_3 + 4.5\mu_2\mu_4 + 27\mu_3\mu_4] \right) \end{aligned}$$

## Step 4: Constraints

The dual problem is subject to the following constraints:

$$\mu_1 d_1 + \mu_2 d_2 + \mu_3 d_3 + \mu_4 d_4 = 0$$

Substitute the values of  $d_i$ :

$$\mu_1(-1) + \mu_2(-1) + \mu_3(1) + \mu_4(1) = 0$$



$$-\mu_1 - \mu_2 + \mu_3 + \mu_4 = 0$$

And the positivity constraints:

$$\mu_1 \geq 0, \quad \mu_2 \geq 0, \quad \mu_3 \geq 0, \quad \mu_4 \geq 0$$

Now this dual problem can be solved using the quadratic programming. Formulation of this QPP is given below.

### 5.3.1 Formation of QPP for the dual

The dual problem can be expressed in matrix form:

$$L(\mu) = \mu^T \mathbf{1} - \frac{1}{2} \mu^T Q \mu$$

Where  $Q$  is defined as:

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

Calculating  $Q$  using given labels and Kernel matrix :

$$Q = \begin{bmatrix} 1 & 1 & -6 & -9 \\ 1 & 0.25 & -3 & -4.5 \\ -6 & -3 & 18 & 27 \\ -9 & -4.5 & 27 & 41 \end{bmatrix}$$

Using the quadratic programming technique, we can derive the optimal Lagrange multipliers  $\mu$ . The solution yields the optimal values for  $\mu$  and identifies the support vectors.

Once the optimal  $\mu$  values are found, the decision boundary (classifier) can be expressed as:

$$f(x) = w^T x - \gamma$$

Where  $w$  is calculated as:

$$w = \sum_{i=1}^m \mu_i d_i x_i$$

The offset  $\gamma$  can be determined from any support vector  $x_s$  using:

$$\gamma = w^T x_s - 1$$

Matlab code to solve the primal problem is given below.

```
1 % Clear workspace
2 clear;
3 clc;
4
5 % Add CVX path (adjust this path as necessary)
6 addpath('C:\path\to\cvx'); % Replace with your CVX path
7 cvx_setup;
8
```

```

9  % Data points
10 X = [1, 1; 0.5, 0.5; 3, 3; 4, 5]; % Feature matrix
11 d = [-1; -1; 1; 1]; % Labels
12
13 % Number of samples
14 m = size(X, 1);
15
16 % CVX for solving the primal problem
17 cvx_begin quiet
18     % Declare the variables
19     variable w(2); % Weight vector
20     variable b; % Bias term
21     variable xi(m); % Slack variables for margin
22
23     % Objective: Minimize ||w||^2 / 2 + C * sum(xi) for hinge
24     % loss
25     C = 1; % Regularization parameter
26     minimize(0.5 * (w' * w) + C * sum(xi))
27
28     % Subject to: Constraints for hard-margin SVM
29     subject to
30         d .* (X * w + b) >= 1 - xi; % Margin constraints
31         xi >= 0; % Slack variables should be non-negative
32
33 cvx_end
34
35 % Display results
36 fprintf('Optimal weights (w):\n');
37 disp(w);
38 fprintf('Optimal bias (b):\n');
39 disp(b);
40
41 % Plot the data points and decision boundary
42 figure;
43 hold on;
44 gscatter(X(:,1), X(:,2), d);
45 xlim([0 5]);
46 ylim([0 6]);
47
48 % Create a grid for decision boundary
49 x1_range = linspace(0, 5, 100);
50 x2_range = -(w(1)/w(2))*x1_range - b/w(2); % Adjusting for bias
51 plot(x1_range, x2_range, 'k--', 'LineWidth', 2); % Decision
52 % boundary
53
54 xlabel('x_1');
55 ylabel('x_2');
56 title('SVM Decision Boundary');
57 legend('Class -1', 'Class +1', 'Decision Boundary');
58 grid on;
59 hold off;

```

Output of the above code is shown below.

```

-----

num. of constraints = 5
dim. of socp var = 4, num. of socp blk = 1
dim. of linear var = 8
dim. of free var = 1 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT 1 0.000 1 0
it pstep dstep pinfeas dinfeas gap prim-obj dual-obj cputime
-----
number of iterations = 12
primal objective value = 2.50000002e-01
dual objective value = 2.49999998e-01
gap := trace(XZ) = 1.58e-08
relative gap = 1.05e-08
actual relative gap = 3.01e-09
rel. primal infeas (scaled problem) = 1.67e-14
rel. dual " " " = 4.16e-09
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 2.8e+00, 4.3e-01, 2.1e+00
norm(A), norm(b), norm(C) = 9.9e+00, 3.2e+00, 3.1e+00
Total CPU time (secs) = 0.11
CPU time per iteration = 0.01
termination code = 0
DIMACS: 2.7e-14 0.0e+00 6.5e-09 0.0e+00 3.0e-09 1.1e-08
-----

-----

Status: Solved
Optimal value (cvx_optval): +0.25

Optimal weights (w):
0.5000
0.5000

Optimal bias (b):
-2.0000

```

## 2. Get the equation for the classifier

### SOLUTION

Given the optimal weights  $w$  and bias  $b$  from the SVM primal problem, we can formulate the decision boundary (classifier equation).

### Given Output

$$\text{Optimal weights: } w = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\text{Optimal bias: } b = -2$$

## Classifier Equation

The equation of the decision boundary (classifier) for a linear SVM can be expressed as:

$$w^T x + b = 0$$

Substituting the values of  $w$  and  $b$  into the equation:

$$\begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 2 = 0$$

This simplifies to:

$$0.5x_1 + 0.5x_2 - 2 = 0$$

## Rearranging the Equation

To express  $x_2$  in terms of  $x_1$ :

$$0.5x_2 = 2 - 0.5x_1$$

$$x_2 = 4 - x_1$$

Thus, the equation of the classifier (decision boundary) is:

$$x_2 = 4 - x_1$$

## RESULTS

1. Basics of Support Vectors with hard margin is revisited.
2. Both primal and dual problem of the associated Lagrange multiplier model is formulated and solved using `matlab`.

## 6 | Assignment 55

# An Interesting Aspect of Lagrangian Function in Terms of Hessian Matrix at the Optimal Point

### 6.1 Introduction

The Hessian matrix of the Lagrange function plays a crucial role in determining whether a solution to a constrained optimization problem represents a local maximum, minimum, or saddle point. For constrained optimization problems involving **equality** and **inequality constraints**, we need to examine the second-order conditions of optimality using the Hessian of the Lagrangian function.

#### 6.1.1 Optimization Problem Setup

Consider a general constrained optimization problem in the following form:

**Minimization Problem:**

$$\text{Minimize } f(x)$$

Subject to:

$$g_i(x) = 0 \quad \text{for } i = 1, \dots, k \quad (\text{equality constraints}),$$

$$h_j(x) \leq 0 \quad \text{for } j = 1, \dots, l \quad (\text{inequality constraints}).$$

**Maximization Problem:**

$$\text{Maximize } f(x)$$

Subject to:

$$g_i(x) = 0 \quad \text{for } i = 1, \dots, k \quad (\text{equality constraints}),$$

$$h_j(x) \leq 0 \quad \text{for } j = 1, \dots, l \quad (\text{inequality constraints}).$$

The goal is to find the point  $x^*$  that optimizes the objective function  $f(x)$  while satisfying the constraints.

#### 6.1.2 Lagrange Function

To solve such problems, the **Lagrange function** (or Lagrangian) is constructed by introducing Lagrange multipliers  $\mu_i$  for the equality constraints and  $\lambda_j$  for the inequality constraints. The Lagrangian is defined as:

$$\mathcal{L}(x, \mu, \lambda) = f(x) + \sum_{i=1}^k \mu_i g_i(x) + \sum_{j=1}^l \lambda_j h_j(x),$$

where:

- $\mu_i$  are the Lagrange multipliers for equality constraints,
  - $\lambda_j$  are the Lagrange multipliers for inequality constraints.
-

**6.1.3 First-Order Optimality Conditions (Karush-Kuhn-Tucker Conditions)**

For a candidate solution  $x^*$  to be a local optimum, it must satisfy the **Karush-Kuhn-Tucker (KKT) conditions**:

**1. Stationarity:**

$$\nabla_x \mathcal{L}(x^*, \mu^*, \lambda^*) = \nabla f(x^*) + \sum_{i=1}^k \mu_i^* \nabla g_i(x^*) + \sum_{j=1}^l \lambda_j^* \nabla h_j(x^*) = 0.$$

**2. Primal Feasibility:**

$$g_i(x^*) = 0 \quad \forall i, \quad h_j(x^*) \leq 0 \quad \forall j.$$

**3. Dual Feasibility:**

$$\lambda_j^* \geq 0 \quad \forall j.$$

**4. Complementary Slackness:**

$$\lambda_j^* h_j(x^*) = 0 \quad \forall j.$$

This means that if  $h_j(x^*) < 0$ , the corresponding  $\lambda_j^*$  must be zero.

**6.1.4 Second-Order Optimality Conditions**

To determine whether  $x^*$  is a local minimum or maximum, we need to examine the **Hessian matrix** of the Lagrangian function. The Hessian matrix is the matrix of second-order partial derivatives of  $\mathcal{L}(x, \mu, \lambda)$  with respect to  $x$ .

Let  $H(x^*, \mu^*, \lambda^*)$  denote the Hessian matrix of the Lagrange function at the point  $x^*$ . The second-order conditions depend on the nature of this matrix.

- For **minimization problems**,  $x^*$  is a local minimum if the Hessian matrix  $H(x^*, \mu^*, \lambda^*)$  is **positive semi-definite** on the feasible directions that satisfy the constraints.
- For **maximization problems**,  $x^*$  is a local maximum if the Hessian matrix  $H(x^*, \mu^*, \lambda^*)$  is **negative semi-definite** on the feasible directions.

**6.1.5 Hessian Matrix with Inequality Constraints**

In the case of inequality constraints, only the **active constraints** (those for which  $h_j(x^*) = 0$ ) need to be considered in the second-order condition.

The Hessian of the Lagrange function for inequality-constrained optimization problems is defined as:

$$H_{\mathcal{L}}(x^*, \mu^*, \lambda^*) = \nabla^2 f(x^*) + \sum_{i=1}^k \mu_i^* \nabla^2 g_i(x^*) + \sum_{j=1}^l \lambda_j^* \nabla^2 h_j(x^*).$$

- **Minimization:** The Hessian matrix should be **positive semi-definite** when restricted to the subspace defined by the equality and active inequality constraints.
- **Maximization:** The Hessian matrix should be **negative semi-definite** in the same subspace.

### 6.1.6 Example: Quadratic Minimization with Inequality Constraints

Consider the following quadratic minimization problem with an inequality constraint:

$$\text{Minimize } f(x) = x_1^2 + x_2^2,$$

Subject to:

$$h_1(x) = x_1 + x_2 - 1 \leq 0.$$

The Lagrangian is given by:

$$\mathcal{L}(x, \lambda_1) = x_1^2 + x_2^2 + \lambda_1(x_1 + x_2 - 1).$$

The Hessian of the Lagrangian is:

$$H_{\mathcal{L}} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

Since this matrix is positive definite, the second-order condition confirms that the solution is a local minimum if the first-order conditions (KKT conditions) are also satisfied.

## 6.2 Takeaway

The Hessian matrix of the Lagrange function provides valuable insight into the nature of the optimal solution in constrained optimization problems. It helps confirm whether a candidate solution is a local maximum, minimum, or saddle point by analyzing the curvature of the Lagrange function in feasible directions. For inequality-constrained problems, the second-order conditions are verified using the Hessian restricted to the subspace determined by the active constraints.

By incorporating both first-order (KKT) and second-order conditions, we can rigorously identify and classify the optimal solutions in both minimization and maximization problems with constraints.

## 6.3 Task

Solve the following constrained optimization problems using *Lagrange multiplier method*.

1.  $\min x_1^2 + x_2^2$  subject to  $x_1 + x_2 \geq 2$

### SOLUTION

Consider the given constrained optimization problem:

$$\min f(x_1, x_2) = x_1^2 + x_2^2$$

subject to the inequality constraint:

$$g(x_1, x_2) = x_1 + x_2 - 2 \geq 0.$$

The Lagrangian function for this problem is given by:

$$\mathcal{L}(x_1, x_2, \lambda) = x_1^2 + x_2^2 + \lambda(2 - x_1 - x_2),$$

where  $\lambda \geq 0$  is the Lagrange multiplier associated with the inequality constraint.

To find the critical points, we take the first-order partial derivatives of the Lagrangian and set them to zero:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 2x_1 - \lambda = 0,$$

$$\frac{\partial \mathcal{L}}{\partial x_2} = 2x_2 - \lambda = 0,$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 2 - x_1 - x_2 = 0.$$

From the first two equations, we find:

$$\lambda = 2x_1, \quad \lambda = 2x_2 \implies 2x_1 = 2x_2 \implies x_1 = x_2.$$

Substituting into the constraint  $x_1 + x_2 = 2$ :

$$2x_1 = 2 \implies x_1 = x_2 = 1.$$

Thus, the critical point is  $(x_1, x_2) = (1, 1)$ .

The Hessian matrix of the Lagrangian function is defined by the second-order partial derivatives:

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial x_1^2} &= 2, & \frac{\partial^2 \mathcal{L}}{\partial x_2^2} &= 2, \\ \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial x_2} &= 0, & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial \lambda} &= -1, & \frac{\partial^2 \mathcal{L}}{\partial x_2 \partial \lambda} &= -1, \\ \frac{\partial^2 \mathcal{L}}{\partial \lambda^2} &= 0. \end{aligned}$$

Thus, the Hessian matrix  $H$  is:

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

The eigenvalues of the Hessian matrix can be found by solving the characteristic equation:

$$\det(H - \lambda I) = 0,$$

where  $I$  is the identity matrix. Substituting  $H$  and  $I$ :

$$H - \lambda I = \begin{bmatrix} 2 - \lambda & 0 & -1 \\ 0 & 2 - \lambda & -1 \\ -1 & -1 & -\lambda \end{bmatrix}.$$

The determinant of this matrix is:

$$\det(H - \lambda I) = (2 - \lambda) [(-\lambda)(2 - \lambda) - (-1)(-1)] = (2 - \lambda) [\lambda^2 - 2\lambda - 1].$$

Setting the characteristic equation to zero:

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

From  $2 - \lambda = 0$ , we obtain:



$$\lambda_1 = 2.$$

From  $\lambda^2 - 2\lambda - 1 = 0$ , we apply the quadratic formula:

$$\lambda = \frac{2 \pm \sqrt{4+4}}{2} = \frac{2 \pm \sqrt{8}}{2} = 1 \pm \sqrt{2}.$$

Thus, the eigenvalues are:

$$- \lambda_1 = 2, - \lambda_2 = 1 + \sqrt{2} > 0, - \lambda_3 = 1 - \sqrt{2} < 0.$$

The eigenvalues of the Hessian matrix indicate the nature of the critical point  $(x_1, x_2) = (1, 1)$ :

$$- \lambda_1 > 0 \text{ (positive)}, - \lambda_2 > 0 \text{ (positive)}, - \lambda_3 < 0 \text{ (negative)}.$$

Since we have both positive and negative eigenvalues, the Hessian matrix is **indefinite**. This suggests that the critical point is a **saddle point** rather than a minimum or maximum.

The eigenvalue analysis of the Hessian matrix reveals that the critical point  $(x_1, x_2) = (1, 1)$  is a **saddle point** for the given constrained optimization problem.

2. Solve  $\text{Max } f(x) = e^{-x}$ ; Subject to  $X \geq 1$ .

### SOLUTION

The given constrained optimization problem is:

$$\max f(x) = e^{-x}$$

subject to the constraint

$$g(x) = x - 1 \geq 0,$$

we will use the method of Lagrange multipliers. The Lagrangian function is defined as:

$$\mathcal{L}(x, \lambda) = e^{-x} + \lambda(x - 1),$$

where  $\lambda$  is the Lagrange multiplier.

We find the first-order partial derivatives of the Lagrangian:

$$\frac{\partial \mathcal{L}}{\partial x} = -e^{-x} + \lambda = 0,$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = x - 1 = 0.$$

From the second equation, we have:

$$x = 1.$$

Substituting  $x = 1$  into the first equation gives:

$$-e^{-1} + \lambda = 0 \implies \lambda = e^{-1}.$$

We check if this solution satisfies the constraint  $x \geq 1$ . Since  $x = 1$  satisfies the constraint, it is a candidate for the optimal solution.

Now we evaluate the objective function at the critical point:

$$f(1) = e^{-1} \approx 0.3679.$$

Since this is a maximization problem with a constraint  $x \geq 1$ , we also need to check the behavior of the function as  $x$  increases. As  $x \rightarrow \infty$ ,

$$f(x) = e^{-x} \rightarrow 0.$$

The Hessian matrix of the Lagrangian is defined as:

$$H(x, \lambda) = \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial x^2} & \frac{\partial^2 \mathcal{L}}{\partial x \partial \lambda} \\ \frac{\partial^2 \mathcal{L}}{\partial \lambda \partial x} & \frac{\partial^2 \mathcal{L}}{\partial \lambda^2} \end{bmatrix}.$$

Calculating the components, we find:

1. The second derivative with respect to  $x$ :

$$\frac{\partial^2 \mathcal{L}}{\partial x^2} = e^{-x}.$$

2. The mixed derivatives are zero since  $\lambda$  does not depend on  $x$ :

$$\frac{\partial^2 \mathcal{L}}{\partial x \partial \lambda} = 1, \quad \frac{\partial^2 \mathcal{L}}{\partial \lambda \partial x} = 1.$$

3. The second derivative with respect to  $\lambda$ :

$$\frac{\partial^2 \mathcal{L}}{\partial \lambda^2} = 0.$$

Thus, the Hessian matrix becomes:

$$H(x, \lambda) = \begin{bmatrix} e^{-x} & 1 \\ 1 & 0 \end{bmatrix}.$$

At  $x = 1$ , the Hessian evaluates to:

$$H(1, e^{-1}) = \begin{bmatrix} e^{-1} & 1 \\ 1 & 0 \end{bmatrix}.$$

To find the eigenvalues, we solve the characteristic polynomial:

$$\det(H - \lambda I) = 0 \implies \det \begin{bmatrix} e^{-1} - \lambda & 1 \\ 1 & -\lambda \end{bmatrix} = 0.$$

This results in:

$$(e^{-1} - \lambda)(-\lambda) - 1 = 0 \implies \lambda^2 - e^{-1}\lambda - 1 = 0.$$

Using the quadratic formula:

$$\lambda = \frac{e^{-1} \pm \sqrt{(e^{-1})^2 + 4}}{2}.$$

Calculating the discriminant:

$$(e^{-1})^2 + 4 = e^{-2} + 4 = \frac{1 + 4e^2}{e^2}.$$

The eigenvalues are given by:

$$\lambda_1 = \frac{e^{-1} + \sqrt{(e^{-1})^2 + 4}}{2}, \quad \lambda_2 = \frac{e^{-1} - \sqrt{(e^{-1})^2 + 4}}{2}.$$

To confirm whether the critical point is a local maximum, we analyze the eigenvalues:

- If both eigenvalues are negative, the critical point is a local maximum.
- If one or both eigenvalues are positive, the critical point is a local minimum.
- If the eigen values are of different sign, the critical point is a saddle point.

Given that  $e^{-1} > 0$  and  $\sqrt{(e^{-1})^2 + 4} > e^{-1}$ , it can be shown that  $\lambda_1 > 0$  and  $\lambda_2 < 0$ .

Since the eigenvalues have mixed signs, the critical point is a saddle point.

3. Minimize  $f(x_1, x_2) = x_1^2 + x_2^2 - 3x_1x_2$  subject to:  $2x_1 + x_2 \geq 2$ .

#### SOLUTION

Given constrained optimization problem is :

Minimize  $f(x_1, x_2) = x_1^2 + x_2^2 - 3x_1x_2$  subject to  $2x_1 + x_2 \geq 2$ .

The Lagrangian function is given by:

$$L(x_1, x_2, \lambda) = x_1^2 + x_2^2 - 3x_1x_2 - \lambda(2x_1 + x_2 - 2)$$

#### First Order Conditions

The first order conditions for a minimum are given by:

$$\frac{\partial L}{\partial x_1} = 2x_1 - 3x_2 - 2\lambda = 0$$

$$\frac{\partial L}{\partial x_2} = 2x_2 - 3x_1 - \lambda = 0$$

$$\frac{\partial L}{\partial \lambda} = -(2x_1 + x_2 - 2) \geq 0$$

$$\lambda(2x_1 + x_2 - 2) = 0$$

#### Solve the System of Equations

From the first two equations, we can express them as:

$$2x_1 - 3x_2 - 2\lambda = 0$$

$$2x_2 - 3x_1 - \lambda = 0$$

Rearranging these equations gives:

$$\lambda = \frac{2x_1 - 3x_2}{2} \quad (1)$$

$$\lambda = 2x_2 - 3x_1 \quad (2)$$

Equating (1) and (2):

$$\frac{2x_1 - 3x_2}{2} = 2x_2 - 3x_1$$

Multiplying through by 2 to eliminate the fraction:

$$2x_1 - 3x_2 = 4x_2 - 6x_1$$

$$8x_1 - 7x_2 = 0$$

Thus, we have:

$$8x_1 = 7x_2 \quad (3)$$

Also, from the constraint  $2x_1 + x_2 = 2$ :

$$2x_1 + x_2 = 2 \quad (4)$$

Now we can solve the equations (3) and (4) together. There for the solution is  $x_1 = \frac{28}{22}$ ,  $x_2 = -\frac{12}{22}$

### Calculate the Hessian

The Hessian matrix  $H$  consists of the second partial derivatives of  $L$ :

$$H(x_1, x_2, \lambda) = \begin{bmatrix} \frac{\partial^2 L}{\partial x_1^2} & \frac{\partial^2 L}{\partial x_1 \partial x_2} & \frac{\partial^2 L}{\partial x_1 \partial \lambda} \\ \frac{\partial^2 L}{\partial x_2 \partial x_1} & \frac{\partial^2 L}{\partial x_2^2} & \frac{\partial^2 L}{\partial x_2 \partial \lambda} \\ \frac{\partial^2 L}{\partial \lambda \partial x_1} & \frac{\partial^2 L}{\partial \lambda \partial x_2} & \frac{\partial^2 L}{\partial \lambda^2} \end{bmatrix} = \begin{bmatrix} 2 & -3 & -2 \\ -3 & 2 & -1 \\ -2 & -1 & 0 \end{bmatrix}$$

### Evaluate the Hessian

To analyze the nature of the critical point, we will compute the eigenvalues of the Hessian matrix  $H$ :

$$H - \lambda I = \begin{bmatrix} 2 - \lambda & -3 & 2 \\ -3 & 2 - \lambda & 1 \\ 2 & 1 & -\lambda \end{bmatrix}$$

Eigen values of  $H$  are 1.59, -2.70, 5.11. So the Hessian matrix is indefinite.

