الجمهوريــة الجزائريــة الديمقراطيــة الشعبيــة
The Peoples's Democratic Republic of Algeria
و البحـث الـعلـمـي وزارة الـتعلـيـم الـعـالي
Ministry of Higher Education And Scientific Research

University of MUSTAPHA Stambouli
Mascara

جامعة مصطفى اسطمبولي
معسكر

Faculty of Economics, Business and Management Sciences

Department of Economics

Pedagogical Handbook

Module:

Optimization Methods

Intended for First-Year Master's Students – Specialization in Quantitative Economics.

Prepared by:

Dr. GUELLIL Mohammed Seghir

Academic Year: 2024/2025

Contents

General Introduction

Optimization is a critical field in applied mathematics, engineering, economics, and computer science, providing methods and techniques for finding the best possible solution to problems under specific constraints. The goal of optimization is to either maximize or minimize an objective function, which is a quantitative measure of success, such as profit, efficiency, or cost. These problems appear in virtually every sector, ranging from designing systems that maximize energy efficiency to developing algorithms in machine learning that optimize predictive models.

The concept of optimization has a long and rich history, stretching back thousands of years. Early examples of optimization can be found in ancient Greek mathematics, where optimization was used in the study of geometry, such as finding the shortest distance between two points or maximizing the volume of geometric solids. However, it was only in the 17th and 18th centuries, with the development of calculus, that optimization began to take shape as a formal mathematical discipline. The pivotal role of calculus, particularly the work of mathematicians like **Isaac Newton** (1643–1727) and **Gottfried Wilhelm Leibniz** (1646–1716), was essential for formulating optimization problems in a more structured way.

Pierre de Fermat (1607–1665), a French mathematician, is often credited with laying the foundation of optimization. Fermat's principle of least time, which states that light follows the path that requires the least time to travel between two points, was an early form of optimization applied to physics. His work paved the way for later advancements in variational calculus, which would become a cornerstone of optimization theory.

The 19th century witnessed the formalization of optimization methods. **Augustin-Louis Cauchy** (1789–1857), a French mathematician, made significant contributions to optimization theory, particularly with his development of the **method of steepest descent** in 1847. This iterative technique, which adjusts variables in the direction of the steepest gradient, remains a cornerstone of optimization methods used to solve unconstrained optimization problems. Cauchy's work was instrumental in establishing optimization as a distinct subfield of mathematics.

As the 20th century dawned, the development of optimization methods accelerated, with the advent of **linear programming** and the introduction of new techniques for solving both linear and nonlinear optimization problems. In 1947, **George Dantzig** (1914–2005), an American mathematician, revolutionized optimization with the invention of the **Simplex Method**, which is still widely used for solving linear optimization problems. Dantzig's work was a milestone in optimization, providing an efficient method for finding the optimal

solution in linear programming, and making optimization a critical tool in operations research, economics, and logistics.

During the mid-20th century, optimization continued to evolve with the introduction of **dynamic programming** by **Richard Bellman** (1920–1984) in the 1950s. Bellman's method provided a systematic approach for solving problems that could be broken down into simpler subproblems, with applications ranging from resource allocation to control systems. This approach has since become a fundamental concept in optimization, especially in fields like computer science and artificial intelligence.

The latter half of the 20th century saw further advancements in the theory and practice of nonlinear optimization. **John von Neumann** (1903–1957), a Hungarian-American mathematician, made important contributions to game theory and the application of optimization in strategic decision-making, while **Ralph E. Gomory** (1929–2022) made pioneering advances in integer programming. Their work helped to establish optimization as a crucial tool in economics, operations research, and decision theory.

The rise of **convex optimization** in the late 20th and early 21st centuries further expanded the scope and applicability of optimization methods. **Stephen Boyd** (born 1953) and **Lieven Vandenberghe** (born 1964), both professors at Stanford University, are key figures in the development of modern convex optimization. Their seminal textbook, *Convex Optimization* (2004), has become the standard reference for practitioners in engineering, economics, and applied mathematics. Their work on convex optimization theory, including the characterization of convex functions and optimality conditions, has made it possible to solve large-scale optimization problems more efficiently.

Optimization is not limited to theoretical developments but has also led to a wide range of practical applications. For example, in machine learning, optimization techniques such as **gradient descent** are used to train models by adjusting parameters to minimize an error function. **Deep learning**, a subfield of machine learning, relies heavily on optimization methods to adjust millions of parameters in neural networks, a process that would be impossible without efficient optimization algorithms.

In engineering, optimization is central to fields such as structural design, control theory, and signal processing. The ability to optimize designs, whether it is minimizing material use or maximizing the efficiency of control systems, has direct implications for innovation and technological progress. In finance, optimization is employed in portfolio management, where the objective is to maximize return while minimizing risk.

Optimization problems can be broadly categorized into two types: **unconstrained optimization**, where there are no restrictions on the decision variables, and **constrained optimization**, where the variables must satisfy specific limitations. The development of algorithms to efficiently solve both types of problems has been a major area of research. Some of the most well-known algorithms include **Newton's method**, **the Nelder-Mead algorithm**, **the Simplex method**, and **gradient-based techniques** such as **steepest descent**.

This handout is dedicated to students in both **licence** and **master's programs**, with the aim of providing them with a deep and thorough understanding of optimization methods. The content is designed to take students from the foundational concepts through to more advanced topics in optimization, equipping them with the tools needed to approach real-world problems.

The handout is divided into five key chapters to provide a comprehensive understanding of optimization techniques. The structure is as follows:

1. **General Concept**: This chapter introduces the fundamental concepts of optimization, including a historical review, the structure of an optimization problem, and methods for solving them. It also covers the critical topics of convexity, gradient vectors, directional derivatives, and the Hessian matrix.
2. **Convex Optimization Overview**: This chapter provides an in-depth look at convex optimization, including the theory of convex functions and the conditions under which optimization problems become easier to solve. It covers first and second-order characterizations of convex functions, as well as the implications of strict convexity for the uniqueness of optimal solutions.
3. **1-D Optimization Algorithms**: Focused on optimization problems involving a single variable, this chapter explores algorithms like the **bisection method**, **Newton-Raphson method**, and **golden section method**. These techniques are critical for solving simpler problems and serve as building blocks for more complex methods.
4. **Unconstrained Optimization**: This chapter discusses techniques for solving optimization problems without constraints, such as the **steepest descent method**, **Newton's method**, and **Nelder-Mead algorithm**. It also introduces additional test functions used to evaluate optimization methods.
5. **Constrained Optimization**: The final chapter explores optimization problems that involve constraints. It includes methods like the **penalty function method**, **Lagrange multipliers**, and **sequential quadratic programming**, and provides real-world applications in areas like structural design.

Each chapter is designed to provide a blend of theoretical foundations and practical solutions, with accompanying highlights, formulae charts, and problems to enhance understanding. By the end of this handout, students will have a solid grasp of optimization methods and be equipped to apply these techniques to solve complex real-world problems.

Chapter I : General Concepts

1.1 Introduction :

Optimization is the process of making something as effective or functional as possible. In mathematical terms, it involves finding the maximum or minimum value of a function subject to certain constraints. Optimization problems arise in almost every field, including engineering, economics, logistics, machine learning, and more. The goal of this course is to provide a solid foundation in optimization theory, methods, and applications.

1.2 Historical Review:

The field of optimization has a rich history that spans centuries, evolving from simple geometric solutions to sophisticated algorithms powered by modern computing. Understanding this history provides context for the development of optimization techniques and their applications today. Below is a detailed timeline of key developments:

Ancient Beginnings

The roots of optimization can be traced back to ancient civilizations, where early mathematicians and philosophers sought to solve practical problems. For example:

- **Ancient Greece (300 BC)**: Mathematicians like **Euclid** and **Archimedes** explored geometric optimization problems, such as finding the shortest path between two points (a precursor to the modern concept of **geodesics**).
- **Heron of Alexandria** (10–70 AD): Known for **Heron's Principle**, which states that light takes the shortest path between two points, an early example of optimization in nature.

17th Century: The Birth of Calculus

The development of calculus by **Isaac Newton** and **Gottfried Wilhelm Leibniz** in the 17th century provided the mathematical tools necessary for solving optimization problems. Key contributions include:

- **Newton's Method**: Originally developed for finding roots of equations, this iterative method became a cornerstone for solving optimization problems numerically.
- **Fermat's Principle**: Pierre de Fermat's work on finding maxima and minima using derivatives laid the groundwork for modern optimization techniques.

18th Century: Lagrange and Constrained Optimization

The 18th century saw significant advancements in optimization, particularly with the work of **Joseph-Louis Lagrange**:

- **Lagrange Multipliers**: Lagrange introduced a method for solving constrained optimization problems, which remains a fundamental tool in economics, physics, and engineering. For example, in economics, it is used to maximize utility subject to a budget constraint.

19th Century: Cauchy and Gradient Descent

In the 19th century, **Augustin-Louis Cauchy** made groundbreaking contributions to optimization:

- **Gradient Descent**: Cauchy developed the gradient descent method, an iterative algorithm for finding the minimum of a function. This method is now widely used in machine learning and numerical optimization.

20th Century: Linear Programming and the Simplex Method

The 20th century marked the rise of **linear programming** and the development of efficient algorithms for solving large-scale optimization problems:

- **George Dantzig**: In 1947, Dantzig introduced the **Simplex Method**, a powerful algorithm for solving linear programming problems. This method revolutionized fields like operations research, logistics, and economics.
- **John von Neumann**: His work on duality theory provided a deeper understanding of linear programming and its applications.

21st Century: Modern Optimization

The 21st century has seen the rise of advanced optimization techniques driven by the need to solve complex problems in machine learning, artificial intelligence, and data science:

- **Convex Optimization**: The development of efficient algorithms for convex optimization has enabled the solution of problems with millions of variables.
- **Stochastic Optimization**: Techniques like stochastic gradient descent are widely used in training deep learning models.
- **Metaheuristics**: Algorithms like genetic algorithms and simulated annealing are used to solve non-convex and combinatorial optimization problems.

1.3 Optimization Problem:

An optimization problem involves finding the best solution from a set of feasible alternatives. It can be formally defined as follows:[1]

$$\begin{gathered} Minimize\ f(x) \quad or \quad Maximize\ f(x) \\ Subject\ to: g_i(x) \leq 0, \quad i = 1,2, \dots, m \\ h_j(x) = 0, \quad j = 1,2, \dots, p \end{gathered}$$

Where:

$* f(x): The\ \boldsymbol{objective\ function}\ to\ be\ minimized\ or\ maximized.$
$* x: The\ vector\ of\ \boldsymbol{decision\ variables}.$
$* g_i(x): Inequality\ constraints.$
$* h_j(x): Equality\ constraints.$

[1] Nocedal, J., & Wright, S. J. (2006). *Numerical optimization*. Springer Series in Operations Research. New York: Springer Science+Business Media.

Types of Optimization Problems

1. **Linear Programming (LP)**: The objective function and constraints are linear. Example:

$$Maximize\ z = c_1x_1 + c_2x_2$$
$$Subject\ to: a_{11}x_1 + a_{12}x_2 \leq b_1$$
$$a_{21}x_1 + a_{22}x_2 \leq b_2$$
$$x_1, x_2 \geq 0$$

2. **Nonlinear Programming (NLP)**: The objective function or constraints are nonlinear. Example:

$$Minimize\ f(x) = x^2 + \sin(x)$$
$$Subject\ to: x \geq 0$$

3. **Convex Optimization**: The objective function and feasible region are convex. Convex problems have unique global minima.
4. **Integer Programming**: Decision variables are restricted to integer values.

