UNIVERSITY BELHADJ BOUCHAIB OF AIN TEMOUCHENT
Faculty of Sciences and Technology
Department of Mechanical Engineering

Educational handout

File number : 12/PG/DGM/FST/UATBB/2025

Title

OPTIMIZATION

Course intended for students of

Master: first grad mechanical construction / second grad energetic

Authors : Ahmed SAIMI
And Abdelhamid LEBAL
2025

Preface

Optimization is a discipline that focuses on finding the best solution to problems involving limited resources and competing objectives. This field is fundamental to operational research and addresses a variety of practical and theoretical challenges, from improving production processes to solving complex mathematical problems.

The objective of this module is to enable students to understand the nature of optimization problems and to master the tools and algorithms necessary to solve them efficiently. This course serves as a foundation for further studies in operational research and computational methods.

This Optimization course book is intended for students specializing in mechanical engineering (first grade master's degree in mechanical construction, and second grade master's degree in energy). The course has been structured to meet the Optimization course program present in the National Program canvas.

This course on optimization is designed for students with a foundational knowledge of mathematics, particularly linear algebra and matrix algebra. The content is structured to align with the requirements of advanced studies in optimization techniques. It consists of four chapters:

The first chapter focuses on *linear optimization*. It introduces the general formulation of linear programming problems, with practical examples such as production, blending, cutting, and transportation problems. The resolution of linear programming problems is explored through the Simplex method, including initialization techniques like the Two-Phase Method.

The second chapter delves into *non-linear optimization without constraints*. This section covers essential mathematical concepts such as positivity, convexity, and gradient methods. It also discusses necessary and sufficient conditions for finding minima, along with optimization techniques like Newton's method, quasi-Newton methods, and conjugate directions.

The third chapter introduces *non-linear optimization with constraints*. It covers critical topics such as Lagrange multipliers, the Karush-Kuhn-Tucker (KKT) conditions, penalty methods, and sequential quadratic programming. These methods address optimization problems with equality and inequality constraints.

The fourth and final chapter explores *stochastic optimization methods*, focusing on modern algorithms like genetic algorithms and particle swarm optimization. These methods are especially suited for solving complex, non-linear problems with multiple local optima.

This course is designed to balance theoretical concepts with practical applications, offering students a pathway to becoming proficient in solving optimization problems. It equips learners with skills that are indispensable in fields ranging from engineering to economics and beyond.

Author:

ahmed.saimi@univ-temouchent.edu.dz

Teaching objective

This handout aims to provide a comprehensive introduction to optimization methods, covering both theoretical and practical aspects. Through four chapters, we will explore linear and nonlinear optimization techniques, as well as modern stochastic approaches. This course is aimed at students with a solid foundation in mathematics and programming, and aims to prepare them to solve complex problems in various scientific and industrial fields.

Teaching Objectives

This section describes the skills and knowledge that students are expected to acquire by the end of the course. Below is a proposed set of objectives for each chapter:

Linear Optimization

- Understand the fundamental concepts of linear optimization and their importance in mathematical modeling.
- Master the simplex method for solving linear optimization problems.
- Be able to interpret geometrically the solutions of a linear problem.
- Learn to formulate real-world problems as linear programs.

Nonlinear Unconstrained Optimization

- Gain an understanding of numerical methods for solving unconstrained nonlinear optimization problems.
- Master classical algorithms such as gradient descent, Newton's method, and quasi-Newton methods.
- Be able to analyze the convergence and properties of optimization algorithms.

Nonlinear Optimization with Constraints

- Understand the optimality conditions (KKT) for constrained problems.

- Learn to use methods such as Sequential Quadratic Programming (SQP) or penalty methods.

- Be able to solve practical problems involving equality and inequality constraints.

Stochastic Optimization Methods

- Discover stochastic approaches for solving optimization problems under uncertainty.

- Understand the principles of evolutionary algorithms, Monte Carlo methods, and reinforcement learning techniques.

- Be able to apply these methods to real-world problems where data is noisy or uncertain.

Global Objective

By the end of this course, students will be able to:

- Identify types of optimization problems and choose appropriate tools to solve them.

- Implement optimization algorithms using software or programming languages (e.g., Python, MATLAB).

- Apply optimization methods to practical problems from various fields (engineering, economics, data science, etc.).

Recommended Prerequisites

To successfully tackle this course, it is recommended that students possess the following knowledge:

Fundamental Mathematics

- Differential and integral calculus: concepts of partial derivatives, gradients, Hessians, and local optimization.

- Linear algebra: manipulation of matrices, vectors, systems of linear equations, and vector spaces.

- Basic understanding of convex functions and their properties.

Programming and Computer Science

- Knowledge of a programming language such as Python, MATLAB, to implement algorithms.

- Basics of numerical programming: loops, functions, and conditional structures.

Statistics and Probability (for Chapter 4)

- Elementary probability concepts: random variables, distributions, and mathematical expectation.

- Understanding of stochastic simulation methods (e.g., Monte Carlo).

Practical Knowledge

- Familiarity with scientific computing tools (e.g., OptimTool, Linprog in Matlab).

- Ability to interpret numerical results and analyze data.

Note: Students who do not possess all of these prerequisites may still take the course, but they will need to invest additional time to fill in any gaps.

Contents

Chapter 1

Mathematical Reminders

1.1 Mathematical Reminders

This chapter serves as a refresher on fundamental mathematical concepts that are essential for understanding advanced topics in optimization and related fields.

1.1.1 Introduction

The purpose of this chapter is to review key mathematical tools, including vectors, matrices, and eigenvalues. These concepts form the backbone of many algorithms used in modern computational sciences.

1.1.1.1 Notations

We adopt standard mathematical notations throughout this document:

- Scalars are represented by lowercase letters (e.g., x).
- Vectors are denoted by bold lowercase letters (e.g., v) and are assumed to be column vectors unless stated otherwise.
- Matrices are represented by bold uppercase letters (e.g., A).
- The transpose of a vector or matrix is denoted by a superscript $^\top$ (e.g., $v^\top$, $A^\top$).
- The norm of a vector v is denoted by $\|v\|$.

1.1.1.2 Vector

A vector $v \in R^n$ is an ordered list of n real numbers:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

Vectors can represent points in space, directions, or features in machine learning models. Operations on vectors include addition, scalar multiplication, dot product, and cross product. For example, the dot product of two vectors u and v is defined as:

$$u^\top v = \sum_{i=1}^{n} u_i v_i.$$

1.1.1.3 Matrix

A matrix $\mathbf{A} \in R^{m \times n}$ is a rectangular array of numbers with m rows and n columns:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Matrices are widely used to represent linear transformations, systems of equations, and data in machine learning. Key operations include matrix addition, multiplication, inversion, and transposition.

1.1.1.4 Eigenvalues

Eigenvalues and eigenvectors are critical for understanding the behavior of square matrices. Given a square matrix $A \in R^{n \times n}$, the eigenvalue problem is defined as:

$$Av = \lambda v,$$

where $\lambda \in R$ is the eigenvalue and $v \neq 0$ is the corresponding eigenvector. Eigenvalues provide insights into the stability, scaling, and transformation properties of a matrix.

For example, the eigenvalues of a symmetric positive definite matrix are always positive, which is crucial in optimization problems.

Chapter 2

Linear Optimization

2.1 Introduction

Linear optimization (also known as linear programming, LP) is a mathematical technique used for optimizing a linear "objective function", subject to a set of linear constraints. The method has widespread applications in diverse fields such as economics, business, engineering, transportation, and manufacturing.

Linear optimization involves:

- An "objective function": A function to be maximized or minimized (e.g., profit, cost, or efficiency).
- Constraints: Conditions represented as linear inequalities or equations that restrict the solution space.
- Decision variables: Variables that determine the outcome, which are to be solved for.

The graphical method is typically used for small problems, while larger problems rely on computational algorithms such as the Simplex Method and Interior Point Methods.

2.2 Definition

Linear optimization can be defined mathematically as follows:

$$\text{Maximize (or Minimize): } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\textit{subject to } \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n (\leq, =, \geq) b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n (\leq, =, \geq) b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n (\leq, =, \geq) b_m \end{cases}$$

where:

- Z: The "objective function".
- $x_1, x_2, \ldots, x_n$: Decision variables.
- $c_1, c_2, \ldots, c_n$: Coefficients of the "objective function".
- a_{ij}: Coefficients of the constraints.
- b_i: Right-hand side constants of the constraints.

2.3 Linear Programs

Linear optimization problems come in many forms. Below are detailed examples to illustrate common applications.

2.3.1 Production Problem

Scenario: Two products, P_1 and P_2 are fabricated by a factory. Each product requires labor and materials. Profits per unit of P_1 and P_2 are \$30 and \$40, respectively. The factory has:

- 120 hours of labor available.
- 100 units of material available.

Goal: Maximize profit.

Steps to Determine the "objective function" and Constraints:

1. **Identify the decision variables:**

$$x_1 = \text{Number of } P_1 \text{ units produced}, \quad x_2 = \text{Number of } P_2 \text{ units produced}.$$

2. **Define the "objective function":**

$$Z = 30x_1 + 40x_2 \quad \text{(Maximize } Z\text{)}.$$

3. **Determine the constraints:**

   - **Labor Constraint:** Each P_1 requires 2 hours of labor, and P_2 requires 3 hours. Total labor cannot exceed 120 hours:

$$2x_1 + 3x_2 \leq 120.$$

   - **Material Constraint:** Each P_1 requires 4 units of material, and P_2 requires 2 units. Total material cannot exceed 100 units:

$$4x_1 + 2x_2 \leq 100.$$

   - **Non-negativity:** The factory cannot produce negative units:

$$x_1 \geq 0, \quad x_2 \geq 0.$$

2.3.2 Mixing Problem

Scenario: A company produces a chemical mixture using two raw materials, R_1 and R_2. Each raw material has specific properties:

- Unit Cost of R_1: \$5.
- Unit Cost of R_2: \$7.

The mixture must meet quality standards, requiring at least 50 units of a key property. R_1 contributes 3 units of the property per unit, while R_2 contributes 2 units.

Goal: Minimize cost.

Steps to Determine the "objective function" and Constraints:

1. **Identify the decision variables:**

$$x_1 = \text{Quantity of } R_1 \text{ used}, \quad x_2 = \text{Quantity of } R_2 \text{ used}.$$

2. **Define the "objective function":**

$$Z = 5x_1 + 7x_2 \quad \text{(Minimize } Z\text{)}.$$

3. **Determine the constraints:**

   - **Quality Constraint:** The total contribution from R_1 and R_2 must be at least 50 units:
   $$3x_1 + 2x_2 \geq 50.$$
   - **Availability Constraints:** R_1 is limited to 20 units and R_2 to 30 units:
   $$x_1 \leq 20, \quad x_2 \leq 30.$$
   - **Non-negativity:**
   $$x_1 \geq 0, \quad x_2 \geq 0.$$

2.3.3 Cutting Problem

Scenario: A paper manufacturer needs to cut large rolls of paper (100 meters each) into smaller sizes to meet customer demand:

- 30 rolls of 10 meters.
- 20 rolls of 5 meters.

Goal: Minimize waste.

Steps to Determine the "objective function" and Constraints:

1. **Identify the decision variables:**
   $$x_1 = \text{Number of 10-meter rolls cut}, \quad x_2 = \text{Number of 5-meter rolls cut}.$$
2. **Define the "objective function":**
   $$Z = 100 - (10x_1 + 5x_2) \quad (\text{Minimize } Z).$$
3. **Determine the constraints:**

   - **Demand Constraints:**
   $$x_1 \geq 30, \quad x_2 \geq 20.$$
   - **Roll Size Constraint:** Total paper used cannot exceed the roll size:
   $$10x_1 + 5x_2 \leq 100.$$
   - **Non-negativity:**
   $$x_1 \geq 0, \quad x_2 \geq 0.$$

2.3.4 Transport Problem

Scenario: A company must ship products from two warehouses (A and B) to two stores (X and Y):

- Warehouse A: 100 units available.
- Warehouse B: 200 units available.
- Store X: 150 units needed.
- Store Y: 150 units needed.

The transportation costs per unit are:

$$\text{A to X: \$4, A to Y: \$3, B to X: \$2, B to Y: \$5.}$$

Goal: Minimize transportation cost.

Steps to Determine the "objective function" and Constraints:

1. **Identify the decision variables:**

$$x_{AX}, x_{AY}, x_{BX}, x_{BY} = \text{Quantities transported from A and B to X and Y.}$$

2. **Define the "objective function":**

$$Z = 4x_{AX} + 3x_{AY} + 2x_{BX} + 5x_{BY} \quad \text{(Minimize } Z\text{)}.$$

3. **Determine the constraints:**

   - **Supply Constraints:**

$$x_{AX} + x_{AY} \leq 100 \quad \text{(Supply at A)},$$

$$x_{BX} + x_{BY} \leq 200 \quad \text{(Supply at B)}.$$

   - **Demand Constraints:**

$$x_{AX} + x_{BX} \geq 150 \quad \text{(Demand at X)},$$

$$x_{AY} + x_{BY} \geq 150 \quad \text{(Demand at Y)}.$$

   - **Non-negativity:**

$$x_{AX}, x_{AY}, x_{BX}, x_{BY} \geq 0.$$

2.4 Graphical Method for Linear Optimization

2.4.1 Problem Statement

A company produces two products, P_1 and P_2, with the following constraints:

- Each unit of P_1 requires 2 hours of labor and 1 unit of material.
- Each unit of P_2 requires 1 hour of labor and 2 units of material.
- A maximum of 40 hours of labor and 50 units of material are available.

The profit per unit is:

- \$30 for P_1.
- \$20 for P_2.

Objective: Maximize profit.

2.4.2 Formulation

Decision Variables

$$x_1 = \text{Number of units of } P_1 \text{ produced.}$$

$$x_2 = \text{Number of units of } P_2 \text{ produced.}$$

2.4.3 "objective function"

$$\text{Maximize: } Z = 30x_1 + 20x_2.$$

2.4.3.1 Constraints

$2x_1+x_2 \leq 40 \quad$ (Labor constraint)$x_1+2x_2 \leq 50 \quad$ (Material constraint)$x_1, x_2 \geq 0 \quad$ (Non-negativity const

2.4.4 Graphical Solution

The feasible region is determined by plotting the constraints and finding the intersection points. Below is the graphical illustration.

2.4.5 Steps for the Graphical Method

The graphical method is used to solve a linear programming problem by finding the optimal solution visually. Below are the detailed steps:

2.4.5.1 Step 1: Define the Problem

1. **Identify the decision variables:** Define the variables x_1 and x_2, representing the quantities to optimize.

2. **Set up the "objective function":** Write the function to maximize or minimize. For example:
$$Z = 30x_1 + 20x_2 \quad \text{(maximize profit)}.$$

3. **Formulate the constraints:** Convert resource limitations into inequalities. For instance:

$$2x_1 + x_2 \leq 40 \quad \text{(labor constraint)} x_1 + 2x_2 \leq 50 \quad \text{(material constraint)} x_1, x_2 \geq 0 \quad \text{(non-negativity)}.$$

2.4.5.2 Step 2: Graph the Constraints

1. **Convert inequalities into equations:** Replace inequality signs ($\leq$) with equality ($=$) to identify boundary lines.
   - For $2x_1 + x_2 \leq 40$, the line is $2x_1 + x_2 = 40$.
   - For $x_1 + 2x_2 \leq 50$, the line is $x_1 + 2x_2 = 50$.

2. **Find boundary points for each line:**
   - For $2x_1 + x_2 = 40$:
$$x_1 = 0 \implies x_2 = 40, \quad x_2 = 0 \implies x_1 = 20.$$
   - For $x_1 + 2x_2 = 50$:
$$x_1 = 0 \implies x_2 = 25, \quad x_2 = 0 \implies x_1 = 50.$$

3. **Plot each line on a graph:** Use the boundary points to draw straight lines for each constraint.

4. **Determine inequality regions:** For each inequality, shade the area below the line (for $\leq$) to represent the feasible region.

2.4.5.3 Step 3: Identify the Feasible Region

1. The feasible region is the area where all constraints overlap, including the non-negativity region ($x_1, x_2 \geq 0$).
2. Mark the boundary points (corner points) of this region.
3. In this example, the corner points are:

$$(0, 25), \quad (20, 0), \quad (10, 20), \quad (0, 0)$$

2.4.5.4 Step 4: Evaluate the "objective function"

1. Substitute each corner point into the "objective function" $Z = 30x_1 + 20x_2$:

$$\begin{aligned}
\text{At } A(10, 20): &\quad Z = 30(10) + 20(20) = 700. \\
\text{At } B(20, 0): &\quad Z = 30(20) + 20(0) = 600. \\
\text{At } C(0, 25): &\quad Z = 30(0) + 20(25) = 500. \\
\text{At } D(0, 0): &\quad Z = 30(0) + 20(0) = 0.
\end{aligned}$$

2.4.5.5 Step 5: Graphical Illustration

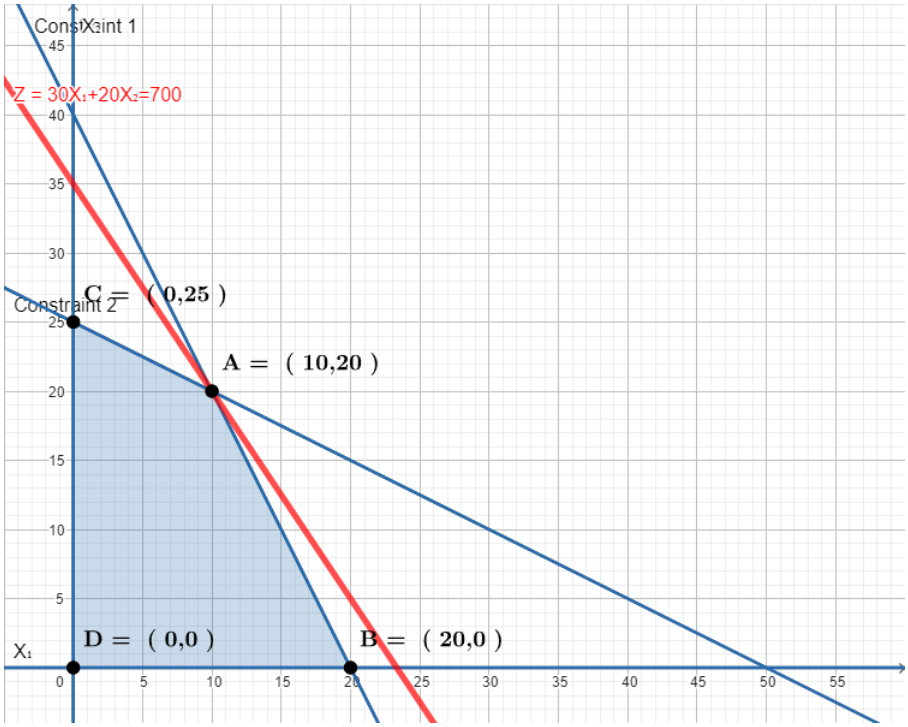

Figure 2.1: Graphical resolution illustration

2.4.5.6 Step 6: Interpret the Results

2. **Select the optimal value:**
   - If maximizing, choose the highest Z: $Z = 700$ at $(10, 20)$.
   - If minimizing, choose the lowest Z.