To confirm the nature of the critical point at  $(x_1, x_2) = (\frac{28}{22}, -\frac{12}{22})$ , the eigenvalues have mixed signs,  $H$  is indefinite, indicating a saddle point.

4. Minimize  $f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2$  subject to  $x_1 + x_2 \geq 2$ .

### SOLUTION

The Lagrangian function for the given problem is defined as:

$$L(x_1, x_2, \lambda) = x_1^2 + x_2^2 - x_1 x_2 - \lambda(x_1 + x_2 - 2)$$

### First Order Conditions

The first-order conditions for a minimum are given by setting the partial derivatives to zero:

$$\frac{\partial L}{\partial x_1} = 2x_1 - x_2 - \lambda = 0$$

$$\frac{\partial L}{\partial x_2} = 2x_2 - x_1 - \lambda = 0$$

$$\frac{\partial L}{\partial \lambda} = 2 - x_1 - x_2 \geq 0$$

$$\lambda(2 - x_1 - x_2) = 0$$

### Solve the System of Equations

From the first two equations, we can express  $\lambda$  in terms of  $x_1$  and  $x_2$ : 1.  $2x_1 - x_2 - \lambda = 0$  (1) 2.  $2x_2 - x_1 - \lambda = 0$  (2)

Rearranging these equations gives:

$$\lambda = 2x_1 - x_2 \quad (3)$$

$$\lambda = 2x_2 - x_1 \quad (4)$$

Setting (3) equal to (4):

$$2x_1 - x_2 = 2x_2 - x_1$$

Rearranging gives:

$$3x_1 - 3x_2 = 0 \implies x_1 = x_2$$

Substituting  $x_2 = x_1$  into the constraint  $x_1 + x_2 \geq 2$ :

$$2x_1 = 2 \implies x_1 = 1 \quad \text{and thus} \quad x_2 = 1$$

Therefore  $\lambda = 1$ .

### Calculate the Hessian

The Hessian matrix  $H$  of the Lagrangian function  $L$  is given by:

$$H(x_1, x_2, \lambda) = \begin{bmatrix} \frac{\partial^2 L}{\partial x_1^2} & \frac{\partial^2 L}{\partial x_1 \partial x_2} & \frac{\partial^2 L}{\partial x_1 \partial \lambda} \\ \frac{\partial^2 L}{\partial x_2 \partial x_1} & \frac{\partial^2 L}{\partial x_2^2} & \frac{\partial^2 L}{\partial x_2 \partial \lambda} \\ \frac{\partial^2 L}{\partial \lambda \partial x_1} & \frac{\partial^2 L}{\partial \lambda \partial x_2} & \frac{\partial^2 L}{\partial \lambda^2} \end{bmatrix} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

### Eigenvalues of Hessian

To determine the nature of the critical point, we compute the eigenvalues of the Hessian:

Thus, the eigenvalues are:

$$\lambda = -1, 2, 3$$

Since  $H$  is indefinite, the critical point  $(1, 1)$  is a saddle point.

## RESULTS

1. Lagrange multiplier method is revisited.
2. Nature of Hessian matrix is used to determine whether a critical point is a local maxima, local minima or a saddle point.
3. These concepts are used in the given constrained optimization problem.



# 7 | Using CVX for Solving LP Problems

## 7.1 Introduction

Linear Programming (LP) is a powerful mathematical technique used for optimizing a linear objective function, subject to linear equality and inequality constraints. It has widespread applications in various fields, including economics, engineering, logistics, and operations research.

## 7.2 Historical Background of Linear Programming

The origins of linear programming date back to the early 20th century. Some key milestones in the development of LP include:

- **1940s:** The foundation of linear programming was laid during World War II, primarily to optimize military logistics and resource allocation. In 1947, George B. Dantzig formulated the Simplex Method, a groundbreaking algorithm that efficiently solves LP problems. This method allowed for the determination of optimal solutions by traversing the vertices of the feasible region defined by constraints.
- **1950s:** The Simplex Method gained significant attention and was widely adopted in various industries for solving complex optimization problems. Researchers and practitioners began exploring its applications in operations research and economic planning.
- **1970s:** The advent of computer technology revolutionized the field of optimization. Algorithms for solving LP problems were implemented in software, making it accessible to a broader audience. The introduction of interior-point methods provided alternative approaches to solving LP problems, complementing the Simplex Method.
- **1980s and Beyond:** The field of linear programming continued to grow, with advancements in algorithms, theory, and software tools. LP has become a standard tool for decision-making in various domains, such as transportation, finance, manufacturing, and telecommunications.

## 7.3 Overview of CVX in MATLAB

CVX is a popular modeling framework for convex optimization in MATLAB. It allows users to formulate and solve linear programming (LP) problems easily, without requiring an in-depth understanding of optimization algorithms. The simplicity of CVX makes it a valuable tool for researchers, engineers, and students.

### 7.3.1 Key Features of CVX

- **User-Friendly Syntax:** CVX employs a high-level syntax that mimics mathematical notation. This allows users to express optimization problems intuitively and concisely.
-

- **Integration with MATLAB:** CVX seamlessly integrates with MATLAB, leveraging its numerical computing capabilities. Users can utilize MATLAB's built-in functions and visualization tools alongside their optimization tasks.
- **Support for Various Problem Types:** In addition to LP, CVX supports a wide range of convex optimization problems, including quadratic programming, semidefinite programming, and more.
- **Built-in Solvers:** CVX automatically selects appropriate solvers based on the problem formulation, providing users with flexibility in choosing solvers that best suit their needs.
- **Extensive Documentation and Community:** CVX comes with comprehensive documentation and an active user community, making it easy for users to find resources, tutorials, and examples.

## 7.4 Example of Using CVX for LP Problems

Here is a simple example of how to use CVX to solve a linear programming problem:

### 7.4.1 Problem Statement

Minimize the objective function:

$$z = 2x_1 + 3x_2$$

subject to the constraints:

$$x_1 + 2x_2 \geq 10$$

$$3x_1 + x_2 \leq 12$$

$$x_1, x_2 \geq 0$$

### 7.4.2 MATLAB Code Using CVX

```
1 cvx_begin quiet
2     variable x(2) % Define decision variables
3     minimize(2*x(1) + 3*x(2)) % Objective function
4     subject to
5         x(1) + 2*x(2) >= 10; % Constraint 1
6         3*x(1) + x(2) <= 12; % Constraint 2
7         x >= 0; % Non-negativity constraints
8 cvx_end
9
10 % Display results
11 disp('Optimal solution:');
12 disp(['x1 = ', num2str(x(1))]);
13 disp(['x2 = ', num2str(x(2))]);
14 disp(['Optimal value of objective function = ', num2str(cvx_optval)
15     ]);
```

Output of the above code is shown below.

```
Calling SDPT3 4.0: 4 variables, 2 equality constraints
```

```
-----
```

```
num. of constraints = 2
dim. of linear var  = 4
```



```

*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations      = 7
primal objective value    = 1.50000000e+01
dual  objective value     = 1.50000000e+01
gap := trace(XZ)          = 7.34e-09
relative gap              = 2.37e-10
actual relative gap       = 1.29e-10
rel. primal infeas (scaled problem) = 5.76e-14
rel. dual      "          "          " = 1.21e-10
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual      "          "          " = 0.00e+00
norm(X), norm(y), norm(Z) = 8.6e+00, 1.5e+00, 1.6e+00
norm(A), norm(b), norm(C) = 5.1e+00, 1.7e+01, 4.6e+00
Total CPU time (secs)    = 0.05
CPU time per iteration   = 0.01
termination code         = 0
DIMACS: 7.4e-14  0.0e+00  1.4e-10  0.0e+00  1.3e-10  2.4e-10
-----

-----
Status: Solved
Optimal value (cvx_optval): +15

Optimal solution:
x1 = 3.3731e-09
x2 = 5
Optimal value of objective function = 15

```

An alternative matrix approach is also possible with CVX. This approach is demonstrated using the following example.

#### 1. Solve the LPP

$$\max_x f(x_1, x_2, x_3, x_4) = 5x_1 + 6x_2 + 9x_3 + 8x_4$$

Subject to:

$$x_1 + 2x_2 + 3x_3 + x_4 \leq 5$$

$$x_1 + x_2 + 2x_3 + 3x_4 \leq 3$$

$$x_i \geq 0$$

$$i = 1, 2, 3, 4$$

#### SOLUTION

The given problem can be written in the form

$$\min_x c^T X$$

S.to

$$AX \leq b$$

$$X \geq 0$$

Matlab code to solve this problem is given below.

```

1 m = 2; % Matrix A is 2x4
2 n = 4;
3 A = [1 2 3 1; 1 1 2 3];
4 b = [5 3]';
5 c = [ 5 6 9 8]';
6 cvx_begin quiet
7 variables x(n)
8 maximize(c' * x)
9 subject to
10 A * x <= b;
11 x>=0;
12 cvx_end
13
14 disp(x)

```

Output of the above code is:

```

Calling SDPT3 4.0: 6 variables, 2 equality constraints
-----

num. of constraints = 2
dim. of linear var = 6
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT      1      0.000 1      0
it pstep dstep pinfeas dinfeas gap      prim-obj      dual-obj      cputime
-----

number of iterations = 8
primal objective value = -1.70000000e+01
dual objective value = -1.70000000e+01
gap := trace(XZ) = 6.78e-08
relative gap = 1.94e-09
actual relative gap = 1.92e-09
rel. primal infeas (scaled problem) = 1.30e-12
rel. dual " " " = 2.89e-11
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 2.2e+00, 4.1e+00, 6.8e+00
norm(A), norm(b), norm(C) = 6.7e+00, 6.8e+00, 1.5e+01
Total CPU time (secs) = 0.05
CPU time per iteration = 0.01
termination code = 0
DIMACS: 1.5e-12 0.0e+00 4.4e-11 0.0e+00 1.9e-09 1.9e-09
-----

Status: Solved
Optimal value (cvx_optval): +17

1.0000
2.0000
0.0000
0.0000

```

2. Solve the dual of the above problem.

**SOLUTION**

Dual of the previous problem is

$$\min f(y_1, y_2) = 5y_1 + 3y_2$$

Subject to:

$$y_1 + y_2 \geq 5$$

$$2y_1 + y_2 \geq 6$$

$$3y_1 + 2y_2 \geq 9$$

$$y_1 + 3y_2 \geq 8$$

$$y_1, y_2 \geq 0$$

Matlab code to solve this dual problem is given below.

```
1 m = 2; % Matrix A is 2x4
2 n = 4;
3 A = [1 2 3 1; 1 1 2 3];
4 b = [5 3]';
5 c = [ 5 6 9 8]';
6 cvx_begin
7 variables x(n)
8 maximize(c' * x)
9 subject to
10 A * x <= b;
11 x >= 0;
12 cvx_end
13
14 disp(x)
15
16 m = 4; % matrix AT is 4x2
17 n = 2;
18 AT = [1 2 3 1; 1 1 2 3]';
19 b = [5 3]';
20 c = [ 5 6 9 8]';
21 cvx_begin quiet
22     variables y(n)
23     minimize(b' * y)
24     subject to
25         AT * y >= c;
26         y >= 0;
27 cvx_end
28
29 disp(x)
```

Output of the above code is shown below.

```
Calling SDPT3 4.0: 6 variables, 2 equality constraints
For improved efficiency, SDPT3 is solving the dual problem.
-----
```

```
num. of constraints = 2
```

```

dim. of linear var = 6
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations = 8
primal objective value = -1.70000000e+01
dual  objective value = -1.70000000e+01
gap := trace(XZ)      = 6.78e-08
relative gap          = 1.94e-09
actual relative gap   = 1.92e-09
rel. primal infeas (scaled problem) = 1.30e-12
rel. dual    "        "        "    = 2.89e-11
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual    "        "        "    = 0.00e+00
norm(X), norm(y), norm(Z) = 2.2e+00, 4.1e+00, 6.8e+00
norm(A), norm(b), norm(C) = 6.7e+00, 6.8e+00, 1.5e+01
Total CPU time (secs) = 0.04
CPU time per iteration = 0.01
termination code      = 0
DIMACS: 1.5e-12  0.0e+00  4.4e-11  0.0e+00  1.9e-09  1.9e-09
-----

-----
Status: Solved
Optimal value (cvx_optval): +17

1.0000
2.0000
0.0000
0.0000

```

3. Solve Min  $f(x, y) = 3x + 9y$  Subject to  $x + 3y \leq 60; x + y \geq 10; x, y \geq 0$ .

#### SOLUTION

Matlab code to solve this system of equation is given below.

```

1 m = 2;
2 n = 2;
3 A = [1 3; -1 -1];
4 b = [60 -10]';
5 c = [3 9]';
6
7 % cvx_solver sedumi
8 cvx_begin quiet
9     variables x(n)
10    minimize(c' * x)
11    subject to
12        A * x <= b
13 cvx_end
14
15 disp(x)

```

Output of the above code is shown below.

```

Calling SDPT3 4.0: 2 variables, 0 equality constraints
-----

num. of constraints = 1
dim. of linear var = 3
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations   = 5
residual of dual infeasibility
certificate X          = 1.33e-14
reldist to infeas.    <= 3.98e-14
Total CPU time (secs) = 0.15
CPU time per iteration = 0.03
termination code       = 2
DIMACS: 1.6e-05  0.0e+00  1.1e+00  0.0e+00  -1.0e+00  9.4e-05
-----

-----

Status: Unbounded
Optimal value (cvx_optval): -Inf

      0.1667
     -0.1667

```

The problem has an unbounded solution.

4. Solve Max  $f(x, y) = 3x + 9y$  Subject to  $x + 3y \leq 60$ ;  $x + y \geq 10$ ;  $x, y \geq 0$ .

#### SOLUTION

Matlab code for solving the given problem is shown below.

```

1 m = 2;
2 n = 2;
3 A = [1 3; -1 -1];
4 b = [60 -10]';
5 c = [3 9]';
6
7 % cvx_solver sedumi
8 cvx_begin quiet
9     variables x(n)
10    maximize(c' * x)
11    subject to
12        A * x <= b
13 cvx_end
14
15 disp(x)

```

Output of the above code is shown below.

```

Calling SDPT3 4.0: 2 variables, 0 equality constraints
-----

```

```
num. of constraints = 1
dim. of linear var = 3
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations   = 7
primal objective value = 1.82505469e-09
dual  objective value = -6.25894135e-10
gap := trace(XZ)       = 9.13e-09
relative gap           = 9.13e-09
actual relative gap    = 2.45e-09
rel. primal infeas (scaled problem) = 3.76e-14
rel. dual      "      "      "      = 5.22e-10
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual      "      "      "      = 0.00e+00
norm(X), norm(y), norm(Z) = 4.7e+00, 6.3e-10, 3.0e+00
norm(A), norm(b), norm(C) = 2.0e+00, 2.0e+00, 4.0e+00
Total CPU time (secs) = 0.05
CPU time per iteration = 0.01
termination code      = 0
DIMACS: 3.8e-14  0.0e+00  5.2e-10  0.0e+00  2.5e-09  9.1e-09
-----

-----

Status: Solved
Optimal value (cvx_optval): +180

-8.1858
22.7286
```

## RESULTS

1. Backgrounds of development of Linear Programming Problem is revisited.
2. CVX library is used to solve the given Linear Programming Problems.

## 8 | Assignment 56

# Loss and Penalty Function

### 8.1 Introduction

Loss function is one of the most important keyword in Statistical Estimation theory , decision science and in Machine learning.

To understand what it is, let us take an example from quality control. (Most probably, the keyword must have come from this domain). Every item produced in a factory must satisfy the given specification. Any deviation from the target is assumed to be associated with a loss (a societal loss- loss to the manufacturer and customer). This loss is generally assumed to be proportional to:

- absolute value of deviation
- square of the deviation for up to some range of deviation, loss is quadratic and further it is linear (huber loss function).
- for up to some range of deviation, there is no loss and further it is linear (epsilon-insensitive loss function)

#### 8.1.1 Role of loss function in quality control- a real story

Sony in 1970s had two divisions -one in Japan and the other in USA-for producing electronic components. Sony japan followed quadratic loss function for their quality control operations. They aimed always at producing items at target value. Any deviation is considered as loss of quality. But the Sony-USA followed the first model. In the long run, what happened is that, Sony japan survived all market competition and thrived in the world market but Sony USA had to closed down. Since then, the word “Goal-Post” syndrome came into existence in quality control parlour because quality control departments of many companies were under the impression that their job was to weed out the defective components rather than improving the process of manufacturing to reduce the quality loss. So wrong quality philosophy in terms of loss function has led to closure of a company.

Another term associated with estimation theory is penalization function. An equivalent term for this is regularization function.

We will try to understand the difference between these concepts through examples.

Let  $X$  be a feature matrix of size  $m \times n$  and  $Y$  be target vector of size  $m \times 1$  for regression. For example.

$$X = \begin{bmatrix} 1 & 2 \\ 4 & -5 \\ -7 & 8 \\ -9 & 11 \end{bmatrix}$$
$$Y = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 7 \end{bmatrix}$$

---

Here  $m = 4, n = 2$ . We are trying to fit a plane of the form  $y = \beta_1 x_1 + \beta_2 x_2 + \gamma$ . So we have 3 (that is  $n + 1$ ) model parameters to be estimated.

### 8.1.2 Loss function

Then Loss function is a mapping from  $L: \mathbb{R}^m \rightarrow \mathbb{R}_+$ . We have  $m$  data points and total loss for this  $m$  data points together is a positive number. Hence the above mapping. If we are trying for linear least square regression estimation, our loss function is quadratic because we are trying to adjust model parameters so as to have least-square error (least square deviation from target values).

### 8.1.3 Penalized function

Penalization function is a mapping  $P: \mathbb{R}^{n+1} \rightarrow \mathbb{R}_+$  so that we find  $\beta \in \mathbb{R}^n$  and  $\gamma \in \mathbb{R}$  that minimize

$$L(Y - [\beta X + \gamma I]) + P(\beta, \gamma)$$

In many applications, we will try to control the model parameters. One main reason for this is to reduce over fitting of the data to the model. We may put the condition like its individual value or squared length of the vector must be small or its value must be positive etc.

### 8.1.4 Quadratic loss function

As an example, for the above data, find regression parameters that solve

$$\min_{\beta, \gamma} \|Y - X\beta - \gamma I\|_2^2$$

For this formulation, we are applying quadratic loss function. There is no penalty function.

### 8.1.5 Task 1

*Solve above regression problem using CVX. Verify the result using pseudo inverse.*

#### SOLUTION

Matlab code for this task is given below.

```
1 % Data
2 x = [1 4 -7 9; 2 -5 8 11]'; % 4x2 matrix
3 y = [1 3 2 7]'; % 4x1 vector
4
5 % Plot data points
6 figure;
7 plot3(x(:,1), x(:,2), y, 'o');
8 hold on;
9 xlabel('x1');
10 ylabel('x2');
11 zlabel('y');
12
13 % CVX linear regression
14 cvx_begin quiet
15     variables m(2) c;
16     minimize( norm(y - x*m - c*ones(4,1)) ) % Linear model: y = x1*
        m1 + x2*m2 + c
17 cvx_end
18
19 % Display the results
20 disp('Estimated slope m:');
```



## 8. Assignment 56

### Loss and Penalty Function

---

```
21 disp(m);
22 disp('Estimated intercept c:');
23 disp(c);
24
25 % Plot the fitted plane
26 [X1, X2] = meshgrid(min(x(:,1)):1:max(x(:,1)), min(x(:,2)):1:max(x
    (:,2)));
27 Y_fit = m(1)*X1 + m(2)*X2 + c;
28 mesh(X1, X2, Y_fit);
29 hold on;
30
31 plot3(x(:,1), x(:,2), y, 'o');
32 grid on;
```

Output of the above code is shown below.

```
Calling SDPT3 4.0: 5 variables, 4 equality constraints
  For improved efficiency, SDPT3 is solving the dual problem.
-----

num. of constraints = 4
dim. of socp var = 5, num. of socp blk = 1
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0

it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations   = 6
primal objective value = -1.93824332e+00
dual  objective value = -1.93824332e+00
gap := trace(XZ)       = 1.28e-08
relative gap           = 2.62e-09
actual relative gap    = 1.19e-09
rel. primal infeas (scaled problem) = 6.77e-13
rel. dual    "         "         "   = 7.81e-10
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual    "         "         "   = 0.00e+00
norm(X), norm(y), norm(Z) = 1.4e+00, 2.8e+00, 2.7e+00
norm(A), norm(b), norm(C) = 2.0e+01, 2.0e+00, 8.9e+00
Total CPU time (secs) = 0.06
CPU time per iteration = 0.01
termination code       = 0
DIMACS: 6.8e-13  0.0e+00  8.7e-10  0.0e+00  1.2e-09  2.6e-09
-----

-----
Status: Solved
Optimal value (cvx_optval): +1.93824

Estimated slope m:

    0.2956
    0.1926
Estimated intercept c:
    1.9624
```

Solution of the system  $y = \beta_1 x_1 + \beta_2 x_2 + \gamma$  can also be found by using the `pinv()` function. Matlab code for this task is shown below.

```
1 x = [1 4 -7 9; 2 -5 8 11]';
2 y = [1 3 2 7]';
3 X = [x, ones(4,1)];
4 beta = pinv(X) * y;
5 m = beta(1:2);
6 c = beta(3);
7 disp('Estimated slopes (m1, m2):');
8 disp(m);
9 disp('Estimated intercept c:');
10 disp(c);
```

Output of above code is shown below.

```
Estimated slopes (m1, m2):
    0.2956
    0.1926
Estimated intercept c:
    1.9624
```

### 8.1.6 Non-negative least square problem

For a linear regression problem,

$$Y = X\beta + \gamma I,$$

the non-negative least square problem can be defined as

$$\min_{\beta, \gamma} \|Y - X\beta - \gamma I\|_2^2; \quad \text{Subject to } \beta \geq 0$$

We assumed a model of the form

$$y_i = \beta_1 x_{1i} + \beta_2 x_{2i} + \gamma + \epsilon_i$$

where  $\epsilon_i$  is the error term.