1.4 Modeling of the Optimization Problem

Modeling is the process of translating a real-world problem into a mathematical framework. This involves identifying the key components of the problem and expressing them in mathematical terms. The steps involved in modeling an optimization problem are:

1. **Define the Decision Variables**:
Decision variables are the unknowns that need to be determined to solve the problem. These variables represent the choices or decisions that can be made. For example, in a production planning problem, the decision variables could represent the quantities of different products to produce.
2. **Formulate the Objective Function**:
The objective function is a mathematical expression that represents the goal of the optimization problem. It could be to maximize profit, minimize cost, or achieve some other desired outcome. The objective function is typically expressed in terms of the decision variables.
3. **Identify Constraints**:
Constraints are the limitations or requirements that must be satisfied. These could include resource limitations, physical laws, or other restrictions. Constraints are expressed as mathematical inequalities or equalities involving the decision variables.
4. **Validate the Model**:
Once the model is formulated, it is important to validate it to ensure that it accurately represents the real-world problem. This may involve checking the model against historical data or using sensitivity analysis to test how changes in the parameters affect the solution.

Example: Production Planning

Let's consider a detailed example of a production planning problem.

Problem Statement:
A company produces two products, P_1 and P_2, with profits of 50 *and* 70 p er unit, respectively. The production process is subject to the following constraints:

- Machine A can work for up to 40 hours per week, and each unit of P_1 and P_2 requires 2 and 3 hours, respectively.
- Machine B can work for up to 30 hours per week, and each unit of P_1 and P_2 requires 4 and 2 hours, respectively.

Step 1: Define the Decision Variables

Let x_1 be the number of units of P_1 to produce, and x_2 be the number of units of P_2 to produce.

Step 2: Formulate the Objective Function

The goal is to maximize profit. The profit from producing x_1 units of P_1 and x_2 units of P_2 is:

$$Maximize\ z = 50x_1 + 70x_2$$

Step 3: Identify Constraints

The constraints are based on the available machine hours:

1. Machine A constraint:

$$2x_1 + 3x_2 \leq 40$$

2. Machine B constraint:

$$4x_1 + 2x_2 \leq 30$$

3. Non-negativity constraints:

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

Step 4: Solve the Problem

We can solve this linear programming problem using the **graphical method**.

1. **Plot the Constraints**:

- $For\ 2x_1 + x_2 \leq 4$:

$When\ x_1 = 0, \quad x_2 = 40/3 \approx 13.33$.
$When\ x_2 = 0, \quad x_1 = 20.$

- $For\ 4x_1 + 2x_2 \leq 30$:

$When\ x_1 = 0,\ \ x_2 = 15.$
$When\ x_2 = 0,\ \ x_1 = 7.5$.

2. **Identify the Feasible Region**:

The feasible region is the area where all constraints are satisfied. This is a polygon bounded by the intersection points of the constraints.

3. **Find the Corner Points**:

The optimal solution lies at one of the corner points of the feasible region. The corner points are:

- (0,0)

- (0,13.33)
- Intersection of $2x_1 + 3x_2 = 40 \; and \; 4x_1 + 2x_2 = 30$:
  Solving these equations simultaneously:

$$\begin{cases} 2x_1 + 3x_2 = 40 \\ 4x_1 + 2x_2 = 30 \end{cases}$$

  Multiply the first equation by 2:

$$4x_1 + 6x_2 = 80$$

  Subtract the second equation:

$$4x_2 = 50 \Rightarrow x_2 = 12.5$$

  Substitute x_2=12.5 into the first equation:

$$2x_1 + 3(12.5) = 40 \Rightarrow 2x_1 = 40 - 37.5 \Rightarrow x_1 = 1.25$$

  So, the intersection point is $(1.25,12.5)$.

4. **Evaluate the Objective Function at Each Corner Point**:

- $At\ (0,0): z = 50(0) + 70(0) = 0$
- $At\ (0,13.33): z = 50(0) + 70(13.33) = 933.1$
- $At\ (1.25,12.5): z = 50(1.25) + 70(12.5) = 62.5 + 875 = 937.5$
- $At\ (7.5,0): z = 50(7.5) + 70(0) = 375$

5. **Determine the Optimal Solution**:

The maximum profit is \$937.5, achieved by producing 1.25 units of P_1 and 12.5 units of P_2.

1.5 Solution with the Graphical Method:

The graphical method is a simple and intuitive way to solve linear programming problems with two decision variables. It involves plotting the constraints on a graph and identifying the feasible region. The optimal solution is found at one of the corner points of the feasible region.

Steps of the Graphical Method:[2]

1. **Plot the Constraints**:
   Convert each constraint into an equation and plot it on a graph. Shade the feasible region that satisfies all constraints.
2. **Identify the Feasible Region**:
   The feasible region is the area where all constraints overlap. It is typically a polygon.
3. **Find the Corner Points**:
   The optimal solution lies at one of the corner points (vertices) of the feasible region.

[2] Dantzig, G. B. (1949). **Programming of interdependent activities: II mathematical model.** *Econometrica, 17*(3), 200–211.

4. **Evaluate the Objective Function**:
   Calculate the value of the objective function at each corner point.
5. **Determine the Optimal Solution**:
   The corner point with the highest (for maximization) or lowest (for minimization) value of the objective function is the optimal solution.

Example: Graphical Method

Consider the following linear programming problem:

$$Maximize\ z = 3x_1 + 4x_2$$

$$Subject\ to: 2x_1 + x_2 \leq 10$$

$$x_1 + 3x_2 \leq 15$$

$$x_1, x_2 \geq 0$$

Step 1: Plot the Constraints

- For $2x_1 + x_2 \leq 10$:
  When $x_1 = 0, x_2 = 10$.
  When $x_2 = 0, x_1 = 5$.
- For $x_1 + 3x_2 \leq 15$:
  When $x_1 = 0, x_2 = 5$.
  When $x_2 = 0, x_1 = 15$.

Step 2: Identify the Feasible Region

The feasible region is the area bounded by the points $(0,0)$, $(0,5)$, $(3,4)$, and $(5,0)$.

Step 3: Find the Corner Points

The corner points are:

- $(0,0)$
- $(0,5)$
- Intersection of $2x_1 + x_2 = 10$ and $x_1 + 3x_2 = 15$:
  Solving these equations simultaneously:

$$\begin{cases} 2x_1 + x_2 = 10 \\ x_1 + 3x_2 = 15 \end{cases}$$

  Multiply the first equation by 3:

$$6x_1 + 3x_2 = 30$$

  Subtract the second equation:

$$5x_1 = 15 \Rightarrow x_1 = 3$$

  Substitute $x_1=3$ into the first equation:

$$2(3) + x_2 = 10 \Rightarrow x_2 = 4$$

  So, the intersection point is $(3,4)$.
- $(5,0)$

Step 4: Evaluate the Objective Function

- At $(0,0): z = 3(0) + 4(0) = 0$
- At $(0,5): z = 3(0) + 4(5) = 20$
- At $(3,4): z = 3(3) + 4(4) = 9 + 16 = 25$
- At $(5,0): z = 3(5) + 4(0) = 15$

Step 5: Determine the Optimal Solution
The maximum value of z is 25, achieved at $(3,4)$.

1.6 Convexity:

Convexity is a fundamental concept in optimization that ensures the existence of a unique global minimum (or maximum) for a given problem. Understanding convexity is crucial because it allows us to determine whether an optimization problem is "well-behaved" and can be solved efficiently.

Definition of Convex Sets

A set $S \subseteq \mathrm{R}^n$ is called **convex** if, for any two points $\mathbf{x}_1, \mathbf{x}_2 \in S$, the line segment connecting them lies entirely within S. Mathematically, this is expressed as:

$$\lambda x_1 + (1-\lambda)x_2 \epsilon\, S \; for\; all\; \lambda \epsilon [0,1].$$

Here, λ is a scalar between 0 and 1.

Example of a Convex Set:
A circle in 2D space is a convex set because any line segment connecting two points within the circle lies entirely inside the circle.

Example of a Non-Convex Set:
A crescent shape is not convex because there exist points in the set for which the connecting line segment lies partially outside the set.

Definition of Convex Functions:[3]

A function $f: \mathrm{R}^n \rightarrow \mathrm{R}$ is called **convex** if its domain is a convex set and for any two points $\mathbf{x}_1, \mathbf{x}_2$ in its domain, the following inequality holds:

$$(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \; for\; all\; \lambda \epsilon [0,1].$$

This inequality states that the function lies below the line segment connecting any two points on its graph.

Example of a Convex Function:
The function $f(x)=x_2$ is convex because its graph is a parabola that curves upward, and any line segment connecting two points on the parabola lies above the curve.

Example of a Non-Convex Function:
The function $f(x)=\sin(x)$ is not convex because its graph oscillates, and there exist points where the line segment connecting them lies below the curve.

Importance of Convexity in Optimization:[4]

[3] Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (p. 112). Springer Series in Operations Research.

Convexity is important because:

1. **Global Minimum**: A convex function has a unique global minimum. This means that any local minimum is also the global minimum.
2. **Efficient Algorithms**: Convex optimization problems can be solved efficiently using algorithms like gradient descent, Newton's method, and interior-point methods.
3. **Duality**: Convex problems have strong duality properties, meaning the primal and dual problems have the same optimal value.

Testing for Convexity

To determine whether a function is convex, we can use the following methods:

1. **First-Order Condition**:
   A differentiable function f is convex if:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \; for \; all \; x, y \; \epsilon \; dom(f).$$

   This means the function lies above its tangent plane at every point.

2. **Second-Order Condition**:
   A twice-differentiable function f is convex if its Hessian matrix $\nabla^2 f(x)$ is positive semi-definite for all $x \; \epsilon \; dom(f)$. That is:

$$v^T \nabla^2 f(x) v \geq 0 \; for \; all \; v \epsilon R^n.$$

Example: Testing Convexity of a Quadratic Function
Consider the function $f(x_1, x_2) = x_1^2 + 2x_2^2 + 3x_1 x_{2..}$ To test its convexity:

1. Compute the Hessian matrix:

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}.$$

2. Check if the Hessian is positive semi-definite:
   The eigenvalues of the Hessian are λ_1=5.37 and λ_2=0.63, both of which are positive. Therefore, the Hessian is positive definite, and the function is convex.

1.7 Gradient Vector, Directional Derivative, and Hessian Matrix:

These concepts are essential for understanding how optimization algorithms work, particularly in multivariable optimization.

Gradient Vector

The gradient of a function f: $\mathrm{R}^n \rightarrow \mathrm{R}$ is a vector of its partial derivatives with respect to each variable. It points in the direction of the steepest ascent of the function. Mathematically:

[4] Griva, I., Nash, S. G., & Sofer, A. (2009). Linear and nonlinear optimization (p. 98). Philadelphia: SIAM.

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

Example:
For $f(x,y)=x^2+y^2$, the gradient is:

$$\nabla f(x,y) = \begin{bmatrix} 2x \\ 3y \end{bmatrix}$$

Directional Derivative

The directional derivative of f in the direction of a vector $\mathbf{v}$ measures the rate of change of f along $\mathbf{v}$. It is given by:

$$D_{\mathbf{v}} f(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{v}.$$

Example:
For $f(x,y) = x^2 + y^2$ and $\mathbf{v}=[1,1]^T$, the directional derivative at $(1,1)$ is:

$$Dvf(1,1) = \begin{bmatrix} 2(1) \\ 2(1) \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 4.$$

Hessian Matrix

The Hessian matrix is a square matrix of second-order partial derivatives of a function. It provides information about the curvature of the function. For $f : \mathrm{R}^n \rightarrow \mathrm{R}$, the Hessian is:

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

Example:
For $f(x,y)=x^2+2xy+y^2$, the Hessian is:

$$\nabla^2 f(x,y) = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

1.8 Linear and Quadratic Approximations:

Linear and quadratic approximations are powerful tools in optimization and numerical analysis. They allow us to approximate complex functions using simpler forms, making it easier to analyze and solve optimization problems.