- The optimal solution is $Z = 700$, achieved at $(10, 20)$.
- This means producing 10 units of P_1 and 20 units of P_2 will yield the maximum profit.

2.5 Simplex Method

The Simplex Method is an iterative algorithm used to solve linear programming problems. It is particularly powerful for problems with many variables and constraints, making it the most widely used algorithm for linear optimization.

The method involves:

- Moving from one vertex (basic feasible solution) of the feasible region to another, improving the "objective function" at each step.
- Reaching the optimal solution when no further improvement is possible.

2.6 Basics and Basic Solutions of Linear Programs

2.6.1 Converting to Standard Form

To use the Simplex Method:

1. All constraints must be expressed as equations by introducing slack variables.
2. The "objective function" is rewritten in terms of decision and slack variables.

Example:

$$\begin{aligned} \text{Maximize } Z &= 3x_1 + 2x_2 \\ x_1 + x_2 &\leq 4 \\ \text{Subject to: } 2x_1 + x_2 &\leq 6 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Step 1: Add Slack Variables

- For $x_1 + x_2 \leq 4$, introduce slack s_1: $x_1 + x_2 + s_1 = 4$.
- For $2x_1 + x_2 \leq 6$, introduce slack s_2: $2x_1 + x_2 + s_2 = 6$.

The problem becomes:

$$\begin{aligned} \text{Maximize } Z &= 3x_1 + 2x_2 + 0s_1 + 0s_2 \\ x_1 + x_2 + s_1 &= 4 \\ \text{Subject to: } 2x_1 + x_2 + s_2 &= 6 \\ x_1, x_2, s_1, s_2 &\geq 0 \end{aligned}$$

2.7 The Simplex Algorithm

The Simplex Algorithm is a systematic method for finding the optimal solution. It involves the following steps:

2.7.1 Steps

1. **Formulate the problem in standard form.**
2. **Set up the initial table:**
   - Each row represents a constraint.
   - The columns represent the decision variables, slack variables, and the right-hand side values.
3. **Identify the entering variable (pivot column):**
   - Select the column with the most negative coefficient in the "objective function" row.
4. **Identify the leaving variable (pivot row):**
   - Divide the right-hand side by the pivot column entries and select the smallest non-negative ratio.
5. **Perform the pivot operation:**
   - Adjust the table to make the pivot element 1 and all other elements in the pivot column 0.
6. **Repeat until optimality is reached:**
   - The solution is optimal when there are no negative coefficients in the objective row.

2.7.2 Example: Solving Using the Simplex Method

$$\begin{aligned} \text{Maximize } Z &= 3x_1 + 2x_2 \\ x_1 + x_2 &\leq 4 \\ \text{Subject to: } 2x_1 + x_2 &\leq 6 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Step 1: Convert to Standard Form

$$\text{Maximize } Z = 3x_1 + 2x_2 + 0s_1 + 0s_2$$

$$x_1 + x_2 + s_1 = 4$$
$$\text{Subject to: } 2x_1 + x_2 + s_2 = 6$$
$$x_1, x_2, s_1, s_2 \geq 0$$

Step 2: Set Up the Initial table

BV	x_1	x_2	s_1	s_2	RHS
s_1	1	1	1	0	4
s_2	2	1	0	1	6
Z	-3	-2	0	0	0

Step 3: Identify the Pivot

1. **Entering variable:** Select the column with the most negative value in the Z-row. Here, x_1 with -3.

2. **Leaving variable:** Calculate the ratio of the RHS to the pivot column for each row:

$$\frac{4}{1} = 4, \quad \frac{6}{2} = 3 \quad \text{(smallest ratio: row 2, variable } s_2\text{)}.$$

3. **Pivot element:** 2 (intersection of the x_1 column and s_2 row).

Step 4: Adjust the Pivot Row Make the pivot element 1 by dividing the entire pivot row (s_2) by the pivot element (2):

$$\text{New } s_2 \text{ row: } \frac{\text{current } s_2 \text{ row}}{\text{pivot element}}.$$

$$\text{New } s_2 \text{ row: } \frac{1}{2} \times [2\ 1\ 0\ 1\ 6] = [1\ 0.5\ 0\ 0.5\ 3].$$

Step 5: Adjust Other Rows to Make Pivot Column Zero For each non-pivot row, subtract a multiple of the pivot row such that the pivot column becomes 0.

Adjust s_1 row:

$$\text{New } s_1 \text{ row: Current } s_1 \text{ row} - (\text{pivot column coefficient in } s_1 \text{ row}) \times \text{New pivot row}.$$

$$[1\ 1\ 1\ 0\ 4] - 1 \times [1\ 0.5\ 0\ 0.5\ 3] = [0\ 0.5\ 1\ -0.5\ 1].$$

Adjust Z-row:

$$\text{New } Z \text{ row: Current } Z \text{ row} - (\text{pivot column coefficient in } Z \text{ row}) \times \text{New pivot row}.$$

$$[-3\ -2\ 0\ 0\ 0] - (-3) \times [1\ 0.5\ 0\ 0.5\ 3] = [0\ -0.5\ 0\ 1.5\ 9].$$

Step 5: Updated table The new table after the pivot operation is:

BV	x_1	x_2	s_1	s_2	RHS
s_1	0	0.5	1	-0.5	1
x_1	1	0.5	0	0.5	3
Z	0	-0.5	0	1.5	9

Explanation of Adjustments

- The pivot row is adjusted to make the pivot element 1.
- All other rows (including Z-row) are adjusted to ensure the pivot column contains 0.
- The new table is a valid simplex table that moves closer to the optimal solution.

The process is repeated with the new table: selecting a new entering variable, determining the leaving variable, and performing another pivot operation until no negative coefficients remain in the Z-row.

Final Table

BV	x_1	x_2	s_1	s_2	RHS
x_2	0	1	1	-1	2
x_1	1	0	-0.5	1	2
Z	0	0	1.5	0	12

Optimal Solution:

$$x_1 = 2, \quad , x_2 = 2,$$

2.8 Introduction to the Dual Simplex Method

The Dual Simplex Method is a variation of the Simplex Method used to solve linear programming problems. It is particularly useful when the initial table satisfies the optimality condition but violates feasibility (i.e., some right-hand side (RHS) values are negative). The dual Simplex Method iteratively improves feasibility while maintaining optimality.

2.8.1 When to Use the Dual Simplex Method

The Dual Simplex Method is used in specific situations where the standard Simplex Method may not be applicable or efficient. It is particularly useful when:

1. **The optimality condition is satisfied, but feasibility is violated:**
    - The objective row (reduced costs) has no negative coefficients, indicating that the optimality condition is satisfied.
    - However, at least one right-hand side (RHS) value is negative, indicating an infeasible solution.
    - The Dual Simplex Method adjusts the solution to restore feasibility while maintaining optimality.
2. **Infeasible basic solution at the start:**
    - When the initial basic solution (e.g., after adding artificial variables for $\geq$ constraints) is infeasible, the Dual Simplex Method works to make the solution feasible.
3. **Degeneracy or alternate optimality in the primal Simplex Method:**
    - If the primal Simplex Method leads to cycling or slow convergence due to degeneracy, the Dual Simplex Method can resolve the issue.
4. **Post-optimality analysis:**
    - After minor changes to a solved linear programming problem (e.g., updates to RHS values or constraints), the Dual Simplex Method can quickly re-optimize without starting over.

2.8.2 Basic Idea of the Dual Simplex Method

- Instead of selecting the entering variable based on the objective row, we select the leaving variable based on the most negative RHS value.
- Pivoting operations are performed to make the table feasible while preserving optimality.

2.9 Determining the Starting Basic Solution in Linear Programming

2.9.1 What Is a Basic Solution?

A basic solution is obtained by:

1. Selecting a subset of variables (called **basic variables**) equal to the number of constraints (m).
2. Setting all other variables (called **non-basic variables**) to zero.
3. Solving the resulting system of linear equations for the basic variables.

A **basic feasible solution** satisfies:

- All constraints in the problem.
- The non-negativity conditions ($x_i \geq 0$).

2.9.2 Steps to Determine the Starting Basic Solution

Step 1: Convert the Problem to Standard Form

To set up the problem for the Simplex or Dual Simplex Method:

- Express all constraints as equalities by adding slack, surplus, or artificial variables.
- Ensure all variables have non-negativity constraints ($x_i \geq 0$).

Step 2: Introduce Slack, Surplus, or Artificial Variables

- **Slack Variables** are added for $\leq$ constraints:

$$a_{11}x_1 + a_{12}x_2 \leq b_1 \implies a_{11}x_1 + a_{12}x_2 + s_1 = b_1, \quad s_1 \geq 0.$$

- **Surplus Variables** are subtracted for $\geq$ constraints:

$$a_{11}x_1 + a_{12}x_2 \geq b_1 \implies a_{11}x_1 + a_{12}x_2 - s_1 = b_1, \quad s_1 \geq 0.$$

- **Artificial Variables** are introduced to handle surplus constraints or infeasible cases.

Step 3: Identify Basic Variables

- Initially, slack and artificial variables are treated as basic variables.
- Decision variables ($x_1, x_2, \ldots$) and surplus variables are non-basic variables.

Step 4: Solve for the Basic Variables

Set all non-basic variables to zero and solve the system of linear equations for the basic variables.

Step 5: Check Feasibility

The starting basic solution must satisfy all constraints and the non-negativity condition ($x_i \geq 0$).

2.9.3 Example: Finding a Feasible Starting Basic Solution

Problem Statement

$$\begin{aligned} \text{Maximize } Z &= 3x_1 + 2x_2 \\ x_1 + x_2 &\leq 4, \\ \text{Subject to: } 2x_1 + x_2 &\leq 6, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Step 1: Convert to Standard Form

Add slack variables s_1, s_2:

$$\begin{aligned} x_1 + x_2 + s_1 &= 4, \\ 2x_1 + x_2 + s_2 &= 6, \\ x_1, x_2, s_1, s_2 &\geq 0. \end{aligned}$$

Step 2: Identify Basic and Non-Basic Variables

- **Basic variables:** s_1, s_2 (slack variables).
- **Non-basic variables:** x_1, x_2 (decision variables).

Step 3: Solve for the Basic Variables

Set $x_1 = 0$ and $x_2 = 0$:

$$s_1 = 4,$$
$$s_2 = 6.$$

Step 4: Verify Feasibility

The starting solution is:

$$x_1 = 0,\ x_2 = 0,\ s_1 = 4,\ s_2 = 6.$$

All variables are non-negative, so the solution is feasible.

2.9.4 Example: Infeasible Starting Solution

Problem Statement

$$\begin{aligned} \text{Minimize } Z &= 5x_1 + 4x_2 \\ x_1 + 2x_2 &\geq 8, \\ \text{Subject to: } 3x_1 + x_2 &\geq 6, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Step 1: Convert to Standard Form

Add surplus variables:

$$x_1 + 2x_2 - s_1 = 8,$$
$$3x_1 + x_2 - s_2 = 6,$$
$$x_1, x_2, s_1, s_2 \geq 0.$$

Step 2: Identify Basic and Non-Basic Variables

- **Basic variables:** s_1, s_2 (artificial variables).
- **Non-basic variables:** x_1, x_2.

Step 3: Solve for the Basic Variables

Set $x_1 = 0, x_2 = 0$:

$$s_1 = -8,$$
$$s_2 = -6.$$

the solution doesn't satisfy the constraint

$$x_1, x_2, s_1, s_2 \geq 0$$

Step 4: add artificial variables

artificial variable are added in constraint with

$$\geq 0 \ \ and \ \ = 0$$

$$x_1 + 2x_2 - s_1 + a_1 = 8,$$
$$3x_1 + x_2 - s_2 + a_2 = 6,$$
$$x_1, x_2, s_1, s_2 \geq 0.$$

Set $x_1 = 0, x_2 = 0, s_1 = 0, s_2 = 0$:

$$a_1 = 8,$$
$$a_2 = 6.$$

Step 4: Verify Feasibility

The artificial variables $a_1 = 8$ and $a_2 = 6$ indicate infeasibility, as artificial variables must eventually leave the basis for a solution to be feasible.

2.9.5 Conclusion

- The starting basic solution is determined by assigning slack, surplus, or artificial variables as basic variables and solving the resulting system.
- Feasibility is verified by checking the non-negativity of all variables.
- If the starting basic solution is infeasible, methods like the **Dual Simplex Method** or the **Big M Method** are required.

2.10 The Dual Simplex Method Algorithm

Steps of the Dual Simplex Method

1. **Identify the initial table:** Ensure the optimality condition is satisfied (no negative values in the objective row).
2. **Choose the leaving variable:** Select the row with the most negative RHS value.

3. **Determine the entering variable:** Compute the ratio of the pivot row coefficients in the "objective function" column to the absolute values of the pivot column coefficients. Select the smallest non-negative ratio.

4. **Perform the pivot operation:** Adjust the table to make the pivot element 1 and all other elements in the pivot column 0.

5. **Repeat until feasibility is restored:** Continue until all RHS values are non-negative.

2.11 Example: Solving Using the Dual Simplex Method

2.11.1 Problem Statement

Solve the following linear program using the dual Simplex Method:

$$\text{Maximize } Z = x_1 + 2x_2$$

$$\text{Subject to: } \begin{aligned} x_1 + x_2 &\leq 5 \\ x_1 + 2x_2 &\geq 2 \\ x_1 &\geq 1 \\ x_1, x_2 &\geq 0. \end{aligned}$$

2.11.2 Step 1: Convert to Standard Form

Since the constraints are $\geq$, introduce surplus and artificial variables:

$$\text{Maximize } Z = x_1 + 2x_2 + 0s_1 + 0s_2 + a_1 + a_2,$$

$$\text{Subject to: } \begin{aligned} x_1 + x_2 + s_1 &= 5 \\ x_1 + 2x_2 - s_2 + a_1 &= 2 \\ x_1 - s_3 + a_2 &= 1 \\ x_1, x_2, s_1, s_2, a_1, a_2 &\geq 0. \end{aligned}$$

2.11.3 Step 2: phase 1 of the dual simplex

in the first phase it is necessary to minimize the function

$$W = a_1 + a_2$$

2.11.4 Step 3: Set Up the Initial Table

The initial table is:

	BV	x_1	x_2	s_1	s_2	s_3	a_1	a_2	C_i
0	s_1	1	1	1	0	0	0	0	5
1	a_1	1	2	0	-1	0	1	0	2
1	a_2	1	0	0	0	-1	0	1	1
	b_i	0	0	0	0	0	1	1	
	W_i	2	2	0	-1	-1	1	1	3
	$b_i - W_i$	-2	-2	0	1	1	0	0	

Step 3: Identify the Pivot to identify the entering variable, we must:

1. **in case of Maximization:** Select the column with the most positive value in the $b_i - W_i$-row
2. **in case of Minimization:** Select the column with the most negative value in the $b_i - W_i$-row

to identify the leaving variable, we must:

1. **in case all cases:** Calculate the ratio of the C_i-column to the pivot column for each row, and choose the most non-negative value. (excluding $0, -inf, +inf$.

in this example:

1. **Entering variable:** the most negative value in the $b_i - W_i$-row. Here, x_1 with -2.
2. **Leaving variable:**

$$\frac{5}{1} = 5, \quad \frac{2}{1} = 2, \quad \frac{1}{1} = 1 \quad \text{(smallest ratio: row 3, variable } a_2\text{)}.$$

3. **Pivot element:** 1 (intersection of the x_1 column and a_2 row).

Step 4: Adjust the Pivot Row Make the pivot element 1 by dividing the entire pivot row (a_2) by the pivot element (1):

$$\text{New } a_2 \text{ row: } \frac{\text{current } a_2 \text{ row}}{\text{pivot element}}.$$

$$\text{New } a_2 \text{ row: } \frac{1}{1} \times [1\ 0\ 0\ 0\ \ -1\ 0\ 1\ 1] = [1\ 0\ 0\ 0\ \ -1\ 0\ 1\ 1].$$

Step 5: Adjust Other Rows to Make Pivot Column Zero For each non-pivot row, subtract a multiple of the pivot row such that the pivot column becomes 0.

Adjust s_1 row:

New s_1 row: Current s_1 row $-$ (pivot column coefficient in s_1 row) $\times$ New pivot row.

$$[1\,1\,1\,0\,0\,0\,0\,5] - 1 \times [1\,0\,0\,0\ -1\,0\,1\,1] = [0\,1\,1\,0\,1\,0\ -1\,4].$$

Adjust a_1 row:

New a_1 row: Current a_1 row $-$ (pivot column coefficient in a_1 row) $\times$ New pivot row.

$$[1\,2\,0\ -1\,0\,1\,0\,2] - 1 \times [1\,0\,0\,0\ -1\,0\,1\,1] = [0\,2\,0\ -1\,1\,1\ -1\,1].$$

Step 5: Updated table The new table after the pivot operation is:

	BV	x_1	x_2	s_1	s_2	s_3	a_1	a_2	C_i
0	s_1	0	1	1	0	1	0	-1	4
1	x_2	0	2	0	-1	1	1	-1	1
0	x_1	1	0	0	0	-1	0	1	1
	b_i	0	0	0	0	0	1	1	
	W_i	0	2	0	-1	1	1	-1	1
	$b_i - W_i$	0	-2	0	1	-1	0	2	

The process is repeated until $W_i = 0$ in C_i column to stop the phase 1, and proceed to phase 2.