### 8.1.7 Task 2

*Solve the above linear regression problem as a non-negative least square problem using Matlab.*

#### SOLUTION

Matlab code for this task is given below.

```
1 x = [1 4 -7 9; 2 -5 8 11]';
2 y = [1 3 2 7]';
3 cvx_begin
4     variables m(2) c;
5     minimize( norm(y - (x(:,1)*m(1) + x(:,2)*m(2) + c)) )
6     subject to
7         m >= 0;
8         c >= 0;
9 cvx_end
10 disp('Estimated slopes (m1, m2):');
11 disp(m);
12 disp('Estimated intercept c:');
13 disp(c);
```

Output of the above code is shown below.

## 8. Assignment 56

### Loss and Penalty Function

---

Calling SDPT3 4.0: 8 variables, 4 equality constraints

```
-----
num. of constraints = 4
dim. of socp var = 5, num. of socp blk = 1
dim. of linear var = 3
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas gap      prim-obj      dual-obj      cputime
-----

number of iterations = 9
primal objective value = 1.93824334e+00
dual objective value = 1.93824326e+00
gap := trace(XZ) = 7.25e-08
relative gap = 1.49e-08
actual relative gap = 1.49e-08
rel. primal infeas (scaled problem) = 1.86e-12
rel. dual " " " = 3.50e-12
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 3.4e+00, 1.0e+00, 1.4e+00
norm(A), norm(b), norm(C) = 2.0e+01, 8.9e+00, 2.0e+00
Total CPU time (secs) = 0.10
CPU time per iteration = 0.01
termination code = 0
DIMACS: 2.1e-12 0.0e+00 3.5e-12 0.0e+00 1.5e-08 1.5e-08
-----

-----
Status: Solved
Optimal value (cvx_optval): +1.93824
Estimated slopes (m1, m2):
    0.2956
    0.1926
Estimated intercept c:
    1.9624
```

#### 8.1.8 LASSO regression

Least Absolute Shrinkage Selection Operator is a regularization approach which involves the least square loss function with L1 penalized term. Mathematically LASSO regression is defined as:

$$\min_{\beta, \gamma} \|Y - X\beta - \gamma I\|_2^2 + \lambda \|\beta\|_1; \quad \lambda = 1$$

Here, loss function is quadratic and penalty function is L1 norm.

#### 8.1.9 Task 3

*Solve the above regression problem using LASSO*

##### **SOLUTION**

Matlab code for this task is given below.

```
1 x = [1 4 -7 9; 2 -5 8 11]';
2 y = [1 3 2 7]';
```

## 8. Assignment 56

### Loss and Penalty Function

---

```
3 lambda = 0.1;
4 cvx_begin quiet
5     variables m(2) c;
6     minimize( norm(y - (x(:,1)*m(1) + x(:,2)*m(2) + c), 2) + lambda
              * (norm(m, 1) + abs(c)) )
7 cvx_end
8
9 disp('Estimated slopes (m1, m2):');
10 disp(m);
11 disp('Estimated intercept c:');
12 disp(c);
```

Output of above code is shown below.

```
Calling SDPT3 4.0: 11 variables, 4 equality constraints
-----

num. of constraints = 4
dim. of socp var = 11, num. of socp blk = 4
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations   = 7
primal objective value = 2.18024848e+00
dual  objective value = 2.18024845e+00
gap := trace(XZ)       = 3.53e-08
relative gap           = 6.59e-09
actual relative gap    = 5.58e-09
rel. primal infeas (scaled problem) = 1.72e-12
rel. dual    "         "         "   = 1.24e-09
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual    "         "         "   = 0.00e+00
norm(X), norm(y), norm(Z) = 3.9e+00, 1.0e+00, 1.4e+00
norm(A), norm(b), norm(C) = 2.0e+01, 8.9e+00, 2.0e+00
Total CPU time (secs) = 0.07
CPU time per iteration = 0.01
termination code      = 0
DIMACS: 1.9e-12  0.0e+00  1.2e-09  0.0e+00  5.6e-09  6.6e-09
-----

-----
Status: Solved
Optimal value (cvx_optval): +2.18025
Estimated slopes (m1, m2):
    0.2967
    0.1965
Estimated intercept c:
    1.8963
```

Choice of loss functions is very crucial for some problems For example, consider the regression problem with 'outlier' data points. Outliers are data points away from the common crowd of data points. Consider data

points given below:

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}; \quad y = \begin{bmatrix} 5 \\ 6 \\ 10 \\ 40 \\ 12 \\ 17 \end{bmatrix}$$

Let us fit a regression line of the form  $y = mx + c$ . The problem with this data is that the point (4,40) is an outlier. Plotting of the data will reveal it. However in high dimensional case, it is impossible to plot and decide the outliers. If we use quadratic loss function, the estimated parameters  $m$  and  $c$  will greatly deviate and cannot capture the general trend of the data in the model. Instead, if we use sum of absolute value of deviation loss function, the effect of the presence of outlier can be greatly reduced. Let us do the computational experiment. Code for the estimation of parameters

Matlab code for this task is given below.

```

1 x=[1 2 3 4 5 6]';
2 y=[5 6 10 40 12 17]';
3 plot(x,y, '*')
4 hold on
5 cvx_begin
6     variable m;
7     variable c;
8     minimize    norm(y-m*x-c*ones(6,1))
9 cvx_end
10
11 disp('Slope');
12 disp(m);
13 disp("Intercept")
14 disp(c);
15 plot(x, m*x+c);

```

Output of the above code is shown below.

```

Calling SDPT3 4.0: 7 variables, 3 equality constraints
  For improved efficiency, SDPT3 is solving the dual problem.
-----

num. of constraints = 3
dim. of socp var = 7,  num. of socp blk = 1
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas gap    prim-obj    dual-obj    cputime
-----
0|0.000|0.000|8.2e-01|1.4e+00|1.3e+02| 0.000000e+00  0.000000e+00| 0:0:00| chol  1  1
1|0.945|1.000|4.5e-02|2.1e-03|8.9e+00|-2.675955e+01 -3.257704e+01| 0:0:00| chol  1  1
2|1.000|0.991|5.9e-08|4.3e-04|1.1e-01|-2.599703e+01 -2.608205e+01| 0:0:00| chol  1  1
3|0.989|0.989|1.4e-08|2.5e-05|1.2e-03|-2.602604e+01 -2.602599e+01| 0:0:00| chol  1  1
4|0.989|0.989|2.6e-09|2.8e-07|1.3e-05|-2.602636e+01 -2.602636e+01| 0:0:00| chol  1  1
5|0.989|0.989|2.9e-11|3.4e-09|1.6e-07|-2.602636e+01 -2.602636e+01| 0:0:00|
stop: max(relative gap, infeasibilities) < 1.49e-08
-----

number of iterations = 5
primal objective value = -2.60263602e+01
dual objective value = -2.60263602e+01

```

## 8. Assignment 56

### Loss and Penalty Function

---

```
gap := trace(XZ)          = 1.56e-07
relative gap              = 2.94e-09
actual relative gap      = -1.47e-10
rel. primal infeas (scaled problem) = 2.92e-11
rel. dual      "      "      "      = 3.39e-09
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual      "      "      "      = 0.00e+00
norm(X), norm(y), norm(Z) = 1.4e+00, 2.7e+01, 3.7e+01
norm(A), norm(b), norm(C) = 1.1e+01, 2.0e+00, 4.8e+01
Total CPU time (secs)    = 0.06
CPU time per iteration   = 0.01
termination code         = 0
DIMACS: 2.9e-11  0.0e+00  4.0e-09  0.0e+00  -1.5e-10  2.9e-09
```

---

```
-----
Status: Solved
Optimal value (cvx_optval): +26.0264
Slope
    3.0857
Intercept
    4.2000
```

Note that, outlier has greatly affected the estimation of parameters and hence the regression lines. The fitted lines is  $y = 3.09x + 4.2$  Let us try now sum of absolute value of deviation loss function. The code is as follows.

```
1 x=[1 2 3 4 5 6]';
2 y=[5 6 10 40 12 17]';
3 plot(x,y, '*')
4 hold on
5 cvx_begin quiet
6     variable m;
7     variable c;
8     minimize    norm(y-m*x-c*ones(6,1),1) % note the change here
9 cvx_end
10
11 disp('Slope');
12 disp(m);
13 disp('Intercept');
14 disp(c);
15 plot(x, m*x+c);
```

Output of the code is shown below.

```
Calling SDPT3 4.0: 14 variables, 6 equality constraints
-----

num. of constraints = 6
dim. of socp var = 12, num. of socp blk = 6
dim. of free var = 2 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT      1      0.000 1      0
it pstep dstep pinfeas dinfeas gap      prim-obj      dual-obj      cputime
```

```
-----  
number of iterations    = 9  
primal objective value = 3.20000000e+01  
dual  objective value = 3.20000000e+01  
gap := trace(XZ)       = 8.28e-08  
relative gap           = 1.27e-09  
actual relative gap    = 4.19e-10  
rel. primal infeas (scaled problem) = 6.17e-15  
rel. dual      "      "      "      = 1.29e-08  
rel. primal infeas (unscaled problem) = 0.00e+00  
rel. dual      "      "      "      = 0.00e+00  
norm(X), norm(y), norm(Z) = 4.0e+01, 2.0e+00, 3.2e+00  
norm(A), norm(b), norm(C) = 1.5e+01, 4.8e+01, 3.4e+00  
Total CPU time (secs) = 0.12  
CPU time per iteration = 0.01  
termination code      = 0  
DIMACS: 7.2e-15  0.0e+00  2.2e-08  0.0e+00  4.2e-10  1.3e-09  
-----
```

```
-----  
Status: Solved  
Optimal value (cvx_optval): +32  
Slope  
    2.4000  
Intercept  
    2.6000
```

The fitted lines is  $y = 2.4x + 2.6$ . Now the fitted line is somewhat okay. It captures the trend in the data.

## RESULTS

1. Concept of loss function and penalized function are revisited.
2. Quadratic loss function, L2 and LASSO regression models are developed and solved using CVX.





# 9 | Assignment 58-1

## Logistic Regression

### 9.1 From Linear Regression to Logistic Regression

#### 9.1.1 Recap- Linear regression

In linear regression, the goal is to model the relationship between the features  $X$  and the continuous target variable  $y$ . The model is:

$$y = \sum_{i=0}^n \beta_i x_i$$

The mean squared error (MSE) is typically used as the loss function to minimize during training. The MSE loss function is:

$$MSE = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

Where:

- $y_i$  is the true value of the target.
- $\hat{y}_i$  is the predicted value from the model.

The objective is to find the model parameters (coefficients) that minimize this loss function.

#### 9.1.2 The Motivation for Logistic Regression

Let's start our journey with a real context. Suppose we have a data as shown in Table 9.1. Here we want to predict whether a patient has a disease based on given features. The target variable  $y$  is either 0 (no disease) or 1 (disease present).

Table 9.1: Sample Dataset for Diabetes Prediction

Age	Glucose	Blood Pressure	BMI	Diabetes (Output)
25	85	70	22.0	0
30	95	75	28.5	1
45	120	80	31.0	1
35	105	68	25.5	0
50	130	85	29.0	1

Let's begin with the **problem** of using **linear regression** for the prediction task. Given input features  $X$ , the goal is to predict whether a sample gives 0 or 1 as output.

Consider a dataset with 5 training examples where each example is a pair  $(x_i, y_i)$ , with  $x_i \in \mathbb{R}^4$  representing the feature vector and  $y_i \in \{0, 1\}$  representing the output. Suppose we apply **linear regression** to predict the binary target. The linear regression model is given by:

$$\hat{y} = w^T x = w_0 + w_1 x_1 + \dots + w_5 x_5$$

Where  $w$  is the weight vector, and  $w^T x$  is the linear combination of the input features. However, since linear regression can output values in  $(-\infty, \infty)$ , it **fails** to correctly handle the binary classification problem, as the output should be constrained to  $[0, 1]$ .

Let us check this general statement using our dataset. Mat lab code for this task is given below.

```

1  % Sample data
2  % Features: [Age, Glucose, Blood Pressure, BMI]
3  X = [25, 85, 70, 22.0;
4        30, 95, 75, 28.5;
5        45, 120, 80, 31.0;
6        35 105 68 25.5;
7        50 130 85 29.0];
8
9  % Output: Diabetes (0 or 1)
10 y = [0; 1; 1; 0; 1];
11
12 % Adding a bias term (intercept) to the features
13 X = [ones(size(X, 1), 1), X]; % Add a column of ones to X for the
    intercept
14
15 % Performing Linear Regression using the Normal Equation
16 theta = (X' * X) \ (X' * y); % Calculate theta using the Normal
    Equation
17
18 % Display the coefficients
19 disp('Coefficients (Theta):');
20 disp(theta);
21
22 % Make predictions
23 predictions = X * theta;
24 X_new=[1 52 100 60 45];
25 % Display the predictions
26 disp('Predictions:');
27 disp(predictions);
28 disp('Prediction on new data:');
29 disp(X_new*theta);

```

Output of the above code is shown below.

Coefficients (Theta):

```

-8.9265
-0.1324
0.0529
0.0735
0.1176

```

Predictions:

```

-0.0000
1.0000
1.0000
0.0000
1.0000

```

```
Prediction on new data:  
-0.8088
```

From the output, the linear regression model learned from the training data is:

$$y = -8.9265 - 0.1324\text{Age} + 0.0529\text{Glucose} + 0.0735\text{Blood Pressure} + 0.1176\text{BMI} \quad (9.1)$$

Interestingly, this least square error learn *well* on the training data. That's why it predicts perfectly correct output as the ground truth. But while testing on a new patients data, this so called perfect model produces an output  $-0.8088$  ! Absolutely a nonsense prediction.

This leads to the need for a model that **maps the output of the linear regression model to a probability value** in the range  $[0, 1]$ , which is where **logistic regression** comes in.

### 9.1.3 Introducing the Sigmoid Function

To convert the continuous output of the linear regression model to a value between 0 and 1, we apply the **sigmoid function** (also known as the logistic function) to the linear regressor:

$$\hat{y} = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

Where: -  $w^T x$  is the linear combination of the input features. -  $\sigma(z)$  is the sigmoid function, which transforms the input  $z$  into a value between 0 and 1.

This transformation ensures that the output of the model is interpreted as the **probability** of the instance belonging to class 1, i.e.,  $P(y = 1|x) = \hat{y}$ .

### 9.1.4 Interpreting Logistic Regression Output as Probability

The output of the sigmoid function can be interpreted as a **probability**:

$$P(y = 1|x) = \hat{y} = \frac{1}{1 + e^{-w^T x}}$$
$$P(y = 0|x) = 1 - \hat{y} = 1 - \frac{1}{1 + e^{-w^T x}} = \frac{e^{-w^T x}}{1 + e^{-w^T x}}$$

### 9.1.5 Log-Odds and the Decision Boundary

The sigmoid function allows us to model the **log-odds** of the probability of class 1. The **odds** of an event is the ratio of the probability that the event occurs to the probability that it does not occur:

$$\text{Odds}(y = 1|x) = \frac{P(y = 1|x)}{P(y = 0|x)} = \frac{\hat{y}}{1 - \hat{y}} = e^{w^T x}$$

Taking the logarithm of the odds gives the **log-odds** (also called the logit function):

$$\log\left(\frac{\hat{y}}{1 - \hat{y}}\right) = w^T x$$

Thus, logistic regression is linear in the **log-odds** but non-linear in the probabilities.

### 9.1.6 Defining the Logistic Loss Function

The next step is to define an appropriate **loss function** for logistic regression. We want to maximize the likelihood that the predicted probabilities match the true labels. This leads to the **log-likelihood** function.

For a single data point  $i$ , the probability that the model correctly predicts the label is:

- If  $y_i = 1$ :  $P(y_i = 1|x_i) = \hat{y}_i$

- If  $y_i = 0$ :  $P(y_i = 0|x_i) = 1 - \hat{y}_i$

The **likelihood** for all training examples is the product of these probabilities:

$$\mathcal{L}(w) = \prod_{i=1}^m P(y_i|x_i) = \prod_{i=1}^m \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i}$$

The corresponding **log-likelihood** (which we maximize) is:

$$\log \mathcal{L}(w) = \sum_{i=1}^m (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

### 9.1.7 Minimizing the Negative Log-Likelihood (Logistic Loss)

Instead of maximizing the log-likelihood, we typically minimize the **negative log-likelihood**, also known as the **logistic loss function** (or binary cross-entropy):

$$J(w) = - \sum_{i=1}^m (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

Substituting  $\hat{y}_i = \sigma(w^T x_i)$ , we have:

$$J(w) = - \sum_{i=1}^m \left( y_i \log \left( \frac{1}{1 + e^{-w^T x_i}} \right) + (1 - y_i) \log \left( 1 - \frac{1}{1 + e^{-w^T x_i}} \right) \right)$$

Simplifying, the logistic loss function becomes:

$$J(w) = \sum_{i=1}^m \log(1 + e^{-d_i w^T x_i})$$

Where  $d_i = 2y_i - 1$  is the transformed label  $d_i \in \{-1, 1\}$ .

### 9.1.8 Adding Regularization

To improve generalization and prevent over fitting, we introduce **L2 regularization**. The regularized loss function is:

$$J(w) = \sum_{i=1}^m \log(1 + e^{-d_i w^T x_i}) + \frac{\lambda}{2} w^T w$$

Where: - The first term is the **logistic loss**, which penalizes incorrect predictions. - The second term is the **L2 regularization** term, which penalizes large weights and controls model complexity. -  $\lambda$  is the regularization hyperparameter.

It is one of the popular model used in machine learning for classification. Similar to SVM, it is a binary classifier. But by formulation, it is an unconstrained optimization problem. It is an unconstrained, strongly convex optimization problem. Further the objective function is twice differentiable.

### 9.1.9 What does optimization tries to do?

For objective function to take small value,  $\exp(-d_i w^T x_i)$  must be small for which it is desirable that  $d_i w^T x_i$  is positive for all  $i$ . So the vector  $x_i$  will try to take its elemental values such that  $W^T x_i$  is positive for  $d_i = 1$  and negative value for  $d_i = -1$ . Further to reduce over-fitting of the data, elemental values of  $w$  vector is made to be small in magnitude by the minimization of second term  $\frac{\lambda}{2} w^T w$ . Generally we call this term as generalization term. In SVM, this term is used to achieve maximum margin classifier.

## 9.2 Computational Part of Logistic Regression

### 9.2.1 Creating first logistic regression on the dummy dataset

To perform logistic regression on the provided diabetes dataset using MATLAB with L2 regularization one can write native matlab code or use CVX library. Matlab code for the lateral approach is shown below.

```
1 % Features (Age, Glucose, Blood Pressure, BMI)
2 X = [25, 85, 70, 22.0;
3       30, 95, 75, 28.5;
4       45, 120, 80, 31.0;
5       35, 105, 68, 25.5;
6       50, 130, 85, 29.0];
7
8 % Output: Diabetes (0 or 1)
9 y = [0; 1; 1; 0; 1];
10
11 % Adding a bias term (intercept) to the features
12 X = [ones(size(X, 1), 1), X]; % Add a column of ones to X for the
    intercept
13
14 % Transform labels: d = 2y - 1
15 d = 2 * y - 1;
16
17 % Set the regularization parameter
18 lambda = 0.1; % Regularization strength (adjustable)
19
20 % Perform Logistic Regression with L2 Regularization using CVX
21 cvx_begin quiet
22     variable w(size(X, 2)); % Initialize parameter vector w
23     minimize( (1/length(d)) * sum(log(1 + exp(-d .* (X * w)))) + (
        lambda / 2) * sum_square(w(2:end)) ); % Logistic loss with
        L2 regularization
24 cvx_end
25
26 % Make predictions using the logistic function
27 predictions = 1 ./ (1 + exp(-X * w)) >= 0.5; % Predict 1 if
    sigmoid >= 0.5
28
29 % Display the predictions
30 disp('Predicted Diabetes (0 or 1) for Training Data:');
31 disp(predictions);
32
33 % Test the performance on new data
34 X_new = [1, 52, 100, 60, 45];
35 new_prediction = 1 ./ (1 + exp(-X_new * w)) >= 0.5; % Predict 1 if
    sigmoid >= 0.5
36 disp('Predicted Diabetes (0 or 1) for New Data Point:');
37 disp(new_prediction);
```

Output of the above code is shown below.

```
Successive approximation method to be employed.
SDPT3 will be called several times to refine the solution.
```

```
Original size: 42 variables, 21 equality constraints
10 exponentials add 80 variables, 50 equality constraints
```

```
-----

Status: Solved
Optimal value (cvx_optval): +0.0577769
Predicted Diabetes (0 or 1) for Training Data:
0
1
1
0
1
Predicted Diabetes (0 or 1) for New Data Point:
1
```

### 9.2.2 Breast Cancer Prediction

The Diagnostic Wisconsin Breast Cancer Dataset is a well-known dataset used in machine learning and statistical analysis, particularly for classification tasks aimed at predicting the malignancy of breast tumors. This dataset is readily available in the UCI Machine Learning Repository and serves as an essential resource for developing and evaluating classification algorithms.

#### Key Details

- **Number of Instances:** 569 samples
- **Number of Features:** 30 numeric features
- **Target Variable:** The diagnosis indicating whether a tumor is malignant or benign:
  - **Malignant (M):** 1
  - **Benign (B):** 0

#### Features

The dataset contains 30 features derived from digitized images of fine needle aspirates (FNA) of breast masses. Key features include:

- **Radius:** Mean of distances from the center to the perimeter.
- **Texture:** Standard deviation of gray-scale values.
- **Perimeter:** Perimeter of the tumor.
- **Area:** Area of the tumor.
- **Smoothness:** Local variation in radius lengths.
- **Compactness:** Calculated as  $(\text{perimeter}^2 / \text{area} - 1.0) \times \text{radius}$ .
- **Concavity:** Severity of concave portions of the contour.
- **Concave Points:** Number of concave portions of the contour.
- **Symmetry:** Symmetry of the tumor.
- **Fractal Dimension:** A measure of complexity, termed “coastline approximation.”