Linear Approximation

A linear approximation of a function $f: \mathrm{R}^n \to \mathrm{R}$ near a point $\boldsymbol{x}_0$ is given by the first-order Taylor expansion:[5]

$$f(x) \approx f(x_0) + \nabla f(x_0)^T (x - x_0).$$

This approximation is valid when $\boldsymbol{x}$ is close to $\boldsymbol{x}_0$.

Interpretation:
The linear approximation represents the tangent hyperplane to the function at $\boldsymbol{x}_0$. It captures the local behavior of the function in the neighborhood of $\boldsymbol{x}_0$.

Example:
Consider the function $f(x,y)=x^2+y^2$ at the point $\boldsymbol{x}_0=(1,1)$. The gradient at $\boldsymbol{x}_0$ is:

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

The linear approximation near $\boldsymbol{x}_0$ is:

$$f(x,y) \approx f(1,1) + \begin{bmatrix} 2 \\ 2 \end{bmatrix}^T \begin{bmatrix} x-1 \\ y-1 \end{bmatrix} - 2 + 2(x-1) + 2(y-1).$$

Simplifying, we get:

$$f(x,y) \approx 2x + 2y - 2$$

Quadratic Approximation

A quadratic approximation of a function $f: \mathrm{R}^n \to \mathrm{R}$ near a point $\boldsymbol{x}_0$ is given by the second-order Taylor expansion:

$$f(x) \approx f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

This approximation includes both the gradient and the Hessian matrix, providing a more accurate representation of the function near $\boldsymbol{x}_0$.

Interpretation:
The quadratic approximation represents the local curvature of the function at $\boldsymbol{x}_0$. It captures both the slope and the curvature of the function in the neighborhood of $\boldsymbol{x}_0$.

Example:
Consider the same function $f(x,y)=x^2+y^2$ at the point $\boldsymbol{x}_0=(1,1)$. The Hessian matrix at $\boldsymbol{x}_0$ is:

$$\nabla^2 f(1,1) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The quadratic approximation near $\boldsymbol{x}_0$ is:

[5] Epperson, J. F. (2010). *An introduction to numerical methods and analysis* (p. 88). Hoboken, NJ: John Wiley & Sons.

$$f(x,y) \approx f(1,1) + \begin{bmatrix}2\\2\end{bmatrix}^T \begin{bmatrix}x-1\\y-1\end{bmatrix} + \frac{1}{2}\begin{bmatrix}x-1\\y-1\end{bmatrix}^T \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}\begin{bmatrix}x-1\\y-1\end{bmatrix}$$

Simplifying, we get:

$$f(x,y) \approx 2 + 2(x-1) + 2(y-1) + (x-1)^2 + (y-1)^2$$

Applications in Optimization:[6]

1. **Gradient Descent**:

   Gradient descent uses the linear approximation (gradient) to iteratively move toward the minimum of a function. At each step, the algorithm updates the current point $\boldsymbol{x}_k$ as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k),$$

   where α is the learning rate.

2. **Newton's Method**:

   Newton's method uses the quadratic approximation (Hessian) to find the minimum of a function. At each step, the algorithm updates the current point $\boldsymbol{x}_k$ as:

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

   This method converges faster than gradient descent but requires computing and inverting the Hessian matrix.

   Example: Gradient Descent vs. Newton's Method

   Consider the function $f(x)=x^4-3x^3+2$.

- **Gradient Descent**: Uses the gradient $f'(x)=4x^3-9x^2$ to iteratively update x.
- **Newton's Method**: Uses both the gradient and the Hessian $f''(x)=12x^2-18x$ to update xx.

  Newton's method converges faster because it accounts for the curvature of the function.

  Practical Applications of Approximations

  Linear and quadratic approximations are not just theoretical tools; they are widely used in optimization algorithms, physics, and engineering to simplify complex problems.

  Example: Robot Motion Planning

  In robotics, quadratic approximations model the cost of moving a robot from one configuration to another. For instance, minimizing energy consumption while avoiding obstacles can be approximated using quadratic functions, enabling efficient path planning.

  1.9 Applications of Optimization:

  Optimization has a wide range of applications in various fields. Here are a few examples:[7]

1. **Machine Learning**:

- Training models by minimizing loss functions (e.g., linear regression, neural networks).
- Algorithms like gradient descent and stochastic gradient descent are widely used.

[6] Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (p. 152). Springer Series in Operations Research.
[7] Rao, S. S. (2009). Engineering optimization: Theory and practice (p. 134). Hoboken, NJ: John Wiley & Sons.

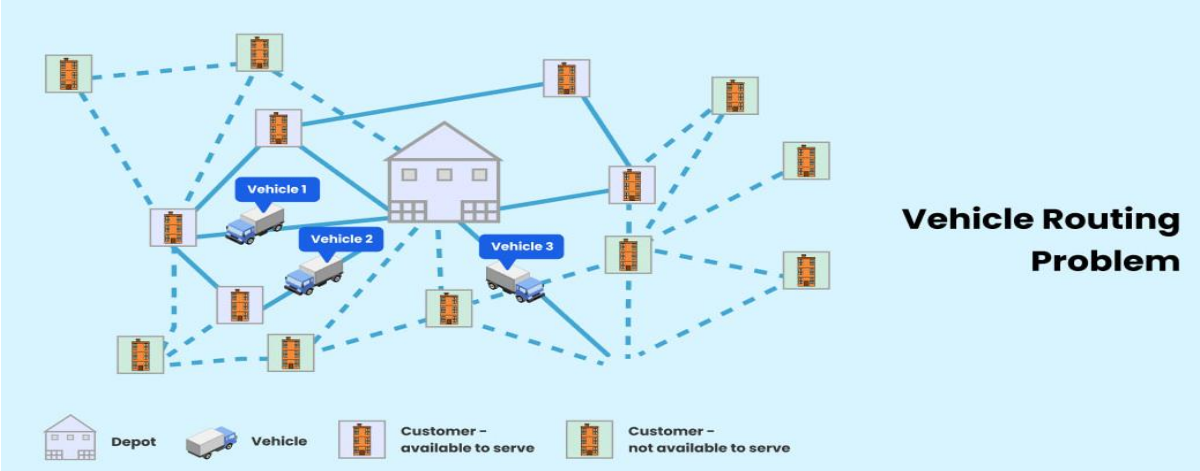
2. **Operations Research**:
   - Resource allocation, scheduling, and logistics (e.g., the traveling salesman problem).
   - Linear programming and integer programming are commonly used.
3. **Economics and Finance**:
   - Portfolio optimization to maximize returns while minimizing risk.
   - Utility maximization subject to budget constraints.
4. **Engineering**:
   - Structural optimization to minimize weight while maintaining strength.
   - Control systems optimization to achieve desired performance.
5. **Data Science**:
   - Clustering and classification problems (e.g., k-means clustering).
   - Feature selection and dimensionality reduction.

1.10 Figures and Schemas:

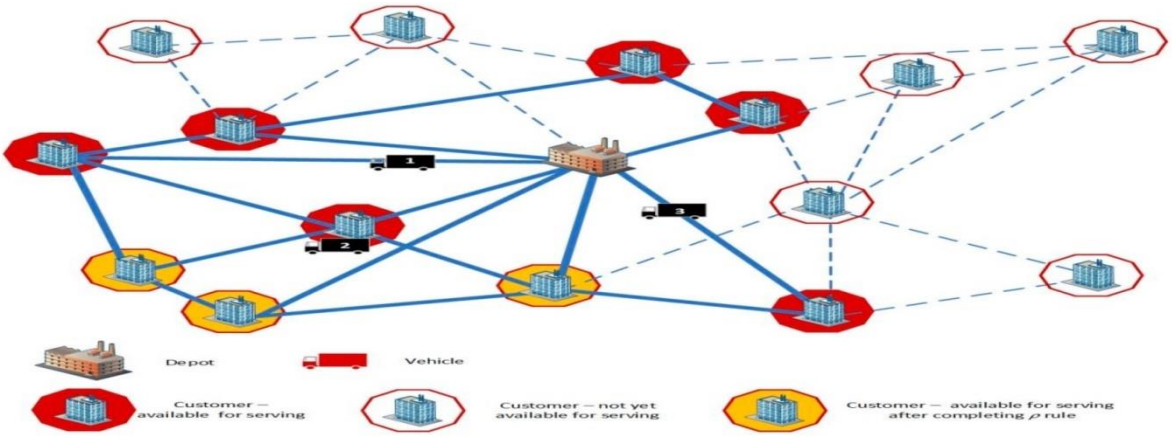
a) *Transportation Optimization*:

Transportation optimization enhances **route efficiency, cost reduction, and timely deliveries** by leveraging data-driven strategies and advanced logistics planning. This concept will be **explained through schemas and figures**, illustrating optimized vehicle routing, resource allocation, and supply chain improvements.

- **Figure 1**: Illustration of vehicle routing optimization with multiple delivery points.[8]



[8]https://www.researchgate.net/publication/287796502_The_Vehicle_Routing_Problem_State_of_the_Art_Classification_and_Review



The **Vehicle Routing Problem (VRP)** illustrated in these images highlights the critical role of **transportation optimization** in logistics and supply chain management. Efficient routing offers several key benefits:

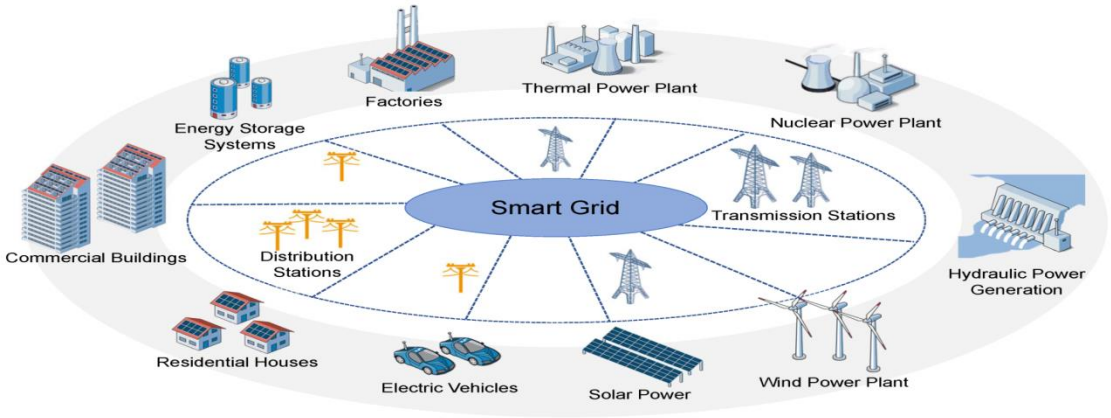
1. **Cost Reduction** : Optimized routes minimize fuel consumption, vehicle maintenance, and labor costs.
2. **Time Efficiency** : Reducing travel distance and optimizing delivery schedules ensures faster service.
3. **Customer Satisfaction** : Timely and reliable deliveries improve customer trust and business reputation.
4. **Resource Utilization** : Balancing workloads across multiple vehicles prevents under- or over-utilization.
5. **Sustainability** : Lower fuel usage leads to reduced carbon emissions, supporting eco-friendly operations.