Final Table of phase 1

	BV	x_1	x_2	s_1	s_2	s_3	a_1	a_2	C_i
0	s_1	0	0	1	1/2	1/2	$-1/2$	$-1/2$	7/2
0	x_2	0	1	0	$-1/2$	1/2	1/2	$-1/2$	1/2
0	x_1	1	0	0	0	-1	0	1	1
	b_i	0	0	0	0	0	1	1	
	W_i	0	0	0	0	0	0	0	0
	$b_i - W_i$	0	0	0	0	0	1	1	

2.11.5 Step 4: phase 2 of the dual simplex

in phase 2, we inject the original "objective function" but without artificial variables in the final table of phase 1:

$$\text{Maximize } Z = x_1 + 2x_2 + 0s_1 + 0s_2,$$

	BV	x_1	x_2	s_1	s_2	s_3	C_i
0	s_1	0	0	1	1/2	1/2	7/2
2	x_2	0	1	0	$-1/2$	1/2	1/2
1	x_1	1	0	0	0	-1	1
	b_i	1	2	0	0	0	
	Z_i	1	2	0	-1	0	2
	$b_i - Z_i$	0	0	0	1	0	

repeat the same steps to identify the pivot.

	BV	x_1	x_2	s_1	s_2	s_3	C_i
0	s_2	0	0	2	1	1	7
2	x_2	0	1	1	0	1	4
1	x_1	1	0	0	0	-1	1
	b_i	1	2	0	0	0	
	Z_i	1	2	2	0	1	9
	$b_i - Z_i$	0	0	-2	0	-1	

the optimality criterion is satisfied when the all elements in $b_i - Z_i$-row are ≤ 0 in case of Maximization.

Optimal Solution:

$$x_1 = 1, \quad , x_2 = 4,$$

$$s_1 = 0, \quad s_2 = 7, \quad s_3 = 0,$$

$$Z = 9,$$

The Dual Simplex Method is a powerful tool for solving linear programming problems, especially when the initial table is optimal but infeasible. It iteratively adjusts the table to restore feasibility while maintaining optimality.

2.12 Exercises

Graphical Method

Exercise 1 Maximize the "objective function":

$$Z = 3x_1 + 4x_2$$

Subject to the constraints:

$$x_1 + 2x_2 \leq 8$$

$$2x_1 + x_2 \leq 10$$

$$x_1, x_2 \geq 0$$

Exercise 2 Minimize the "objective function":

$$Z = 5x_1 + 2x_2$$

Subject to the constraints:

$$3x_1 + 2x_2 \geq 12$$

$$x_1 + x_2 \leq 6$$

$$x_1, x_2 \geq 0$$

Exercise 3 Maximize the "objective function":

$$Z = 7x_1 + 9x_2$$

Subject to the constraints:

$$2x_1 + x_2 \leq 14$$

$$x_1 + 3x_2 \leq 15$$

$$x_1 \geq 0, \quad x_2 \geq 0$$

Exercise 4 Minimize the "objective function":

$$Z = 4x_1 + 6x_2$$

Subject to the constraints:

$$x_1 + x_2 \geq 5$$

$$2x_1 + 3x_2 \leq 12$$

$$x_1, x_2 \geq 0$$

Simplex Method

Exercise 1 Maximize the "objective function":

$$Z = 6x_1 + 8x_2$$

Subject to the constraints:

$$4x_1 + 6x_2 \leq 24$$

$$2x_1 + 3x_2 \leq 15$$

$$x_1, x_2 \geq 0$$

Exercise 2 Maximize the "objective function":

$$Z = 5x_1 + 3x_2$$

Subject to the constraints:

$$x_1 + x_2 \leq 7$$

$$2x_1 + x_2 \leq 10$$

$$x_1 + 3x_2 \leq 12$$

$$x_1, x_2 \geq 0$$

Exercise 3 Minimize the "objective function":

$$Z = 3x_1 + 2x_2$$

Subject to the constraints:

$$x_1 + 2x_2 \geq 6$$

$$2x_1 + x_2 \geq 8$$

$$x_1, x_2 \geq 0$$

Exercise 4 Maximize the "objective function":

$$Z = 10x_1 + 12x_2$$

Subject to the constraints:

$$2x_1 + 3x_2 \leq 18$$

$$x_1 + x_2 \leq 8$$

$$3x_1 + 2x_2 \leq 21$$

$$x_1, x_2 \geq 0$$

Dual Simplex Method

Exercise 1 Minimize the "objective function":

$$Z = 2x_1 + 4x_2$$

Subject to the constraints:

$$3x_1 + x_2 \geq 6$$

$$2x_1 + 5x_2 \geq 10$$

$$x_1, x_2 \geq 0$$

Exercise 2 Minimize the "objective function":

$$Z = 5x_1 + 8x_2$$

Subject to the constraints:

$$4x_1 + 2x_2 \geq 12$$

$$x_1 + 3x_2 \geq 9$$

$$x_1, x_2 \geq 0$$

Exercise 3 Maximize the "objective function":

$$Z = 3x_1 + 6x_2$$

Subject to the constraints:

$$2x_1 + 3x_2 \leq 12$$

$$4x_1 + x_2 \leq 10$$

$$x_1, x_2 \geq 0$$

Exercise 4 Minimize the "objective function":

$$Z = 7x_1 + 4x_2$$

Subject to the constraints:

$$3x_1 + 2x_2 \geq 14$$

$$5x_1 + 3x_2 \geq 15$$

$$x_1, x_2 \geq 0$$

Chapter 3

Nonlinear Unconstrained Optimization

3.1 Introduction

Nonlinear optimization without constraints focuses on finding the minimum or maximum of a nonlinear objective function $f(x)$, where x is a vector of decision variables:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

The problem can be formally stated as:

$$\min_{x \in R^n} f(x), \quad \text{or} \quad \max_{x \in R^n} f(x).$$

- **Nonlinearity:** The function $f(x)$ is nonlinear, meaning it cannot be expressed as a linear combination of the decision variables x_i.
- **Unconstrained Nature:**There are no explicit constraints on the variables x. Unlike constrained optimization, the feasible region is the entire R^n.

The goal is to find the optimal point x^* such that:

$$f(x^*) \leq f(x) \quad \forall x \in R^n \quad \text{(for minimization)},$$

or

$$f(x^*) \geq f(x) \quad \forall x \in R^n \quad \text{(for maximization)}.$$

- **Local vs. Global Optima:** A nonlinear function may have multiple local minima or maxima, making it challenging to ensure the solution is globally optimal.
- **Non-Convexity:** is non-convex, standard optimization methods may converge to a local optimum.
- **Computational Complexity:** Solving high-dimensional nonlinear problems often requires efficient numerical algorithms.

The analysis of $f(x)$ relies on the following mathematical tools:

- **Gradient**.
- **Hessian**.

These tools form the basis for solving optimization problems using methods such as gradient descent, Newton's method, and quasi-Newton methods.

3.2 Positivity, Convexity, Minimum

The three fundamental concepts in nonlinear optimization: positivity, convexity, and minimum play a crucial role in characterizing and solving optimization problems.

3.2.1 Positivity

A function $f(x)$ is said to be **positive definite** if it satisfies the following condition:

$$f(x) > 0 \quad \forall x \neq 0, \quad \text{and} \quad f(0) = 0.$$

Positive definiteness ensures that the function always has a unique minimum at $x = 0$.

For example, the quadratic function:

$$f(x) = x^2 + y^2$$

is positive definite since:

$$f(x, y) > 0 \quad \text{for all} \quad (x, y) \neq (0, 0), \quad \text{and} \quad f(0, 0) = 0.$$

A related concept is **positive semi-definiteness**, where:

$$f(x) \geq 0 \quad \forall x \in R^n.$$

3.2.2 Convexity

A function $f(x)$ is **convex** if, for all $x_1, x_2 \in R^n$ and $\lambda \in [0, 1]$, the following inequality holds:

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2).$$

This property ensures that any line segment drawn between two points on the graph of $f(x)$ lies above or on the graph itself.

For twice-differentiable functions, convexity can be characterized using the Hessian matrix $\nabla^2 f(x)$. Specifically:

- A twice-differentiable function $f(x)$ is **convex** if and only if its Hessian matrix $\nabla^2 f(x)$ is **positive semi-definite** for all $x \in R^n$. This means that:

$$v^\top \nabla^2 f(x) v \geq 0 \quad \forall v \in R^n.$$

Equivalently, all eigenvalues of $\nabla^2 f(x)$ must be non-negative.

- A function is **strictly convex** if its Hessian matrix $\nabla^2 f(x)$ is **positive definite** for all $x \in R^n$. This means that:

$$v^\top \nabla^2 f(x) v > 0 \quad \forall v \neq 0.$$

To rigorously verify positive semi-definiteness of the Hessian matrix, one can:

1. Compute the eigenvalues of $\nabla^2 f(x)$ and confirm that they are all non-negative.
2. Alternatively, use the principal minors test: check that all leading principal minors of $\nabla^2 f(x)$ are non-negative.

Why Positive Semi-Definiteness Ensures Convexity

The condition that $\nabla^2 f(x)$ is positive semi-definite ensures that the second-order term in the Taylor expansion of $f(x)$ around any point x_0 satisfies:

$$f(x) \geq f(x_0) + \nabla f(x_0)^\top (x - x_0),$$

which is a defining property of convex functions. This result is a standard theorem in optimization theory (see, e.g., Boyd and Vandenberghe, *Convex Optimization*, Chapter 3).

Examples of Convex Functions

1. $f(x) = x^2$ is convex because its second derivative $\frac{d^2 f}{dx^2} = 2 > 0$, making the Hessian positive definite.

2. $f(x) = e^x$ is convex since $\frac{d^2f}{dx^2} = e^x > 0$ for all $x \in R$, ensuring the Hessian is positive definite.

3. $f(x) = -\ln(x)$ is convex on its domain $x > 0$ because $\frac{d^2f}{dx^2} = \frac{1}{x^2} > 0$, making the Hessian positive definite.

3.2.3 Examples of Convex Functions

1. $f(x) = x^2$ is convex because its second derivative $\frac{d^2f}{dx^2} = 2 > 0$. 2. $f(x) = e^x$ is convex since $\frac{d^2f}{dx^2} = e^x > 0$ for all $x \in R$. 3. $f(x) = \ln(x)$ is convex on its domain $x > 0$ as $\frac{d^2f}{dx^2} = -\frac{1}{x^2} < 0$.

3.2.4 Minimum

The **minimum** of a function $f(x)$ occurs at a point x^* if:

$$f(x^*) \leq f(x), \quad \forall x \in R^n.$$

The function value $f(x^*)$ is called the **global minimum** if the inequality holds for all x. If it holds only within a neighborhood of x^*, x^* is a **local minimum**.

3.2.5 Example: Quadratic Function

Consider $f(x) = x_1^2 + 2x_2^2 - 2x_1x_2 + 3$. To analyze the convexity of this function, we proceed as follows:

3.2.5.1 Compute the Hessian Matrix

The Hessian matrix of $f(x)$ is:

$$\nabla^2 f(x) = \begin{bmatrix} 2 & -1 \\ -1 & 4 \end{bmatrix}.$$

3.2.5.2 Verify Positive Definiteness Using Principal Minors

To determine whether the Hessian matrix is positive definite (and hence confirm that $f(x)$ is strictly convex), we use the **principal minor criterion**.

Definition of Principal Minors For a symmetric matrix A, the **principal minors** are the determinants of the submatrices formed by taking the top-left corners of A:

- The **first principal minor**, denoted Δ_1, is the determinant of the top-left 1×1 submatrix.

- The **second principal minor**, denoted Δ_2, is the determinant of the entire 2×2 matrix.

In general, for an $n \times n$ matrix, there are n principal minors, corresponding to the determinants of the $1 \times 1, 2 \times 2, \ldots, n \times n$ leading submatrices.

Application to the Hessian Matrix For the Hessian matrix $\nabla^2 f(x)$, we compute the principal minors as follows:

1. **First principal minor (Δ_1)**:

$$\Delta_1 = \det([2]) = 2.$$

   Since $\Delta_1 > 0$, the first principal minor is positive.

2. **Second principal minor (Δ_2)**:

$$\Delta_2 = \det\left(\begin{bmatrix} 2 & -1 \\ -1 & 4 \end{bmatrix}\right) = (2)(4) - (-1)(-1) = 8 - 1 = 7.$$

   Since $\Delta_2 > 0$, the second principal minor is also positive.

Interpretation of Results For a symmetric matrix to be **positive definite**, all its principal minors must be positive. In this case:

$$\Delta_1 = 2 > 0 \quad \text{and} \quad \Delta_2 = 7 > 0.$$

Thus, the Hessian matrix $\nabla^2 f(x)$ is positive definite.

3.2.5.3 Conclusion

Since the Hessian matrix is positive definite, the quadratic function $f(x)$ is strictly convex. Therefore, $f(x)$ has a unique global minimum.

3.2.5.4 Additional Clarification on Positive Semi-Definiteness

If we were checking for **positive semi-definiteness** instead of positive definiteness, we would require that all principal minors be non-negative ($\Delta_i \geq 0$) rather than strictly positive. For example:

- $\Delta_1 \geq 0$,
- $\Delta_2 \geq 0$.

In this specific example, since both $\Delta_1 > 0$ and $\Delta_2 > 0$, the Hessian matrix is not only positive semi-definite but also positive definite.

3.2.6 Geometric Interpretation of Convexity

A convex function appears "bowl-shaped," meaning any local minimum is also a global minimum. This property simplifies optimization since gradient-based methods will converge to the global minimum.

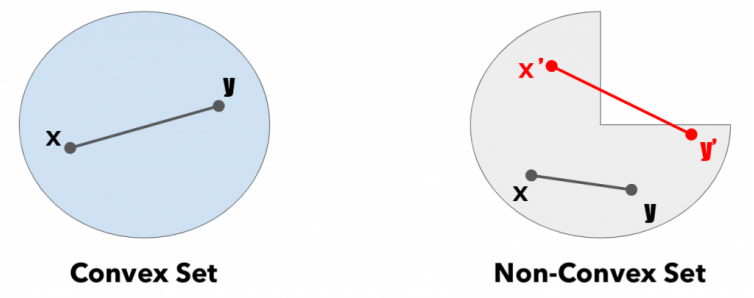

Figure 3.1: convex and non-convex sets.

3.3 Gradient and Hessian

The gradient and Hessian are fundamental tools for analyzing and solving nonlinear optimization problems.

3.3.1 Gradient

The gradient of $f(x)$, denoted $\nabla f(x)$, is the vector of partial derivatives:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}.$$

3.3.2 Hessian

The Hessian of $f(x)$, denoted $\nabla^2 f(x)$, is the matrix of second-order partial derivatives:

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}.$$

3.3.3 Example

For $f(x_1, x_2) = x_1^2 + 3x_1x_2 + 2x_2^2$:

$$\nabla f(x) = \begin{bmatrix} 2x_1 + 3x_2 \\ 3x_1 + 4x_2 \end{bmatrix}, \quad \nabla^2 f(x) = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}.$$

3.4 Necessary Conditions for a Minimum

For a differentiable function $f(x)$, x^* is a stationary point if:

$$\nabla f(x^*) = 0.$$

If $\nabla^2 f(x^*)$ is positive definite, x^* is a local minimum.

3.5 Sufficient Conditions for a Minimum

For a twice-differentiable function $f(x)$, x^* is a local minimum if:

$$\nabla f(x^*) = 0 \quad \text{and} \quad \nabla^2 f(x^*) > 0.$$

3.6 Local Methods

Local methods are optimization techniques that iteratively refine a candidate solution to approach a local minimum of a function. These methods rely on information from the function's derivatives, such as the gradient and Hessian, to guide the search for the optimal solution. Local methods are particularly effective when the initial guess is near the optimal point and the function is smooth and differentiable.

3.6.1 Key Idea of Local Methods

The general principle behind local methods is to start with an initial point x_0 and iteratively improve it using the relationship:

$$x_{k+1} = x_k + d_k,$$

where d_k is the search direction determined by the method, and k represents the iteration number.

The search direction d_k is chosen based on: - First-order information (gradient). - Second-order information (Hessian) for more accurate updates.

3.6.2 Descent Property

Local methods ensure that the new point x_{k+1} improves the objective function value:

$$f(x_{k+1}) < f(x_k).$$

To guarantee improvement, d_k must be a descent direction, satisfying:

$$\nabla f(x_k)^\top d_k < 0.$$

This property ensures that the direction d_k points downhill, reducing the function value.

3.6.3 General Steps of Local Methods

- **1. Initialization**: Choose an initial point x_0 and set the iteration counter $k = 0$.
- **2. Compute the Search Direction** : Calculate d_k based on the method used (e.g., gradient descent, Newton's method).
- **3. Line Search**: Determine the step size α_k by solving:

$$\alpha_k = \arg\min_{\alpha>0} f(x_k + \alpha d_k).$$

- **4. Update**: Compute the next point:

$$x_{k+1} = x_k + \alpha_k d_k.$$

- **5. Convergence Check**: Stop if $\|\nabla f(x_k)\|$ is below a predefined tolerance, or continue with $k = k + 1$.

3.6.4 Common Local Methods

Several local methods are commonly used in optimization. These include:

- **One-dimensional search methods**
- **Gradient methods**
- **Conjugate directions methods**
- **Newton method**
- **Quasi-Newton methods**

3.7 One-Dimensional Search Methods

One-dimensional search methods are optimization techniques used to find the minimum or maximum of a function $f(x)$ along a single direction. These methods are widely used as subroutines in multidimensional optimization to determine the optimal step size in iterative algorithms, such as gradient descent or Newton's method.

3.7.1 Problem Definition

Given a continuous function $f(x)$, the goal is to find:

$$x^* = \arg\min_{x \in [a,b]} f(x),$$

where $[a, b]$ is a closed interval. The methods iteratively reduce the size of the interval containing the minimum until convergence is achieved.

3.7.2 Key Concepts

- **Bracketing**: An interval $[a, b]$ is said to bracket a minimum if:

$$f(a) > f(x^*) \quad \text{and} \quad f(b) > f(x^*).$$

- **Convergence**: The interval size $|b - a|$ decreases with each iteration until x^* is sufficiently close to the true minimum.

3.7.3 Popular One-Dimensional Search Methods

3.7.3.1 Golden Section Search

The golden section search reduces the interval size by evaluating the function at two points within the interval:

$$x_1 = a + \phi(b - a), \quad x_2 = b - \phi(b - a),$$

where $\phi = \frac{\sqrt{5}-1}{2} \approx 0.618$ is the golden ratio.