## Data Characteristics

The features are typically scaled between 0 and 1, enhancing the performance of machine learning algorithms. The dataset exhibits a class imbalance, with approximately 37% of the cases being malignant and 63% benign, making it suitable for studying classification techniques that address such imbalances.

## Applications

This dataset is utilized in:

- **Classification Problems:** Building algorithms to categorize tumors as malignant or benign.
- **Performance Evaluation:** Testing and comparing various machine learning models like logistic regression, support vector machines, and decision trees.
- **Feature Selection:** Analyzing the importance of individual features in predicting breast cancer diagnoses.

The Diagnostic Wisconsin Breast Cancer Dataset can be accessed at the UCI Machine Learning Repository: [Breast Cancer Wisconsin \(Diagnostic\)](#).

### 9.2.3 Logistic regression model

To build a logistic regression model and use it to predict the breast cancer using the above dataset, CVX library is used. Matlab code for this task is given below.

```
1 data = readtable('wdbc.csv', 'ReadVariableNames', false);
2 X = table2array(data(:, 3:end));
3 d = strcmp(table2array(data(:, 2)), 'M');
4 [m, n] = size(X);
5 X = [X ones(m, 1)];
6 lambda = 1;
7 n = n + 1;
8 cvx_begin quiet
9     variable w(n)
10     minimize (log_sum_exp([-d .* (X * w)]) + lambda/2 * sum_square(w)
11             )
12 cvx_end
13 classification_accuracy = 1 - mean((d .* (X * w) < 1));
14 disp(['Classification Accuracy: ', num2str(classification_accuracy)
15     ]);
```

Output of the above code is shown below.

```
Successive approximation method to be employed.
For improved efficiency, SDPT3 is solving the dual problem.
SDPT3 will be called several times to refine the solution.
Original size: 1740 variables, 601 equality constraints
569 exponentials add 4552 variables, 2845 equality constraints
-----

Status: Solved
Optimal value (cvx_optval): +5.87781
Classification Accuracy: 0.37258
```

## RESULTS

1. Concept of logistic regression and its relevance is revisited in comparison with linear regression.
2. Loss function for the logistic regression is derived.
3. Logistic regression is computationally implemented using CVX library to predict the breast cancer using the UCI-wdb dataset.



# 10 | Assignment 59

## $\epsilon$ – Insensitive Loss Functions

### 10.1 Introduction

The  $\epsilon$ -insensitive loss function is commonly used in support vector regression (SVR). It allows for a certain margin of tolerance ( $\epsilon$ ) where errors within this margin are not penalized. Specifically, there is no loss if points lie inside a tube defined by width  $2\epsilon$  around the prediction line. Only outlier points (those outside the tube) are associated with a loss, making the optimization problem more focused on minimizing these outliers.

The  $\epsilon$ -insensitive loss function is designed to ignore small errors, effectively allowing a margin of tolerance around the predicted values. This is particularly useful in scenarios where small deviations from the true values are not significant or do not warrant penalty, which can lead to more robust models.

#### Definition

For each training sample  $(x_i, y_i)$ :

- The prediction is given by  $f(x_i) = wx_i - \gamma$ .
- The goal is to ensure that most predictions fall within the  $\epsilon$ -tube around the true values  $y_i$ .

The  $\epsilon$ -insensitive loss can be expressed as:

$$L(y, f(x)) = \begin{cases} 0 & \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & \text{if } |y - f(x)| > \epsilon \end{cases}$$

where: -  $y$  is the true value, -  $f(x)$  is the predicted value from the model, -  $\epsilon$  is a user-defined threshold that specifies the insensitivity margin.

In essence, if the absolute error between the true value and the predicted value is less than or equal to  $\epsilon$ , no loss is incurred. If the error exceeds  $\epsilon$ , the loss increases linearly with the deviation.

#### 10.1.1 Formulating the Optimization Problem

To formulate this in terms of optimization, we introduce slack variables  $\xi_i^+$  and  $\xi_i^-$  to account for deviations above and below the  $\epsilon$ -margin.

- **Above the Margin:** If  $f(x_i) < y_i - \epsilon$ , we have a positive error, which we denote by  $\xi_i^+$ .
- **Below the Margin:** If  $f(x_i) > y_i + \epsilon$ , we have a negative error, denoted by  $\xi_i^-$ .

Thus, we can rewrite the constraints:

1. For positive deviations:

$$y_i \leq (wx_i - \gamma) + \epsilon + \xi_i^+$$

---

2. For negative deviations:

$$y_i \geq (wx_i - \gamma) - \epsilon - \xi_i^-$$

### 10.1.2 Optimization Objective

The objective is to find appropriate values of  $w$  and  $\gamma$  such that most points fall within the tube defined by the  $\epsilon$ -margin, thereby minimizing the total penalty from the slack variables. This leads to the formulation:

$$\min_{w, \gamma, \xi^+, \xi^-} \sum_i (\xi_i^+ + \xi_i^-)$$

This optimization problem seeks to minimize the sum of losses associated with outlier points while maximizing the number of points that lie within the  $\epsilon$ -tube.

### 10.1.3 Constraints

Finally, the constraints ensure that we account for the deviations while also ensuring that  $\xi_i^+$  and  $\xi_i^-$  are non-negative:

$$\xi_i^-, \xi_i^+ \geq 0$$

## Relevance

The  $\epsilon$ -insensitive loss function has several advantages that make it relevant in various applications:

**Robustness to Outliers:** By ignoring small errors, the  $\epsilon$ -insensitive loss function reduces the impact of outliers on the model training process, leading to a more stable and reliable regression model.

**Flexibility in Modeling:** The parameter  $\epsilon$  allows practitioners to adjust the sensitivity of the loss function based on the specific characteristics of the data and the problem at hand. This adaptability is beneficial in diverse real-world applications.

**Support Vector Regression:** In SVR, the  $\epsilon$ -insensitive loss function plays a central role in defining the regression problem. It facilitates the identification of a function that fits the data while maintaining a balance between the complexity of the model and its predictive accuracy.

**Application in Real-World Problems** Many real-world problems, such as financial forecasting, environmental monitoring, and engineering applications, require models that can tolerate small errors due to inherent variability in the data. The  $\epsilon$ -insensitive loss function addresses this need effectively.

## 10.2 Example

Consider the problem with input  $X$  and the target  $y$  as given below.

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

$$y = \begin{bmatrix} 5 \\ 6 \\ 10 \\ 40 \\ 12 \\ 17 \end{bmatrix}$$

It should be noted that from the data , the pair (4,40) is an outlier! Let us fit a regression line of the form,  $y = wx - \gamma$ . The constraints can be written in the vector form as follows.

$$\begin{bmatrix} 5 \\ 6 \\ 10 \\ 40 \\ 12 \\ 17 \end{bmatrix} \leq w^* \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} - \gamma \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \epsilon \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} \xi_1^+ \\ \xi_2^+ \\ \xi_3^+ \\ \xi_4^+ \\ \xi_5^+ \\ \xi_6^+ \end{bmatrix}$$

$$\begin{bmatrix} 5 \\ 6 \\ 10 \\ 40 \\ 12 \\ 17 \end{bmatrix} \geq w^* \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} - \gamma \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \epsilon \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} \xi_1^- \\ \xi_2^- \\ \xi_3^- \\ \xi_4^- \\ \xi_5^- \\ \xi_6^- \end{bmatrix}$$

$$\xi_i^+, \xi_i^- \geq 0$$

Matlab code to solve this linear programming problem using  $\epsilon$ - insensitive loss function with CVX solver is given below.

```

1 close(gcf);
2 x=[1 2 3 4 5 6]';
3 y=[5 6 10 40 12 17]';
4 e=[1 1 1 1 1 1]';
5 epsino = 2;
6 n=6;
7 plot(x,y, 'r*')
8 hold on
9 cvx_begin quiet
10 variable w;
11 variable gama;
12 variable psiplus(n);
13 variable psiminus(n);
14 minimize sum(psiplus+psiminus)
15 subject to
16     y<=w*x-gama*e+epsino*e+psiplus;
17     y>= w*x-gama*e-epsino*e-psiminus;
18     psiplus>=0;
```

```

19     psiminus >= 0;
20 cvx_end
21 plot(x, w*x-gama);
22 hold on
23 plot(x, w*x-gama+epsino);
24 hold on
25 plot(x, w*x-gama-epsino);
26 w
27 gama

```

output of the above code is

```

Calling SDPT3 4.0: 26 variables, 12 equality constraints
-----

num. of constraints = 12
dim. of linear var = 24
dim. of free var = 2 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas gap prim-obj dual-obj cputime
-----

number of iterations = 12
primal objective value = 2.60000000e+01
dual objective value = 2.60000000e+01
gap := trace(XZ) = 6.54e-09
relative gap = 1.23e-10
actual relative gap = 3.00e-11
rel. primal infeas (scaled problem) = 3.18e-16
rel. dual " " " = 8.47e-10
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 4.1e+01, 1.2e+00, 3.3e+00
norm(A), norm(b), norm(C) = 2.1e+01, 6.8e+01, 4.5e+00
Total CPU time (secs) = 0.15
CPU time per iteration = 0.01
termination code = 0
DIMACS: 5.0e-16 0.0e+00 1.9e-09 0.0e+00 3.0e-11 1.2e-10
-----

-----
Status: Solved
Optimal value (cvx_optval): +26

w = 2.0000
gama = -4.0000

```

Visualization of the  $\epsilon$  tube for given dataset along with the outlier is shown in Figure [10.1](#).

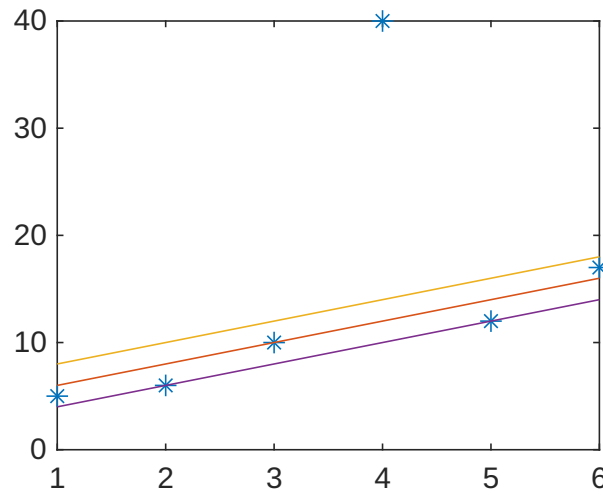


Figure 10.1:  $\epsilon$  tube generated with the solution of  $\epsilon$ – insensitive loss function

The equation of the regression line is  $y = 2x - 4$ . The fitted line for the ordinary regression model was  $y = 2.4x + 2.6$ !. So the  $\epsilon$ – insensitive loss function successfully eliminate the outlier and create a practical model to that capture the general trend in the data.

### 10.3 Task

For the following data, fit a regression equation of the form  $y = w_1x_1 + w_2x_2 - \gamma$  using  $\epsilon = 2$ .  $X =$

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 2 \\ 3 & 3 \\ 3 & 4 \\ 4 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 22 \\ 27 \\ 33 \\ 80 \\ 58 \\ 47 \end{bmatrix}.$$

#### SOLUTION

Here the objective function is

$$\min_{w, \gamma, \xi^+, \xi^-} \sum_i (\xi_i^+ + \xi_i^-)$$

subject to the constraints:

$$\begin{bmatrix} 22 \\ 27 \\ 33 \\ 80 \\ 58 \\ 47 \end{bmatrix} \leq w_1^* \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \\ 3 \\ 4 \end{bmatrix} + w_2^* \begin{bmatrix} 2 \\ 1 \\ 2 \\ 3 \\ 4 \\ 3 \end{bmatrix} - \gamma \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \epsilon \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} \xi_1^+ \\ \xi_2^+ \\ \xi_3^+ \\ \xi_4^+ \\ \xi_5^+ \\ \xi_6^+ \end{bmatrix}$$

$$\begin{bmatrix} 22 \\ 27 \\ 33 \\ 80 \\ 58 \\ 47 \end{bmatrix} \geq w_1^* \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \\ 3 \\ 4 \end{bmatrix} + w_2^* \begin{bmatrix} 2 \\ 1 \\ 2 \\ 3 \\ 4 \\ 3 \end{bmatrix} - \gamma \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \epsilon \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} \xi_1^- \\ \xi_2^- \\ \xi_3^- \\ \xi_4^- \\ \xi_5^- \\ \xi_6^- \end{bmatrix}$$

$$\xi_i^+, \xi_i^- \geq 0$$

Matlab code for solving the task is given below.

```

1 close(gcf);
2 clear;
3 X = [1 2 3 3 3 4; 2 1 2 3 4 3];
4 y_new = [22; 27; 33; 80; 58; 47];
5 n = length(y_new);
6 e = ones(n, 1);
7 epsino = 2;
8 cvx_begin quiet
9     variable w_new(2);
10    variable gama_new;
11    variable psiplus_new(n);
12    variable psiminus_new(n);
13    minimize(sum(psiplus_new + psiminus_new))
14
15    % Constraints
16    subject to
17        y_new <= X' * w_new - gama_new + epsino + psiplus_new;
18        y_new >= X' * w_new - gama_new - epsino - psiminus_new;
19        psiplus_new >= 0;
20        psiminus_new >= 0;
21 cvx_end
22 disp('New Weight (w_new):');
23 disp(w_new);
24 disp('New Bias (gama_new):');
25 disp(gama_new);

```

Output of the above code is shown below.

Calling SDPT3 4.0: 27 variables, 12 equality constraints

```

-----

num. of constraints = 12
dim. of linear var = 24
dim. of free var = 3 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0

it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj      cputime
-----

number of iterations = 12
primal objective value = 4.05000000e+01
dual objective value = 4.05000000e+01
gap := trace(XZ) = 1.29e-08
relative gap = 1.57e-10
actual relative gap = 2.76e-11
rel. primal infeas (scaled problem) = 6.91e-16
rel. dual " " " = 1.61e-09
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 5.2e+01, 2.1e+00, 3.3e+00
norm(A), norm(b), norm(C) = 2.1e+01, 1.7e+02, 4.5e+00
Total CPU time (secs) = 0.07

```

## 10. Assignment 59

### $\epsilon$ – Insensitive Loss Functions

---

```
CPU time per iteration = 0.01
termination code       = 0
DIMACS: 1.4e-15  0.0e+00  3.6e-09  0.0e+00  2.8e-11  1.6e-10
```

-----

-----

```
Status: Solved
Optimal value (cvx_optval): +40.5
New Weight (w_new):
    4.3320
    10.5000
New Bias (gama_new):
    -1.0041
```

So the linear regression model using  $\epsilon$ – insensitive loss function is given by;

$$y = 4.3320x_1 + 10.50x_2 - 1.0041$$

## RESULTS

1. Theoretical frame work of the  $\epsilon$ – insensitive loss function and its derivation is revisited.
2. Problems with outliers are solved using the new approach.





# 11 | Assignment 60

## SVM Classifier With Soft Margin

### 11.1 Introduction

Support Vector Machines (SVMs) are supervised learning models primarily used for classification tasks. The objective of an SVM is to find the optimal hyperplane that separates data points of different classes in a high-dimensional space. The SVM aims to maximize the margin between the separating hyperplane and the closest data points from each class (support vectors). A larger margin leads to better generalization.

#### 11.1.1 Hard margin SVM

In the case of **hard margin SVM**, the algorithm finds a hyperplane that perfectly separates two classes without any misclassification. This approach works well for linearly separable data but fails when the data is noisy or not perfectly separable.

#### 11.1.2 Soft margin SVM

To address the limitations of hard margin SVM, **soft margin SVM** was introduced. The soft margin SVM allows for some misclassification while still maximizing the margin, making the model more flexible and applicable to real-world data.

Let's start with a simple context to understand the principles of SVM. Consider labelled data points as shown in Figure 11.1.

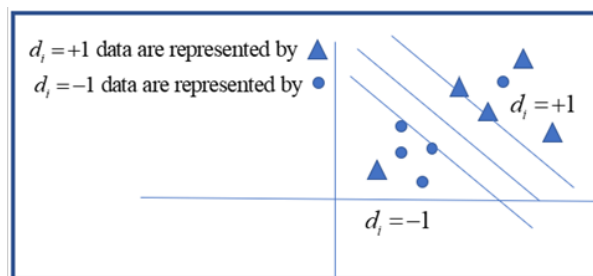


Figure 11.1: Hyper planes in SVM Classification problem

Let the central line is  $w_1 x_1 + w_2 x_2 - \gamma = 0$ . This form the classifier line. The top bounding plane is  $w_1 x_1 + w_2 x_2 - \gamma = 1$  and The bottom bounding plane is  $w_1 x_1 + w_2 x_2 - \gamma = -1$ . All triangular shaped data points are supposed to satisfy following relation,  $w^T x_i - \gamma \geq +1$ . But one point in the data cannot satisfy that relation. To satisfy, we add a positive quantity on the left or equivalently subtract a positive quantity from right. This is a sort of 'error allowance' given to such points. But in advance we don't know which points require that error allowance. So, we allow all points to have it first, but form the

objective function in such a that only the needy few points take that allowance. So, we write

$$(w^T x_i - \gamma) + \xi_i \geq 1$$

$$\text{or } (w^T x_i - \gamma) \geq 1 - \xi_i; \quad \xi_i \geq 0$$

Multiplying left side by  $d_i (= 1)$  on the left and right side with +1, we obtain

$$d_i (w^T x_i - \gamma) \geq 1 - \xi_i \quad (11.1)$$

Similarly for a correct classification all circular shaped data points are supposed to satisfy the relation

$$w^T x_j - \gamma \leq 1$$

But one point in the data cannot satisfy that relation. To satisfy, we should subtract a positive quantity on the left. So, we write

$$(w^T x_j - \gamma) + \xi_j \leq -1; \quad \xi_j \geq 0$$

Multiplying left side by  $d_j (= -1)$  and right side with -1, we obtain

$$d_j ((w^T x_j - \gamma) + \xi_j) \geq 1$$

$$d_j (w^T x_j - \gamma) + d_j \xi_j \geq 1$$

$$d_j (w^T x_j - \gamma) \geq 1 - \xi_j; \quad \xi_j \geq 0$$

But as mentioned earlier, we should allow only those points which are stranded to opposite side utilize the error allowances. This is imposed by taking

$$\min_{w, \gamma, \xi} \sum_i \xi_i$$

as the objective function Thus, our optimization problem boils down to

$$\min_{w, \gamma, \xi} \sum_i \xi_i$$

Subject to:

$$d_i (w^T x_i - \gamma) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \forall i$$

It is a linear programming problem.

### 11.1.3 Solution using Matlab

We can use the CVX library to solve this optimization problem. In matlab notation this can be specified by assuming

Number of data points is m

Number of elements in w is n

We then write the problem in CVX as follows.

```
e=ones(m,1)
cvx_begin quiet
variable w(n)
variable gama
variable psi(m)
minimize sum(psi)
subject to
    d.*(X*w-gama*e)>=1-psii;
    psii>=0;
cvx_end
```

After getting the  $(w, \gamma)$  we put the classifier as  $(w^T x - \gamma)$ .

### 11.1.4 Computational example

Find a linear classifier for following data,

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 2 & 2.5 \\ 2 & 2 \\ 2 & 3 \\ 3 & 2 \\ -1 & 0 \end{bmatrix}; \quad y = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

#### SOLUTION

Mathematical model of the given problem is

$$\begin{aligned} & \min_{w, \gamma, \xi} \sum_i \xi_i \\ & \text{Subject to:} \\ & d_i(w^T x_i - \gamma) \geq 1 - \xi_i \\ & \xi_i \geq 0 \forall i \end{aligned}$$

This LPP can be solved using the CVX library. Matlab code for this task is given below.

```
1 X=[1 0 0 2 2 2 3 -1;0 1 0 2.5 2 3 2 0]';
2 d=[-1 -1 -1 -1 1 1 1 1]';
3 e=ones(8,1);
4 cvx_begin quiet
5 variable w(2);
6 variable gama;
7 variable psii(8); % matlab will assume psii as a column vector
8 minimize sum(psii)
9 subject to
10     d.*(X*w-gama*e)>=1-psii;
11     psii>=0;
12 cvx_end
13 plot(X(1:4,1), X(1:4,2), '*'); % plot -1 points
14 hold on
15 plot(X(5:8,1), X(5:8,2), 'o'); % plot +1 points
16 hold on
17 x1=-1:3;
18 x2=-(w(1)/w(2))*x1+(gama/w(2));
19 plot(x1,x2); % draw the classifier line
20
21 hold on
22 x2=-(w(1)/w(2))*x1+((-1+gama)/w(2));
23 plot(x1,x2) % draw the lower bounding line
24 disp("Optimal values of w:")
25 disp(w);
26 disp("Optimal values of gamma:")
27 disp(gama);
28 hold on
29 x2=-(w(1)/w(2))*x1+((1+gama)/w(2));
30 plot(x1,x2) % draw the upper bounding line
```

Output of the above code is shown below.

```

Calling SDPT3 4.0: 19 variables, 8 equality constraints
-----

num. of constraints = 8
dim. of linear var = 16
dim. of free var = 3 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version  predcorr  gam  expon  scale_data
NT      1      0.000  1      0
it pstep dstep pinfeas dinfeas gap      prim-obj      dual-obj      cputime
-----
0|0.000|0.000|7.4e-01|1.4e+01|2.2e+03| 8.000000e+01  0.000000e+00| 0:0:00| chol  1  1
1|1.000|0.974|1.3e-06|4.6e-01|1.2e+02| 7.407906e+01  3.908924e+00| 0:0:00| chol  1  1
2|1.000|0.686|6.8e-07|1.5e-01|3.3e+01| 2.701793e+01  3.683567e+00| 0:0:00| chol  1  1
3|0.900|0.489|7.6e-07|7.8e-02|8.3e+00| 1.053947e+01  3.958191e+00| 0:0:00| chol  1  1
4|0.990|0.478|4.0e-07|4.1e-02|2.4e+00| 6.359193e+00  4.377868e+00| 0:0:00| chol  1  1
5|1.000|0.594|4.2e-08|1.7e-02|9.0e-01| 5.656266e+00  4.895228e+00| 0:0:00| chol  1  1
6|1.000|0.839|1.4e-08|2.7e-03|1.1e-01| 5.293551e+00  5.204960e+00| 0:0:00| chol  1  1
7|0.978|0.977|1.6e-09|6.1e-05|2.7e-03| 5.251138e+00  5.248871e+00| 0:0:00| chol  1  1
8|0.989|0.989|4.2e-10|7.0e-07|3.0e-05| 5.250013e+00  5.249987e+00| 0:0:00| chol  1  2
9|0.989|0.989|2.7e-11|4.2e-07|1.7e-06| 5.250000e+00  5.250000e+00| 0:0:00| chol  1  1
10|1.000|0.989|3.7e-10|2.3e-08|9.2e-08| 5.250000e+00  5.250000e+00| 0:0:00| chol  2  2
11|1.000|0.989|2.0e-11|1.3e-09|5.0e-09| 5.250000e+00  5.250000e+00| 0:0:00|
stop: max(relative gap, infeasibilities) < 1.49e-08
-----

number of iterations = 11
primal objective value = 5.25000000e+00
dual objective value = 5.25000000e+00
gap := trace(XZ) = 5.04e-09
relative gap = 4.38e-10
actual relative gap = 7.12e-11
rel. primal infeas (scaled problem) = 2.01e-11
rel. dual " " " = 1.28e-09
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 3.9e+00, 2.1e+00, 2.6e+00
norm(A), norm(b), norm(C) = 1.2e+01, 3.8e+00, 3.8e+00
Total CPU time (secs) = 0.14
CPU time per iteration = 0.01
termination code = 0
DIMACS: 3.8e-11 0.0e+00 2.5e-09 0.0e+00 7.1e-11 4.4e-10
-----

Status: Solved
Optimal value (cvx_optval): +5.25
Optimal values of w:
0.5000
0.5000
Optimal values of gamma:
1.5000

```

From the output, it is clear that the separating plane is  $0.5x_1 + 0.5x_2 = 1.5$ . Skill of this SVM classifier is shown in Figure 11.2.