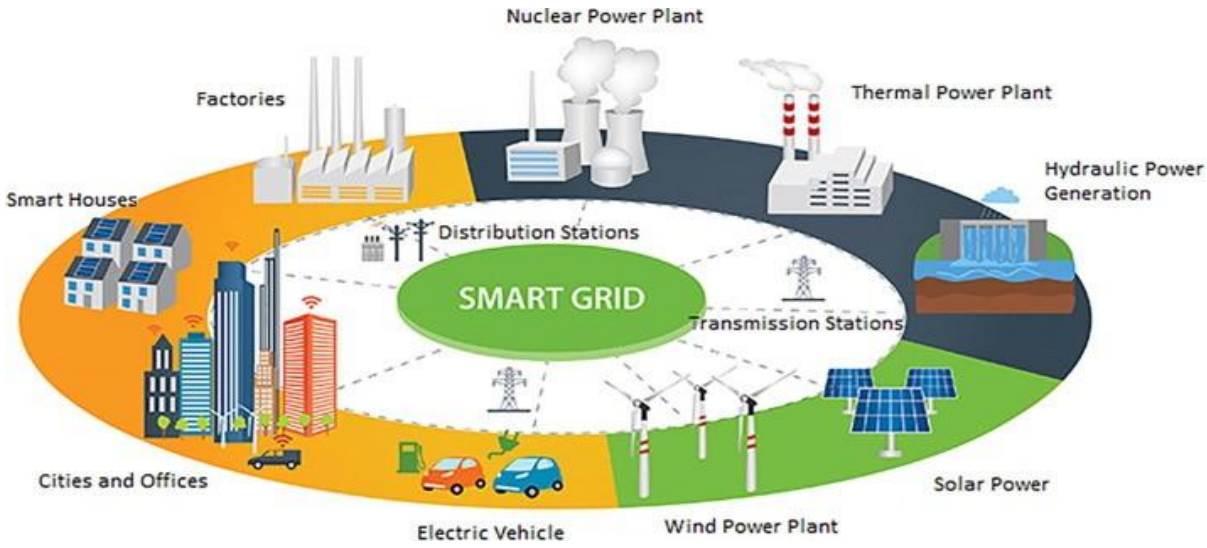
By implementing smart **transportation optimization**, businesses can enhance **profitability, efficiency, and sustainability**, making logistics **more responsive and cost-effective**.

b) *Energy System Optimization*:

Energy system optimization ensures **efficient generation, distribution, and consumption of energy**, reducing waste and enhancing sustainability. This will be **explained through schemas and figures**, showcasing smart grids, renewable integration, and optimized energy resource allocation.

- **Figure 2**: Diagram showing the optimization of energy generation and distribution in a smart grid.





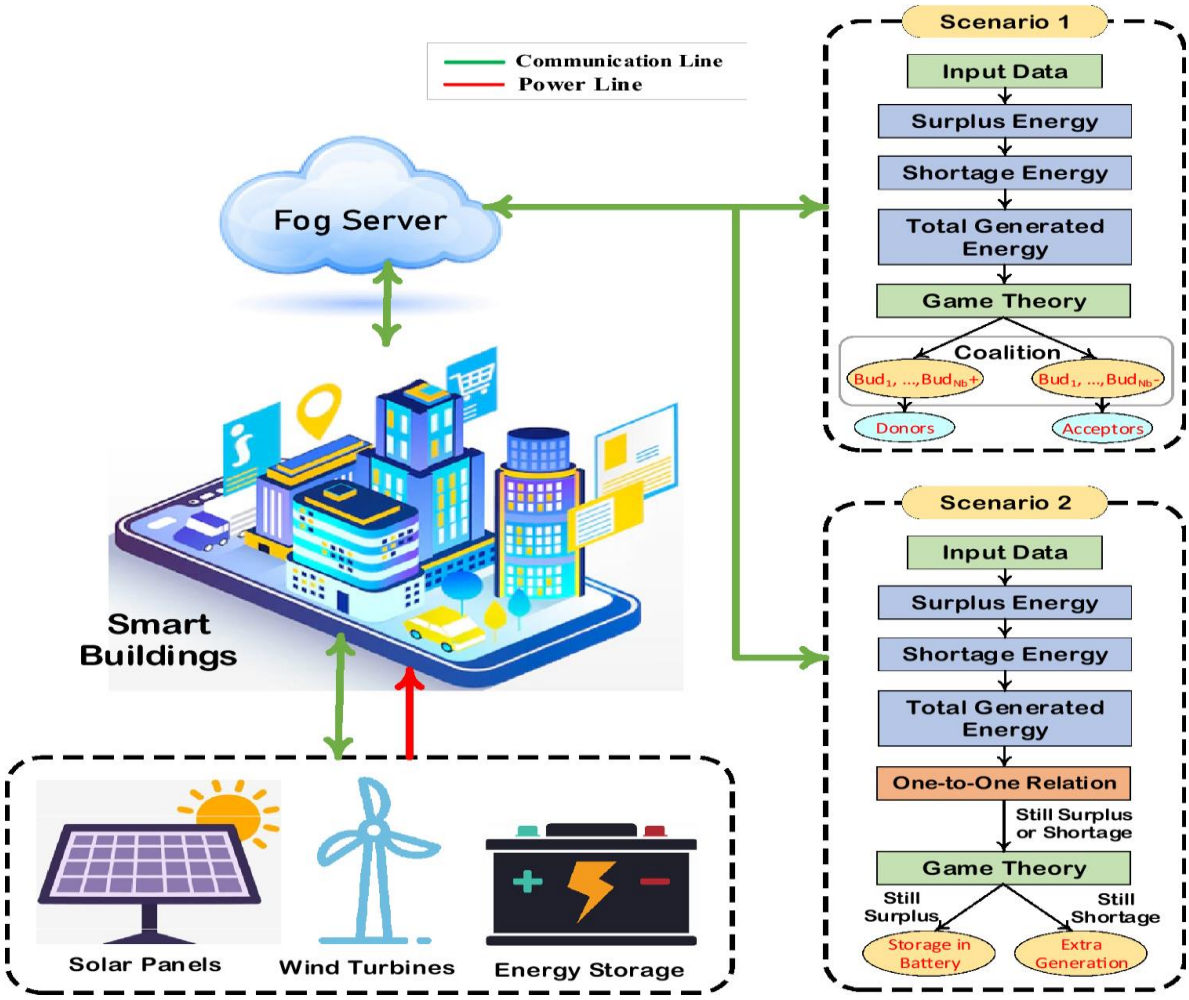
These diagrams illustrate the **Smart Grid**, an advanced energy system that efficiently manages **generation, distribution, and consumption** using multiple energy sources.

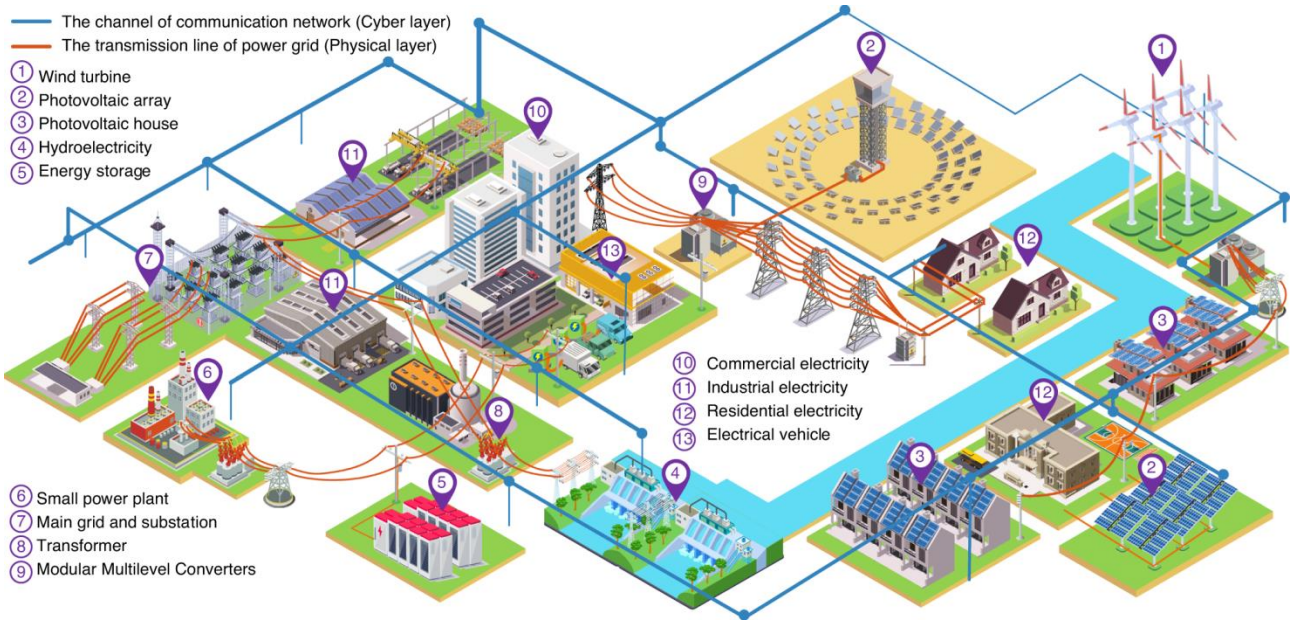
1. **Balanced Energy Distribution** – Smart grids dynamically **adjust power flow** between **renewable (solar, wind, hydro) and traditional (nuclear, thermal) sources** to meet demand efficiently.
2. **Grid Stability & Reliability** – **Optimized transmission and distribution** reduce blackouts and ensure uninterrupted power supply to **residential, commercial, and industrial users**.
3. **Sustainability & Cost Savings** – **Integrating renewable sources and storage systems** minimizes reliance on fossil fuels, lowering both **carbon footprint and energy costs**.
4. **Smart Load Management** – Real-time monitoring and automation ensure **optimized energy usage** for factories, electric vehicles, and homes.

By leveraging **intelligent energy optimization**, smart grids **enhance efficiency, reduce waste, and promote sustainability**, ensuring a **resilient and eco-friendly power future**.

- **Schema 2**: Graphical representation of the optimization process for energy resource allocation.[9]

[9] Mendoza, J. E., Castanier, B., Guéret, C., Medaglia, A. L., & Renaud, J. (2010). The Vehicle Routing Problem: State of the Art Classification and Review. *Pesquisa Operacional*, 30(2), 215-258.





These Schemas representations highlight the role of **smart energy management** in optimizing resource allocation across **renewable sources, storage, and consumption points**.

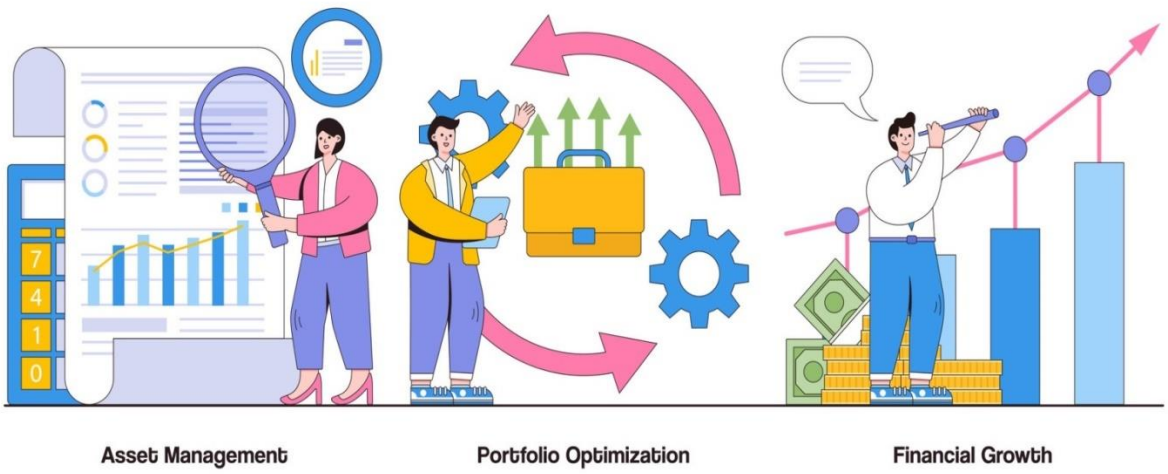
1. **Efficient Energy Utilization** – Smart grids distribute **surplus energy** to areas experiencing shortages, reducing waste and maximizing renewable energy usage.
2. **Real-Time Decision Making** – **Fog computing and cyber-physical layers** enable intelligent decisions for energy transfer between **solar panels, wind turbines, energy storage, and smart buildings**.
3. **Game Theory-Based Optimization** – The system forms a **coalition of energy donors and acceptors**, ensuring balanced energy distribution and sustainable operations.
4. **Enhanced Grid Stability & Reliability** – Smart grids **prevent power failures** by dynamically adjusting energy flow based on demand and supply conditions.
5. **Sustainability & Cost Savings** – **Energy storage solutions and smart allocation techniques** reduce dependency on fossil fuels and lower energy costs.