Algorithm

- **1. Initialize**: a, b and compute x_1, x_2.
- **2. Evaluate**: $f(x_1)$ and $f(x_2)$:
  - - If $f(x_1) < f(x_2)$, update $b = x_2$ (narrow the interval to the left).
  - - If $f(x_1) \geq f(x_2)$, update $a = x_1$ (narrow the interval to the right).
- **3. Repeat until**: $|b - a|$ is smaller than a predefined tolerance ϵ.
- **4. The approximate minimum is**: $x^* = \frac{a+b}{2}$.

Example Minimize $f(x) = (x - 2)^2$ over $[1, 5]$:

- **1. Compute**: $x_1 = 1 + 0.618(5 - 1) = 3.472$ and $x_2 = 5 - 0.618(5 - 1) = 2.528$.
- **2. Evaluate**: $f(x_1) = (3.472 - 2)^2 = 2.168$ and $f(x_2) = (2.528 - 2)^2 = 0.278$.
- **3. Since**: $f(x_2) < f(x_1)$, update $b = x_1 = 3.472$.
- **4. Repeat until**: $|b - a| < \epsilon$.

3.7.3.2 Bisection Method

The bisection method divides the interval $[a, b]$ into two equal parts and evaluates the derivative $f'(x)$ at the midpoint:

$$x_m = \frac{a+b}{2}.$$

Algorithm

- **1. Evaluate**: $f'(x_m)$:
  - - If $f'(x_m) = 0$, x_m is the minimum.
  - - If $f'(x_m) > 0$, update $b = x_m$ (narrow the interval to the left).
  - - If $f'(x_m) < 0$, update $a = x_m$ (narrow the interval to the right).
- **2. Repeat until**: $|b - a| < \epsilon$.
- **3. The approximate minimum is**: $x^* = x_m$.

Example Minimize $f(x) = x^2 - 4x + 4$ over $[0, 4]$:

- **1. Compute**: $x_m = (0 + 4)/2 = 2$.
- **2. Evaluate**: $f'(x) = 2x - 4$:
  - - At $x_m = 2$, $f'(2) = 0$.
- **3. The minimum is at**: $x^* = 2$.

3.7.3.3 Newton's Method (1D)

Newton's method uses second-order derivative information to find the minimum:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

Algorithm

- **1. Start with an initial guess**: x_0.
- **2. Update**: x_k iteratively using the above formula.
- **3. Stop when**: $|f'(x_k)| < \epsilon$.

Example

Minimize $f(x) = x^2 - 4x + 4$:

- **1. Compute**: $f'(x) = 2x - 4$ and $f''(x) = 2$.
- **2. Starting with**: $x_0 = 3$:
  - - $x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)} = 3 - \frac{2(3)-4}{2} = 2$.
- **3. The minimum is at**: $x^* = 2$.

3.7.3.4 Secant Method

The secant method approximates the derivative using two points:

$$x_{k+1} = x_k - f'(x_k)\frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}.$$

3.7.4 Comparison of Methods

Method	Advantages	Limitations
Golden Section Search	Does not require derivatives	Converges slowly
Bisection Method	Simple and robust	Requires derivatives
Newton's Method	Fast convergence for smooth functions	Requires second derivative
Secant Method	Faster than bisection, no second derivative	Sensitive to initial guess

3.8 Gradient Methods

Gradient methods are iterative optimization techniques that utilize the gradient of a function to find its minimum. They are widely used for solving unconstrained nonlinear optimization problems due to their simplicity and efficiency in practical applications.

3.8.1 Concept of Gradient Methods

The gradient $\nabla f(x)$ of a function $f(x)$ provides the direction of steepest ascent. To minimize $f(x)$, we move in the opposite direction of the gradient. Gradient methods generate a sequence of points $\{x_k\}$ that iteratively approach the minimum.

The general update rule is:

$$x_{k+1} = x_k + \alpha_k d_k,$$

where:

- x_k: the current point,

- d_k: the search direction, often chosen as $-\nabla f(x_k)$ (steepest descent),
- $\alpha_k > 0$: the step size or learning rate.

3.8.2 Types of Gradient Methods

Gradient methods can vary depending on how the step size α_k and direction d_k are chosen. Below are the most common types:

- **Steepest Descent Method:** The search direction is the negative gradient:

$$d_k = -\nabla f(x_k).$$

- **Fixed Step Size:** The step size α_k remains constant throughout the iterations.
- **Adaptive Step Size:** The step size α_k is dynamically determined using a line search to ensure sufficient reduction in $f(x)$.
- **Gradient Descent with Momentum:** A memory term is added to accelerate convergence:

$$v_{k+1} = \beta v_k - \alpha_k \nabla f(x_k), \quad x_{k+1} = x_k + v_{k+1},$$

  where $\beta \in [0, 1)$ controls the momentum.

3.8.3 Convergence of Gradient Methods

Gradient methods converge to a local minimum under the following conditions:

- The objective function $f(x)$ is continuously differentiable.
- The step size α_k satisfies sufficient descent conditions.
- The initial point x_0 is within the basin of attraction of the minimum.

3.8.4 Advantages and Limitations

Advantages:

- Easy to implement and computationally inexpensive.
- Effective for smooth and well-behaved functions.

Limitations:

- Slow convergence near the minimum for poorly conditioned problems.
- Sensitive to the choice of step size α_k.

3.8.5 Example: Gradient Descent for Minimizing

We aim to minimize the function:

$$f(x_1, x_2) = x_1^2 + 2x_2^2.$$

Steps

- **Gradient:** The gradient of $f(x_1, x_2)$ is:

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 \\ 4x_2 \end{bmatrix}.$$

- **Update Rule:** Using the gradient descent update formula:

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} - \alpha \nabla f(x_1^{(k)}, x_2^{(k)}),$$

  where α is the step size.

- **Initialization:** Start at $(x_1^{(0)}, x_2^{(0)}) = (1, 1)$.
- **Step Size:** Set $\alpha = 0.1$.
- **Convergence Criterion:** Stop when:

$$\|\nabla f(x_1^{(k)}, x_2^{(k)})\| < \epsilon,$$

  with $\epsilon = 10^{-6}$.

Iterations The first few iterations of the gradient descent algorithm are as follows:

- Iteration 0: $(x_1, x_2) = (1, 1)$, Gradient $\nabla f = [2, 4]$, Updated $(x_1, x_2) = (0.8, 0.6)$.
- Iteration 1: $(x_1, x_2) = (0.8, 0.6)$, Gradient $\nabla f = [1.6, 2.4]$, Updated $(x_1, x_2) = (0.64, 0.36)$.
- Iteration 2: $(x_1, x_2) = (0.64, 0.36)$, Gradient $\nabla f = [1.28, 1.44]$, Updated $(x_1, x_2) = (0.512, 0.216)$.

- Iteration 3: $(x_1, x_2) = (0.512, 0.216)$, Gradient $\nabla f = [1.024, 0.864]$, Updated $(x_1, x_2) = (0.4096, 0.1296)$.

Results After several iterations, the algorithm converges to the minimum at:

$$x^* = (0, 0), \quad f(x^*) = 0.$$

Convergence Plot

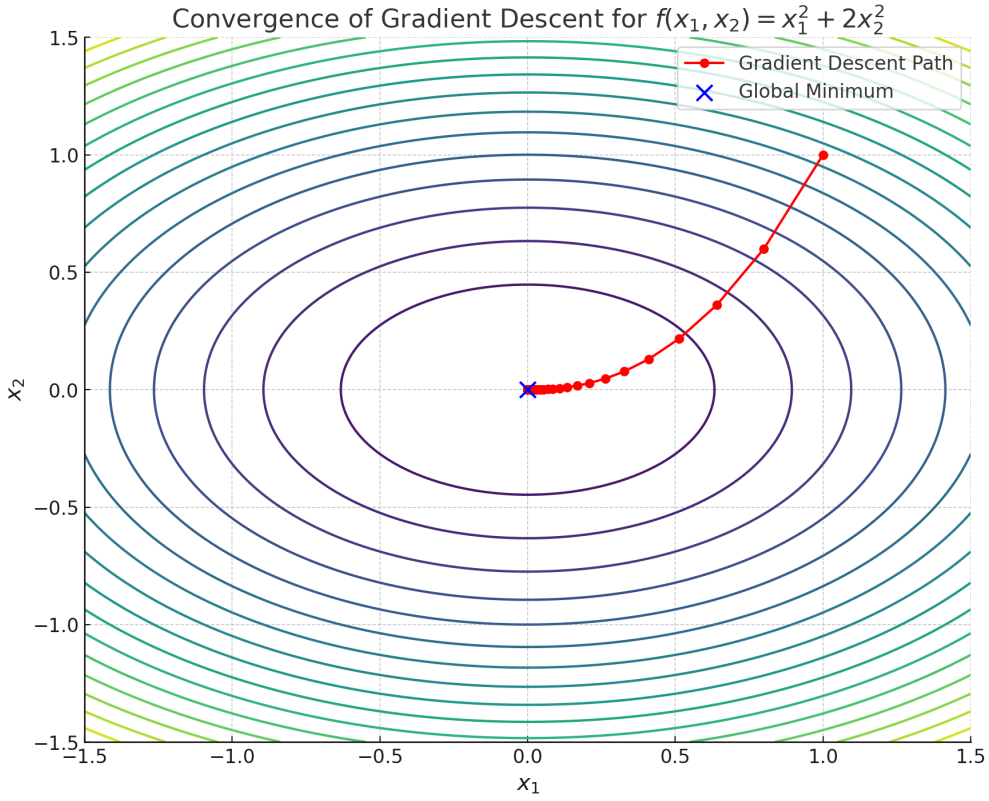

Figure 3.2: Convergence of Gradient Descent for $f(x_1, x_2) = x_1^2 + 2x_2^2$.

Description of the Plot

- The **contour plot** represents the objective function $f(x_1, x_2) = x_1^2 + 2x_2^2$, with ellipses indicating equal function values.
- The **red dots** show the points evaluated during each iteration of gradient descent.
- The **red dashed line** illustrates the path of convergence to the global minimum.

This example demonstrates how gradient descent iteratively reduces the value of the objective function by leveraging the negative gradient direction and systematically approaches the minimum.

3.9 Conjugate Direction Methods

3.9.1 Overview

Conjugate Direction Methods are optimization techniques for solving systems of linear equations and unconstrained quadratic minimization problems. These methods improve over traditional gradient-based methods by selecting search directions that are mutually conjugate with respect to the Hessian matrix A. This ensures faster convergence, especially for quadratic functions.

3.9.2 Problem Definition

Conjugate Direction Methods solve problems of the form:

$$\min_x f(x) = \frac{1}{2}x^T Ax - b^T x,$$

where:

- A is a symmetric positive definite matrix ($A^T = A$, and $x^T Ax > 0$ for all $x \neq 0$),
- b is a vector,
- x is the vector of variables to be optimized.

For such problems, the gradient is $\nabla f(x) = Ax - b$, and the Hessian matrix is A.

3.9.3 Key Concepts

- **Conjugacy with Respect to A:** Vectors p_i and p_j are A-conjugate if:

$$p_i^T Ap_j = 0 \quad \text{for } i \neq j.$$

This property ensures that the search directions are independent and do not interfere with each other during optimization.

- **Solution in Finite Steps:** For an n-dimensional quadratic problem, the Conjugate Direction Method converges in at most n iterations, as the A-conjugate directions span the n-dimensional space.

- **Algorithm Outline:** The solution is constructed iteratively:

$$x_{k+1} = x_k + \alpha_k p_k,$$

where:

- α_k is the step size along the search direction p_k,

- p_k is the conjugate search direction.
- **Search Direction Update:** The search direction is updated to maintain conjugacy:

$$p_{k+1} = r_{k+1} + \beta_k p_k,$$

where:

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

- **Residual Update:** The residual represents the gradient of the function:

$$r_k = b - Ax_k.$$

3.9.4 Derivation of the Formula for β_k (Fletcher-Reeves Method)

The formula for β_k arises from the requirement to maintain A-conjugacy between successive search directions. Below, we provide a detailed derivation of the Fletcher-Reeves formula for β_k:

3.9.4.1 Step 1: Define the Search Direction

The search direction at iteration $k+1$ is defined as:

$$p_{k+1} = r_{k+1} + \beta_k p_k,$$

where $r_{k+1} = b - Ax_{k+1}$ is the residual at iteration $k+1$. To ensure conjugacy, the new search direction p_{k+1} must satisfy:

$$p_{k+1}^T A p_k = 0.$$

3.9.4.2 Step 2: Expand the Conjugacy Condition

Substitute $p_{k+1} = r_{k+1} + \beta_k p_k$ into the conjugacy condition:

$$(r_{k+1} + \beta_k p_k)^T A p_k = 0.$$

Expanding this expression:

$$r_{k+1}^T A p_k + \beta_k p_k^T A p_k = 0.$$

3.9.4.3 Step 3: Simplify Using Properties of A-Conjugacy

From the definition of A-conjugacy, $p_k^T A p_k \neq 0$, and $r_{k+1}^T A p_k = 0$ because residuals are orthogonal to previous search directions. Thus, the equation simplifies to:

$$\beta_k p_k^T A p_k = 0.$$

3.9.4.4 Step 4: Solve for β_k

To compute β_k, we use the relationship between the residuals and the search directions. Specifically, in the Fletcher-Reeves method, β_k is chosen to minimize the error in the next iteration. This leads to:

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

3.9.4.5 Significance of β_k

The parameter β_k plays a critical role in ensuring both A-conjugacy of the search directions and rapid convergence of the algorithm:

- **Conjugacy:** By updating p_{k+1} using β_k, the new search direction remains A-conjugate to all previous search directions.

- **Convergence:** The Fletcher-Reeves formula ensures that the algorithm converges in at most n iterations for quadratic problems, leveraging the orthogonality of residuals.

- **Efficiency:** The computation of β_k is straightforward and does not require additional matrix-vector products, making it computationally efficient.

3.9.4.6 Context of the Fletcher-Reeves Method

The Fletcher-Reeves method is one of several conjugate gradient methods used for solving large-scale optimization problems. It is particularly well-suited for quadratic functions due to its finite convergence property. Other variants, such as Polak-Ribière and Hestenes-Stiefel, modify the formula for β_k to improve performance in non-quadratic settings.

3.9.5 Algorithm Steps

1. **Initialize:**

   - Set the initial guess x_0 (e.g., $x_0 = 0$),
   - Compute the initial residual $r_0 = b - Ax_0$,
   - Set the first search direction $p_0 = r_0$.

2. **Iterative Steps (for $k = 0, 1, \ldots, n-1$):**

   - Compute step size:

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}.$$

- Update the solution:

$$x_{k+1} = x_k + \alpha_k p_k.$$

- Update the residual:

$$r_{k+1} = r_k - \alpha_k A p_k.$$

- Compute β_k to ensure conjugacy:

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

- Update the search direction:

$$p_{k+1} = r_{k+1} + \beta_k p_k.$$

3. Repeat until convergence.

3.9.6 Example

Problem Statement:

Minimize:

$$f(x) = \frac{1}{2} x^T A x - b^T x,$$

where:

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Step 1: Initial Residual and Direction

$$r_0 = b - A x_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Set $p_0 = r_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

Step 2: Compute Step Size α_0:

$$\alpha_0 = \frac{r_0^T r_0}{p_0^T A p_0}.$$

Substitute values:

$$r_0^T r_0 = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 1^2 + 2^2 = 5.$$

$$p_0^T A p_0 = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \end{bmatrix} = 1(6) + 2(7) = 20.$$

$$\alpha_0 = \frac{5}{20} = \frac{1}{4}.$$

Step 3: Update Solution and Residual

$$x_1 = x_0 + \alpha_0 p_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \end{bmatrix}.$$

$$r_1 = r_0 - \alpha_0 A p_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \frac{1}{4} \begin{bmatrix} 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} \frac{3}{2} \\ \frac{7}{4} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{4} \end{bmatrix}.$$

Step 4: Compute Next Direction

$$\beta_0 = \frac{r_1^T r_1}{r_0^T r_0}.$$

$$r_1^T r_1 = \begin{bmatrix} -\frac{1}{2} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{4} \end{bmatrix} = \frac{1}{4} + \frac{1}{16} = \frac{5}{16}.$$

$$\beta_0 = \frac{\frac{5}{16}}{5} = \frac{1}{16}.$$

$$p_1 = r_1 + \beta_0 p_0 = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{4} \end{bmatrix} + \frac{1}{16} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} + \frac{1}{16} \\ \frac{1}{4} + \frac{2}{16} \end{bmatrix}.$$

Repeat until convergence.

3.9.7 Advantages

- Guarantees convergence in n steps for quadratic problems.
- Efficient for large-scale optimization problems when combined with preconditioning.

3.9.8 Applications

- Numerical solutions for partial differential equations.
- Structural analysis in engineering.
- Machine learning and data fitting.

3.10 Newton's Method

3.10.1 Overview

Newton's Method is a widely used iterative optimization technique for solving nonlinear equations and unconstrained optimization problems. It is particularly effective for problems where the objective function $f(x)$ is twice differentiable. The method leverages second-order information (the Hessian matrix) to achieve quadratic convergence near the solution, making it faster than first-order methods like gradient descent in many cases.

3.10.2 Problem Definition

Newton's Method is used to solve optimization problems of the form:

$$\min_x f(x),$$

where $f(x)$ is a smooth, twice differentiable function. The iterative update rule is based on a second-order Taylor series expansion of $f(x)$.

3.10.3 Key Concepts

- **Taylor Series Approximation:** Near a point x_k, the function $f(x)$ can be approximated as:

$$f(x) \approx f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k),$$

  where:

  - $\nabla f(x_k)$ is the gradient at x_k,
  - $H(x_k)$ is the Hessian matrix (matrix of second derivatives) at x_k.

- **Update Rule:** The minimizer of the quadratic approximation is used to compute the next iteration:

$$x_{k+1} = x_k - H(x_k)^{-1} \nabla f(x_k).$$

- **Convergence:** When x_k is close to the optimal solution x^*, the method exhibits quadratic convergence, meaning the error decreases at an exponential rate.

- **Requirements:**

- The Hessian $H(x_k)$ must be positive definite to ensure that $f(x)$ is locally convex.
- Computing $H(x_k)^{-1}$ can be computationally expensive, especially for high-dimensional problems.