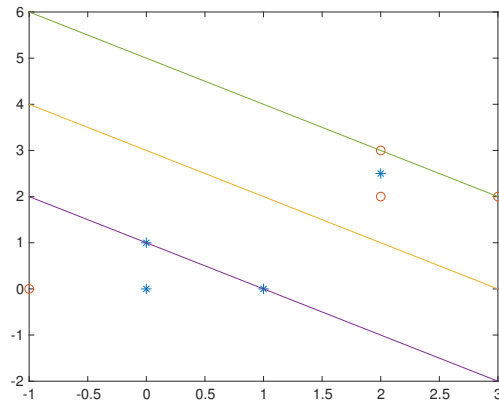


Figure 11.2: Visualization of the SVM Classifier modelled through LPP

From Figure 11.2, it is clear that still there are misclassification! Soft margin SVM introduces a **regularization parameter**  $C$ , which balances two competing objectives:

- **Maximizing the margin:** Ensuring a large margin between classes.
- **Minimizing misclassification errors:** Allowing some points to be misclassified but penalizing them based on the slack variables.

## 11.2 Optimization in Soft Margin SVM

The optimization problem for soft margin SVM can be formulated as a **Linear Programming Problem (LPP)**:

$$\min_{\mathbf{w}, \gamma, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to:

$$d_i(\mathbf{w} \cdot \mathbf{x}_i - \gamma) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Where:

- $\mathbf{w}$  represents the weights of the hyperplane.
- $\gamma$  is the **bias term** (also known as the intercept).
- $d_i$  is the **class label** for data point  $i$ , where  $d_i = +1$  for one class and  $d_i = -1$  for the other class.
- $\xi_i$  are the **slack variables** that allow some points to be misclassified.
- $C$  is the regularization parameter that controls the trade-off between maximizing the margin and minimizing misclassifications.

Matlab code to implement the regularized version is given below.

```

1 % Data points and labels
2 X = [1 0 0 2 2 2 3 -1; 0 1 0 2.5 2 3 2 0]';
3 d = [-1 -1 -1 -1 1 1 1 1]';
4 e = ones(8,1);
5
6 % Regularization parameter
7 C = 0.5;
8
9 % CVX optimization
10 cvx_begin
11     variable w(2);           % Weight vector
12     variable gama;           % Bias term

```

```

13     variable psii(8);           % Slack variables (column vector)
14
15     % Objective: Minimize regularized objective function
16     minimize (0.5*sum_square(w) + C*sum(psii))
17
18     % Subject to constraints
19     subject to
20         d.*(X*w - gama*e) >= 1 - psii; % SVM margin constraints
21         psii >= 0;                     % Slack variables non-
            negative
22 cvx_end
23
24 % Display the optimal values of w and gama
25 disp("Optimal values of w:")
26 disp(w);
27 disp("Optimal values of gamma:")
28 disp(gama);
29
30 % Plotting the data points and decision boundary
31 figure;
32 plot(X(1:4,1), X(1:4,2), '*'); % Plot -1 class points
33 hold on
34 plot(X(5:8,1), X(5:8,2), 'o'); % Plot +1 class points
35 hold on
36
37 % Plot the decision boundary
38 x1 = -1:3; % Range for x1
39 x2 = -(w(1)/w(2))*x1 + (gama/w(2));
40 plot(x1, x2, 'k', 'LineWidth', 2); % Decision boundary
41
42 % Plot the margin boundaries
43 hold on
44 x2_lower = -(w(1)/w(2))*x1 + ((-1 + gama)/w(2));
45 plot(x1, x2_lower, '--r', 'LineWidth', 1); % Lower margin boundary
46
47 hold on
48 x2_upper = -(w(1)/w(2))*x1 + ((1 + gama)/w(2));
49 plot(x1, x2_upper, '--b', 'LineWidth', 1); % Upper margin boundary
50
51 % Add labels and legend
52 xlabel('x_1');
53 ylabel('x_2');
54 title('SVM with Soft Margin and Regularization (C = 0.5)');
55 legend('-1 class', '+1 class', 'Decision Boundary', 'Lower Margin',
        'Upper Margin');
56 hold off;

```

Output of the above code is shown below.

```

Calling SDPT3 4.0: 21 variables, 9 equality constraints
-----

```

```

num. of constraints = 9
dim. of socp var = 4, num. of socp blk = 1

```

```

dim. of linear var = 16
dim. of free var = 1 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT 1 0.000 1 0
it pstep dstep pinfeas dinfeas gap prim-obj dual-obj cputime
-----
0|0.000|0.000|1.0e+00|1.5e+01|1.6e+03| 4.200000e+01 0.000000e+00| 0:0:00| chol 1 1
1|1.000|0.989|8.0e-07|2.7e-01|6.5e+01| 4.048903e+01 7.704888e-01| 0:0:00| chol 1 1
2|1.000|0.634|1.6e-07|1.0e-01|1.1e+01| 1.003469e+01 1.407374e+00| 0:0:00| chol 1 1
3|0.974|0.294|3.3e-07|7.4e-02|3.3e+00| 4.340897e+00 1.718256e+00| 0:0:00| chol 1 1
4|0.920|0.881|7.3e-07|8.8e-03|4.9e-01| 3.085932e+00 2.635943e+00| 0:0:00| chol 1 1
5|1.000|0.676|1.1e-08|2.9e-03|1.3e-01| 2.901001e+00 2.784417e+00| 0:0:00| chol 1 1
6|1.000|0.855|1.4e-08|4.2e-04|1.9e-02| 2.866653e+00 2.849065e+00| 0:0:00| chol 1 1
7|0.929|0.933|1.7e-09|2.8e-05|1.5e-03| 2.860648e+00 2.859286e+00| 0:0:00| chol 1 1
8|0.974|0.827|1.9e-09|1.8e-05|2.6e-04| 2.860069e+00 2.859856e+00| 0:0:00| chol 1 1
9|0.975|0.980|1.8e-10|3.3e-06|1.2e-05| 2.860002e+00 2.859997e+00| 0:0:00| chol 1 1
10|1.000|0.987|3.4e-11|1.4e-07|4.4e-07| 2.860000e+00 2.860000e+00| 0:0:00| chol 1 1
11|1.000|0.987|1.3e-11|5.5e-09|1.7e-08| 2.860000e+00 2.860000e+00| 0:0:00|
stop: max(relative gap, infeasibilities) < 1.49e-08
-----
number of iterations = 11
primal objective value = 2.86000000e+00
dual objective value = 2.86000000e+00
gap := trace(XZ) = 1.68e-08
relative gap = 2.49e-09
actual relative gap = 9.84e-10
rel. primal infeas (scaled problem) = 1.26e-11
rel. dual " " " = 5.53e-09
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual " " " = 0.00e+00
norm(X), norm(y), norm(Z) = 3.4e+00, 1.2e+00, 1.6e+00
norm(A), norm(b), norm(C) = 1.0e+01, 4.0e+00, 2.6e+00
Total CPU time (secs) = 0.08
CPU time per iteration = 0.01
termination code = 0
DIMACS: 2.5e-11 0.0e+00 9.5e-09 0.0e+00 9.8e-10 2.5e-09
-----

-----
Status: Solved
Optimal value (cvx_optval): +2.86
Optimal values of w:
0.4000
0.4000
Optimal values of gamma:
1.0000

```

### 11.3 LPP Formulation as an Advantage

The **LPP formulation** of soft margin SVM offers several advantages:

- **Convex Optimization:** The problem remains a convex quadratic optimization problem, ensuring that it can be solved efficiently using standard optimization techniques.
- **Control Over Misclassifications:** By adjusting the regularization parameter  $C$ , one can control

the degree to which the model prioritizes maximizing the margin versus minimizing misclassifications.

- A **high value of  $C$**  focuses on minimizing misclassifications, potentially leading to over fitting.
- A **low value of  $C$**  allows for a larger margin with more tolerance for misclassified points, leading to better generalization in noisy datasets.
- **Flexibility with Slack Variables:** Slack variables  $\xi_i$  give the model flexibility in handling overlapping or non-linearly separable data points.

## 11.4 Kernel Trick in Soft Margin SVM

The soft margin SVM can handle non-linearly separable data using the **kernel trick**. The kernel function implicitly maps the input data into a higher-dimensional feature space where it is easier to find a separating hyperplane. Popular kernels include:

- **Linear kernel:** For linearly separable data.
- **Polynomial kernel:** For data separable by polynomial decision boundaries.
- **Radial Basis Function (RBF) kernel:** For more complex, non-linearly separable data.

## 11.5 Advantages of Soft Margin SVM with LPP Formulation

- **Generalization to Real-World Data:** The soft margin SVM handles noisy, overlapping, and non-separable data more effectively by allowing misclassifications. This leads to better performance on unseen data.
- **Scalability and Flexibility:** The LPP-based formulation ensures that the optimization problem can be solved efficiently even for large datasets.
- **Trade-off Control:** The regularization parameter  $C$  offers a clear mechanism for controlling the trade-off between margin maximization and misclassification tolerance.
- **Kernel Efficiency:** The kernel trick, combined with the soft margin approach, makes SVM a powerful classifier even in high-dimensional spaces.

## 11.6 Applications of Soft Margin SVM

Soft margin SVM has broad applications, particularly in tasks requiring high accuracy and robustness:

- **Text classification:** For tasks like spam detection and sentiment analysis.
- **Image recognition:** For tasks like object detection and face recognition.
- **Bioinformatics:** In tasks such as gene expression classification and protein function prediction.

## 11.7 Task

Use LP formulation of SVM classifier to classify UCI breast cancer dataset.

### SOLUTION

The Diagnostic Wisconsin Breast Cancer Dataset is a well-known dataset used in machine learning and statistical analysis, particularly for classification tasks aimed at predicting the malignancy of breast tumors. This dataset is readily available in the UCI Machine Learning Repository and serves as an essential resource for developing and evaluating classification algorithms.



## Key Details

- **Number of Instances:** 569 samples
- **Number of Features:** 30 numeric features
- **Target Variable:** The diagnosis indicating whether a tumor is malignant or benign:
  - **Malignant (M):** 1
  - **Benign (B):** 0

## Features

The dataset contains 30 features derived from digitized images of fine needle aspirates (FNA) of breast masses. Key features include:

- **Radius:** Mean of distances from the center to the perimeter.
- **Texture:** Standard deviation of gray-scale values.
- **Perimeter:** Perimeter of the tumor.
- **Area:** Area of the tumor.
- **Smoothness:** Local variation in radius lengths.
- **Compactness:** Calculated as  $(\text{perimeter}^2 / \text{area} - 1.0) \times \text{radius}$ .
- **Concavity:** Severity of concave portions of the contour.
- **Concave Points:** Number of concave portions of the contour.
- **Symmetry:** Symmetry of the tumor.
- **Fractal Dimension:** A measure of complexity, termed “coastline approximation.”

## Data Characteristics

The features are typically scaled between 0 and 1, enhancing the performance of machine learning algorithms. The dataset exhibits a class imbalance, with approximately 37% of the cases being malignant and 63% benign, making it suitable for studying classification techniques that address such imbalances.

Matlab code for the task is given below.

```
1 data = readtable('wdbc.csv', 'ReadVariableNames', false);
2 X = table2array(data(:, 3:end)); % Extract features
3 d = strcmp(table2array(data(:, 2)), 'M'); % Labels: Convert 'M' to
   1, others to 0
4 d = 2 * d - 1; % Convert d to +1 and -1
5 [m, n] = size(X); % m: number of data points, n: number of
   features
6 e = ones(m,1); % Vector of ones for bias term
7
8 % Regularization parameter
9 C = 0.5;
10
11 % CVX optimization
12 cvx_begin
```

## 11. Assignment 60

### SVM Classifier With Soft Margin

```

13     variable w(n);           % Weight vector with n elements (number
    of features)
14     variable gama;          % Bias term (scalar)
15     variable psii(m);       % Slack variables (m elements, one for
    each data point)
16
17     % Objective: Minimize regularized objective function
18     minimize (0.5*sum_square(w) + C*sum(psii))
19
20     % Subject to constraints
21     subject to
22         d.*(X*w - gama*e) >= 1 - psii; % SVM margin constraints
23         psii >= 0;                    % Slack variables non-
    negative
24 cvx_end
25
26 % Display the optimal values of w and gama
27 disp("Optimal values of w:")
28 disp(w);
29 disp("Optimal values of gamma:")
30 disp(gama);
31
32 % Calculate and display classification accuracy
33 classification_accuracy = 1 - mean((d .* (X * w - gama) < 1));
34 disp(['Classification Accuracy: ', num2str(classification_accuracy)
    ]);

```

Output of the above code is shown below.

Calling SDPT3 4.0: 1171 variables, 570 equality constraints

-----

```

num. of constraints = 570
dim. of socp var = 32, num. of socp blk = 1
dim. of linear var = 1138
dim. of free var = 1 *** convert ublk to lblk
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT 1 0.000 1 0
it pstep dstep pinfeas dinfeas gap prim-obj dual-obj cputime
-----
0|0.000|0.000|9.8e-01|3.9e+02|2.6e+06| 1.919762e+04 0.000000e+00| 0:0:00| chol 1 1
1|1.000|0.678|1.8e-04|1.3e+02|1.2e+06| 2.687214e+04 -1.164910e+03| 0:0:00| chol 1 1
2|1.000|0.805|4.5e-06|2.5e+01|2.6e+05| 2.586179e+04 -2.180943e+02| 0:0:00| chol 1 1
3|1.000|0.133|5.0e-06|2.2e+01|2.5e+05| 2.584860e+04 -1.921474e+02| 0:0:00| chol 1 1
4|1.000|0.828|1.6e-06|3.7e+00|6.0e+04| 2.181686e+04 -1.037319e+02| 0:0:00| chol 1 1
5|0.610|0.432|1.1e-05|2.1e+00|3.9e+04| 1.616194e+04 -1.297325e+02| 0:0:00| chol 1 1
6|0.971|0.427|3.8e-06|1.2e+00|2.7e+04| 1.084041e+04 -2.889765e+02| 0:0:00| chol 1 1
7|0.563|0.314|3.5e-06|8.2e-01|2.4e+04| 9.931706e+03 -3.185510e+02| 0:0:00| chol 1 1
8|0.284|0.342|3.4e-06|5.4e-01|2.0e+04| 9.808614e+03 -3.127259e+02| 0:0:00| chol 1 1
9|0.207|0.481|3.0e-06|2.8e-01|1.5e+04| 9.559508e+03 -2.544086e+02| 0:0:01| chol 1 1
10|0.300|0.197|2.2e-06|2.3e-01|1.4e+04| 8.436309e+03 -2.450072e+02| 0:0:01| chol 1 1
11|0.338|0.218|1.5e-06|1.8e-01|1.2e+04| 7.627973e+03 -2.579558e+02| 0:0:01| chol 1 1
12|0.278|0.148|1.1e-06|1.5e-01|1.1e+04| 7.053327e+03 -2.719061e+02| 0:0:01| chol 1 1

```

## 11. Assignment 60

### SVM Classifier With Soft Margin

```

13|0.243|0.097|8.4e-07|1.4e-01|1.1e+04| 6.813621e+03 -2.828205e+02| 0:0:01| chol 1 1
14|0.447|0.076|4.5e-07|1.3e-01|1.1e+04| 6.481714e+03 -3.001959e+02| 0:0:01| chol 1 1
15|0.113|0.144|4.0e-07|1.1e-01|1.0e+04| 6.446485e+03 -3.309733e+02| 0:0:01| chol 1 1
16|0.194|0.526|3.2e-07|5.1e-02|8.2e+03| 6.228793e+03 -2.380634e+02| 0:0:01| chol 1 1
17|0.406|0.121|1.9e-07|4.5e-02|7.6e+03| 5.761819e+03 -2.627605e+02| 0:0:01| chol 1 1
18|0.912|0.898|1.6e-08|4.6e-03|3.7e+03| 3.401701e+03 -1.722020e+02| 0:0:01| chol 1 1
19|0.987|0.449|1.9e-08|2.5e-03|2.9e+03| 2.673912e+03 -1.367530e+02| 0:0:01| chol 1 1
20|0.613|0.467|7.4e-07|1.3e-03|2.2e+03| 2.057303e+03 -1.380605e+02| 0:0:01| chol 1 1
21|1.000|0.270|8.4e-08|9.8e-04|9.7e+02| 8.507979e+02 -1.082213e+02| 0:0:01| chol 1 1
22|1.000|0.591|3.2e-07|4.0e-04|4.4e+02| 3.979602e+02 -4.244095e+01| 0:0:01| chol 1 2
23|1.000|0.333|2.4e-08|2.7e-04|2.4e+02| 2.161537e+02 -2.511164e+01| 0:0:01| chol 2 2
24|1.000|0.424|1.2e-07|1.5e-04|1.4e+02| 1.308791e+02 -8.842995e+00| 0:0:01| chol 2 2
25|0.945|0.360|2.8e-07|9.9e-05|8.2e+01| 8.192363e+01 7.464416e-02| 0:0:01| chol 2 2
26|0.323|0.104|7.6e-07|8.8e-05|8.0e+01| 8.262994e+01 2.487816e+00| 0:0:01| chol 2 2
27|1.000|0.309|8.8e-08|6.1e-05|6.0e+01| 6.753539e+01 7.158838e+00| 0:0:01| chol 2 2
28|0.997|0.307|1.1e-06|4.2e-05|4.1e+01| 5.148179e+01 1.056641e+01| 0:0:01| chol 2 2
29|1.000|0.287|1.1e-06|3.0e-05|3.6e+01| 4.929799e+01 1.338485e+01| 0:0:01| chol 2 2
30|0.717|0.320|6.5e-07|2.1e-05|2.8e+01| 4.364233e+01 1.553924e+01| 0:0:01| chol 2 2
31|0.900|0.260|1.1e-06|1.5e-05|2.2e+01| 3.939470e+01 1.730264e+01| 0:0:01| chol 2 2
32|0.710|0.295|8.0e-07|1.1e-05|1.8e+01| 3.663586e+01 1.894426e+01| 0:0:01| chol 2 2
33|0.751|0.220|8.2e-07|8.4e-06|1.6e+01| 3.586345e+01 1.988955e+01| 0:0:01| chol 2 2
34|0.898|0.303|9.9e-07|5.9e-06|1.2e+01| 3.253502e+01 2.096587e+01| 0:0:02| chol 3 2
35|0.947|0.301|4.2e-06|4.2e-06|9.1e+00| 3.095766e+01 2.181210e+01| 0:0:02| chol 3 3
36|0.711|0.163|4.5e-06|3.6e-06|8.6e+00| 3.078727e+01 2.218522e+01| 0:0:02| chol 2 2
37|0.898|0.401|3.3e-06|2.4e-06|6.9e+00| 2.981566e+01 2.295968e+01| 0:0:02| chol 2 2
38|1.000|0.907|1.7e-06|8.1e-07|3.9e+00| 2.835032e+01 2.448367e+01| 0:0:02| chol 2 3
39|0.614|0.607|1.8e-06|6.6e-07|3.0e+00| 2.787741e+01 2.484927e+01| 0:0:02| chol 2 2
40|0.808|0.603|9.6e-07|6.2e-07|2.0e+00| 2.713362e+01 2.517748e+01| 0:0:02| chol 2 2
41|1.000|0.457|1.8e-06|5.3e-07|1.4e+00| 2.674433e+01 2.533737e+01| 0:0:02| chol 2 3
42|0.811|0.428|1.0e-06|5.9e-07|1.1e+00| 2.658512e+01 2.547070e+01| 0:0:02| chol 3 2
43|1.000|0.581|1.0e-06|4.5e-07|6.3e-01| 2.631364e+01 2.568104e+01| 0:0:02| chol 2 3
44|1.000|0.553|9.7e-07|4.0e-07|3.6e-01| 2.618493e+01 2.582474e+01| 0:0:02| chol 2 2
45|0.985|0.749|2.1e-07|2.9e-07|1.1e-01| 2.608998e+01 2.597621e+01| 0:0:02| chol 2 2
46|0.977|0.925|6.9e-08|1.6e-07|9.0e-03| 2.605843e+01 2.604962e+01| 0:0:02| chol 2 2
47|1.000|0.830|9.9e-11|5.4e-08|2.5e-03| 2.605734e+01 2.605483e+01| 0:0:02| chol 2 1
48|1.000|0.960|8.6e-10|6.2e-08|2.5e-04| 2.605670e+01 2.605645e+01| 0:0:02| chol 2 2
49|1.000|0.986|9.9e-10|8.4e-09|1.1e-05| 2.605661e+01 2.605660e+01| 0:0:02| chol 2 2
50|1.000|0.988|2.9e-09|3.7e-10|3.1e-07| 2.605661e+01 2.605661e+01| 0:0:02|

```

```

stop: max(relative gap, infeasibilities) < 1.49e-08

```

```

-----
number of iterations      = 50
primal objective value    = 2.60566084e+01
dual  objective value    = 2.60566081e+01
gap := trace(XZ)          = 3.11e-07
relative gap              = 5.86e-09
actual relative gap       = 5.20e-09
rel. primal infeas (scaled problem) = 2.85e-09
rel. dual      "      "      "      = 3.70e-10
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual      "      "      "      = 0.00e+00
norm(X), norm(y), norm(Z) = 1.2e+02, 4.3e+00, 1.2e+01
norm(A), norm(b), norm(C) = 3.1e+04, 2.5e+01, 1.3e+01
Total CPU time (secs)    = 2.23
CPU time per iteration   = 0.04
termination code         = 0
DIMACS: 3.5e-08  0.0e+00  3.2e-09  0.0e+00  5.2e-09  5.9e-09
-----

```

## 11. Assignment 60

### SVM Classifier With Soft Margin

---

```
Status: Solved
Optimal value (cvx_optval): +26.0566
Optimal values of w:
-0.8198
-0.0513
 0.1334
-0.0072
 0.1685
 0.1956
 0.4636
 0.2441
 0.2405
 0.0285
 0.0634
-0.6870
-0.1684
 0.0513
 0.0307
-0.0102
 0.0681
 0.0343
 0.0488
-0.0057
 0.1978
 0.2115
 0.0467
 0.0048
 0.3158
 0.6267
 1.1936
 0.4499
 0.6727
 0.0946
Optimal values of gamma:
14.4276
Classification Accuracy: 0.90158
```

## RESULTS

1. Mathematical concepts and formulation of LPP model for SVM classifier is revisited.
2. Computationally solve classification problems using LPP model of SVM soft margin classifier with and without regularization.
3. SVM classifier with soft margin and regularization produces better classification accuracy.