   By **optimizing energy resource allocation**, smart grids create a **resilient, cost-effective, and eco-friendly** power system, ensuring efficient electricity distribution for residential, commercial, and industrial needs.

c) *Financial Portfolio Optimization*:

Financial portfolio optimization aims to **maximize returns while minimizing risks** through strategic asset allocation and diversification. This will be **explained through schemas and figures**, illustrating decision trees, risk assessment models, and optimal investment strategies.

- **Figure 3**: Visualization of an optimized investment portfolio with balanced asset allocation.



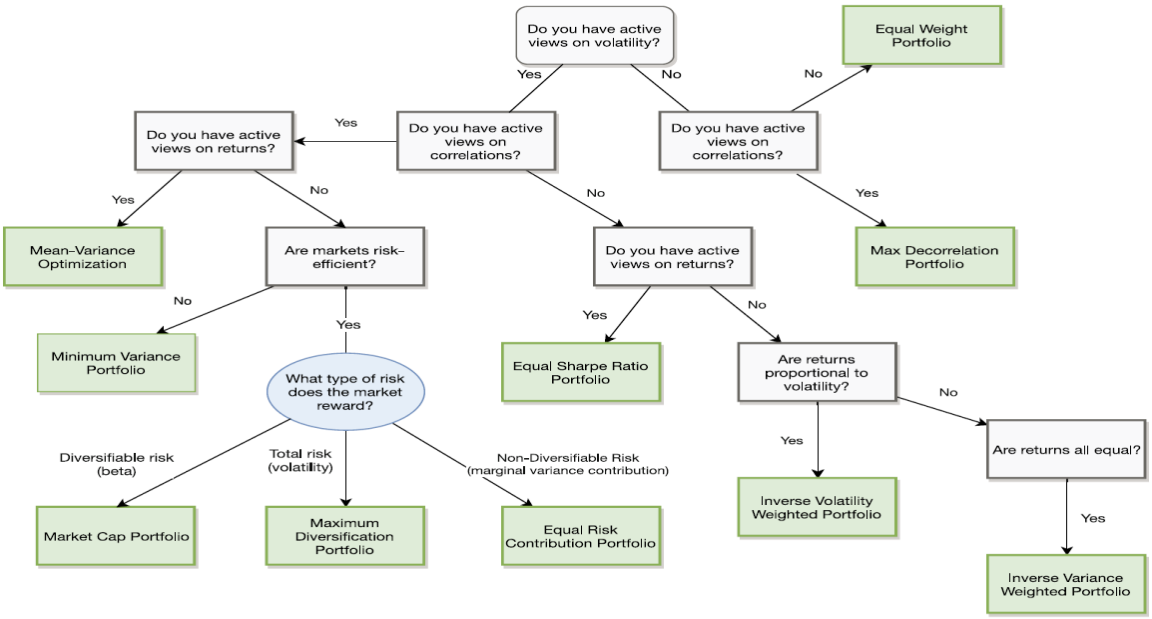
This visualization highlights the **strategic approach to investment management**, ensuring **financial growth through asset optimization**.

1. **Asset Management for Risk Control** – Proper allocation **diversifies investments**, reducing risks and improving stability in fluctuating markets.
2. **Portfolio Optimization for Maximum Returns** – A well-structured portfolio balances **growth, income, and safety**, adapting to market conditions for **optimal financial performance**.

3. **Long-Term Financial Growth** – Smart investment strategies enhance **wealth accumulation** and **financial security** over time.
4. **Data-Driven Decision Making** – Continuous analysis and adjustments ensure investments align with **market trends and financial goals**.

   By **optimizing portfolio allocation**, investors achieve **sustained growth, minimized risk, and improved profitability**, leading to **greater financial success**.

- **Schema 3**: Decision tree outlining the steps in optimizing asset allocation for portfolio management.



This decision tree outlines a structured approach to **portfolio optimization**, helping investors **choose the best allocation strategy** based on market conditions and risk preferences.

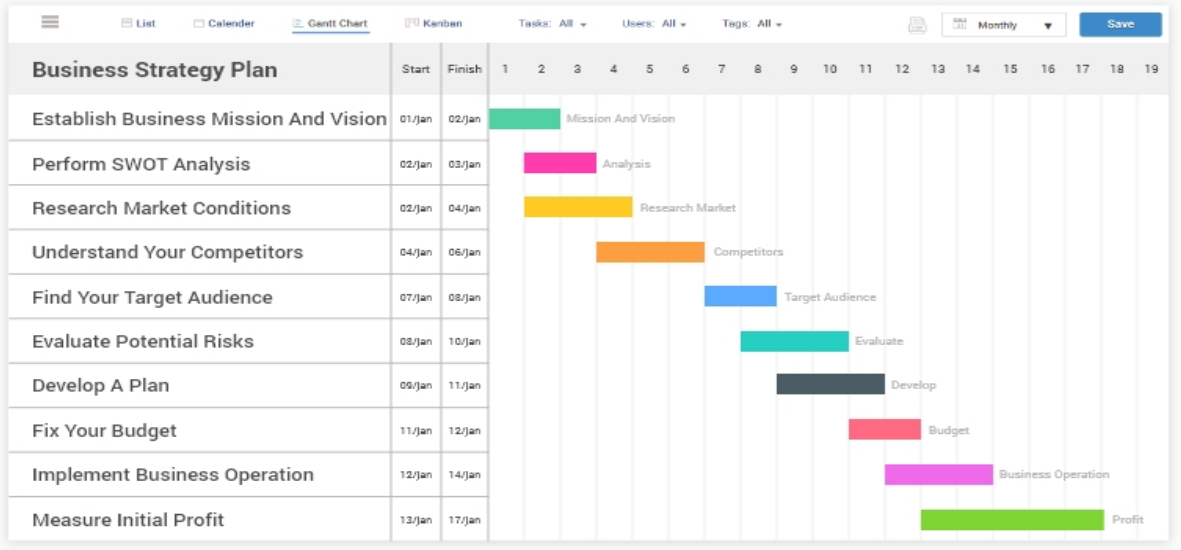
1. **Customizable Strategies** – Investors can **tailor portfolios** based on their views on **returns, volatility, and correlations** to align with financial goals.
2. **Risk-Reward Balance** – Identifies **whether markets reward diversifiable or non-diversifiable risk**, ensuring an optimal mix of assets.
3. **Efficient Portfolio Selection** – Guides investors toward **Mean-Variance Optimization, Minimum Variance, Maximum Diversification, or Equal Weight strategies** depending on risk efficiency.
4. **Data-Driven Decision Making** – A systematic framework **eliminates guesswork** and enhances **investment stability and profitability**.

   By **optimizing asset allocation**, investors achieve **risk-adjusted returns, diversification benefits, and long-term financial growth**, ensuring a **resilient and profitable portfolio**.

d) *Production Planning Optimization*:

Production planning optimization enhances **efficiency, resource utilization, and workflow scheduling**, ensuring minimal waste and maximum output. This will be **explained through schemas and figures**, illustrating Gantt charts, resource allocation models, and optimized manufacturing processes.

- **Figure 4**: Gantt chart demonstrating optimized production scheduling for a manufacturing plant.

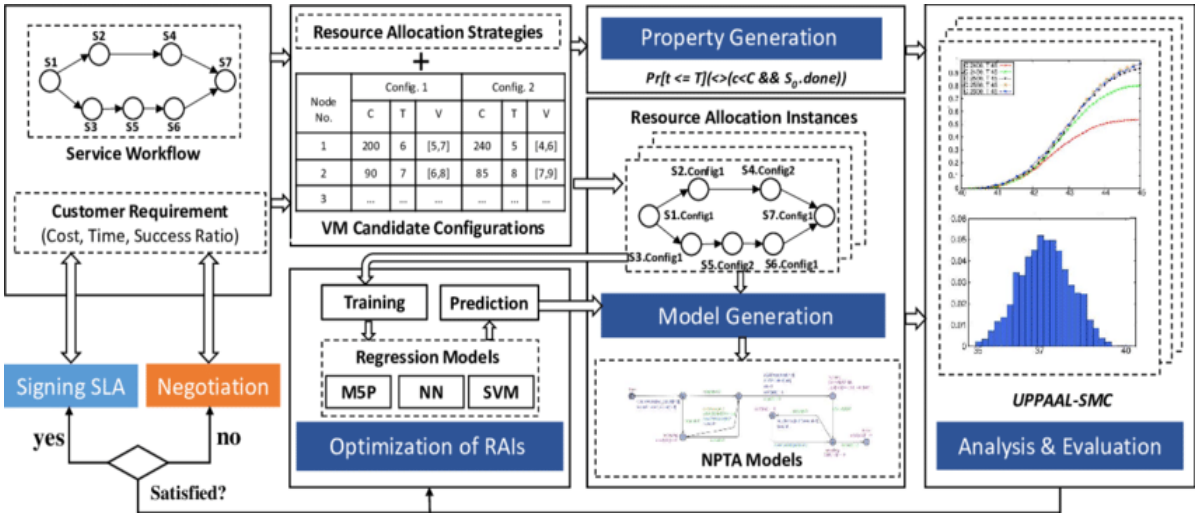


This Gantt chart illustrates **optimized production scheduling**, a crucial element in **efficient business strategy planning and manufacturing operations**.

1. **Improved Workflow Efficiency** – Sequential task organization ensures a **structured production process**, minimizing downtime and delays.
2. **Resource Optimization** – Helps in **allocating manpower, materials, and equipment efficiently**, reducing waste and maximizing output.
3. **Time Management** – Establishing **clear start and finish dates** enhances **project tracking and deadline adherence**.
4. **Risk Mitigation** – Identifies potential bottlenecks (e.g., budget constraints, market research delays) and enables proactive adjustments.
5. **Increased Profitability** – Structured execution of tasks like **market research, risk evaluation, and financial planning** leads to **higher productivity and profitability**.

By **optimizing production scheduling**, businesses can achieve **better coordination, cost savings, and streamlined operations**, ensuring **sustained growth and competitiveness**.

- **Schema 4**: Schematic diagram illustrating the optimization of resource allocation in production planning.



This schematic diagram illustrates the **optimization of resource allocation** in **production planning**, leveraging **machine learning models and workflow strategies**.

1. **Efficient Resource Utilization** – Ensures optimal allocation of **virtual machines (VMs), computational resources, and time**, minimizing waste.
2. **Automated Decision-Making** – Uses **regression models (M5P, NN, SVM)** to predict and optimize **resource allocation strategies** dynamically.
3. **Enhanced Performance & Cost Efficiency** – Balances **cost, time, and success ratio**, ensuring production meets customer requirements.
4. **Scalability & Adaptability** – The model continuously adjusts based on **real-time analysis & evaluation (UPPAAL-SMC)** for improved efficiency.
5. **Data-Driven Optimization** – Uses **statistical modeling and machine learning** to optimize production workflows and reduce inefficiencies.

By implementing **resource allocation optimization**, production planning becomes **more cost-effective, scalable, and responsive**, leading to **higher efficiency and improved service reliability**.