3.10.4 Algorithm Steps

1. **Initialize:**
   - Choose an initial guess x_0.
2. **Iterative Steps (for $k = 0, 1, 2, \ldots$):**
   - Compute the gradient:
   $$g_k = \nabla f(x_k).$$
   - Compute the Hessian matrix:
   $$H_k = H(x_k).$$
   - Update the solution:
   $$x_{k+1} = x_k - H_k^{-1} g_k.$$
3. **Stop:** When $\|g_k\|$ is below a predefined tolerance, x_k is considered the optimal solution.

3.10.5 Example

Problem Statement:

Minimize:

$$f(x) = x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2.$$

Step 1: Compute Gradient and Hessian

The gradient of $f(x)$ is:

$$\nabla f(x) = \begin{bmatrix} 2x_1 - 2x_2 - 4 \\ 4x_2 - 2x_1 - 6 \end{bmatrix}.$$

The Hessian matrix of $f(x)$ is:

$$H(x) = \begin{bmatrix} 2 & -2 \\ -2 & 4 \end{bmatrix}.$$

Step 2: Initialize

Let the initial guess be:

$$x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Compute:

$$\nabla f(x_0) = \begin{bmatrix} -4 \\ -6 \end{bmatrix}, \quad H(x_0) = \begin{bmatrix} 2 & -2 \\ -2 & 4 \end{bmatrix}.$$

Step 3: Compute Update

The update rule is:

$$x_1 = x_0 - H(x_0)^{-1} \nabla f(x_0).$$

Compute the inverse of $H(x_0)$:

$$H(x_0)^{-1} = \frac{1}{4} \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}.$$

Multiply:

$$H(x_0)^{-1} \nabla f(x_0) = \frac{1}{4} \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} -4 \\ -6 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}.$$

Update:

$$x_1 = x_0 + \begin{bmatrix} -2 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}.$$

Step 4: Repeat

Using x_1, repeat the process until convergence. The optimal solution is $x^* = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$.

3.10.6 Advantages

- Quadratic convergence near the solution.
- Utilizes second-order information for more accurate updates.

3.10.7 Disadvantages

- Computationally expensive due to Hessian inversion.
- Requires $H(x_k)$ to be positive definite.

3.10.8 Applications

- Nonlinear optimization problems in engineering and physics.
- Machine learning (e.g., logistic regression, support vector machines).
- Solving systems of nonlinear equations.

3.11 Quasi-Newton Methods

3.11.1 Overview

Quasi-Newton Methods are iterative optimization techniques used to solve nonlinear optimization problems. They aim to approximate Newton's Method by avoiding the explicit computation of the Hessian matrix, which can be computationally expensive. Instead, Quasi-Newton Methods build an approximation to the Hessian matrix or its inverse using gradient information from previous iterations. These methods are especially useful for large-scale optimization problems.

3.11.2 Problem Definition

Quasi-Newton Methods solve optimization problems of the form:

$$\min_x f(x),$$

where $f(x)$ is a smooth, twice differentiable function. Instead of directly using the Hessian matrix $H(x_k)$, these methods iteratively update an approximation B_k (or B_k^{-1}) based on gradients.

3.11.3 Key Concepts

- **Hessian Approximation:** The Hessian matrix $H(x)$ is approximated by B_k, which is updated iteratively to satisfy the *secant equation*:

$$B_{k+1}s_k = y_k,$$

where:

- $s_k = x_{k+1} - x_k$ is the step vector,
- $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ is the change in gradients.

- **Update Rule:** The general update formula for the inverse Hessian approximation B_k^{-1} is:

$$B_{k+1}^{-1} = B_k^{-1} + \text{correction terms.}$$

- **Invertibility of B_k:** The matrix B_k must remain invertible throughout the iterations for the method to proceed. This invertibility is guaranteed under certain conditions:

  - **Positive Definiteness:** In methods like BFGS, B_k remains positive definite if the curvature condition holds:

$$s_k^T y_k > 0.$$

  This ensures that B_k is not only invertible but also well-conditioned. - **Secant Equation:** The secant equation $B_{k+1}s_k = y_k$ ensures that the updates preserve consistency with the true Hessian's behavior along the search direction. - **Initialization:** Typically, B_0 is initialized as the identity matrix, which is positive definite and invertible. Subsequent updates maintain this property under the above conditions.

- **What Happens if B_k Becomes Singular?** If B_k becomes singular or ill-conditioned, the method may fail because the search direction $p_k = -B_k^{-1}\nabla f(x_k)$ cannot be computed reliably. In such cases:

- A restart mechanism may be employed, reinitializing B_k as the identity matrix.

- Alternatively, numerical safeguards, such as regularization (adding a small multiple of the identity matrix), can be applied to restore invertibility.

- **Quasi-Newton Direction:** The search direction is given by:

$$p_k = -B_k^{-1}\nabla f(x_k).$$

- **Convergence:** Quasi-Newton Methods converge superlinearly and are less computationally expensive than Newton's Method.

3.11.4 Popular Quasi-Newton Algorithms

- **DFP (Davidon-Fletcher-Powell):** Updates B_k^{-1} using both s_k and y_k while ensuring positive definiteness.

$$B_{k+1}^{-1} = B_k^{-1} + \frac{s_k s_k^T}{s_k^T y_k} - \frac{(B_k^{-1}y_k)(B_k^{-1}y_k)^T}{y_k^T B_k^{-1} y_k}.$$

- **BFGS (Broyden-Fletcher-Goldfarb-Shanno):** One of the most widely used Quasi-Newton methods due to its numerical stability and efficiency. The update formula for B_k^{-1} is:

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}.$$

The BFGS update preserves positive definiteness as long as $s_k^T y_k > 0$, ensuring B_k remains invertible.

- **Limited-Memory BFGS (L-BFGS):** A memory-efficient variant of BFGS suitable for high-dimensional problems. Instead of storing the full matrix B_k^{-1}, it uses a limited number of recent updates. The L-BFGS method also relies on the curvature condition $s_k^T y_k > 0$ to maintain numerical stability.

3.11.5 Algorithm Steps (General Quasi-Newton Method)

1. **Initialize:**

   - Choose an initial guess x_0,

   - Set an initial Hessian approximation $B_0 = I$ (identity matrix),

   - Compute the initial gradient $g_0 = \nabla f(x_0)$.

2. **Iterative Steps (for $k = 0, 1, 2, \dots$):**

   - Compute the search direction:

   $$p_k = -B_k^{-1} \nabla f(x_k).$$

   - Update the solution:

   $$x_{k+1} = x_k + \alpha_k p_k,$$

   where α_k is determined using a line search.

   - Compute:

   $$s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k).$$

   - Check the curvature condition:

   $$s_k^T y_k > 0.$$

   If the condition is violated, apply a safeguard (e.g., restart B_k as the identity matrix or regularize B_k).

   - Update B_k^{-1} or B_k using one of the Quasi-Newton update formulas (e.g., BFGS or DFP).

3. **Stop:** When $\|\nabla f(x_k)\|$ is below a predefined tolerance, x_k is considered the optimal solution.

3.11.6 Example

Problem Statement:

Minimize:

$$f(x) = x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2.$$

Step 1: Initialization

Let:

$$x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad B_0 = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Compute:

$$\nabla f(x_0) = \begin{bmatrix} -4 \\ -6 \end{bmatrix}.$$

Step 2: Compute Search Direction

$$p_0 = -B_0 \nabla f(x_0) = - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -4 \\ -6 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}.$$

Step 3: Update Solution

Perform a line search to determine α_0. Assume $\alpha_0 = 0.25$:

$$x_1 = x_0 + \alpha_0 p_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.25 \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}.$$

Step 4: Update B_k^{-1}

Compute:

$$s_0 = x_1 - x_0 = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}, \quad y_0 = \nabla f(x_1) - \nabla f(x_0).$$

Update B_0^{-1} using BFGS or DFP formulas. Repeat the process until convergence.

3.11.7 Advantages

- Avoids explicit computation of the Hessian matrix.
- Requires fewer iterations than gradient-based methods.
- Converges superlinearly for convex problems.

3.11.8 Disadvantages

- More computationally intensive than first-order methods.
- Requires gradient evaluations at each step.
- Storage requirements for B_k or B_k^{-1} can be high for large-scale problems (mitigated by L-BFGS).

3.11.9 Applications

- Machine learning (e.g., logistic regression, support vector machines).
- Nonlinear optimization problems in engineering and physics.
- Data fitting and regression analysis.

3.12 Exercises

One-Dimensional Search Methods

1. Given $f(x) = x^4 - 3x^3 + 2$, use the Golden Section Search method to find the minimum in the interval $[0, 2]$ with a tolerance of 10^{-3}.
2. Apply the Bisection method to minimize $f(x) = (x-1)^2 + e^x$ over $[0, 2]$. Perform three iterations and calculate the approximate minimum.
3. For the function $f(x) = \sin(x) + x^2$, implement the Fibonacci Search method to find the minimum within $[0, \pi]$. Stop when the interval length is less than 0.1.
4. Minimize $f(x) = \ln(1+x^2)$ using the Dichotomous Search method in the interval $[-1, 2]$ with $\epsilon = 0.01$.
5. Use the Secant method to minimize $f(x) = x^3 - 2x + 1$ with initial points $x_0 = 0$ and $x_1 = 1$. Perform three iterations.

Gradient Methods

1. Solve $\min f(x_1, x_2) = x_1^2 + x_2^2 - 2x_1x_2 + x_1 - x_2$ using the Gradient Descent method. Start at $(x_1, x_2) = (1, 1)$ with a step size of $\alpha = 0.1$. Perform three iterations.

2. Find the minimum of $f(x, y) = x^2 + y^2 + 4xy - 4x - 6y + 9$ using the Steepest Descent method starting at $(x, y) = (0, 0)$. Perform two iterations and calculate the gradients at each step.

3. For $f(x, y) = x^2 + 2y^2 - 2x - 4y$, implement a Gradient Descent method with exact line search. Start at $(x, y) = (0, 0)$ and find the minimum.

4. Minimize $f(x_1, x_2) = 2x_1^2 + x_2^2 + x_1x_2 + x_1 - x_2$ using Gradient Descent with $\alpha = 0.2$. Start at $(x_1, x_2) = (0, 0)$ and perform two iterations.

5. Prove that the Gradient Descent method converges for $f(x) = \frac{1}{2}x^TQx - b^Tx$ when Q is symmetric positive definite.

Conjugate Directions Methods

1. Solve min $f(x_1, x_2) = x_1^2 + x_2^2 - 2x_1x_2 - 4x_1 - 6x_2$ using the Conjugate Direction Method. Start at $(x_1, x_2) = (0, 0)$ and perform two iterations.

2. Minimize $f(x) = \frac{1}{2}x^TAx - b^Tx$, where $A = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, using the Conjugate Gradient method. Start at $x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

3. Show that the Conjugate Gradient method converges in at most n iterations for an n-dimensional quadratic problem with a symmetric positive definite A.

4. Implement the Conjugate Gradient method for $f(x_1, x_2, x_3) = x_1^2 + 3x_2^2 + 5x_3^2 - 2x_1x_2 - 4x_2x_3$ with initial guess $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$.

5. Compare the convergence rates of the Steepest Descent and Conjugate Gradient methods for $f(x) = x_1^2 + 10x_2^2$.

Newton Method

1. Minimize $f(x) = x^2 - 4x + 4$ using Newton's Method. Start at $x_0 = 0$ and perform two iterations.

2. For $f(x_1, x_2) = x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2$, compute the gradient and Hessian at $(x_1, x_2) = (0, 0)$. Use Newton's Method to find the minimum.

3. Solve $\min f(x) = e^x - x^2$ using Newton's Method starting at $x_0 = 1$. Perform three iterations.

4. Prove that Newton's Method has quadratic convergence when applied to $f(x) = \frac{1}{2}x^T Ax - b^T x$, where A is positive definite.

5. Implement Newton's Method for $f(x, y) = x^4 + y^4 - 2x^2y^2 + 2x^2 + 2y^2$ and find the stationary points.

Quasi-Newton Methods

1. Solve $\min f(x_1, x_2) = x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2$ using the BFGS algorithm. Start at $(x_1, x_2) = (0, 0)$ and perform two iterations.

2. For $f(x, y) = x^2 + y^2 + 4xy - 4x - 6y$, implement the DFP method to find the minimum. Start at $(x, y) = (1, 1)$.

3. Apply the Limited-Memory BFGS (L-BFGS) algorithm to minimize $f(x_1, x_2) = x_1^2 + 3x_2^2 + x_1x_2 - x_1 - x_2$ with an initial guess $(x_1, x_2) = (0, 0)$.

4. Derive the BFGS update formula for B_k^{-1} and verify that it satisfies the secant equation.

5. Compare the performance of the Gradient Descent and Quasi-Newton methods (e.g., BFGS) for minimizing $f(x_1, x_2) = x_1^4 + x_2^4 - 2x_1^2x_2^2 + x_1^2 + x_2^2$.

Chapter 4

Nonlinear optimization with constraints

4.1 Introduction

In the real world, many optimization problems are nonlinear in nature, meaning that either the objective function or the constraints (or both) are nonlinear. These problems arise across a wide range of applications, from engineering design to economics, operations research, and machine learning.

Nonlinear optimization with constraints involves finding the minimum (or maximum) of a nonlinear objective function while satisfying a set of equality and/or inequality constraints. The constraints define a feasible region within which the solution must lie, adding complexity compared to unconstrained optimization.

4.1.1 General Problem Formulation

A constrained nonlinear optimization problem can be written as:

$$\min_x f(x) \quad \text{subject to} \quad \begin{aligned} &g_i(x) \leq 0, \quad i = 1, \ldots, m \quad \text{(inequality constraints)}, \\ &h_j(x) = 0, \quad j = 1, \ldots, p \quad \text{(equality constraints)}, \end{aligned}$$

where:

- $f(x)$: Objective function to be minimized (or maximized),
- $g_i(x)$: Nonlinear inequality constraints,
- $h_j(x)$: Nonlinear equality constraints,

- $x \in R^n$: Decision variables.

The solution involves balancing the trade-off between minimizing the objective function and satisfying the constraints.

4.1.2 Challenges in Nonlinear Constrained Optimization

Nonlinear optimization with constraints poses several unique challenges:

- **Feasibility:** Ensuring the solution remains within the feasible region defined by the constraints.
- **Nonconvexity:** Nonlinear functions can lead to nonconvex problems, making it difficult to guarantee a global minimum.
- **Complexity:** Solving nonlinear equations for gradients, Hessians, and constraint interactions can be computationally intensive.
- **Divergence:** Iterative methods may diverge if not properly initialized or tuned.

4.1.3 Types of Constraints

Constraints in nonlinear optimization can be broadly classified into:

- **Equality Constraints:** These are equations of the form $h_j(x) = 0$ that must be satisfied exactly.
- **Inequality Constraints:** These are inequalities of the form $g_i(x) \leq 0$ that define bounds or limits on the feasible region.
- **Box Constraints:** These are simple bounds on the variables, such as $l_i \leq x_i \leq u_i$.

4.1.4 Optimality Conditions

For constrained optimization problems, optimality is determined by both the objective function and the constraints. Two key theoretical foundations are:

- **Lagrange Multipliers:** These introduce auxiliary variables to enforce constraints while minimizing the objective.

- **Karush-Kuhn-Tucker (KKT) Conditions:** These are necessary conditions for a solution to be optimal, provided certain regularity conditions are satisfied. The KKT conditions generalize the concept of Lagrange multipliers to include inequality constraints.

4.1.5 Solution Methods

Several methods have been developed to solve nonlinear optimization problems with constraints, including:

- **Penalty and Barrier Methods:** Transform the constrained problem into a series of unconstrained problems by penalizing or restricting constraint violations.
- **Sequential Quadratic Programming (SQP):** Solve a sequence of quadratic approximations to the problem.
- **Interior-Point Methods:** Maintain iterates strictly within the feasible region and gradually approach the optimal solution.
- **Augmented Lagrangian Methods:** Combine the benefits of penalty methods and Lagrange multipliers for improved convergence.

4.1.6 Applications

Constrained nonlinear optimization has widespread applications:

- **Engineering:** Structural optimization, resource allocation, and control systems.
- **Economics:** Utility maximization and equilibrium analysis.
- **Machine Learning:** Constrained optimization is often used in regularization and support vector machines.
- **Operations Research:** Optimization of logistics, transportation, and scheduling problems.

4.2 Lagrange Multipliers

The method of Lagrange multipliers is a powerful mathematical technique used to solve optimization problems with equality constraints. It provides a way to incorporate constraints into

the optimization process by introducing additional variables, known as Lagrange multipliers. This method transforms a constrained optimization problem into a system of equations that can be solved simultaneously.

4.2.1 Problem Formulation

Consider the problem of minimizing (or maximizing) a nonlinear objective function $f(x)$ subject to equality constraints $h_j(x) = 0$, for $j = 1, \ldots, p$. The problem can be expressed as:

$$\min_x f(x) \quad \text{subject to} \quad h_j(x) = 0, \quad j = 1, \ldots, p.$$

To solve this, the Lagrangian function is defined as:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{j=1}^{p} \lambda_j h_j(x),$$

where:

- $x \in R^n$: Decision variables,
- λ_j: Lagrange multipliers associated with each equality constraint $h_j(x) = 0$.

4.2.2 Optimality Conditions

The optimal solution x^* and the corresponding Lagrange multipliers λ^* must satisfy the following conditions, known as the **Lagrange conditions**:

1. **Stationarity:**
$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) + \sum_{j=1}^{p} \lambda_j^* \nabla h_j(x^*) = 0.$$
This ensures that the gradient of the objective function and the gradients of the constraints are aligned.

2. **Primal Feasibility:**
$$h_j(x^*) = 0, \quad \forall j.$$
The solution must satisfy all equality constraints.

3. **Dual Feasibility:** The values of the Lagrange multipliers λ_j^* are unconstrained in this case because the method only involves equality constraints.

4.2.3 Geometric Interpretation

The method of Lagrange multipliers can be interpreted geometrically as finding the point where the gradients of the objective function $f(x)$ and the constraints $h_j(x)$ are parallel. This ensures that the direction of steepest ascent (or descent) is restricted to the feasible region defined by the constraints.