# 12 | Assignment 61

## Linear Programming for Signal Processing

### 12.1 Introduction

In many signal processing applications, smooth signals are often corrupted by abrupt random spikes. Linear programming allows us to effectively separate these two components.

- **Smooth Signal:** Slowly varying component, often representing the low-frequency part.
- **Spiky Signal:** Abrupt changes or noise, representing high-frequency components or outliers.

The smooth part of the signal can be expressed as a **linear combination of DCT bases**, and the spiky part can be expressed as a **linear combination of columns of the identity matrix**. By constructing an overcomplete dictionary and applying sparsity constraints through  $\ell_1$ -norm penalty functions, we can decompose the noisy signal into its smooth and spiky components.

### 12.2 Overcomplete Dictionary: Combining DCT and Identity Matrix

#### 12.2.1 DCT Bases for Smooth Signal Representation

The **Discrete Cosine Transform (DCT)** is widely used in signal processing to represent smooth, low-frequency components of a signal. The DCT basis functions are cosine waves of varying frequencies. Given a signal  $S \in \mathbb{R}^{128}$ , we construct a matrix  $D \in \mathbb{R}^{128 \times 128}$ , where each column represents a DCT basis function. The smooth part of the signal is expressible as a **linear combination of these DCT bases**:

$$S_{\text{smooth}} = D\alpha_{\text{smooth}}$$

where  $\alpha_{\text{smooth}}$  is the coefficient vector associated with the smooth part.

#### 12.2.2 Identity Matrix for Spiky Signal Representation

The **spiky part** of the signal can be expressed as a linear combination of columns of the **identity matrix**. Each column of the identity matrix represents a unit spike at a specific position. Let  $I \in \mathbb{R}^{128 \times 128}$  represent the identity matrix. The spiky component of the signal is then:

$$S_{\text{spiky}} = I\alpha_{\text{spiky}}$$

where  $\alpha_{\text{spiky}}$  is the coefficient vector representing the spiky part of the signal.

---

### 12.2.3 Overcomplete Dictionary Construction

To separate the smooth and spiky components simultaneously, we concatenate the DCT basis matrix  $D$  and the identity matrix  $I$  to form an **overcomplete dictionary**:

$$B = [D \ I]$$

where  $B \in \mathbb{R}^{128 \times 256}$ . This matrix contains more basis vectors than necessary, allowing for multiple possible representations of the signal.

Thus, the noisy, corrupted signal  $f$  can be expressed as:

$$S = B\alpha$$

where  $\alpha \in \mathbb{R}^{256}$  is the coefficient vector combining both smooth and spiky parts.

## 12.3 Sparse Representation Using $\ell_1$ -Norm

Given that the overcomplete dictionary  $B$  provides infinite possible representations of  $f$ , we introduce a **sparsity constraint** by using the  $\ell_1$ -norm on the coefficient vector  $\alpha$ .

### 12.3.1 $\ell_1$ -Norm for Sparsity

The  $\ell_1$ -norm is widely used for inducing sparsity in linear programming problems. By minimizing the  $\ell_1$ -norm of  $\alpha$ , we encourage most of the coefficients to be zero:

$$\min_{\alpha} \|\alpha\|_1 \quad \text{subject to} \quad S = B\alpha$$

This ensures that the signal is represented by only a few DCT bases (for the smooth part) and a few identity matrix columns (for the spiky part).

### 12.3.2 Interpretation of Coefficients

- The **first 128 coefficients** of  $\alpha$  correspond to the DCT bases, representing the **smooth part** of the signal.
- The **remaining 128 coefficients** correspond to the identity matrix columns, representing the **spiky part** of the signal.

This formulation allows us to efficiently separate the smooth and spiky components.

## 12.4 Signal Generation

Let us fix the signal length as  $n = 128$ , assuming we are sampling 1 second of a signal. The following steps describe the signal generation process:

#### 1. Signal Generation:

- Define the angle  $\Theta$  as:

$$\Theta = \frac{\pi}{2}n + (0 : n - 1) \cdot \frac{2\pi}{n}$$

- Define the frequencies:

$$\text{Fre1} = 12, \quad \text{Fre2} = 25$$

- Generate the signal  $S$ :

$$S = 5 \cos(\text{Fre1} \cdot \Theta) + 3 \cos(\text{Fre2} \cdot \Theta)$$

## 2. Add Spikes at Specific Locations:

- Define spike locations:

$$\text{Loc1} = 23, \quad \text{Loc2} = 48, \quad \text{Loc3} = 96$$

- Add spikes to the signal:

$$S(\text{Loc1}) = S(\text{Loc1}) + 15$$

$$S(\text{Loc2}) = S(\text{Loc2}) + 15$$

$$S(\text{Loc3}) = S(\text{Loc3}) + 15$$

- Convert  $S$  into a column vector:

$$S = S'$$

## 12.5 Overcomplete Dictionary Construction

### 12.5.1 Create the Overcomplete Dictionary

To separate the smooth and spiky components, we construct an overcomplete dictionary:

- Create the DCT basis matrix:

$$\text{dctB} = \text{dctmtx}(n)'$$

or

$$\text{dctB} = \text{dct}(\text{eye}(n))'$$

- Form the overcomplete dictionary:

$$B = [\text{dctB} \quad \text{eye}(n)]$$

where  $B$  is a  $128 \times 256$  matrix.

## 12.6 Sparse Coefficient Representation

### 12.6.1 Find Sparse Set of Coefficient Representation

Using linear programming, we find the sparse coefficient representation of the signal:

```
cvx_begin
variable c(2*n); % we have 2*n bases
minimize sum(abs(c))
subject to
    B*c == S;
cvx_end
```

## 12.7 Interpretation of Results

The coefficient vector  $c$  can be divided into two parts:

- The **first 128 coefficients** correspond to the DCT bases, representing the **smooth part** of the signal.
- The **remaining 128 coefficients** correspond to the identity matrix columns, representing the **spiky part** of the signal.

### 12.7.1 Extracting Smooth and Spiky Parts

We can separate the smooth and spiky components as follows:

- Smooth part:

$$\text{SmoothPart} = \text{dctB} \cdot c(1:128)$$

- Spiky part:

$$\text{SpikyPart} = c(129:\text{end})$$

## 12.8 Computational Experiment

In this experiment, we demonstrate the decomposition of a noisy signal into smooth and spiky parts using an overcomplete dictionary and sparse representation.

Matlab code for demonstrating the above example is given below.

```
1 close(gcf);
2 %1. Signal generation
3 n=128;
4 Theta=(pi/2*n)+(0:n-1)*2*pi/n;
5 Fre1=12;
6 Fre2=15;
7 S= 5*cos(Fre1*Theta)+3*cos(Fre2*Theta); % signal generation
8 %2. Add spike at some location
9 figure
10 subplot(6,1,1);
11 plot(1:128,S);
12 title('original smooth');
13 Loc1= 23;
14 Loc2=48;
15 Loc3=96;
16 S(Loc1)=S(Loc1)+15; % add spikes
17 S(Loc2)=S(Loc2)+15;% add spikes
18 S(Loc3)= S(Loc3)+15;% add spikes
19 S=S'; % convert into a column vector
20 subplot(6,1,2);
21 plot(1:128,S);
22 title('original smooth+spike');
23 %3. Create ovecomplete dictionary
24 dctB=dct(eye(n))';
25 B=[dctB eye(n)]; % B is a 128 by 256 matrix
26 %4. Find sparse set of coefficient representation
27 cvx_begin
28 variable c(2*n);
29 minimize sum(abs(c))
30 subject to
31     B*c==S;
32     cvx_end
33 %5. Plot coefficient
34 Smoothpart=dctB*c(1:128);
35 Spikypart=c(129:end);%
36
37 subplot(6,1,3)
38 plot(1:128,Smoothpart)
```



## 12. Assignment 61

### Linear Programming for Signal Processing

```

39 title('smooth retrieved');
40 subplot(6,1,4)
41 plot(1:128,Spikypart)
42 title('spike retrieved');
43 % original spike alone
44 sp=zeros(1,128);
45 sp(Loc1)=15;
46 sp(Loc2)=15;
47 sp(Loc3)=15;
48 subplot(6,1,5)
49 plot(1:128,sp)
50 title('original spike alone')
51
52 subplot(6,1,6)
53 stem(1:256,c)
54 title('coefficients')

```

Output of the above code is shown below.

Calling SDPT3 4.0: 512 variables, 128 equality constraints

```

-----
num. of constraints = 128
dim. of socp var = 512, num. of socp blk = 256
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT 1 0.000 1 0
it pstep dstep pinfeas dinfeas gap prim-obj dual-obj cputime
-----
0|0.000|0.000|9.8e-01|1.5e+01|5.7e+04| 3.324946e+03 0.000000e+00| 0:0:00| chol 2 2
1|1.000|0.826|4.3e-08|2.7e+00|1.5e+04| 4.001842e+03 2.015534e+02| 0:0:00| chol 1 1
2|1.000|0.986|3.8e-08|4.6e-02|2.3e+03| 2.248121e+03 1.038402e+02| 0:0:00| chol 1 1
3|0.868|0.677|3.6e-08|1.6e-02|3.7e+02| 4.879206e+02 1.266831e+02| 0:0:00| chol 1 1
4|1.000|0.635|3.0e-08|5.8e-03|8.5e+01| 2.254169e+02 1.421035e+02| 0:0:00| chol 1 1
5|0.958|0.722|1.9e-09|1.6e-03|2.4e+01| 1.730069e+02 1.490702e+02| 0:0:00| chol 1 1
6|0.771|0.976|4.8e-10|4.0e-05|8.1e+00| 1.602722e+02 1.521949e+02| 0:0:00| chol 1 1
7|1.000|0.945|8.6e-10|2.3e-06|1.6e+00| 1.542696e+02 1.526656e+02| 0:0:00| chol 1 1
8|0.950|0.677|2.6e-09|7.4e-07|3.5e-01| 1.531333e+02 1.527797e+02| 0:0:00| chol 1 1
9|1.000|0.679|5.4e-09|2.4e-07|9.7e-02| 1.529298e+02 1.528325e+02| 0:0:00| chol 1 1
10|0.591|0.761|2.2e-09|5.7e-08|4.6e-02| 1.528995e+02 1.528537e+02| 0:0:00| chol 2 2
11|0.793|0.828|4.6e-10|1.0e-08|1.2e-02| 1.528728e+02 1.528608e+02| 0:0:00| chol 2 2
12|0.905|0.933|4.9e-11|7.8e-10|1.6e-03| 1.528643e+02 1.528627e+02| 0:0:00| chol 3 3
13|0.987|0.607|1.1e-07|3.2e-10|3.5e-04| 1.528631e+02 1.528628e+02| 0:0:00| chol 3 3
14|0.935|0.697|7.4e-08|1.1e-10|1.1e-04| 1.528630e+02 1.528629e+02| 0:0:00| chol 5 5
15|1.000|0.799|5.0e-08|4.5e-11|2.8e-05| 1.528629e+02 1.528629e+02| 0:0:00| chol 6 7
16|0.654|0.811|1.7e-08|4.2e-11|1.2e-05| 1.528629e+02 1.528629e+02| 0:0:00| chol 12 13
17|1.000|0.935|1.8e-08|5.2e-11|1.4e-06| 1.528629e+02 1.528629e+02| 0:0:00| chol
warning: symqmr failed: 0.3
switch to LU factor. lu 18 ^10
18|0.572|0.983|2.0e-08|7.6e-11|6.6e-07| 1.528629e+02 1.528629e+02| 0:0:00| lu 11 ^29
19|0.453|0.285|1.4e-07|1.7e-10|4.6e-07| 1.528629e+02 1.528629e+02| 0:0:00| lu 11 ^19
20|0.240|0.150|9.6e-08|3.1e-10|4.1e-07| 1.528629e+02 1.528629e+02| 0:0:00|
lack of progress in infeas
-----
number of iterations = 20

```

```

primal objective value = 1.52862931e+02
dual  objective value = 1.52862932e+02
gap := trace(XZ)      = 1.45e-06
relative gap          = 4.72e-09
actual relative gap   = -3.25e-09
rel. primal infeas (scaled problem) = 1.80e-08
rel. dual    "      "      "      = 5.25e-11
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual    "      "      "      = 0.00e+00
norm(X), norm(y), norm(Z) = 7.5e+01, 7.5e+00, 1.9e+01
norm(A), norm(b), norm(C) = 1.7e+01, 5.5e+01, 1.7e+01
Total CPU time (secs) = 0.28
CPU time per iteration = 0.01
termination code      = 0
DIMACS: 4.4e-08  0.0e+00  4.5e-10  0.0e+00  -3.3e-09  4.7e-09

```

```

-----
Status: Solved
Optimal value (cvx_optval): +152.863

```

A visualization of complete extraction and comparison is shown in Figure 12.1.

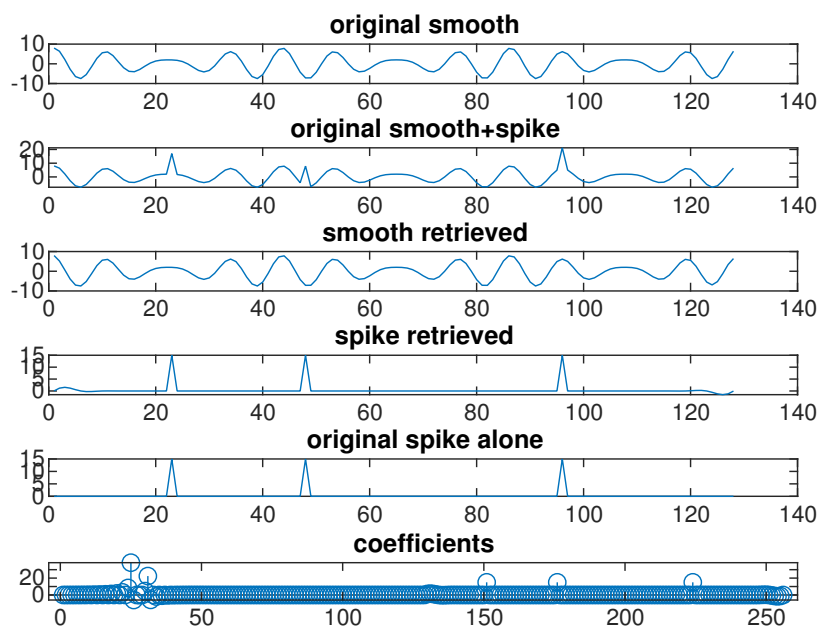


Figure 12.1: Comparison of extraction of spiky components using optimization of LPP model

## 12.9 Task

Repeat the experiment with Sine bases.

### SOLUTION

Matlab code for this task is given below.

```

1 %% Task 1: Signal Generation with Sine Bases
2 % 1. Signal generation
3 n = 128; % Signal length

```

```
4  Theta = (0:n-1)' * (2*pi/n); % Angle for signal generation (column
    vector)
5  Fre1 = 12;
6  Fre2 = 15;
7  S = 5*cos(Fre1*Theta) + 3*cos(Fre2*Theta); % Generate original
    smooth signal
8
9  % 2. Add spikes at specified locations
10 Loc1 = 23;
11 Loc2 = 48;
12 Loc3 = 96;
13 S(Loc1) = S(Loc1) + 15; % Add spike
14 S(Loc2) = S(Loc2) + 15; % Add spike
15 S(Loc3) = S(Loc3) + 15; % Add spike
16 %S = S'; % Convert into a column vector
17
18 % Plot original smooth and corrupted signals
19 figure
20 subplot(6,1,1);
21 plot(1:128, S);
22 title('Original Smooth + Spike');
23
24 %% Task 2: Create Overcomplete Dictionary with Sine Bases
25 % Create sine bases (no zero vectors)
26 frequencies = 1:128; % Use more frequencies to match dimensions
27 sineB = zeros(n, length(frequencies)); % Initialize sine basis
    matrix
28
29 % Generate sine basis columns
30 for i = 1:length(frequencies)
31     sineB(:, i) = sin((1:n)' * (2 * pi * frequencies(i) / n)); %
        Correct sine basis generation
32 end
33
34 % Append identity matrix to form overcomplete dictionary
35 B_sine = [sineB, eye(n)]; % B_sine is a 128 x (128 + 128) = 128 x
    256 matrix
36
37 %% Task 3: Find Sparse Coefficient Representation with Sine Bases
38 % 4. Find sparse representation using CVX
39 cvx_begin
40     variable c_sine(size(B_sine, 2)); % Coefficient variable for the
        number of bases
41     minimize(sum(abs(c_sine))); % Minimize L1 norm
42     subject to
43     B_sine * c_sine == S; % Constraint to match the signal
44     cvx_end
45
46 %% Task 4: Retrieve Smooth and Spiky Parts
47 % Smooth part from sine coefficients
48 Smoothpart_sine = sineB * c_sine(1:n);
49 % Spiky part from identity matrix coefficients
```

```

50 Spikypart_sine = c_sine(n+1:end);
51
52 % Plot smooth and spiky components
53 subplot(6,1,2);
54 plot(1:128, Smoothpart_sine);
55 title('Smooth Retrieved with Sine Bases');
56
57 subplot(6,1,3);
58 plot(1:128, Spikypart_sine);
59 title('Spiky Retrieved with Sine Bases');
60
61 % Original spike alone for comparison
62 sp = zeros(1,128);
63 sp(Loc1) = 15;
64 sp(Loc2) = 15;
65 sp(Loc3) = 15;
66
67 subplot(6,1,4);
68 plot(1:128, sp);
69 title('Original Spike Alone');
70
71 % Coefficients Plot
72 subplot(6,1,5);
73 stem(1:length(c_sine), c_sine);
74 title('Coefficients with Sine Bases');

```

Output of the code is shown below.

Calling SDPT3 4.0: 512 variables, 128 equality constraints

```

-----
num. of constraints = 128
dim. of socp var = 512, num. of socp blk = 256
*****
SDPT3: Infeasible path-following algorithms
*****
version predcorr gam expon scale_data
NT 1 0.000 1 0
it pstep dstep pinfeas dinfeas gap prim-obj dual-obj cputime
-----
0|0.000|0.000|9.8e-01|1.5e+01|2.3e+04| 1.335959e+03 0.000000e+00| 0:0:00| chol 1 1
1|1.000|0.757|2.2e-06|3.7e+00|9.1e+03| 1.945483e+03 5.641972e+02| 0:0:00| chol 1 1
2|1.000|1.000|1.4e-06|9.4e-03|1.1e+03| 1.325594e+03 2.493474e+02| 0:0:00| chol 1 1
3|0.885|0.901|5.5e-07|1.8e-03|1.3e+02| 4.876231e+02 3.546714e+02| 0:0:00| chol 1 1
4|0.677|0.732|1.9e-07|5.5e-04|5.2e+01| 4.312010e+02 3.791701e+02| 0:0:00| chol 1 1
5|0.996|0.701|3.3e-09|1.7e-04|1.5e+01| 4.011484e+02 3.863982e+02| 0:0:00| chol 1 1
6|0.621|0.835|1.8e-09|2.9e-05|7.1e+00| 3.972495e+02 3.901883e+02| 0:0:00| chol 1 1
7|0.742|0.810|1.6e-10|5.6e-06|2.8e+00| 3.942565e+02 3.914607e+02| 0:0:00| chol 1 1
8|0.776|0.869|2.8e-10|7.4e-07|1.2e+00| 3.930636e+02 3.919098e+02| 0:0:00| chol 1 1
9|0.805|0.873|2.5e-10|9.5e-08|3.2e-01| 3.924597e+02 3.921426e+02| 0:0:00| chol 1 1
10|0.897|0.999|2.6e-11|2.1e-10|7.9e-02| 3.922828e+02 3.922035e+02| 0:0:00| chol 1 1
11|0.967|0.835|8.4e-13|4.8e-11|5.9e-03| 3.922245e+02 3.922186e+02| 0:0:00| chol 1 1
12|0.975|0.982|2.2e-14|1.8e-12|1.3e-04| 3.922219e+02 3.922218e+02| 0:0:00| chol 1 1
13|0.992|0.992|9.5e-15|1.0e-12|1.9e-06| 3.922218e+02 3.922218e+02| 0:0:00|
stop: max(relative gap, infeasibilities) < 1.49e-08
-----

```

```

number of iterations    = 13
primal objective value = 3.92221812e+02
dual  objective value = 3.92221810e+02
gap := trace(XZ)       = 1.87e-06
relative gap           = 2.38e-09
actual relative gap    = 2.38e-09
rel. primal infeas (scaled problem) = 9.53e-15
rel. dual    "         "         "   = 1.02e-12
rel. primal infeas (unscaled problem) = 0.00e+00
rel. dual    "         "         "   = 0.00e+00
norm(X), norm(y), norm(Z) = 7.4e+01, 1.1e+01, 2.1e+01
norm(A), norm(b), norm(C) = 9.2e+01, 5.5e+01, 1.7e+01
Total CPU time (secs) = 0.14
CPU time per iteration = 0.01
termination code      = 0
DIMACS: 2.3e-14  0.0e+00  8.6e-12  0.0e+00  2.4e-09  2.4e-09

```

```

-----
Status: Solved
Optimal value (cvx_optval): +392.222

```

A visualization of complete extraction and comparison is shown in Figure 12.2.