❖ Chapter Highlights:

- An optimization problem involves formulating an objective function to be either maximized or minimized, subject to inequality and equality constraints. These functions depend on design variables, which are assessed using optimization techniques.
- Design variables can be real numbers or take discrete, binary, or integer forms.
- Modelling entails representing a problem mathematically using fundamental operations such as addition, subtraction, multiplication, division, and various functions, ensuring appropriate units are applied.
- The **gradient** at a given point defines the slope of the tangent at that point.
- If both the objective function and constraints are linear in terms of design variables, the problem is classified as **linear programming**, which excludes multiplicative terms like x_1x_2 or x^2.
- The **graphical method** is applicable for solving optimization problems involving up to

three design variables.

- Functions with multiple local minima or maxima are termed **multimodal functions**.
- The concept of **convexity** is crucial in determining whether a function possesses a single minimum. A convex function guarantees a global minimum.
- Optimization algorithms are typically designed for **minimization**. If an objective function requires maximization, it is transformed into an equivalent minimization problem by negation.
- The **necessary condition** for optimality—whether maximum or minimum—is that the gradient at the given point must be zero.
- At an optimal point, the second derivative of the objective function determines the nature of the extremum:

❖ A positive second derivative indicates a **minimum**.

❖ A negative second derivative indicates a **maximum**.

- Numerical differentiation methods, including **forward, backward, and central difference methods**, can estimate the derivative of a function. Among these, the **central difference method** provides the highest accuracy.
- The **directional derivative** measures the instantaneous rate of change of a function in a specified direction.
- The **Hessian matrix (H)** represents the second-order derivatives of a function with multiple variables.
- At a function's minimum, the Hessian matrix must be **positive definite**, meaning all its eigenvalues are positive.
- Quadratic **approximations** are often useful in optimization, particularly for methods like **Newton's method**, which achieve faster convergence with quadratic functions.
- Taylor **series approximation** is employed to derive **linear or quadratic approximations** of functions, depending on the number of terms included in the expansion.

❖ Formulae Chart:

- Forward difference:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- Backward difference:

$$f'(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

- Central difference:

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

- Central difference formula for the second derivative:

$$f''(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2}$$

- Jacobian of three functions with three variables:

$$[\boldsymbol{J}] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}$$

- Hessian for a three-variable function:

$$[\boldsymbol{H}] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \frac{\partial^2 f}{\partial x_1 x_3} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_3} \\ \frac{\partial^2 f}{\partial x_3 x_1} & \frac{\partial^2 f}{\partial x_3 x_2} & \frac{\partial^2 f}{\partial x_3^2} \end{bmatrix}$$

- Quadratic approximation:

$$f(x+\Delta x) = f(x) + \nabla f(x)^T \Delta x + \frac{1}{2}\Delta x^T H \Delta x$$

❖ Optimization Problems

1. An airline company in India uses A320 aircraft to fly passengers from New Delhi to Mumbai. Though the maximum seating capacity of the aircraft is 180, the airline observes that on average it flies only 130 passengers per flight. The regular fare between the two cities is Rs. 15,000. From the market survey, the company knows that for every Rs. 300 reduction in fare, it would attract an additional four passengers. The company would like to find a fare policy that would maximize its revenue. Formulate this as an optimization problem.

2. The average yield in a farm is 300 apples per tree, if 50 apple trees are planted per acre. The yield per tree decreases by 3 apples for each additional tree planted per acre. How many additional trees per acre should be planted to maximize the yield? Formulate this as an optimization problem.

3. Determine the area of the largest rectangle that can be inscribed in a circle of radius 5 cm. Formulate this as an optimization problem by writing down the objective function and the constraint. Solve the problem using the graphical method.

4. A field needs to be enclosed with a fence, with a river flowing on one side of the field. We have 300 m of fencing material. Our aim is to use the available fencing material and cover the maximum area of the field. Formulate this as an optimization problem by writing down the objective function and the constraint and clearly stating the design variables.

5. A traveling salesman has to start from city A, cover all other *n* number of cities, and then come back to city A. The distance between the ith and jth cities is given by y_{ij}. How could he plan the route so to cover the minimum distance? Formulate this as an optimization problem.

6. A company has initial wealth *W* and would like to invest this to get maximum returns. It can get higher returns (r_r) if it invests in risky assets, but the return is not guaranteed. A return (r_s) is guaranteed if it invests in safe assets. How much should the company invest in risky assets (*R*), to maximize its wealth at the end of a stipulated period? Formulate the objective function for the optimization problem.

7. In an experiment, the following observations (see Table 1.3) are made where *x* is an independent variable and *y* is a dependent variable. It is desired to fit these data with a straight line:

$$\hat{y} = mx + c$$

where m and c are to be determined. The data are to be fitted in the least squares sense, that is, $\sum(y_i - \hat{y})^2$ is to be minimized. Formulate this as an optimization problem.

8. The cost of a solar energy system (King 1975) is given by $U = 35A + 208V$, where *A* is the surface area of the collector and *V* is the volume of storage (Figure 1.19). Due to energy balance considerations, the following relation between *A* and *V* is to be satisfied:

$$A\left(290 - \frac{100}{V}\right) = 5833.3$$

TABLE 1.3

Data Observed from an Experiment

x_i	1	2	3	4	5
y_i	45	55	70	85	105

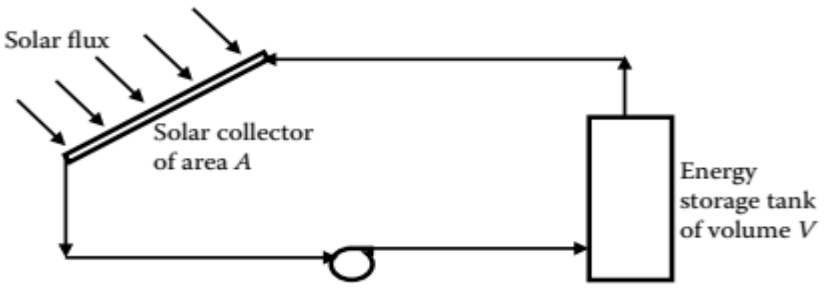


FIGURE 1.19
Solar energy problem.

The design variable *T* is related to *V* as:

$$V = \frac{50}{T - 20}$$

The variable *T* has to be restricted between 40°C and 90°C. The cost U is to be minimized. Formulate this as an optimization problem.

9. Write the gradient and Hessian matrix for the function:

$$f(x) = 5x_1x_2 + \ln(2x_1^2 + 3x_2^2)$$

10. A company manufactures three products: A, B, and C. Each product requires time for three processes: 1, 2, and 3, as given in Table 1.4.

TABLE 1.4

Time Required for Each Process

	Time Required (minute)/Unit		
Product	**A**	**B**	**C**
Process 1	12	25	7
Process 2	11	6	20
Process 3	15	6	5

TABLE 1.5

Maximum Capacity of Each Process

Process	Capacity (minutes)
1	28,000
2	35,000
3	32,000

TABLE 1.6

Profit per Unit of Each Product

Product	Profit/Unit
A	5
B	7
C	4

The maximum available capacity on each process is given in Table 1.5. The profit per unit for the product is given in Table 1.6. What quantities of A, B, and C should be produced to maximize profit? Formulate this as an optimization problem.

11. A company has three factories and five warehouses. The warehouse demand, factory capacity, and shipping cost are given in Table 1.7. Determine the optimal shipment plan to minimize the total cost of transportation. Formulate the optimization problem.

TABLE 1.7

Cost per Unit of Shipment from Factory to Warehouse

	Warehouse					
	A	B	C	D	E	
From Factory	Cost per Unit of Shipment					Capacity (No. of Units)
P	3	7	4	6	5	150
Q	5	4	2	5	1	110
R	6	3	2	2	4	90
Demand	50	100	70	70	60	

12. Plot the function:

$$f(x) = (x + 3)(x - 1)(x + 4)$$

and locate the minimum and maximum in [-4, 0].

13. An oil refinery company blends four raw gasoline types (A, B, C, and D) to produce two grades of automobile fuel, standard and premium. The cost per barrel of different gasoline types, performance rating and number of barrels available each day is given in Table 1.8.

TABLE 1.8

Cost, Performance Rating, and Production Level of Different Gasoline Types

	Cost/Barrel in Dollars	Performance Rating	Barrels/Day
A	60	75	3000
B	65	85	4000
C	70	90	5000
D	80	95	4000

The premium should have a rating greater than 90 while the standard fuel should have a performance rating in excess of 80. The selling prices of standard and premium fuel are 90 dollars and 100 dollars per barrel respectively. The company should produce at least 6000 barrels of fuel per day. Determine how much quantity of fuel (of each type) should be produced to maximize profit? Formulate this as an optimization problem.

14. Check whether the following functions are convex or not:

$$a.\ 2x^2 - 3x + 5 \qquad x \in [-4,4]$$
$$b.\ x^3 - 2x^2 + 4x - 10 \qquad x \in [-3,1]$$
$$c.\ \frac{1}{1-x^2} \qquad x \in [-1.6,-0.8]$$
$$d.\ \sqrt{x^2 + 2x + 5} \qquad x \in [-5,5]$$

15. Write the first three terms of the Taylor series for the function:

$$f(x) = \ln(x - 1) \quad / \text{ at } x = 3.$$

16. Find the linear approximation of the function:

$$f(x) = (1 + x)^{50} + (1 - 2x)^{60} \quad / \text{ at } x = 1.$$

17. Write the Taylor series expansion (up to four terms) for the function e^x centered at x = 3.

18. Write the Taylor series expansion (up to three terms) for the function $e^{\cos(x)}$ centered at x = π.

19. Find the quadratic approximation of the function $f(x) = ln(1 + sin\, x)$ at x = 0.

20. Find the directional derivative of the function: $f(x) = x_1^2 x_2 + x_2^2 x_3 - x_1 x_2\, x_3^2$ at (1, 1, −1) in the direction $\begin{bmatrix}1\\2\\3\end{bmatrix}$.

21. Using MATLAB, plot the functions $\boldsymbol{x^4}$ and $|x|$ and check whether these functions are convex.

22. Solve the following optimization problems using the graphical method:

i. Maximize $z = 125x_1 + 150x_2$
subject to: $6x_1 + 11x_2 \leq 66$,
$8x_1 + 9x_2 \leq 72$,
$x_1, x_2 \geq 0$.

ii. Maximize $z = 3x_1 + 4x_2$,
subject to: $2x_1 + x_2 \leq 30$,
$x_1 + 3x_2 \geq 40$,
$x_1, x_2 \geq 0$.

23. Calculate the Jacobian of the following system of equations:

$$\begin{bmatrix} x_1 + 2x_2^2 + 3x_3^3 \\ x_1^2 x_2 x_3^2 \\ 3x_1x_2 - 2x_1x_3 + 4x_2x_3 \end{bmatrix}$$

Chapter II: Convex Optimization Overview

2.1 Introduction:

In the preceding chapter, our attention was centered on providing an Introduction to Optimization and elucidating the Simplex Method. This chapter transitions our focus towards another critical element within convex optimization, specifically, convex functions. The forthcoming discussion will cover a range of topics, including:

- The distinctions and characteristics of convex, concave, strictly convex, and strongly convex functions.
- First and second-order characterizations of convex functions.
- Conditions for optimality within convex optimization problems.

2.2 Theory of convex functions:

a. Definition

Let us begin by revisiting the definition of a convex function.