4.2.4 Steps for Solving Using Lagrange Multipliers

1. **Define the Lagrangian Function**: Write the Lagrangian $\mathcal{L}(x, \lambda)$ by combining the objective function $f(x)$ and the constraints $h_j(x) = 0$.

2. **Set Up the Stationary Conditions**: Solve the equations:

$$\nabla_x \mathcal{L}(x, \lambda) = 0.$$

3. **Include the Constraints**: Add the constraint equations $h_j(x) = 0$ to the system.

4. **Solve the System**: Solve the resulting system of equations for x^* (optimal solution) and λ^* (Lagrange multipliers).

4.2.5 Applications

The method of Lagrange multipliers is widely used in:

- **Economics:** Solving utility maximization and cost minimization problems with budget constraints.
- **Engineering:** Optimizing system designs subject to resource or performance constraints.
- **Machine Learning:** Constrained optimization in support vector machines and regularization.
- **Physics:** Analyzing systems with constraints, such as the motion of particles on surfaces.

4.2.6 Advantages and Limitations

Advantages:

- Provides an analytical approach to solving constrained problems.
- Offers insights into the relationship between constraints and the objective function through the multipliers.

Limitations:

- Only applies to problems with equality constraints.
- Solving the resulting system of equations can be challenging for high-dimensional or nonconvex problems.

4.2.7 Example

Consider the optimization problem:

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

subject to the constraint:

$$h(x_1, x_2) = x_1 + x_2 - 1 = 0.$$

The goal is to minimize the quadratic objective function while ensuring that the solution lies on the line defined by the constraint.

Geometric Interpretation

The solution occurs at the point where the gradient of the objective function, ∇f, is aligned with the gradient of the constraint, ∇h. This alignment ensures that the search for the minimum respects the constraint.

Mathematical Solution

Minimize the objective function:

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

Subject to the constraint:

$$h(x_1, x_2) = x_1 + x_2 - 1 = 0$$

Solution using Lagrange Multipliers

Step 1: Form the Lagrangian

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

Step 2: Take Partial Derivatives

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

Step 3: Solve the System

From the first two equations:

$$2x_1 = 2x_2 \quad \Rightarrow \quad x_1 = x_2$$

Substitute into the constraint:

$$x_1 + x_1 = 1 \quad \Rightarrow \quad x_1 = \frac{1}{2}$$

Thus:

$$x_1^* = x_2^* = \frac{1}{2}$$

Step 4: Find Lagrange Multiplier

From the first equation:

$$2 \times \frac{1}{2} + \lambda = 0 \quad \Rightarrow \quad \lambda^* = -1$$

Final Answer

The optimal solution is:

$$x_1^* = \frac{1}{2}, \quad x_2^* = \frac{1}{2}, \quad \lambda^* = -1$$

MATLAB Code for Geometric Interpretation

Listing 4.1: MATLAB code for Geometric Interpretation

```
clc;
clear;
close all;

% Define Grid
x1 = linspace(-1, 2, 100);
x2 = linspace(-1, 2, 100);
[X1, X2] = meshgrid(x1, x2);

% Objective Function
F = X1.^2 + X2.^2;

% Plot Contours of f(x1, x2)
figure;
contour(X1, X2, F, 20, 'LineWidth', 1.5);
hold on;

% Constraint Line x1 + x2 = 1
x1_line = linspace(-1, 2, 100);
x2_line = 1 - x1_line;
plot(x1_line, x2_line, 'r-', 'LineWidth', 2);

% Optimal Point
x_opt = 0.5;
x2_opt = 0.5;
plot(x_opt, x2_opt, 'ko', 'MarkerSize', 10, 'MarkerFaceColor', 'k');

% Annotations
text(x_opt+0.1, x2_opt, 'Optimal Point (0.5, 0.5)', 'FontSize', 10);
xlabel('x_1');
ylabel('x_2');
title('Geometric Interpretation of Lagrange Multipliers');
legend('Contours of f(x_1, x_2)', 'Constraint x_1 + x_2 = 1', 'Optimal Point');

axis equal;
grid on;
hold off;
```

Figure

The figure below provides a geometric interpretation of the problem:

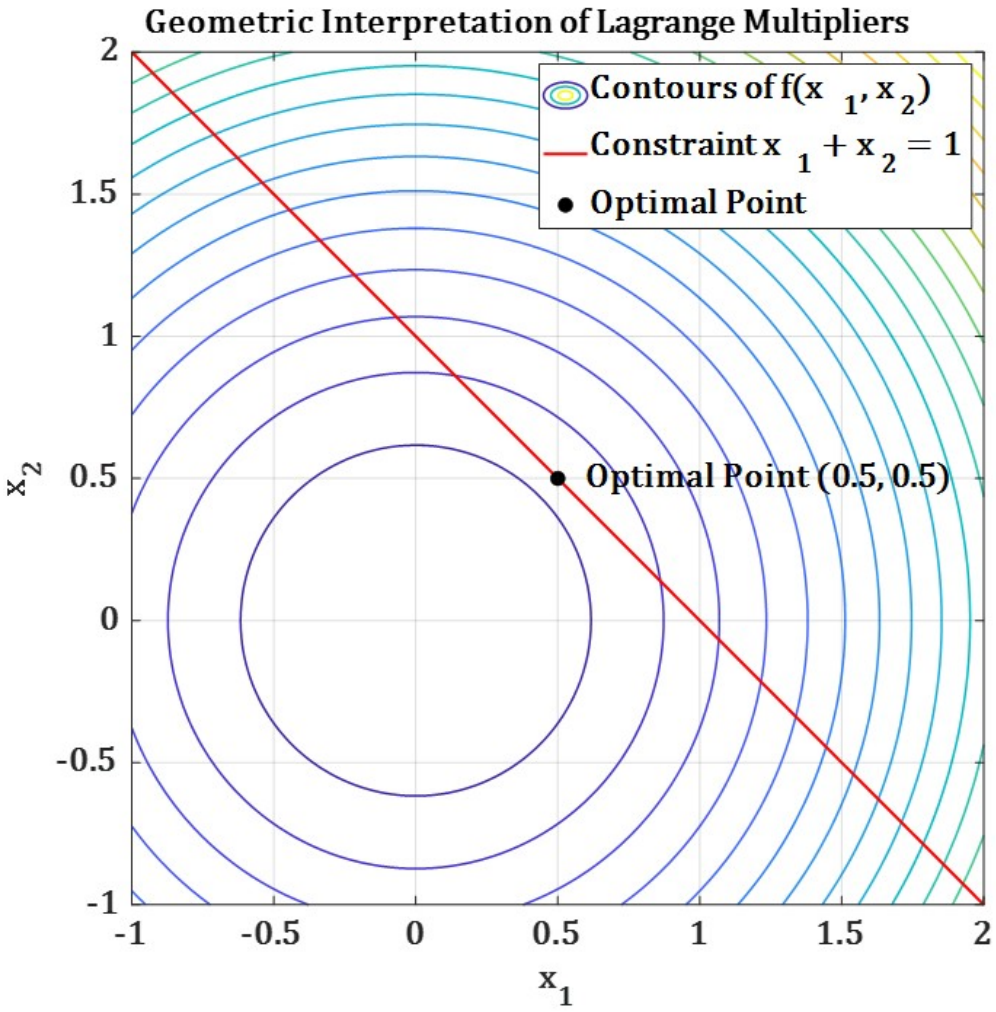

Figure 4.1: Geometric Interpretation of Lagrange Multipliers

Explanation of the Figure

- The contour lines represent the objective function $f(x_1, x_2)$, with lower levels indicating smaller values. - The red line represents the constraint $h(x_1, x_2) = 0$, defining the feasible region. - The blue point at $(x_1^*, x_2^*) = (0.5, 0.5)$ marks the optimal solution, where the gradients ∇f and ∇h are aligned. - The vector ∇f is scaled by the Lagrange multiplier λ^*, ensuring that the constraint is satisfied.

This geometric perspective demonstrates how Lagrange multipliers incorporate constraints

into the optimization process.

4.3 Karush-Kuhn-Tucker (KKT) Conditions

The Karush-Kuhn-Tucker (KKT) conditions are a set of necessary conditions for a solution to be optimal in nonlinear optimization problems with constraints. They generalize the method of Lagrange multipliers by accommodating inequality constraints and provide a powerful theoretical framework for constrained optimization.

The KKT conditions apply to optimization problems of the form:

$$\min_x f(x) \quad \text{subject to} \quad \begin{aligned} &g_i(x) \leq 0, \quad i = 1, \ldots, m \quad \text{(inequality constraints)}, \\ &h_j(x) = 0, \quad j = 1, \ldots, p \quad \text{(equality constraints)}. \end{aligned}$$

4.3.1 Key Assumptions

For the KKT conditions to hold, the following assumptions are typically required:

- The objective function $f(x)$, inequality constraints $g_i(x)$, and equality constraints $h_j(x)$ are differentiable.
- A regularity condition, such as the Linear Independence Constraint Qualification (LICQ), must be satisfied. This ensures that the gradients of active constraints are linearly independent.

4.3.2 KKT Conditions

Let x^* be a candidate solution, and let λ_j and μ_i be the Lagrange multipliers associated with the equality and inequality constraints, respectively. The KKT conditions are:

1. **Stationarity:**
$$\nabla f(x^*) + \sum_{j=1}^{p} \lambda_j \nabla h_j(x^*) + \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) = 0.$$
2. **Primal Feasibility:**
$$g_i(x^*) \leq 0, \quad \forall i, \quad h_j(x^*) = 0, \quad \forall j.$$
3. **Dual Feasibility:**
$$\mu_i \geq 0, \quad \forall i.$$

4. **Complementary Slackness:**

$$\mu_i g_i(x^*) = 0, \quad \forall i.$$

This condition ensures that each inequality constraint is either active ($g_i(x^*) = 0$) or its corresponding multiplier is zero ($\mu_i = 0$).

4.3.3 Geometric Interpretation

The KKT conditions can be interpreted as a balance between:

- The gradient of the objective function (∇f),
- The gradients of the equality constraints (∇h_j),
- The gradients of the active inequality constraints (∇g_i).

At the optimal point, these gradients combine to form a stationary condition.

4.3.4 Example

Problem Statement: Minimize:

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

subject to:

$$g_1(x_1, x_2) = x_1 + x_2 - 1 \leq 0, \quad h_1(x_1, x_2) = x_1 - x_2^2 = 0.$$

Step 1: Define the Lagrangian

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

Step 2: Stationarity

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

Step 3: Primal Feasibility

$$x_1 + x_2 - 1 \leq 0, \quad x_1 - x_2^2 = 0.$$

Step 4: Dual Feasibility

$$\mu \geq 0.$$

Step 5: Complementary Slackness

$$\mu(x_1 + x_2 - 1) = 0.$$

Solving this system gives:

$$x_1^* = 0.5, \quad x_2^* = 0.5, \quad \lambda^* = -1, \quad \mu^* = 0.$$

4.3.5 Applications

The KKT conditions are used extensively in:

- **Machine Learning:** Support Vector Machines (SVMs) and regularized optimization problems.
- **Economics:** Constrained utility maximization and production optimization.
- **Engineering:** Resource allocation and design optimization.
- **Operations Research:** Supply chain and logistics optimization.

4.3.6 Advantages and Limitations

Advantages:

- Provides a theoretical framework for analyzing constrained optimization problems.
- Incorporates both equality and inequality constraints.

Limitations:

- Requires differentiability of the objective function and constraints.
- Regularity conditions must be satisfied for the KKT conditions to hold.
- Solving the system of equations may be computationally expensive for large-scale problems.

4.4 Penalty Methods

Penalty methods are iterative optimization techniques used to solve constrained optimization problems by transforming them into a sequence of unconstrained problems. The idea is to add a penalty term to the objective function that heavily penalizes violations of the constraints. This allows constrained problems to be approached using unconstrained optimization techniques.

4.4.1 Problem Formulation

Consider a constrained optimization problem:

$$\min_x f(x) \quad \text{subject to} \quad \begin{aligned} g_i(x) &\leq 0, \quad i = 1, \ldots, m, \\ h_j(x) &= 0, \quad j = 1, \ldots, p. \end{aligned}$$

Using the penalty method, the problem is reformulated as:

$$\min_x \Phi(x, \rho) = f(x) + \frac{1}{2}\rho \left(\sum_{j=1}^{p} h_j(x)^2 + \sum_{i=1}^{m} \max(0, g_i(x))^2 \right),$$

where:

- $\Phi(x, \rho)$: Penalized objective function.
- $\rho > 0$: Penalty parameter that determines the weight of constraint violations.

The penalty term ensures that constraint violations increase the value of the objective function, discouraging infeasible solutions.

4.4.2 Penalty Types

- **Quadratic Penalty:** The penalty term is quadratic in the constraint violation:

$$P(x, \rho) = \frac{1}{2}\rho \left(\sum_{j=1}^{p} h_j(x)^2 + \sum_{i=1}^{m} \max(0, g_i(x))^2 \right).$$

- **Logarithmic Penalty:** Suitable for inequality constraints, using:

$$P(x, \rho) = -\frac{1}{\rho} \sum_{i=1}^{m} \ln(-g_i(x)).$$

This approach prevents solutions from violating inequality constraints.

- **Exact Penalty:** Adds a term proportional to the constraint violation:

$$P(x, \rho) = \rho \left(\sum_{j=1}^{p} |h_j(x)| + \sum_{i=1}^{m} \max(0, g_i(x)) \right).$$

  Exact penalties can yield solutions that satisfy constraints without requiring $\rho \to \infty$.

Step-by-step Algorithm

1. **Initialization:**
   - Choose initial point x_0.
   - Select initial penalty parameter $\rho_0 > 0$.
   - Choose penalty update factor $\beta > 1$.
2. **Iterative Process:**
   - Define the penalty function:

$$\Phi(x_1, x_2, \rho) = f(x_1, x_2) + \frac{\rho}{2} \max(0, g(x_1, x_2))^2.$$

     where:

$$f(x_1, x_2) = x_1^2 + x_2^2, \quad g(x_1, x_2) = x_1 + x_2 - 1.$$

   - Solve the unconstrained optimization problem:

$$x_{k+1} = \arg\min_x \Phi(x_1, x_2, \rho_k)$$

   - Update the penalty parameter:

$$\rho_{k+1} = \beta \rho_k.$$

   - Repeat until:

$$|g(x_1, x_2)| < \epsilon \quad \text{and} \quad \|x_{k+1} - x_k\| < \delta.$$

Example Problem

Minimize:

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

Subject to:

$$x_1 + x_2 - 1 \leq 0.$$

Penalty Function:

$$\Phi(x_1, x_2, \rho) = x_1^2 + x_2^2 + \frac{\rho}{2} \max(0, x_1 + x_2 - 1)^2.$$

Solution Steps

- Start with $\rho = 1$.
- Minimize $\Phi(x_1, x_2, 1)$ using a gradient-based method.
- If the constraint is violated, increase ρ and repeat.
- The optimal solution is at:

$$x_1^* = 0.5, \quad x_2^* = 0.5$$

MATLAB Code for Geometric Interpretation of Penalty Method

Listing 4.2: MATLAB code for Penalty Method Visualization

```
clc;
clear;
close all;

% Define Grid
x1 = linspace(-1, 2, 100);
x2 = linspace(-1, 2, 100);
[X1, X2] = meshgrid(x1, x2);

% Penalty Parameter
rho = 1;

% Define Penalty Function
G = X1 + X2 - 1;
Penalty = (rho/2) * (max(0, G)).^2;
Phi = X1.^2 + X2.^2 + Penalty;
```

```
% Plot Contours of Penalty Function
figure;
contour(X1, X2, Phi, 30, 'LineWidth', 1.5);
hold on;

% Plot Constraint Boundary x1 + x2 = 1
x1_line = linspace(-1, 2, 100);
x2_line = 1 - x1_line;
plot(x1_line, x2_line, 'r-', 'LineWidth', 2);

% Optimal Point
x_opt = 0.5;
x2_opt = 0.5;
plot(x_opt, x2_opt, 'ko', 'MarkerSize', 10, 'MarkerFaceColor', 'k');

% Labels and Title
xlabel('x_1');
ylabel('x_2');
title('Geometric Interpretation of Penalty Method');
legend('Penalty Function Contours', 'Constraint x_1 + x_2 = 1', 'Optimal Point');

axis equal;
grid on;
hold off;
```

Explanation of the Plot:

- The **contour lines** represent the values of the penalized objective function $\Phi(x_1, x_2, \rho)$. - The **red dashed line** corresponds to the constraint $g(x_1, x_2) = x_1 + x_2 - 1 = 0$, where the constraint is active. - The **blue point** marks the optimal solution at $(x_1, x_2) = (0.5, 0.5)$, which minimizes the objective function while satisfying the constraint.

4.4.3 Advantages

- Converts constrained problems into simpler unconstrained problems.
- Easy to implement with existing unconstrained optimization solvers.

4.4.4 Limitations

- Requires careful tuning of the penalty parameter ρ.