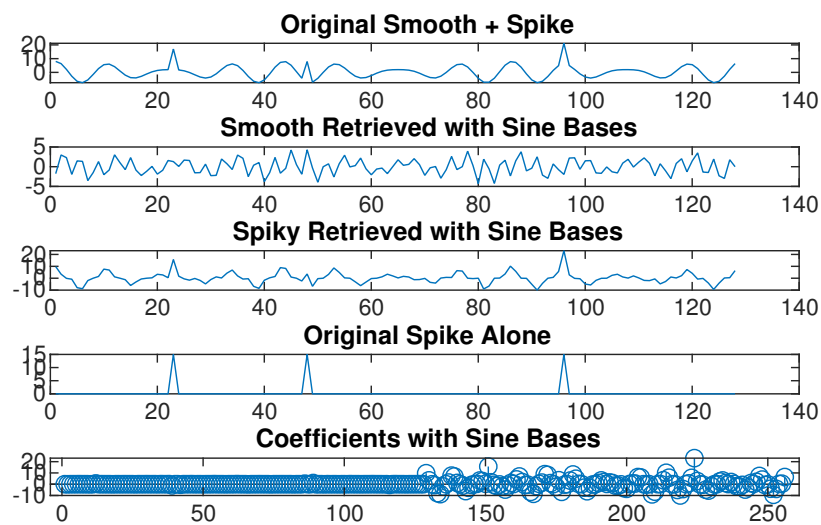


Figure 12.2: Comparison of extraction of spiky components with *sine* bases

## RESULTS

1. Fundamentals of extraction of noise from corrupted signals using LPP approach is revisited.
2. Spikes from corrupted signals are extracted using DCT bases and sine bases with optimization of LPP model.



# 13 | Assignment 62

## Verify Karush-Khun-Tucker Conditions for LP Problems Using CVX

### 13.1 Introduction

In the “bad olden days” it was very difficult to check KKT condition of a real world problems because KKT condition involves both primal and dual (Lagrangian multipliers) variables. To get the dual variables we need to solve a separate problem. Then, again, to get dual variables corresponding to non-negativity constraints, we need to do further computations based on Lagrangian function corresponding to primal problem. In Assignment No: 56, KKT conditions were tested based on above procedure.

Fortunately, modern solvers like CVX is solving using ADMM like algorithm which get primal and all dual variables in one go. In this assignment we demonstrate this capability of CVX.

#### 13.1.1 Example

Consider the primal problem,

$$\begin{aligned} \text{Max}_X \quad & f(X) = 5x_1 + 6x_2 + 9x_3 + 8x_4 \\ \text{Subject to:} \\ & x_1 + 2x_2 + 3x_3 + x_4 \leq 5 \\ & x_1 + x_2 + 2x_3 + 3x_4 \leq 3 \\ & x_i \geq 0; \quad i = 1, 2, 3, 4 \end{aligned}$$

Lagrangian formulation of the above LPP is.

$$\begin{aligned} \mathcal{L}(X, Y, \Lambda) = & 5x_1 + 6x_2 + 9x_3 + 8x_4 + y_1 (5 - (x_1 + 2x_2 + 3x_3 + x_4)) + y_2 (3 - (x_1 + x_2 + 2x_3 + 3x_4)) \\ & + \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 + \lambda_4 y_4 \\ & y_1, y_2, \lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0 \end{aligned}$$

The complementary condition are given by

$$\begin{aligned} y_1 (5 - (x_1 + 2x_2 + 3x_3 + x_4)) &= 0 \\ y_2 (3 - (x_1 + x_2 + 2x_3 + 3x_4)) &= 0 \\ \lambda_1 x_1 &= 0 \\ \lambda_2 x_2 &= 0 \\ \lambda_3 x_3 &= 0 \\ \lambda_4 x_4 &= 0 \end{aligned}$$

---

Now the dual problem is given by

$$\text{Min}_Y \quad g(Y) = 5y_1 + 3y_2$$

Subject to:

$$y_1 + y_2 \geq 5$$

$$2y_1 + y_2 \geq 6$$

$$3y_1 + 2y_2 \geq 9$$

$$y_1 + 3y_2 \geq 8$$

$$y_i \geq 0; \quad i = 1, 2$$

We can use CVX to check complementarity condition. CVX internally uses ADMM. Therefore, when we ask for solving primal in CVX, it is solving primal and dual simultaneously based on Lagrangian function. So, after solving, it can give both primal and dual variable (Lagrangian multipliers) values at the optimal point.

Following matlab code demonstrate the way of getting both by specifying only primal problem. The code also demonstrates how to suppress usual CVX performance outputs.

```

1  m = 2; % Matrix A is 2x4
2  n = 4;
3  A = [1 2 3 1; 1 1 2 3];
4  b = [5 3]'; % must be a column vector
5  c = [5 6 9 8]'; % must be a column vector
6  cvx_begin quiet % suppress cvx process outputs
7      variables x(n)
8      dual variable y % it won't accept dual variable y,z
9      dual variable z
10     maximize(c' * x)
11     subject to
12         y: A * x <= b; % y: generates dual variable(Lagrangian
13             multiplier for each constraint)
14         z: x >= 0; % z: generates dual variable(Lagrangian
15             multiplier for each constraint)
16 cvx_end
17 format bank; % Display values in two decimals
18 primal_optimal_value=c'*x
19 dual_optimal_value=b'*y
20 disp('primal variable x values:')
21 disp(x);
22 disp('main dual variable y values:')
23 disp(y);
24 disp('secondary dual variable z values:')
25 disp(z)
26 main_copm_cond_values=y.*(A*x-b)
27 sec_copm_cond_values=z.*x

```

Output of the code is shown below.

```

primal_optimal_value =
    17.00
dual_optimal_value =
    17.00
primal variable x values:

```



```

1.00
2.00
0.00
0.00
main dual variable y values:
1.00
4.00
secondary dual variable z values:
0.00
0.00
2.00
5.00
Main Complementary Condition Values:
-0.00
-0.00
Secondary Complementary Condition Values:
0.00
0.00
0.00
0.00

```

## 13.2 Tasks

Find primal and dual variables at the optima point . Also check KKT conditions.

1.

$$\text{Max}_{x,y} \quad P = 4x + 4y$$

S.to:

$$x + 3y \leq 30$$

$$2x + y \leq 20$$

$$x, y \geq 0$$

### SOLUTION

The Lagrangian function can be written as:

$$\mathcal{L}(x, y, Y, \Lambda) = 4x + 4y + y_1(30 - x - 3y) + y_2(20 - 2x - y) + \lambda_1 x + \lambda_2 y$$

The complementary conditions are:

$$y_1(30 - x - 3y) = 0$$

$$y_2(20 - 2x - y) = 0$$

$$\lambda_1 x = 0$$

$$\lambda_2 y = 0$$

Dual problem is given by:

$$\underset{y}{\text{Min}} \quad Q = 30y_1 + 20y_2$$

S.to:

$$y_1 + 2y_2 \geq 4$$

$$3y_1 + y_2 \geq 4$$

$$y_1, y_2 \geq 0$$

Matlab code for solving this problem computationally is given below.

```

1  m = 2; % Matrix A is 2X2
2  n = 2;
3  A = [1 3; 2 1];
4  b = [30 20]'; % must be a column vector
5  c = [ 4 4]'; % must be a column vector
6  cvx_begin quiet % suppress cvx process outputs
7      variables x(n)
8      dual variable y % it won't accept dual variable y,z
9      dual variable z
10     maximize(c' * x)
11     subject to
12         y: A * x <= b;
13         z: x >= 0;
14 cvx_end
15 format bank; % Display values in two decimals
16 primal_optimal_value=c'*x
17 dual_optimal_value=b'*y
18 disp('primal variable x values:')
19 disp(x);
20 disp('main dual variable y values:')
21 disp(y);
22 disp('secondary dual variable z values:')
23 disp(z)
24 disp('Main Complementary Condition Values:');
25 disp(y.*(A*x-b));
26 disp('Secondary Complementary Condition Values:');
27 disp(z.*x);

```

Output of the code is shown below.

```

primal_optimal_value =
    56.00
dual_optimal_value =
    56.00
primal variable x values:
    6.00
    8.00
main dual variable y values:
    0.80
    1.60
secondary dual variable z values:
   -0.00

```

```

-0.00
Main Complementary Condition Values:
-0.00
-0.00
Secondary Complementary Condition Values:
-0.00
-0.00

```

2.

$$\text{Min}_{x,y} \quad C = 5x + 5y$$

S.to:

$$x + y \geq 6$$

$$6x + y \geq 16$$

$$x + 6y \geq 16$$

**SOLUTION**

```

1      m = 3; % Matrix A is 2x4
2      n = 2;
3      A = [1 1; 6 1; 1 6];
4      b = [6 16 16]'; % must be a column vector
5      c = [ 5 5]'; % must be a column vector
6      cvx_begin quiet % suppress cvx process outputs
7      variables x(n)
8      dual variable y % it won't accept dual variable y,z
9      dual variable z
10     minimize(c' * x)
11     subject to
12     y: A * x >= b; % y: generates dual variable (Lagrangian
        multiplier for each constraint)
13     z: x >= 0; % z: generates dual variable (Lagrangian
        multiplier for each constraint)
14     cvx_end
15     format bank; % Display values in two decimals
16     primal_optimal_value = c' * x
17     dual_optimal_value = b' * y
18     disp('primal variable x values:')
19     disp(x);
20     disp('main dual variable y values:')
21     disp(y);
22     disp('secondary dual variable z values:')
23     disp(z)
24     disp('Main Complementary Condition Values:');
25     disp(y.*(A*x-b));
26     disp('Secondary Complementary Condition Values:');
27     disp(z.*x);

```

```

primal_optimal_value =
30.00

```

```

dual_optimal_value =
30.00
primal variable x values:
3.00
3.00
main dual variable y values:
5.00
0.00
0.00
secondary dual variable z values:
0.00
0.00

Main Complementary Condition Values:
0.00
0.00
0.00
Secondary Complementary Condition Values:
0.00
0.00

```

3.

$$\text{Min}_{x,y} \quad C = x - 2y$$

S.to:

$$x \geq 2$$

$$x \leq 4$$

$$y \geq 1$$

**SOLUTION**

```

1  m = 3; % Matrix A is 2x4
2  n = 2;
3  A = [1 0; -1 0; 0 1];
4  b = [2 -4 1]'; % must be a column vector
5  c = [1 -1]'; % must be a column vector
6  cvx_begin quiet % suppress cvx process outputs
7  variables x(n)
8  dual variable y % it won't accept dual variable y,z
9  %dual variable z
10 minimize(c' * x)
11 subject to
12 y: A * x >= b; % y: generates dual variable(Lagrangian
    multiplier for each constraint)
13 %z: x >= 0; % z: generates dual variable(Lagrangian
    multiplier for each constraint)
14 cvx_end
15 format bank; % Display values in two decimals
16 primal_optimal_value=c'*x
17 dual_optimal_value=b'*y
18 disp('primal variable x values:')
19 disp(x);

```

```

20     disp('main dual variable y values:')
21     disp(y);
22     disp('secondary dual variable z values:')
23     disp(z)
24     disp('Main Complementary Condition Values:');
25     disp(y.*(A*x-b));
26     disp('Secondary Complementary Condition Values:');
27     %disp(z.*x);

```

```

primal_optimal_value =
-1.00
dual_optimal_value =
NaN
primal variable x values:
    0.00
    1.00
main dual variable y values:
    NaN
    NaN
    NaN
secondary dual variable z values:
    0.00
    0.00
Main Complementary Condition Values:
    NaN
    NaN
    NaN

```

## RESULTS

1. The KKT conditions are revisited.
2. KKT conditions for the Linear Programming Problems are verified using CVX solver in matlab.



# 14 | Assignment 63

## Experiments in CVX with LP Problems to Understand Lagrangian Multipliers

### 14.1 Introduction

In this section, we explore the role of **Lagrangian multipliers** (also called **dual variables**) in **Linear Programming (LP)** problems. Using the CVX framework, we will solve LP problems and examine the insights provided by the Lagrangian multipliers at the optimal solution. The focus will be on understanding:

- Which constraints are active at the optimal point.
- The geometric and economic interpretations of the dual variable values.

Lagrangian multipliers not only provide information about the optimal solution but also reveal how the objective function value can change when constraints are relaxed.

### 14.2 Linear Programming and Lagrangian Multipliers

Consider a standard linear programming (LP) problem of the form:

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax \leq b, \\ & && x \geq 0, \end{aligned}$$

where  $x \in \mathbb{R}^n$  is the variable to be optimized,  $c \in \mathbb{R}^n$  is the cost vector,  $A \in \mathbb{R}^{m \times n}$  is a matrix representing the constraint coefficients, and  $b \in \mathbb{R}^m$  is a vector representing the constraint bounds.

To solve this constrained optimization problem, we introduce the **Lagrangian**:

$$\mathcal{L}(x, \lambda) = c^T x + \lambda^T (Ax - b),$$

where  $\lambda \in \mathbb{R}^m$  are the **Lagrangian multipliers**, or dual variables, corresponding to each constraint  $A_i x \leq b_i$ .

At the optimal solution, the value of each Lagrangian multiplier indicates whether the corresponding constraint is **active** or **inactive**:

- If  $\lambda_i > 0$ , the constraint  $A_i x = b_i$  is active at the optimal point (i.e., the optimal solution lies on the boundary defined by the constraint).
  - If  $\lambda_i = 0$ , the constraint is inactive, meaning the optimal point lies within the feasible region for that constraint.
-

The Lagrangian multipliers also provide insight into how much the objective function could improve if the active constraints were relaxed. This makes Lagrangian multipliers important for both geometric and economic interpretations of optimization problems.

### 14.3 Geometric Interpretation of Lagrangian Multipliers

In the context of an LP problem, Lagrangian multipliers help identify the geometry of the optimal solution. If  $\lambda_i > 0$ , the constraint is said to be **binding** or **active**, meaning the optimal point lies on the boundary of the feasible region for that constraint. Geometrically, the optimal point satisfies  $A_i x = b_i$ , indicating that the point is on the constraint's boundary.

If  $\lambda_i = 0$ , the constraint is **inactive**, meaning the optimal solution is not constrained by  $A_i x \leq b_i$  and lies in the interior with respect to this constraint.

### 14.4 Economic Interpretation of Lagrangian Multipliers

The Lagrangian multiplier  $\lambda_i$  can be interpreted as the **shadow price** of the resource associated with constraint  $A_i x \leq b_i$ . In economic terms:

- A positive  $\lambda_i$  indicates how much the objective function could improve if the constraint  $A_i x \leq b_i$  were relaxed slightly.
- A large  $\lambda_i$  suggests that the constraint is significantly limiting the optimization, and relaxing it could result in substantial improvement in the objective function.

This economic interpretation makes Lagrangian multipliers valuable in practical problems where resources are limited, and decision-makers need to understand the marginal value of relaxing constraints.

### 14.5 Experiment 1: Solving an LP Problem Using CVX

#### 14.5.1 Problem Statement

Consider the following linear programming problem:

$$\begin{aligned} & \text{minimize} && f(x_1, x_2) = x_1 + 2x_2 \\ & \text{subject to:} && x_1 + x_2 \geq 1, \\ & && 2x_1 + x_2 \leq 4, \\ & && x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

We will solve this problem using CVX and extract the Lagrangian multipliers associated with each constraint.

#### 14.5.2 CVX Code

```
cvx_begin
    variables x1 x2
    dual variable lambda1
    dual variable lambda2
    minimize( x1 + 2*x2 )
    subject to
        lambda1: x1 + x2 >= 1    % Dual variable for the first constraint
        lambda2: 2*x1 + x2 <= 4  % Dual variable for the second constraint
        x1 >= 0
```



```

        x2 >= 0
cvx_end

% Display the solution
disp('Optimal solution:')
disp(['x1 = ', num2str(x1), ', x2 = ', num2str(x2)])

% Display the dual variables (Lagrangian multipliers)
disp('Lagrangian multipliers:')
disp(['lambda1 = ', num2str(lambda1)])
disp(['lambda2 = ', num2str(lambda2)])

```

Output of this code is shown below.

```

Optimal solution:
x1 = 1, x2 = 2.9421e-10
Lagrangian multipliers:
lambda1 = 1
lambda2 = 2.098e-10

```

### 14.5.3 Discussion

1. **Active and Inactive Constraints:** After solving the problem, observe which constraints have non-zero Lagrangian multipliers. These are the active constraints that define the geometry of the optimal solution. So, the first constraint is active. Any constraints with zero Lagrangian multipliers are inactive. So, the second constraint is inactive.
2. **Geometric Interpretation:** If  $\lambda_i > 0$ , the corresponding constraint is active, and the optimal point lies on the boundary defined by that constraint. If  $\lambda_i = 0$ , the optimal point lies within the feasible region for that constraint, not on its boundary.
3. **Improvement in the Objective Function:** The value of  $\lambda_i$  also indicates how much the objective function value can improve if the active constraint is relaxed. A larger  $\lambda_i$  means a more significant potential improvement. So, the first constraint has a unit weight and has a positive impact on the optimum solution.

## 14.6 Experiment 2: Economic Interpretation of Lagrangian Multipliers

### 14.6.1 Problem Statement

Consider another LP problem where the goal is to maximize the objective function:

$$\begin{aligned}
 &\text{maximize} && f(x_1, x_2) = 3x_1 + 4x_2 \\
 &\text{subject to} && x_1 + 2x_2 \leq 6, \\
 & && 2x_1 + x_2 \leq 8, \\
 & && x_1 \geq 0, x_2 \geq 0.
 \end{aligned}$$

Matlab code for this task is shown below.

```

cvx_begin
    variables x1 x2
    dual variable lambda1
    dual variable lambda2
    maximize( 3*x1 + 4*x2 )

```

```

subject to
    lambda1: x1 + 2*x2 <= 6      % Dual variable for the first constraint
    lambda2: 2*x1 + x2 <= 8     % Dual variable for the second constraint
    x1 >= 0
    x2 >= 0
cvx_end

% Display the solution
disp('Optimal solution:')
disp(['x1 = ', num2str(x1), ', x2 = ', num2str(x2)])

% Display the dual variables (Lagrangian multipliers)
disp('Lagrangian multipliers:')
disp(['lambda1 = ', num2str(lambda1)])
disp(['lambda2 = ', num2str(lambda2)])

```

Output of the code is shown below.

```

Optimal solution:
x1 = 3.3333, x2 = 1.3333
Lagrangian multipliers:
lambda1 = 1.6667
lambda2 = 0.66667

```

### 14.6.2 Discussion

1. **Effect of Relaxing Constraints:** Examine the values of the Lagrangian multipliers. A positive value suggests that the corresponding constraint is binding and significantly affects the optimal solution. So, in this problem both the constraint affects the optimum solution significantly.
2. **Shadow Price Interpretation:** The magnitude of the Lagrangian multiplier indicates how much the objective function could improve if the active constraint were relaxed slightly. This is the shadow price of relaxing the resource limitation represented by that constraint. So, in this problem first constraint is more influential in the optimum solution of the problem.

### 14.6.3 Solution using matrix approach

Instead of inputting the equations directly, write the constraints in the standard form and pass the coefficient matrix  $A$ , column of constants,  $b$  and cost coefficients in the objective function,  $c$  to the CVX. An example is shown below.

$$\begin{aligned}
 &\underset{y}{\text{Min}} \quad 5y_1 + 3y_2 \\
 &\text{Subject to: } y_1 + y_2 \geq 5 \\
 &\quad 2y_1 + y_2 \geq 6 \\
 &\quad 3y_1 + 2y_2 \geq 9 \\
 &\quad y_1 + 3y_2 \geq 8 \\
 &\quad y_1, y_2 \geq 0
 \end{aligned}$$

Matlab code for primal and dual variable at the optimal point is shown below.

```

1 n = 2; % Matrix A is 4x2
2 A = [1 2 3 1; 1 1 2 3]';
3 c = [5 3]'; % must be a column vector
4 b = [5 6 9 8]'; % must be a column vector

```

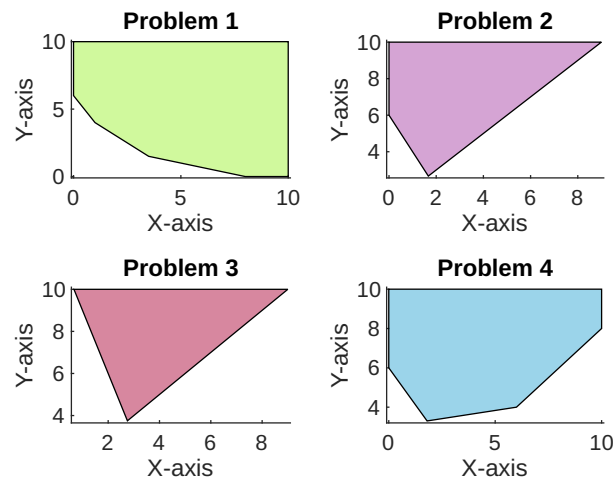


Figure 14.1: Feasible region of LPPs created with plotregion function

```

5  cvx_begin quiet      % suppress cvx process outputs
6      variables y(n)
7      dual variable x
8      dual variable z
9      minimize(c' * y)
10     subject to
11         x: A * y >= b;
12         z: y >= 0
13 cvx_end
14 disp('Solution')
15 disp(y);
16 disp('Lagrange Multipliers')
17 disp(x);

```

Output of the code is shown below.

```

Solution
      2.67
      0.67
Lagrange Multipliers
      1.33
      0.17
      0.00

```

Since the first and second Lagrange multipliers are non-zero, they are active and have a positive impact on the optimum solution.