Definition 1.[10] *A function* $f : \mathrm{R}^n \to \mathrm{R}$ *is convex if its domain is a convex set and for all* x, *in its domain, and all* $\lambda \in [0, 1]$, *we have*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

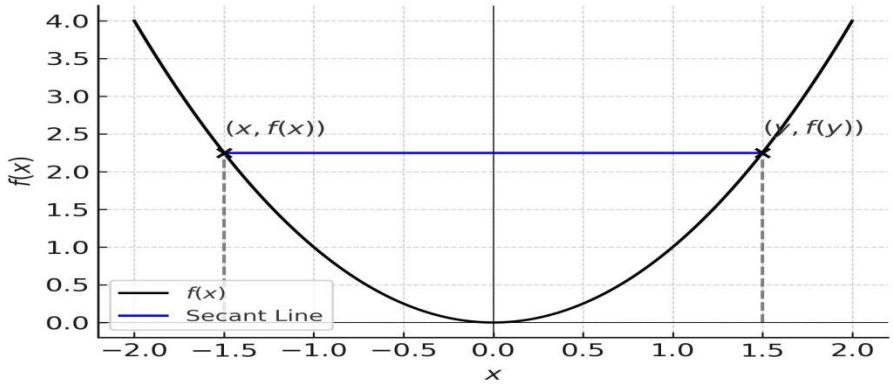


Figure 1: An illustration of the definition of a convex function

- Expressed formally, this implies that for any two points x, y, when function f is evaluated at any convex combination of x and y, the result should not exceed the convex combination of f(x) and f(y).
- From a geometric perspective, the line segment that connects the points (x, f(x)) and (y, f(y)) must remain above the graph of the function f.

[10] Boyd, S., & Vandenberghe, L. (2004). Convex optimization (p. 67). Cambridge University Press.

- In the case of f being continuous, establishing convexity requires only the verification of the definition for λ = 1 (or any other constant λ within the open interval (0, 1)), mirroring the concept of midpoint convex sets previously discussed.
- The function f is defined as concave if the negation of *f*, denoted as *-f*, exhibits convexity.

b. Examples of univariate convex functions

To rigorously ascertain the convexity of specific functions, it is standard practice to utilize the second derivative test for convexity. This test stipulates that a function f(x) is deemed convex over a certain interval if its second derivative f''(x) remains non-negative (f''(x) ≥ 0) across that interval. An examination of the functions in question yields the following insights:

1. For $f(x) = e^{ax}$, the calculation of the first and second derivatives yields $f'(x) = ae^{ax}$ and $f''(x) = a^2e^{ax}$, respectively. Given that e^{ax} consistently holds a positive value and a^2 is inherently non-negative, the condition $f''(x) \geq 0$ is satisfied, affirming the function's convexity.

2. In the case of $f(x) = -\log(x)$, the derivatives are determined to be $f'(x) = -\frac{1}{x}$ and $f''(x) = \frac{1}{x^2}$. The condition $x^2 > 0$ for all $x \neq 0$ ensures that $f''(x) > 0$, thereby confirming the function's convexity.

3. The function $f(x) = x^a$, applicable in the domain $R_{\{++\}}$ for $a \geq 1$ or $a \leq 0$, has its second derivative expressed as $f''(x) = a(a-1)x^{a-2}$. This expression remains non-negative under the specified conditions for a, substantiating the function's convexity within its domain.

4. Conversely, for $f(x) = -x^a$ defined within $R_{\{++\}}$ for $0 \leq a \leq 1$, the second derivative $f''(x) = -a(a-1)x^{a-2}$ indicates a negative value for $a(a-1)$, signaling concavity instead of convexity.

5. With $f(x) = |x|^a$, where $a \geq 1$, the function exhibits convexity for $x \geq 0$ as $f(x) = x^a$ and similarly for $x < 0$ as $f(x) = (-x)^a$. Convexity across the entire domain hinges on the behavior at $x = 0$, which generally aligns with convexity for $a \geq 1$.

6. For $f(x) = x\log(x)$, applicable in $R_{\{++\}}$, the derivatives are $f'(x) = 1 + \log(x)$ and $f''(x) = \frac{1}{x}$. Given that $x > 0$ within $R_{\{++\}}$, $f''(x) > 0$ is upheld, affirming the function's convexity.

c. Strict and strong convexity

Consider a function $f: R^n \rightarrow R$. It is characterized as follows:[11]

- Strictly Convex: For any distinct points $x, y \in R^n$ and any λ in the open interval (0, 1), the function *f* satisfies the inequality $f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda)f(y)$.

[11] Hiriart-Urruty, J.-B., & Lemaréchal, C. (2001). Fundamentals of convex analysis (p. 79). Springer.

- Strongly Convex: There exists a positive scalar $\alpha > 0$ such that the function $f(x) - \alpha||x||^2$ is convex.

Lemma 1: The property of strong convexity implies strict convexity, which in turn implies general convexity. However, the reverse of these implications does not hold.

Proof: The transition from strict convexity to convexity is intuitively straightforward.

To understand why strong convexity leads to strict convexity, consider the strong convexity condition for f, which suggests that for any distinct points $x, y \in R^n$ and any λ in the open interval (0, 1), we have: $f(\lambda x + (1-\lambda)y) - \alpha||\lambda x + (1-\lambda)y||^2 \leq \lambda f(x) + (1-\lambda)f(y) - \lambda\alpha||x||^2 - (1-\lambda)\alpha||y||^2$. The expression $\lambda\alpha||x||^2 + (1-\lambda)\alpha||y||^2 - \alpha||\lambda x + (1-\lambda)y||^2$ is positive for all distinct *x*, *y* and λ in (0, 1), due to the strict convexity of $||x||^2$. This validates the lemma.

The non-equivalence of the converse statements is evident through examples: the function $f(x) = x$ is convex but lacks strict convexity, and the function $f(x) = x^4$ demonstrates strict convexity without being strongly convex.

d. Examples of multivariate convex functions

Affine Functions: An affine function is defined by the equation $f(x) = a^T x + b$, where 'a' is a vector in R^n and $'b'$ is a scalar in *R*. These functions are both convex and concave but not strictly so in either case. This dual property is illustrated by the equation for any λ in [0, 1]:

$$f(\lambda x + (1-\lambda)y) = a^{T(\lambda x + (1-\lambda)y)} + b = \lambda a^T x + (1-\lambda)a^T y + \lambda b + (1-\lambda)b = \lambda f(x) + (1-\lambda)f(y),$$

Demonstrating that affine functions uniquely satisfy both convexity and concavity conditions.

Quadratic Functions: The general form of a quadratic function is $f(x) = x^T Q x + c^T x + d$. The convexity and concavity properties of quadratic functions are determined by the matrix *Q*:

- The function is convex if *Q* is positive semidefinite ($Q >= 0$).

- It is strictly convex if Q is positive definite ($Q > 0$).

- The function is concave if *Q* is negative semidefinite ($Q <= 0$), and strictly concave if *Q* is negative definite ($Q < 0$).

These properties can be established using the second-order conditions for convexity.

Norms: A norm is a function f that satisfies the following conditions for any scalar alpha in *R* and for all vectors *x* and *y*:

1. $f(alpha\, x) = |alpha| f(x)$,

2. $f(x + y) <= f(x) + f(y)$,

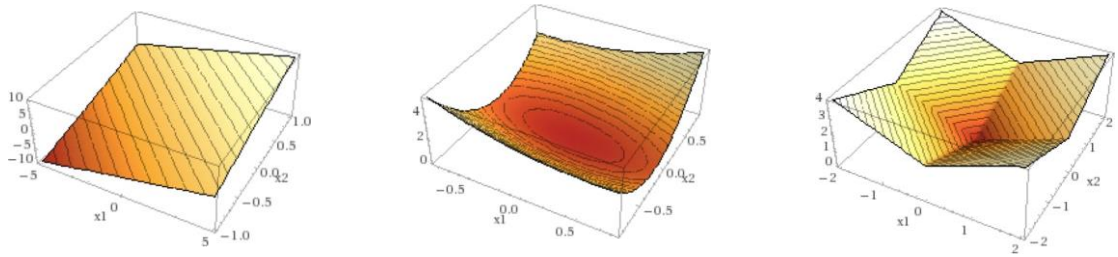
3. $f(x) >= 0$ for all *x*, with $f(x) = 0$ implying that $x = 0$.

These properties ensure that norms measure the size or length of vectors in a consistent and scalable manner.

Proof: For any λ in [0, 1], the following inequality holds:

$$f(\lambda x + (1 - \lambda)y) \leq f(\lambda x) + f\big((1 - \lambda)y\big) = \lambda f(x) + (1 - \lambda)f(y),$$

which is supported by the triangle inequality and the homogeneity property of norms, demonstrating the foundational principles of convexity without relying on the positivity property.



(a) An affine function (b) A quadratic function (c) The 1-norm

Figure 2: Examples of multivariate convex functions

e. Convexity = convexity along all lines

Theorem 1: A function $f: R^n -> R$ is convex if and only if the function g: $R \rightarrow R$, defined by $g(t) = f(x + ty)$, is convex as a univariate function for every x within the domain of *f* and for all *y* in R^n. The domain of *g* includes all *t* for which *x + ty* is within the domain of *f*.[12]

Proof: this proof follows directly from the definition of convexity. Implications and Applications:

- The theorem facilitates numerous fundamental proofs in convex analysis, though it does not significantly ease the task of verifying convexity due to the requirement that the condition be satisfied across an infinite spectrum of lines.
- Within convex optimization, many algorithms focus on iteratively minimizing the function along lines.

This theorem ensures that each such sub-problem is also a convex optimization challenge, thereby reinforcing the applicability of convex optimization strategies.

2.3 First and second order characterizations of convexfunctions:

Theorem 2. Consider a function $f: R^n -> R$ that is twice differentiable over an open domain. The following statements are equivalent:

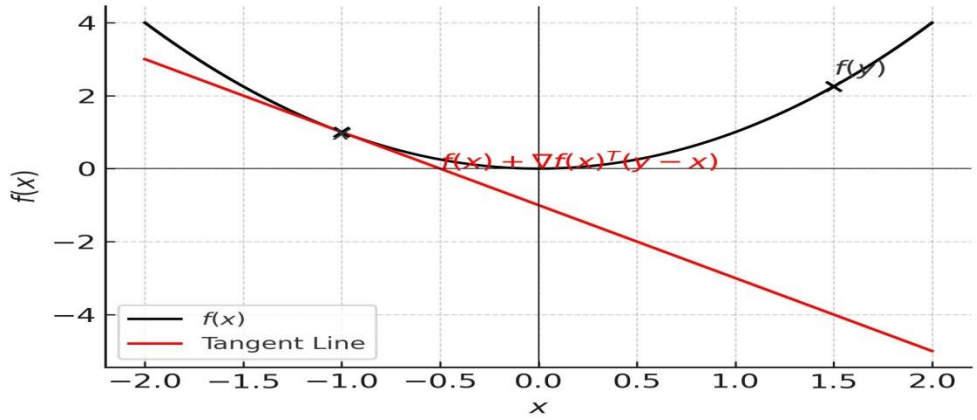
- (i) The function *f* is convex.

- (ii) For all *x*, *y* in the domain of *f*, the inequality $f(y) >= f(x) + \nabla f(x)^{T(y-x)}$ holds. This implies the function *f* satisfies that, for any two points in its domain, the function's value at *y* is at least the first-order Taylor expansion at *x*, evaluated at *y*.

- (iii) The Hessian matrix of $f, \nabla^2 f(x)$, is positive semidefinite $(\nabla^2 f(x) >= 0)$ for all *x* in the domain of *f*.

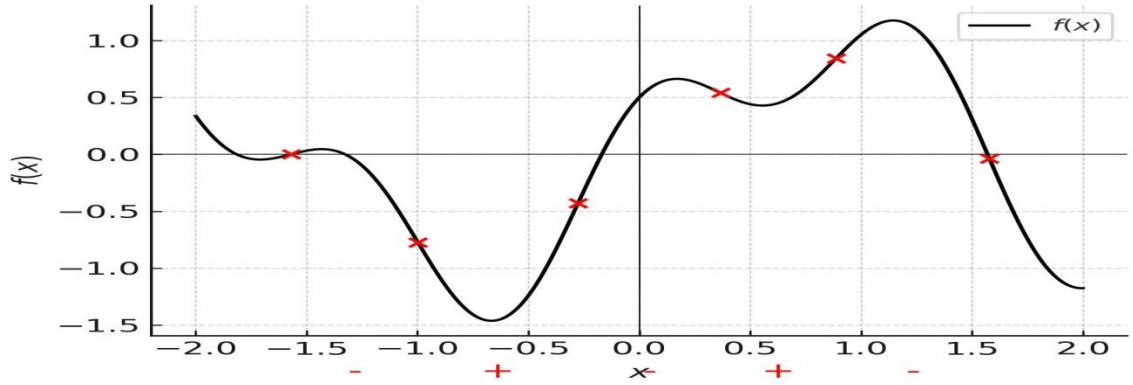
[12] Borwein, J. M., & Lewis, A. S. (2006). Convex analysis and nonlinear optimization (p. 64). Springer.