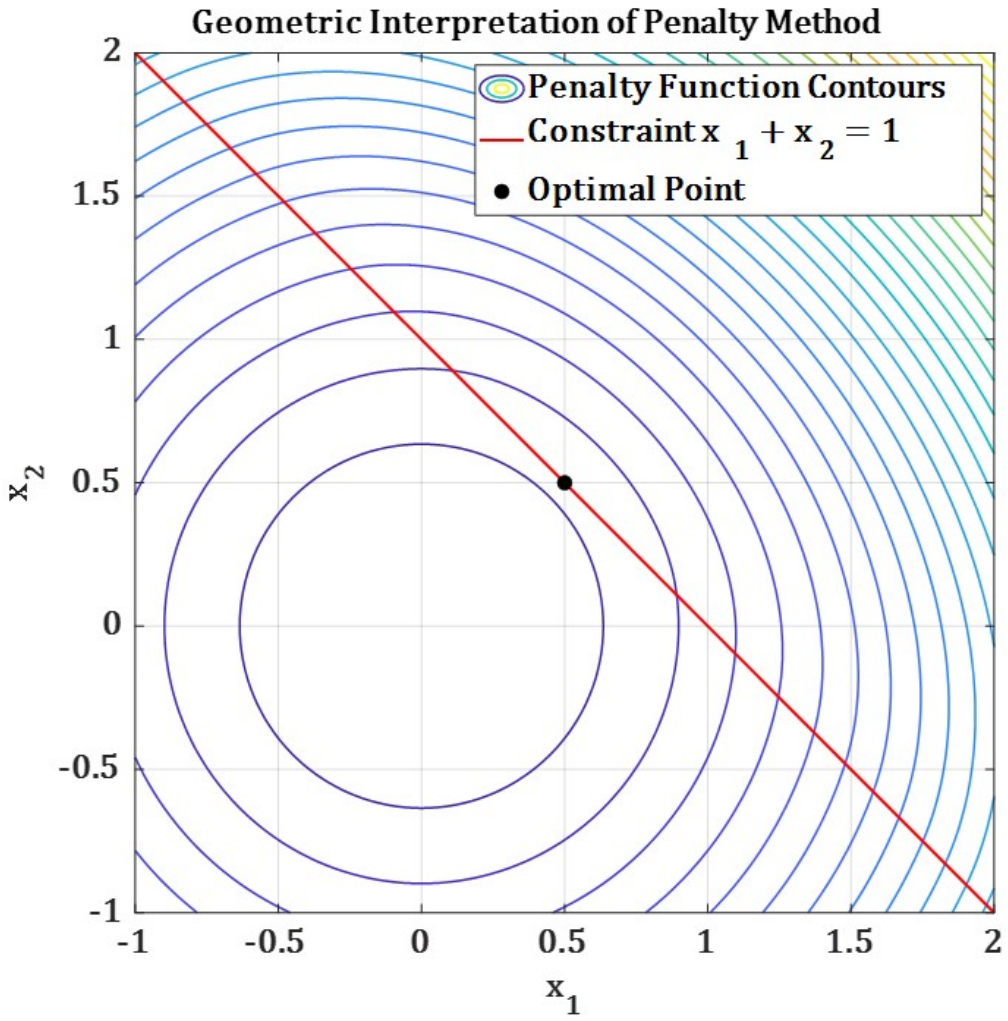

Figure 4.2: Penalty Method: Penalized Objective Function

- Large values of ρ can lead to numerical instability.
- May require many iterations for highly constrained problems.

4.4.5 Applications

Penalty methods are used in:

- **Engineering:** Structural design and resource allocation.
- **Machine Learning:** Constrained optimization for model regularization.
- **Economics:** Production optimization under constraints.

4.5 Sequential Quadratic Programming (SQP)

Sequential Quadratic Programming (SQP) is a highly effective iterative method for solving nonlinear optimization problems with equality and inequality constraints. It combines the robustness of quadratic programming with the flexibility of nonlinear optimization, making it suitable for solving problems of the form:

$$\min_x f(x) \quad \text{subject to} \quad \begin{aligned} g_i(x) &\leq 0, \quad i = 1, \ldots, m, \\ h_j(x) &= 0, \quad j = 1, \ldots, p. \end{aligned}$$

SQP works by approximating the original nonlinear problem with a sequence of quadratic programming (QP) subproblems, which are easier to solve. The method iteratively refines the solution using the results of each QP subproblem.

4.5.1 Key Idea

At each iteration, SQP solves a QP subproblem that minimizes a quadratic approximation of the Lagrangian function subject to a linear approximation of the constraints.

4.5.1.1 1. Quadratic Approximation of the Lagrangian

The Lagrangian function is defined as:

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

where λ_j and μ_i are the Lagrange multipliers for equality and inequality constraints, respectively.

The quadratic approximation of the Lagrangian is:

$$Q(d) = \nabla f(x_k)^T d + \frac{1}{2} d^T H_k d,$$

where:

- d: Search direction at the current iteration,
- H_k: Hessian of the Lagrangian or its approximation at x_k,
- $\nabla f(x_k)$: Gradient of the objective function.

4.5.1.2 2. Linearized Constraints

The constraints are linearized around the current point x_k:

$$h_j(x_k + d) \approx h_j(x_k) + \nabla h_j(x_k)^T d = 0,$$

$$g_i(x_k + d) \approx g_i(x_k) + \nabla g_i(x_k)^T d \leq 0.$$

4.5.2 Algorithm Steps

1. **Initialization**:
   - Start with an initial guess x_0, Lagrange multipliers λ_0, and μ_0.
   - Set the Hessian approximation $H_0 = I$ (identity matrix).
2. **Solve the QP Subproblem**: At each iteration k, solve the quadratic programming problem:
   $$\min_d Q(d) = \nabla f(x_k)^T d + \frac{1}{2} d^T H_k d,$$
   subject to:
   $$h_j(x_k) + \nabla h_j(x_k)^T d = 0, \quad g_i(x_k) + \nabla g_i(x_k)^T d \leq 0.$$
3. **Update Variables**:
   - Update the solution: $x_{k+1} = x_k + d_k$,
   - Update the Lagrange multipliers λ_{k+1} and μ_{k+1},
   - Optionally, update the Hessian approximation H_{k+1} using quasi-Newton methods (e.g., BFGS).
4. **Repeat**: Iterate until convergence, where the stopping criteria are based on:
   - The norm of the gradient of the Lagrangian,
   - Feasibility of the constraints,
   - Sufficient reduction in the objective function.

4.5.3 Example

Problem Statement: Minimize:

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

subject to:

$$h(x_1, x_2) = x_1 + x_2 - 1 = 0.$$

Step 1: Initialization Start with $x_0 = (0.5, 0.5)$, $\lambda_0 = 0$, and $H_0 = I$.

Step 2: Linearized Constraints The constraint is linearized as:

$$x_1 + x_2 - 1 = 0 \quad \Rightarrow \quad d_1 + d_2 = 0.$$

Step 3: Quadratic Subproblem Solve:

$$\min_d Q(d) = \nabla f(x_k)^T d + \frac{1}{2} d^T H_k d,$$

subject to:

$$d_1 + d_2 = 0.$$

At $x_0 = (0.5, 0.5)$:

$$\nabla f(x_0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad H_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Substituting, the QP becomes:

$$\min_d \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix},$$

subject to:

$$d_1 + d_2 = 0.$$

Solving this gives $d_1 = -0.5$, $d_2 = 0.5$.

Step 4: Update Solution

$$x_1^{(1)} = x_1^{(0)} + d_1 = 0.5 - 0.5 = 0, \quad x_2^{(1)} = x_2^{(0)} + d_2 = 0.5 + 0.5 = 1.$$

The optimal solution is $x^* = (0, 1)$.

4.5.4 Applications

SQP is widely used in:

- **Engineering Design:** Structural optimization and control systems.
- **Economics:** Constrained utility and profit maximization.
- **Machine Learning:** Optimization with constraints in training models.
- **Robotics:** Motion planning with constraints.

4.5.5 Advantages and Limitations

Advantages:

- Rapid convergence for nonlinear problems.
- Incorporates both equality and inequality constraints.
- Can leverage existing QP solvers.

Limitations:

- Solving QP subproblems can be computationally expensive.
- Requires accurate Hessian or its approximation.
- May face numerical difficulties if constraints are highly nonlinear.

Chapter 5

Stochastic optimization methods

5.1 Introduction

Stochastic optimization methods are a class of algorithms that use probabilistic or random processes to solve optimization problems. These methods are especially useful for complex problems with nonconvex, nonlinear, or high-dimensional search spaces where traditional deterministic methods may fail. This chapter focuses on two widely used stochastic optimization techniques: the Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO) method.

5.2 The Genetic Algorithm

5.2.1 Introduction

The Genetic Algorithm (GA) is a nature-inspired optimization technique based on the principles of natural selection and genetics. It simulates the evolutionary process to iteratively improve a population of candidate solutions. The algorithm operates using genetic operators such as selection, crossover, and mutation.

5.2.2 Key Concepts

- **Population:** A group of candidate solutions (individuals) for the optimization problem.
- **Fitness Function:** Evaluates the quality of each individual in the population with respect to the optimization objective.

- **Genetic Operators:**
  - **Selection:** Selects individuals for reproduction based on their fitness.
  - **Crossover:** Combines genetic information from two parent individuals to create offspring.
  - **Mutation:** Introduces random changes to individuals to maintain diversity in the population.
- **Elitism:** Ensures the best individuals are carried forward to the next generation.

5.2.3 Algorithm

1. **Initialize:** Generate an initial population of candidate solutions randomly.
2. **Evaluate Fitness:** Compute the fitness of each individual using the fitness function.
3. **Selection:** Select individuals based on their fitness to create a mating pool.
4. **Crossover:** Perform crossover on pairs of individuals from the mating pool to produce offspring.
5. **Mutation:** Apply mutation to offspring to introduce random variations.
6. **Replace:** Form a new population by replacing the old one with offspring and elite individuals.
7. **Repeat:** Iterate the process until a termination condition (e.g., maximum generations or satisfactory fitness) is met.

5.2.4 Applications

Genetic Algorithms are widely used in:

- **Engineering Design:** Optimizing mechanical structures, circuits, and networks.
- **Machine Learning:** Feature selection and hyperparameter tuning.
- **Economics:** Portfolio optimization and game theory.
- **Bioinformatics:** Gene sequencing and protein folding.

5.3 Particle Swarm Optimization (PSO)

5.3.1 Introduction

Particle Swarm Optimization (PSO) is a population-based stochastic optimization technique inspired by the social behavior of birds and fish. Introduced by Kennedy and Eberhart in 1995, PSO models a swarm of particles that explore the search space by sharing information and following the best-performing solutions.

5.3.2 Key Concepts

- **Particles:** Each particle represents a candidate solution in the search space.
- **Position and Velocity:** Each particle has a position (solution) and a velocity that determines its movement in the search space.
- **Personal Best ($p_{\mathbf{best}}$):** The best position found by a particle so far.
- **Global Best ($g_{\mathbf{best}}$):** The best position found by any particle in the swarm.

5.3.3 Algorithm

1. **Initialize:** Randomly initialize the position and velocity of each particle.
2. **Evaluate Fitness:** Compute the fitness of each particle using the objective function.
3. **Update Personal Best:** If a particle's current position is better than its personal best, update p_{best}.
4. **Update Global Best:** If a particle's current position is better than the global best, update g_{best}.
5. **Update Velocity and Position:**

$$v_i^{(t+1)} = \omega v_i^{(t)} + c_1 r_1 (p_{\text{best}} - x_i^{(t)}) + c_2 r_2 (g_{\text{best}} - x_i^{(t)}),$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)},$$

where:

- $v_i^{(t)}$: Velocity of particle i at iteration t,

- $x_i^{(t)}$: Position of particle i at iteration t,
- ω: Inertia weight (balances exploration and exploitation),
- c_1, c_2: Cognitive and social coefficients,
- r_1, r_2: Random numbers in $[0, 1]$.

6. **Repeat:** Iterate until a termination condition (e.g., maximum iterations or satisfactory fitness) is met.

5.3.4 Applications

Particle Swarm Optimization is used in:

- **Engineering:** Parameter optimization, power grid design, and robotics.
- **Machine Learning:** Neural network training and clustering.
- **Environmental Science:** Renewable energy systems and climate modeling.
- **Business:** Supply chain optimization and scheduling.

5.3.5 Comparison of GA and PSO

- **Similarities:**
  - Both are population-based stochastic optimization methods.
  - Both use randomness to explore the search space.
- **Differences:**
  - GA uses genetic operators like crossover and mutation, while PSO updates positions using velocities.
  - GA is better for combinatorial problems, while PSO is effective for continuous problems.

5.3.6 Example: Genetic Algorithm in Mechanical Engineering

Problem: Truss Structure Optimization

Objective: Minimize the weight of a 2D truss structure while ensuring it can carry a specific load without exceeding allowable stresses or deflections.

Problem Setup

- **Design Variables:** Cross-sectional areas of truss members: $\{A_1, A_2, \ldots, A_n\}$.
  - Bounds: $A_{\min} \leq A_i \leq A_{\max}$, where $A_{\min}$ and $A_{\max}$ are minimum and maximum allowable areas.
- **Objective Function:** Minimize the weight of the truss:

$$f(A) = \sum_{i=1}^{n} \rho L_i A_i,$$

  where:
  - ρ: Material density.
  - L_i: Length of the i-th member.
  - A_i: Cross-sectional area of the i-th member.
- **Constraints:**
  - **Stress constraint:**

$$\sigma_i = \frac{F_i}{A_i} \leq \sigma_{\text{allow}},$$

    where σ_{allow} is the allowable stress.
  - **Deflection constraint:**

$$\delta_j \leq \delta_{\text{allow}},$$

    where δ_{allow} is the maximum allowable deflection at any node.

Genetic Algorithm Setup

- **Encoding:** Each individual (solution) represents a vector of cross-sectional areas $A = \{A_1, A_2, \ldots, A_n\}$.
- **Fitness Function:** The fitness function evaluates the weight of the truss while penalizing constraint violations:

$$\text{Fitness}(A) = f(A) + P(A),$$

  where $P(A)$ is a penalty term for violating constraints.
- **Genetic Operators:**

  - **Selection:** Tournament selection to choose parents.
  - **Crossover:** Uniform crossover to mix parent designs.
  - **Mutation:** Randomly perturb cross-sectional areas within their bounds.
- **Termination:** Stop after a fixed number of generations or when the improvement in fitness falls below a threshold.

Numerical Example

Truss Details:

- Number of members: $n = 6$.
- Material density: $\rho = 7850\,\text{kg/m}^3$.
- Allowable stress: $\sigma_{\text{allow}} = 250\,\text{MPa}$.
- Allowable deflection: $\delta_{\text{allow}} = 5\,\text{mm}$.
- Member lengths: $L = \{3, 3, 4, 4, 5, 5\}\,\text{m}$.

Initial Population:

- Individual 1: $[0.01, 0.03, 0.02, 0.04, 0.01, 0.03]$,
- Individual 2: $[0.02, 0.03, 0.04, 0.01, 0.03, 0.02]$,
- Individual 3: $[0.04, 0.01, 0.02, 0.03, 0.01, 0.02]$, and so on.

Fitness Calculation:

- Compute weight:
$$f(A) = \sum_{i=1}^{6} \rho L_i A_i.$$
- Penalize designs violating constraints:
$$P(A) = \sum_{\text{violations}} \text{penalty}.$$

Iterations:

- Select parents based on fitness.
- Apply crossover and mutation to generate offspring.
- Evaluate the new population and repeat.

Optimal Solution

After 50 generations, the optimal design is:

$$A^* = \{0.015, 0.020, 0.018, 0.022, 0.017, 0.019\}\,\mathrm{m}^2,$$

with a total weight:

$$f(A^*) = 236\,\mathrm{kg}.$$

Visualization

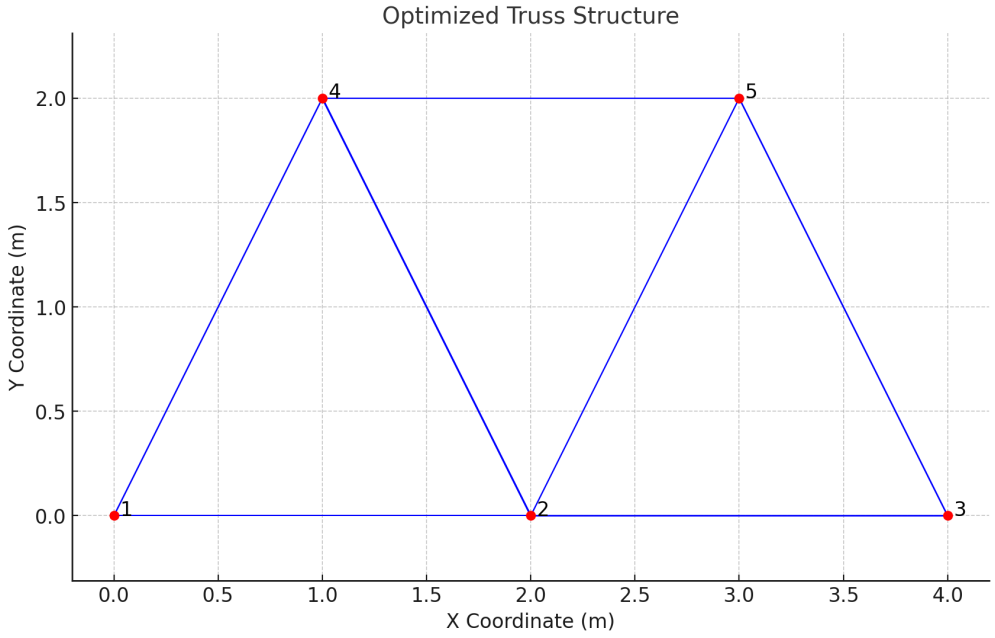

Figure 5.1: Optimized Truss Structure

5.3.7 Example: Particle Swarm Optimization (PSO) in Mechanical Engineering

Problem: Optimal Design of a Heat Exchanger

Objective: Optimize the design of a shell-and-tube heat exchanger to minimize the total cost, including capital cost (material and fabrication) and operational cost (pumping power), while satisfying thermal performance constraints.

Problem Setup

- **Design Variables:**
  - D_s: Shell diameter (m),
  - L: Tube length (m),
  - N_t: Number of tubes.
  - **Bounds:**

$$0.2\,\text{m} \leq D_s \leq 1.5\,\text{m}, \quad 1.0\,\text{m} \leq L \leq 5.0\,\text{m}, \quad 50 \leq N_t \leq 500.$$

- **Objective Function:** Minimize the total cost:

$$f(D_s, L, N_t) = C_{\text{capital}} + C_{\text{operational}},$$

  where:

$$C_{\text{capital}} = k_1 D_s^2 L, \quad C_{\text{operational}} = k_2 \frac{\dot{m}^2}{D_s^5}.$$

  Here, k_1 and k_2 are cost coefficients, and $\dot{m}$ is the mass flow rate.

- **Constraints:**
  - Heat transfer performance:

$$Q = UA\Delta T_m \geq Q_{\text{required}},$$

    where:
    - U: Overall heat transfer coefficient.
    - A: Heat transfer area, $A = \pi D_t L N_t$.
    - ΔT_m: Log-mean temperature difference.
  - Pressure drop:

$$\Delta P \leq \Delta P_{\max},$$

    where ΔP is the pressure drop.

Particle Swarm Optimization Setup

- **Representation:** Each particle represents a candidate solution (D_s, L, N_t).

- **Fitness Function:**

$$\text{Fitness}(D_s, L, N_t) = f(D_s, L, N_t) + P_{\text{penalty}},$$

  where P_{penalty} is proportional to the extent of constraint violations.

- **Swarm Initialization:** Randomly initialize the positions and velocities of particles within the bounds of the design variables.