Matlab code to visualize the feasible region of the LPP is given below.

```

1  A=[1 2; 4 2; -1 1];
2  b=[4 12 1]';
3  lb=[0 0]'; ub=[10 10]';
4  plotregion(A,b,lb,ub)

```

Output is shown in the Problem 1 of Figure 14.1.

## 14.7 Tasks

Draw the feasible region using plotregion function. Find inactive constraints based on dual variables for the following problems.

$$\begin{aligned}
 &\max \quad 2x_1 + 3x_2 \\
 &\text{Subject to: } x_1 + 2x_2 \leq 4 \\
 1. \quad &4x_1 + 2x_2 \leq 12 \\
 &-x_1 + x_2 \leq 1 \\
 &x_1, x_2 \geq 0
 \end{aligned}$$

**SOLUTION**

Matlab code to visualize the feasible region of the given LPP is given below.

```

1 A=[1 2; 4 2; -1 1];
2 b=[4 12 1]';
3 lb=[0 0]'; ub=[10 10]';
4 plotregion(A,b,lb,ub)

```

Output of the code is shown in Problem 2 of Figure 14.1.

Matlab code to solve both primal and dual of the given LPP is given below.

```

1 n = 2; % Matrix A is 4x2
2 A=[1 2; 4 2; -1 1];
3 b=[4 12 1]';
4 c = [2 3]'; % must be a column vector
5 cvx_begin quiet % suppress cvx process outputs
6     variables y(n)
7     dual variable x
8     dual variable z
9     maximize(c' * y)
10    subject to
11        x: A * y <= b;
12        z: y >= 0
13 cvx_end
14 disp('Solution')
15 disp(y);
16 disp('Lagrange Multipliers')
17 disp(x);

```

Output of the code is shown below.

```

Solution
      2.67
      0.67
Lagrange Multipliers
      1.33
      0.17
      0.00

```

It is clear from the result that, the last constraint is inactive in the optimal solution of the LPP.

$$\begin{aligned}
 &\max \quad 5x_1 + 4x_2 \\
 &\text{Subject to: } 3x_1 + 2x_2 \leq 12 \\
 2. \quad &x_1 + 2x_2 \leq 6 \\
 &-x_1 + x_2 \leq 1 \\
 &x_2 \leq 2 \\
 &x_1, x_2 \geq 0
 \end{aligned}$$

**SOLUTION**

Matlab code to visualize the feasible region of the given LPP is given below.

```
1 A=[3 1; 1 2; -1 1;0 1];
2 b=[12 6 1 2]';
3 lb=[0 0]';ub=[10 10]';
4 plotregion(A,b,lb,ub)
```

Output of the code is shown in Problem 3 of Figure 14.1.

Matlab code to solve both primal and dual of the given LPP is given below.

```
1 n = 2; % Matrix A is 4x2
2 A=[3 1;1 2;-1 1;0 1];
3 b=[12 6 1 2]';
4 c = [5 4]'; % must be a column vector
5 cvx_begin quiet % suppress cvx process outputs
6     variables y(n)
7     dual variable x
8     dual variable z
9     maximize(c' * y)
10    subject to
11        x: A * y <= b;
12        z: y >=0
13 cvx_end
14 disp('Solution')
15 disp(y);
16 disp('Lagrange Multipliers')
17 disp(x);
```

Output of the code is shown below.

```
Solution
      3.60
      1.20
Lagrange Multipliers
      1.20
      1.40
      0.00
      0.00
```

It is clear from the result that, the last two constraints are inactive in the optimal solution of the LPP.

$$\begin{aligned} \max \quad & 4x_1 + x_2 \\ \text{Subject to:} \quad & 3x_1 + 2x_2 \geq 12 \\ 3. \quad & 2x_1 - 6x_2 \leq -18 \\ & x_1 - x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

**SOLUTION**

Matlab code to visualize the feasible region of the given LPP is given below.

```
1 A=[3 2;-1 6; -1 1];
2 b=[12 18 -2]'
```

```

3 lb=[0 0]';ub=[10 10]';
4 plotregion(A,b,lb,ub)

```

Output of the code is shown in Problem 4 of Figure 14.1.

Matlab code to solve both primal and dual of the given LPP is given below.

```

1 n = 2; % Matrix A is 4x2
2 A=[3 2;-1 6; -1 1];
3 b=[12 18 -2]'
4 c = [4 1]'; % must be a column vector
5 cvx_begin quiet % suppress cvx process outputs
6     variables y(n)
7     dual variable x
8     dual variable z
9     minimize(c' * y)
10    subject to
11        x: A * y >= b;
12        z: y >=0
13 cvx_end
14 disp('Solution')
15 disp(y);
16 disp('Lagrange Multipliers')
17 disp(x);

```

Output of the code is shown below.

```

Solution
      0.00
      6.00
Lagrange Multipliers
      0.50
      0.00
      0.00

```

It is clear from the result that, the last two constraints are inactive in the optimal solution of the LPP.

## RESULTS

- Lagrangian multipliers play a crucial role in both the geometric and economic interpretations of constrained optimization problems.
- They help identify active constraints and quantify the effect of relaxing those constraints on the objective function.
- Through experiments in CVX, we gain practical insights into the relationship between the primal and dual problems in LP, as well as the importance of dual variables in understanding the optimal solution.

# 15 | Assignment 64

## Hard Margin Support Vector Machine Formulation and Solution by CVX

### 15.1 Introduction

Support Vector Machines (SVMs) are well known supervised learning models used for classification tasks. The hard margin SVM seeks to find a hyperplane that maximally separates two classes of data points while ensuring that the points from each class are on the correct side of the margin.

CVX allows us to get both primal and dual solutions simultaneously. It is very instructive now to understand which point (to be precise which constraints) get nonzero lagrangian multiplier at the optimal solution point. Consider 4 points as given below. Two points belong to class -1 and another two point belongs to class +1. Aim is to find a linear classifier (equation of a plane) that maximally separate the two classes of objects.

Given a dataset with four points can be represented as:

Point ID	$x_1$	$x_2$	$d$
1	1	1	-1
2	0.5	0.5	-1
3	3	3	1
4	4	5	1

The objective of the SVM model is to find the weight vector  $w$  and the bias term  $\gamma$  that define the hyperplane equation:

$$f(x) = w^T x - \gamma = 0$$

The hard margin SVM optimization problem can be formulated as follows:

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

subject to:

$$d_i(A_i w - \gamma) \geq 1 \quad \forall i$$

Where:

$$A = \begin{bmatrix} 1 & 1 \\ 0.5 & 0.5 \\ 3 & 3 \\ 4 & 5 \end{bmatrix}$$
$$d = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

and  $\|w\|$  is the norm of the weight vector.

---

### 15.1.1 CVX Code Implementation

Below is the CVX code to solve the hard margin SVM problem using the specified data points:

```

1  % Data points and labels
2  A = [1 1; 0.5 0.5; 3 3; 4 5]; % Feature matrix
3  d = [-1; -1; 1; 1];          % Class labels
4  e = ones(4, 1);              % Vector of ones
5
6  % CVX optimization
7  cvx_begin quiet
8      variable w(2);            % Weight vector
9      variable gama;            % Bias term
10     dual variable u;           % Dual variables
11     minimize (0.5 * w' * w)    % Objective: Minimize the norm of w
12     subject to
13         u: d .* (A * w - gama * e) >= e; % Constraints with dual
           variable u
14 cvx_end
15
16 % Output results
17 format bank
18 disp('Weight vector w:')
19 disp(w)
20 disp('Bias term gama:')
21 disp(gama)
22 disp('Lagrangian multipliers u:')
23 disp('These correspond to points on the boundary:')
24 disp(u)
25
26 disp('Residuals (d .* (A * w - gama * e) - e):')
27 disp(d .* (A * w - gama * e) - e)
28 disp('Complementary conditions (u .* (d .* (A * w - gama * e) - e))')
29 disp(u .* (d .* (A * w - gama * e) - e))

```

output of the code is shown below.

```

w vector
    0.50
    0.50

gama
    2.00

u vector (lagrange multipliers)
those data point on boundary will have non-zero eigen values
    0.50
    0.00
    0.50
    0.00

(d.*(A*w-gama*e)-e)
those data point on boundary will show zero values
    0.00
    0.50
    0.00

```



```

1.50
u.*(d.*(A*w-gama*e)-e)
(these are complementarity condition)
0.50
0.00
0.50
0.00

```

We obtain the output as:

$$w = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}; \quad \text{gama} = 2; \quad u = \begin{bmatrix} 0.25 \\ 0.0 \\ 0.25 \\ 0.0 \end{bmatrix}$$

Now let's update the given data table with  $u$  values to identify support vectors. Updations are shown in Table 15.1.

Point ID	$x_1$	$x_2$	d	u
1	1	1	-1	0.25
2	0.5	0.5	-1	0.00
3	3	3	1	0.25
4	4	5	1	0.00

Table 15.1: Dataset with support vectors

Note that, first and third points are support vectors. They are on the bounding planes. Corresponding constraints are active. Now the SVM do its job of safely reconstructing the weight vector  $w$  as linear combination of points on the bounding planes as:

$$w = \sum_i u_i d_i x_i = 0.25 \times -1 \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0.25 \times 1 \times \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

### 15.1.2 Interpretation of Results

- **Weight Vector  $w$ :** Represents the direction of the hyperplane.
- **Bias Term  $\gamma$ :** Shifts the hyperplane away from the origin.
- **Lagrangian Multipliers  $u$ :** Points with non-zero multipliers correspond to the data points that lie on the margin (active constraints). They indicate which points are critical in defining the optimal hyperplane.

### 15.1.3 Residuals and Complementary Conditions

- **Residuals:** The values calculated as  $d \cdot (A \cdot w - \gamma \cdot e) - e$  help identify which points are on the boundary.
- **Complementary Conditions:** The product  $u \cdot (d \cdot (A \cdot w - \gamma \cdot e) - e)$  shows the relationship between the active constraints and their corresponding Lagrangian multipliers.

This is analytically proved using Lagrangian function and dual formation of the LPP version of the SVM.

We start the mathematical journey with the primal problem of the hard-margin SVM for  $N$  data points  $\{(x_i, d_i)\}_{i=1}^N$ , where  $x_i \in \mathbb{R}^n$  are the feature vectors and  $d_i \in \{-1, 1\}$  are the class labels. The optimization problem can be formulated as:

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

$$\text{subject to: } d_i(w^T x_i - \gamma) \geq 1, \quad \forall i = 1, \dots, N$$

where  $w \in \mathbb{R}^n$  is the weight vector, and  $\gamma$  is the bias term (intercept).

We construct the Lagrangian for this primal problem by introducing the Lagrange multipliers  $u_i \geq 0$  for each constraint:

$$L(w, \gamma, u) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N u_i [d_i(w^T x_i - \gamma) - 1]$$

### Solving for Stationary Points (Dual Problem)

Taking the partial derivatives of the Lagrangian  $L(w, \gamma, u)$  with respect to  $w$  and  $\gamma$  and setting them to zero gives the conditions for optimality.

Gradient with Respect to  $w$  are given by

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^N u_i d_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^N u_i d_i x_i$$

Gradient with Respect to  $\gamma$  is given by

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^N u_i d_i = 0$$

Thus, the solution for the weight vector  $w$  is expressed as a linear combination of the data points, weighted by the Lagrange multipliers  $u_i$  and the class labels  $d_i$ . The second condition ensures that the sum of the Lagrange multipliers weighted by their corresponding labels is zero.

### Quadratic Minimization Dual Formulation

Substituting  $w = \sum_{i=1}^N u_i d_i x_i$  into the primal objective function, the Lagrangian becomes purely in terms of the dual variables  $u_i$ , and the problem turns into a quadratic minimization problem.

The dual problem is then:

$$\text{Minimize: } \frac{1}{2} u^T Q u - \mathbf{1}^T u$$

$$\text{subject to: } u^T d = 0, \quad u_i \geq 0 \quad \forall i$$

where:

- $u = [u_1, u_2, \dots, u_N]^T$  is the vector of Lagrange multipliers,
- $d = [d_1, d_2, \dots, d_N]^T$  is the vector of class labels,
- $\mathbf{1}$  is a vector of ones,
- $Q = dd^T XX^T$ , where  $X = [x_1^T, x_2^T, \dots, x_N^T]^T \in \mathbb{R}^{N \times n}$  is the matrix of input data points.

## Full Matrix Representation

We now express the quadratic form and all components in matrix form.

The input data matrix  $X \in \mathbb{R}^{N \times n}$  contains the feature vectors of the data points:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nn} \end{bmatrix}$$

The labels are collected in the vector  $d \in \mathbb{R}^N$ :

$$d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}, \quad d_i \in \{-1, 1\}$$

The Quadratic Form Matrix  $Q = dd^T XX^T$  is formed by combining the labels and the data points, capturing the interaction between the input data and their class labels:

$$Q = dd^T XX^T$$

This can be broken down as follows:

1.  $XX^T \in \mathbb{R}^{N \times N}$  is the **Gram matrix** of the data points, where each entry  $(i, j)$  is the inner product  $x_i^T x_j$ , representing the similarity between the data points.

$$XX^T = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \dots & x_1^T x_N \\ x_2^T x_1 & x_2^T x_2 & \dots & x_2^T x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_N^T x_1 & x_N^T x_2 & \dots & x_N^T x_N \end{bmatrix}$$

2.  $dd^T \in \mathbb{R}^{N \times N}$  is the matrix of class labels, where each entry is  $d_i d_j$ :

$$dd^T = \begin{bmatrix} d_1 d_1 & d_1 d_2 & \dots & d_1 d_N \\ d_2 d_1 & d_2 d_2 & \dots & d_2 d_N \\ \vdots & \vdots & \ddots & \vdots \\ d_N d_1 & d_N d_2 & \dots & d_N d_N \end{bmatrix}$$

Thus, the matrix  $Q = dd^T XX^T$  combines these two matrices:

$$Q = \begin{bmatrix} d_1 d_1 x_1^T x_1 & d_1 d_2 x_1^T x_2 & \dots & d_1 d_N x_1^T x_N \\ d_2 d_1 x_2^T x_1 & d_2 d_2 x_2^T x_2 & \dots & d_2 d_N x_2^T x_N \\ \vdots & \vdots & \ddots & \vdots \\ d_N d_1 x_N^T x_1 & d_N d_2 x_N^T x_2 & \dots & d_N d_N x_N^T x_N \end{bmatrix}$$

## Complementary Slackness and Support Vectors

The **Karush-Kuhn-Tucker (KKT) conditions** include complementary slackness, which states that for each  $u_i$ :

$$u_i [d_i (w^T x_i - \gamma) - 1] = 0$$

This means that if  $u_i > 0$ , the constraint  $d_i(w^T x_i - \gamma) = 1$  is active, i.e., the point  $x_i$  is a **support vector**. If  $u_i = 0$ , the point is not a support vector.

Note that it only gives  $u$  based on which we can compute  $w$ . But it does not give  $\gamma$  directly. It can be easily obtained by making use of one of the support vector point.

### Conclusion: Support Vectors and Active Constraints

We have shown that:

- The dual problem is a quadratic minimization problem with the quadratic form  $u^T Q u$ , where  $Q = d d^T X X^T$ .
- The Lagrange multipliers  $u_i$  correspond to the active constraints in the primal problem.
- The support vectors are precisely the data points corresponding to non-zero Lagrange multipliers, as indicated by the KKT conditions.

Let us solve above using CVX. Matlab code for this task is given below.

```

1  % For assign 64 : Solves the dual svm
2  X=[1 1; 0.5 .5; 3 3;4 5];
3  d=[-1;-1;1;1];
4  n=size(X,1);% number of data points
5  e=ones(n,1);
6  Q=(d.*d').*(X*X');
7  cvx_begin quiet
8  variable u(n);
9  minimize (0.5*u'*Q*u-e'*u)
10 subject to
11 u'*d==0;
12 u>=0;
13 cvx_end
14 format bank
15 disp('u vector')
16 disp(u);

```

Output of the above code is shown below.

```

u vector
      0.25
      0.00
      0.25
      0.00

```

Internally, the CVX solves the problem using ADMM and hence it has obtained this  $u$  while solving primal problem itself. Note that values obtained earlier and now are same.

## 15.2 Task

Generate two groups of 2-tuple data (non-overlapping) by using `mvnrnd` with appropriate mean and covariance matrix. Find a hard-margin classifier using svm. In each group take 20 points. Solve for  $w$  using both primal and dual form of svm. Print  $w$  and  $\gamma$  values. Also draw the classifier line.

### SOLUTION

This task is solved in three stages. In the first stage data points are created using the `mvnrnd()` function in such a way that there is two distinguishable sets of points in terms its mean and spread. In

stage two the hard margin SVM primal problem is modelled and is solved using CVX. In stage three its dual is formed and the minimization version of its quadratic form is solved using CVX , support vectors are identified, plot the separating plane and other deliverables from the model.matlab code for this task is given below.

### Data generation and plotting of points

```

1  % Data generation
2  close(gcf)
3  n = 20;
4  mu1 = [-2; 5];
5  sigma1 = [4 3; 3 6];
6  d1 = -1*ones(n,1);
7  mu2 = [2; -5];
8  sigma2 = [6 2; 2 8];
9  d2 = 1*ones(n,1);
10 X1 = mvnrnd(mu1, sigma1, n);
11 X2 = mvnrnd(mu2, sigma2, n);
12 X = [X1; X2];
13 d = [d1; d2];
14
15 % Plotting the generated data points
16 figure('Position', [100, 100, 800, 600]); % Set figure size (width,
    height)
17 plot(X1(:,1), X1(:,2), 'r*');
18 hold on;
19 plot(X2(:,1), X2(:,2), 'b*');
20 hold on;

```

### Solution of the primal problem

```

1  % Define the primal variables
2  [m, n] = size(X);
3  cvx_begin quiet
4      variables w(n) gama xi(m)
5      minimize( (1/2)*w'*w ) % Objective function: (1/2) * ||w||^2
6      subject to
7          d .* (X*w - gama) >= 1 - xi; % SVM constraints
8          xi >= 0; % Non-negative slack variables (for hard-margin,
            xi will be zero)
9  cvx_end
10
11 % Print the primal solution
12 fprintf('Primal form solution:\n');
13 fprintf('w = [%.4f, %.4f]\n', w(1), w(2));
14 fprintf('gama = %.4f\n', gama);

```

### Solution of the dual problem

```

1  % Solve the SVM problem using the dual form
2

```

```

3  % Compute the Gram matrix for the dual form
4  H = (d * d') .* (X * X');
5
6  cvx_begin quiet
7      variable u(m)
8      dual variables y1 y2
9      maximize( sum(u) - (1/2) * u' * H * u ) % Objective function
          for the dual
10     subject to
11         u >= 0; % Non-negative Lagrange multipliers
12         d' * u == 0; % Equality constraint from dual form
13 cvx_end
14
15 % Compute w and gamma from dual variables
16 w_dual = X' * (u .* d);
17 sv = find(u > 1e-4); % Find support vectors
18 gamma_dual = mean(d(sv) - X(sv,:) * w_dual); % Compute gamma from
    support vectors
19
20 % Print the dual solution
21 fprintf('Dual form solution:\n');
22 fprintf('w = [%.4f, %.4f]\n', w_dual(1), w_dual(2));
23 fprintf('gamma = %.4f\n', gamma_dual);
24
25 % Plot the decision boundary (classifier line)
26 x_vals = linspace(min(X(:,1))-1, max(X(:,1))+1, 100);
27
28 % Decision boundary: w_1 * x_1 + w_2 * x_2 = gamma
29 y_vals_primal = -(w(1)*x_vals - gamma)/w(2); % Primal SVM boundary
30 y_vals_dual = -(w_dual(1)*x_vals - gamma_dual)/w_dual(2); % Dual
    SVM boundary
31
32 plot(x_vals, y_vals_primal, 'k-', 'LineWidth', 2); % Primal SVM
    boundary
33 plot(x_vals, y_vals_dual, 'g--', 'LineWidth', 2); % Dual SVM
    boundary
34
35 % Plot the support vectors
36 plot(X(sv,1), X(sv,2), 'ko', 'MarkerSize', 10, 'LineWidth', 2); %
    Highlight support vectors
37 % Adjust axis limits to ensure all points are visible
38 xlim([min(X(:,1))-1, max(X(:,1))+1]);
39 ylim([min(X(:,2))-1, max(X(:,2))+1]);
40 % Add legend and place it below the plot
41 legend('Class 1', 'Class 2', 'Primal SVM Boundary', 'Dual SVM
    Boundary', 'Support Vectors', 'Location', 'southoutside', '
    Orientation', 'vertical');
42 %title('SVM Classifier using Primal and Dual Forms');
43 xlabel('X_1');
44 ylabel('X_2');
45 grid on;
46 hold off;

```

The separating plane, support vectors and skill of the SVM model is shown in Figure 15.1.

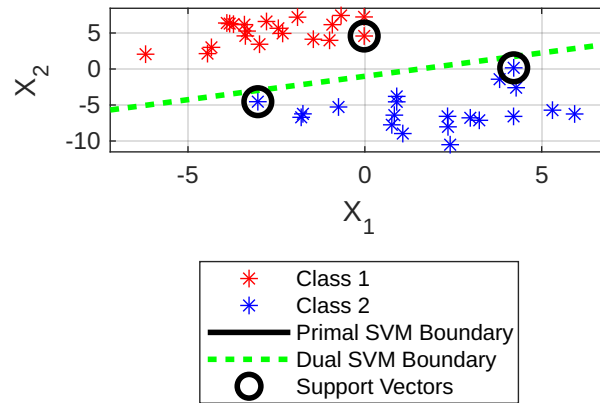


Figure 15.1: Result of hard margin SVM classification through Lagrange Optimization Method

## RESULTS

1. Mathematical machinery of hard margin support vector machines is revisited.
2. Both primal and dual problems are solved and crosschecked with CVX solvers.