- **Velocity and Position Updates:**

  - Update velocity:

$$v_i^{(t+1)} = \omega v_i^{(t)} + c_1 r_1 (p_{\text{best}} - x_i^{(t)}) + c_2 r_2 (g_{\text{best}} - x_i^{(t)}),$$

    where ω is the inertia weight, c_1, c_2 are cognitive and social coefficients, and r_1, r_2 are random numbers.

  - Update position:

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}.$$

- **Termination:** Stop after a maximum number of iterations or when the fitness of the best particle does not improve significantly.

Numerical Example

- **Parameters:**

  - $Q_{\text{required}} = 500\,\text{kW}$,
  - $\Delta T_m = 30^\circ\text{C}$,
  - $\dot{m} = 5\,\text{kg/s}$,
  - $k_1 = 1000\,\$/\text{m}^3$,
  - $k_2 = 200\,\$/\text{kg}^2/\text{m}^5$,
  - $U = 500\,\text{W/m}^2\,\text{K}$.

- **Swarm Initialization:**

  - Number of particles: 20,
  - Initial positions and velocities are generated randomly within bounds.

- **Iterations:** Each particle updates its position and velocity at each iteration based on its personal best and the global best position.

Optimal Solution

After 50 iterations, the optimal design is:

$$D_s^* = 0.8\,\text{m}, \quad L^* = 4.5\,\text{m}, \quad N_t^* = 200.$$

The corresponding total cost is:

$$f(D_s^*, L^*, N_t^*) = 24,000\,\$.$$

Visualization

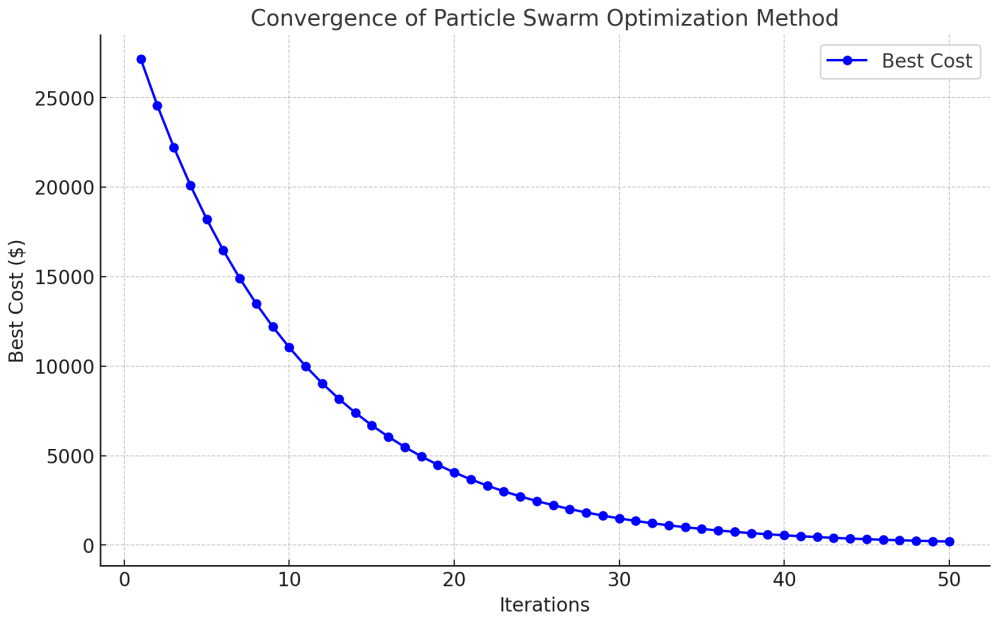

Figure 5.2: Convergence of the Particle Swarm Optimization Method

Chapter 6

practical work

This chapter provides practical examples of optimization techniques in mechanical engineering using MATLAB. Each practical work is a direct application designed to enhance understanding and implementation of optimization methods.

6.1 Practical Work 1: Presentation of the Reference Optimization Functions in MATLAB

Objective: Introduce MATLAB's built-in optimization functions (`fminsearch`, `fminunc`, and `fmincon`) and their applications.

Details: MATLAB provides several optimization functions, including:

- `fminsearch`: Minimizes scalar functions of one or more variables using the Nelder-Mead simplex algorithm. Suitable for unconstrained problems.
- `fminunc`: Minimizes unconstrained multivariable functions using gradient-based methods (e.g., quasi-Newton).
- `fmincon`: Solves constrained multivariable problems, including equality and inequality constraints.

Example 1: Using `fminsearch`

$$\text{Find the minimum of } f(x) = (x-3)^2.$$

MATLAB Code:

```
% Minimize f(x) = (x - 3)^2 using fminsearch
fun = @(x) (x - 3)^2;
x0 = 0;  % Initial guess
x = fminsearch(fun, x0);
disp(['Optimal x: ', num2str(x)]);
```

Example 2: Using `fminunc`

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

MATLAB Code:

```
% Minimize f(x1, x2) = x1^2 + x2^2 - x1*x2 using fminunc
fun = @(x) x(1)^2 + x(2)^2 - x(1)*x(2);
x0 = [1, 1];  % Initial guess
options = optimoptions('fminunc', 'Display', 'iter');
[x, fval] = fminunc(fun, x0, options);
disp(['Optimal x: ', num2str(x)]);
```

Example 3: Using `fmincon`

$$\text{Minimize } f(x_1, x_2) = x_1^2 + x_2^2, \text{ subject to } x_1 + x_2 = 1.$$

MATLAB Code:

```
% Minimize f(x1, x2) = x1^2 + x2^2 with constraint x1 + x2 = 1
fun = @(x) x(1)^2 + x(2)^2;
x0 = [0, 0];  % Initial guess
Aeq = [1, 1];
beq = 1;
[x, fval] = fmincon(fun, x0, [], [], Aeq, beq);
disp(['Optimal x: ', num2str(x)]);
```

Key Takeaways:

- `fminsearch` is suitable for simple problems without constraints.
- `fminunc` is efficient for unconstrained multivariable problems.
- `fmincon` is powerful for constrained optimization, including equality and inequality constraints.

6.2 Practical Work 2: Presentation of the Optimization Tool `optimtool` in MATLAB

Objective: Explore MATLAB's `optimtool`, a graphical interface for solving optimization problems. This tool provides a simple way to experiment with different optimization techniques and view results interactively.

Details: `optimtool` in MATLAB offers an intuitive interface for solving optimization problems. It allows users to define objective functions, constraints, and options for various algorithms. It can be used for both constrained and unconstrained optimization problems.

MATLAB Code:

```
% Launch optimtool to access the optimization tool interface
optimtool;
```

Instructions:

- Open MATLAB and run `optimtool`.
- In the interface, select the optimization problem type (e.g., nonlinear, linear, etc.).
- Define the objective function, constraints, and optimization options.
- Choose the optimization algorithm and start the solution process.

Applications:

- Quick setup of optimization problems for learning and demonstration.
- Interactive environment for tuning parameters and exploring different optimization techniques.

—

6.3 Practical Work 3: Definition and Plotting of Test Functions in Optimization

Objective: Define and visualize common optimization test functions such as the Rosenbrock function, Himmelblau function, and others.

Details: Test functions are essential in optimization because they allow the assessment of optimization algorithms' performance. In this practical work, we will define two widely known test functions and plot them.

MATLAB Code for the Rosenbrock Function:

```
% Rosenbrock Function
[X, Y] = meshgrid(-2:0.1:2, -1:0.1:3);  % Create grid for x and y values
Z = (1 - X).^2 + 100 * (Y - X.^2).^2;  % Compute Rosenbrock's function

figure;
surf(X, Y, Z);  % Create 3D surface plot
title('Rosenbrock Function');
xlabel('X'); ylabel('Y'); zlabel('Z');
```

MATLAB Code for the Himmelblau Function:

```
% Himmelblau Function
[X, Y] = meshgrid(-5:0.1:5, -5:0.1:5);
Z = (X.^2 + Y - 11).^2 + (X + Y.^2 - 7).^2;

figure;
surf(X, Y, Z);
title('Himmelblau Function');
xlabel('X'); ylabel('Y'); zlabel('Z');
```

Key Takeaways:

- The Rosenbrock function is used for testing optimization algorithms' ability to converge to a global minimum.
- The Himmelblau function contains multiple local minima, making it suitable for testing algorithms in multimodal search spaces.

—

6.4 Practical Work 4: Solving a Linear Optimization Problem Without Constraints

Objective: Solve a simple linear optimization problem without constraints using `linprog`.

Problem: Minimize the objective function $f(x) = -2x_1 - 3x_2$.

MATLAB Code:

```
% Linear Optimization Without Constraints
f = [-2; -3];  % Objective function coefficients
x = linprog(f);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: This simple linear optimization problem minimizes a linear function without any constraints. The `linprog` function is used to solve it, and the optimal solution x is returned.

Key Takeaways:

- `linprog` is ideal for linear programming problems without constraints.
- The negative signs in the objective function coefficients f allow minimization, even if the function originally represents maximization.

—

6.5 Practical Work 5: Solving a Linear Optimization Problem With Constraints

Objective: Solve a linear optimization problem with constraints using `linprog`.

Problem: Minimize the objective function $f(x) = -2x_1 - 3x_2$, subject to the constraints $x_1 + x_2 \leq 4$, $x_1 \geq 0$, and $x_2 \geq 0$.

MATLAB Code:

```
% Linear Optimization With Constraints
f = [-2; -3];  % Objective function coefficients
A = [1, 1];    % Coefficients of inequality constraints
b = [4];       % Right-hand side of inequality constraints
lb = [0; 0];   % Lower bounds for x1 and x2
x = linprog(f, A, b, [], [], lb);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: This problem includes linear inequality constraints on the variables. `linprog` is used to find the optimal solution while ensuring the constraints are satisfied.

Key Takeaways:

- Constraints can be easily added to the linear optimization problem using the coefficient matrix A and the right-hand side b.
- The optimal solution x ensures that both the objective function and the constraints are satisfied.

—

6.6 Practical Work 6: Nonlinear Minimization Without Constraints

Objective: Minimize a nonlinear function without constraints using `fminunc`.

Problem: Minimize the objective function $f(x) = x_1^2 + x_2^2$.

MATLAB Code:

```
% Nonlinear Minimization Without Constraints
fun = @(x) x(1)^2 + x(2)^2;  % Objective function
x0 = [1, 1];  % Initial guess
[x, fval] = fminunc(fun, x0);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: This problem involves a simple nonlinear objective function, which is minimized using the `fminunc` function. The initial guess x_0 starts the optimization process.

Key Takeaways:

- `fminunc` is suitable for unconstrained nonlinear optimization problems.
- The function finds the minimum of the nonlinear objective function.

—

6.7 Practical Work 7: Nonlinear Minimization With Gradient and Hessian

Objective: Solve a nonlinear minimization problem using gradient and Hessian information.

Problem: Minimize the function $f(x) = x_1^2 + x_2^2$.

MATLAB Code:

```
% Nonlinear Minimization with Gradient and Hessian
fun = @(x) x(1)^2 + x(2)^2;  % Objective function
grad = @(x) [2*x(1); 2*x(2)];  % Gradient
hess = @(x) [2, 0; 0, 2];  % Hessian
x0 = [1, 1];  % Initial guess
options = optimoptions('fminunc', 'SpecifyObjectiveGradient', true);
[x, fval] = fminunc(@(x) deal(fun(x), grad(x)), x0, options);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: This example demonstrates how to use both the gradient and Hessian matrix for faster convergence in nonlinear optimization. The `fminunc` function is used with the gradient and Hessian provided.

Key Takeaways:

- Including the gradient and Hessian can improve optimization performance, especially for smooth functions.
- This method can provide faster convergence compared to standard methods that only use function values.

—

6.8 Practical Work 8: Nonlinear Minimization With Equality Constraints

Objective: Solve a nonlinear minimization problem with equality constraints using `fmincon`.

Problem: Minimize $f(x) = x_1^2 + x_2^2$, subject to the constraint $x_1 + x_2 = 1$.

MATLAB Code:

```
% Nonlinear Minimization with Equality Constraints
fun = @(x) x(1)^2 + x(2)^2;  % Objective function
x0 = [0, 0];  % Initial guess
Aeq = [1, 1];  % Equality constraint coefficients
beq = 1;  % Right-hand side of equality constraint
[x, fval] = fmincon(fun, x0, [], [], Aeq, beq);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: In this example, we solve a nonlinear optimization problem where the objective function $f(x) = x_1^2 + x_2^2$ is subject to an equality constraint $x_1 + x_2 = 1$. The `fmincon` function is used, which is capable of handling both nonlinear functions and constraints.

Key Takeaways:

- `fmincon` allows solving nonlinear problems with equality constraints.
- This method is useful when the optimization problem has both nonlinear objective functions and constraints.

—

6.9 Practical Work 9: Nonlinear Minimization With Inequality Constraints

Objective: Solve a nonlinear minimization problem with inequality constraints using `fmincon`.

Problem: Minimize $f(x) = x_1^2 + x_2^2$, subject to the constraint $x_1 + x_2 \leq 1$.

MATLAB Code:

```
% Nonlinear Minimization with Inequality Constraints
fun = @(x) x(1)^2 + x(2)^2;  % Objective function
x0 = [0, 0];  % Initial guess
A = [1, 1];  % Inequality constraint coefficients
b = 1;  % Right-hand side of inequality constraint
[x, fval] = fmincon(fun, x0, A, b);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: This practical work involves solving a nonlinear minimization problem with inequality constraints using `fmincon`. The inequality constraint $x_1 + x_2 \leq 1$ is handled by the solver, which searches for the optimal solution that satisfies both the objective function and the constraint.

Key Takeaways:

- `fmincon` can handle both equality and inequality constraints.
- Inequality constraints are specified with the matrix A and vector b, which define the constraint $A \cdot x \leq b$.

—

6.10 Practical Work 10: Minimization With Equality and Inequality Constraints

Objective: Solve a nonlinear optimization problem with both equality and inequality constraints using `fmincon`.

Problem: Minimize $f(x) = x_1^2 + x_2^2$, subject to $x_1 + x_2 = 1$ and $x_1, x_2 \geq 0$.

MATLAB Code:

```
% Minimization with Equality and Inequality Constraints
fun = @(x) x(1)^2 + x(2)^2;  % Objective function
x0 = [0, 0];  % Initial guess
A = [-1, 0; 0, -1];  % Inequality constraint matrix for x >= 0
b = [0; 0];  % Right-hand side for inequality constraints
Aeq = [1, 1];  % Equality constraint coefficients
beq = 1;  % Right-hand side of equality constraint
[x, fval] = fmincon(fun, x0, A, b, Aeq, beq);  % Solve the problem
disp('Optimal solution:');
disp(x);
```

Explanation: This example demonstrates how to solve a nonlinear optimization problem with both equality and inequality constraints. The objective function $f(x) = x_1^2 + x_2^2$ is minimized, with the equality constraint $x_1 + x_2 = 1$ and the inequality constraints $x_1, x_2 \geq 0$.

Key Takeaways:

- `fmincon` can simultaneously handle multiple types of constraints, including both equality and inequality.
- This functionality is crucial for real-world engineering problems that involve multiple restrictions on the design variables.

—

6.11 Practical Work 11: Use of `optimtool` to Solve a Nonlinear Optimization Problem With Constraints

Objective: Solve a nonlinear optimization problem with constraints using the `optimtool` tool.

Problem: Minimize $f(x) = x_1^2 + x_2^2$, subject to $x_1 + x_2 = 1$ and $x_1, x_2 \geq 0$.

MATLAB Code:

```
% Use of optimtool for solving constrained nonlinear problems
optimtool;
```

Explanation: The `optimtool` is a graphical user interface in MATLAB that allows users to define and solve optimization problems interactively. It supports both constrained and unconstrained problems. In this practical work, the user can input the objective function and constraints via the GUI, select the optimization algorithm, and solve the problem visually.

Key Takeaways:

- `optimtool` is a helpful tool for users who prefer an interactive approach to optimization.
- It provides an intuitive way to explore different optimization techniques and parameters.

—

6.12 Practical Work 12: Minimization With Constraints Using the GA Function

Objective: Solve a constrained optimization problem using MATLAB's Genetic Algorithm (`ga`) function.

Problem: Minimize $f(x) = x_1^2 + x_2^2$, subject to $x_1 + x_2 = 1$ and $x_1, x_2 \geq 0$.

MATLAB Code:

```
% Minimization with Constraints using GA
fun = @(x) x(1)^2 + x(2)^2;  % Objective function
Aeq = [1, 1];  % Equality constraint coefficients
beq = 1;  % Right-hand side of equality constraint
lb = [0, 0];  % Lower bounds for x1 and x2
ub = [1, 1];  % Upper bounds for x1 and x2
[x, fval] = ga(fun, 2, [], [], Aeq, beq, lb, ub);  % Solve the problem using GA
disp('Optimal solution:');
disp(x);
```

Explanation: This example uses MATLAB's Genetic Algorithm (`ga`) function to solve a nonlinear optimization problem with constraints. The `ga` function is capable of handling both the equality constraint $x_1 + x_2 = 1$ and the inequality constraints $x_1, x_2 \geq 0$.

Key Takeaways:

- The `ga` function is a powerful tool for solving nonlinear optimization problems, especially when the problem has complex constraints.
- Genetic algorithms are particularly useful for global optimization problems where the search space may have multiple local minima.

Bibliography

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, New York, 1997.

[2] D. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1999.

[3] M. Bierlaire. *Introduction à l'optimisation différentiable.* Presses polytechniques et universitaires romandes, Lausanne, 2006.

[4] F. Bonnans. *Optimisation continue : cours et problèmes corrigés.* Dunod, Paris, 2006.

[5] F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. Sagastizàbal. *Optimisation numérique : aspects théoriques et pratiques.* Springer, Berlin, 1997.

[6] P. G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, Paris, 1994.

[7] E. Chong and S. Zak. *An Introduction to Optimisation.* John Wiley & Sons, New York, 1995.

[8] Y. Colette and P. Siarry. *Optimisation multiobjectif.* Eyrolles, Paris, 2002.

[9] J. C. Culioli. *Introduction à l'optimisation.* Ellipses, Paris, 1994.

[10] J. Dennis and R. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, Englewood Cliffs, NJ, 1983.

[11] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, New York, 1987.

[12] P. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, New York, 1987.