Interpretation of Condition (ii): The first-order Taylor expansion at any point acts as a global underestimator for the function *f*. This condition underlines that the linear approximation of f at any given point within its domain never overestimates the function's true value at any other point, a key characteristic of convex functions.



Interpretation of Condition (iii): This condition asserts that the function f maintains nonnegative curvature across its entire domain. In the one-dimensional case, this is represented by the inequality $f''(x) >= 0$ for all x in the domain of *f*, signifying that the second derivative of *f* is nonnegative. This nonnegative curvature indicates that the function's graph is consistently "bowed" upwards, a characteristic trait of convex functions.

For functions of higher dimensions, the positive semi-definiteness of the Hessian matrix ($\nabla^2 f(x) >= 0$) ensures that all directional second derivatives are nonnegative. This condition is essential for affirming the convexity of twice differentiable functions, as it verifies the absence of local maxima within the function's domain, aligning with the essential attributes of convex functions.



Proof ([2],[1]):

The detailed proof demonstrates the equivalence of three fundamental conditions for the convexity of a function f: $R^n -> R$ that is twice differentiable over an open domain. Here's the structured summary:

1. (i) => (ii): By the convexity of *f*, we establish that $f(y) - f(x)$ is at least the gradient of *f* at *x*, transposed and multiplied by *y-x*, leveraging the approach of λ towards 0.

2. (ii) => (i): Given the first-order condition for all *x, y* in f's domain, we use a convex combination *z* of *x* and *y* to demonstrate f's convexity, illustrating that *f(z)* is less than or equal to a convex combination of *f(x)* and *f(y)*.

3. (ii) <=> (iii) in dimension 1: We show the necessity and sufficiency of the second derivative's non-negativity (or the Hessian in higher dimensions) for the first-order condition, indicating the nonnegative curvature of *f*.

4. Generalization to higher dimensions: Extends the argument to higher dimensions by proving convexity of *f* along all lines, equivalent to the Hessian's non-negativity across f's domain.

Corollary 1: Global Minima in Convex Optimization
Establishes that for convex and differentiable *f*, any point $\bar{x}$ where the gradient of *f* vanishes marks a global minimum, highlighting the global optimality condition in convex problems.

Remarks:
- The gradient of *f* equals 0 is a necessary and sufficient condition for global optimality in convex problems.
- In non-convex settings, this condition may not ensure even local optimality.
- Convex functions inherently satisfy the local optimality condition where the Hessian of *f* is nonnegative.

2.4 Strict convexity:

a. Characterization of Strict Convexity

Strict Convexity Defined:[13]

A function $f: R^n -> R$ is deemed strictly convex if, for any two distinct points *x* and *y* in its domain, and for every λ in the open interval (0, 1), the following inequality holds:

$$f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda)f(y).$$

This definition naturally implies that a strictly convex function is also convex; however, the reverse does not necessarily apply. For example, the function $f(x) = x$ for x in *R* is convex but not strictly convex.

Second Order Sufficient Condition for Strict Convexity:

The condition that the Hessian matrix of *f* is positive definite ($\nabla^2 f(x) > 0$) for all *x* in a subset **Ω** of its domain suggests that *f* is strictly convex over **Ω**. It's important to note, however, that the converse of this statement may not hold.

First Order Characterization:

A function f is strictly convex on a subset **Ω** subseteq R^n if, and only if, for all pairs of distinct points *x, y* within **Ω**, the inequality:

$$f(y) > f(x) + \nabla f(x)^{\mathrm{T}}(y - x)$$

[13] Borwein, J. M., & Lewis, A. S. (2006). Convex analysis and nonlinear optimization (p. 68). Springer.

is satisfied, indicating a stricter condition than mere convexity.

Characterizations of Strong Convexity:

A function *f* is identified as strongly convex if there exists a constant $m > 0$ such that for all *x, y* in the domain of *f*,

$$f(y) \geq f(x) + \nabla f(x)^{\mathrm{T}}(y - x) + m||y - x||^2$$

or equivalently, if the Hessian matrix satisfies

$$\nabla^2 f(x) \geq mI, \forall x \in dom(f).$$

Application of Strict Convexity:

A principal application of strict convexity lies in its ability to guarantee the uniqueness of the optimal solution in optimization problems, showcasing its significance in mathematical optimization and analysis.

b. Strict Convexity and Uniqueness of Optimal Solutions

Theorem 3. *This theorem posits that for an optimization problem aiming to minimize f(x) subject to x belonging to a convex set* Ω*, if* $f : R^n -> R$ *is strictly convex on* Ω*, then the optimal solution, assuming its existence, is unique.*

Proof: Assume the existence of two distinct optimal solutions *x, y* in R^n, implying both are within Ω and $f(x) = f(y) <= f(z)$ for all *z* in Ω. Considering $z = \frac{(x+y)}{2}$, and given Ω's convexity, *z* also lies within Ω. Due to the strict convexity of *f*, it follows that:

$$f\left(\frac{(x+y)}{2}\right) < \frac{1}{2}f(x) + \frac{1}{2}f(y) = f(x) = f(y),$$

which contradicts the assumption that both x and y are optimal. Therefore, the optimal solution must be unique.

Exercise*: Convexity Analysis*

1. $f(x) = (x1 - 3x2)^2$*:*

- This function is strictly convex. The quadratic term implies a parabolic surface that opens upward, indicative of strict convexity.

2. $f(x) = (x1 - 3x2)^2 + (x1 - 2x2)^2$*:*

- This function is also strictly convex as it is a sum of strictly convex functions. The quadratic terms ensure that any line segment on the function's surface lies above the chord connecting its endpoints.

3. $f(x) = (x1 - 3x2)^2 + (x1 - 2x2)^2 + x3$*:*

- This function remains strictly convex. The addition of x3 to the strictly convex quadratic terms does not affect the strict convexity of the overall function.

4. $f(x) = |x| (for\ x\ in\ R)$*:*

- This function is convex. The absolute value function forms a V-shaped graph, which

satisfies the definition of convexity but not strict convexity.

5. $f(x) = \left||x|\right| (for\ x\ in\ R^n)$*:*

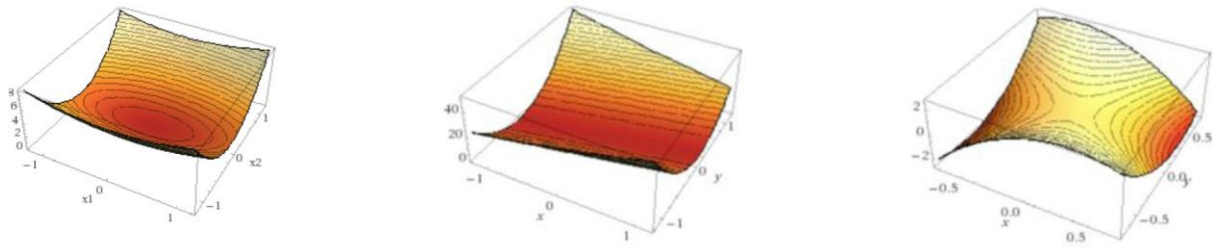
- The norm function is convex. Norms are convex by definition, as they satisfy the triangle inequality and homogeneity. However, they are not strictly convex since the line segment between any two points with the same norm but different directions will lie entirely on the surface of the norm ball.

c. Quadratic functions revisited:

The function *f(x)* is convex if the matrix A is positive semidefinite ($A \geq 0$). This criterion depends solely on the second-order term $x^T Ax$, which relates to the function's curvature. The linear term b and the constant term *c* affect the function's location but not its curvature.[14]

Proofs and Implications:

- ***Non-Convexity with Negative Eigenvalues:*** If *A* is not positive semi-definite (i.e., has a negative eigenvalue), *f(x)* cannot be convex. Demonstrated by considering an eigenvector $\bar{x}$ of *A* with a negative eigenvalue λ, leading to the function being unbounded below as $\alpha \to \infty$, where α is a scalar multiplier of $\bar{x}$.
- ***Strict Convexity with Positive Definite Matrix:*** When *A* is positive definite ($A > 0$), *f(x)* is strictly convex, guaranteeing a unique solution to the optimization problem. This solution is given by $x^* = -\frac{1}{2}A^{(-1)b}$, derived by setting the gradient of *f(x)* to zero.
- ***Convexity with Positive Semidefinite Matrix:*** If A is positive semidefinite ($A \geq 0$), *f(x)* is convex but not necessarily strictly convex. This scenario may result in the optimization problem being unbounded below or having infinitely many solutions, depending on the specific characteristics of *A* and *b*. The condition for a unique solution is more nuanced and depends on whether *b* lies in the range of *A*.



$A \succ 0$ $\quad$ $A \succcurlyeq 0$ (but $A \not\succ 0$) $\quad$ $A \not\succcurlyeq 0$

Figure 3: An illustration of the different possibilities for unconstrained quadratic minimization

3.3.1 Least squares revisited

In the least squares problem, we seek to Minimize the squared difference between Ax and *b*, expressed as Minimize$\left|Ax - b\right|^2$. The unique solution, given by $x = (A^T A)^{-1} A^T b$, hinges on the columns of *A* being linearly independent. This independence ensures A^T , A is positive definite, making the objective function strictly convex. Strict convexity implies that any local Minimize is also a global Minimize, guaranteeing the uniqueness of the solution.

[14] Strang, G. (2009). Introduction to linear algebra (p. 215). Wellesley-Cambridge Press.

The key to this strict convexity is the positive definiteness of A^T A, which stems from the linear independence of A's columns, ensuring the objective's curvature is always positive and thus, strictly convex.

2.5 Optimality conditions for convex optimization:

Theorem. 4. This theorem presents a crucial condition for optimality in convex optimization problems. When the objective function $f : R^n \rightarrow R$ is convex and differentiable, and the feasible set Ω is also convex, a point *x* is optimal (i.e., minimizes f) if and only if it lies within Ω and satisfies the condition:[15]

$$\nabla f(x)^T(y - x) \geq 0, \forall\, y \in \Omega \quad .$$

This condition can be understood as follows:

- **Direction of Increase**: The gradient $\nabla f(x)$ points in the direction of the steepest ascent of f at *x*. The condition $\nabla f(x)^T (y - x) \geq 0$ means that moving from x towards any feasible point y in the direction of $\nabla f(x)$ (or not against it) does not decrease the function value. In other words, any move within Ω from *x* to *y* either increases *f* or leaves it unchanged; implying *x* is at least a local minimum.

- **Hyperplane Support**: The vector $-\nabla f(x)$ (assuming it's nonzero) acts like a supporting hyperplane to the set Ω at point *x*. A hyper-plane is a flat affine subspace of one dimension less than the ambient space (in R^n, it's an *n-1* dimensional space). This hyper-plane 'supports' Ω at *x* in the sense that Ω lies entirely on one side of the hyperplane. The gradient vector $\nabla f(x)$ is perpendicular to this hyperplane, and the condition suggests that *x* is on the boundary of the feasible region where the objective function begins to increase.

In essence, this optimality condition highlights that at the optimal point *x*, any feasible direction does not lead to a decrease in the objective function value. This aligns with the intuitive understanding of a minimum in a convex landscape: at the bottom of a bowl, moving away in any direction only takes you uphill. This theorem formalizes that intuition for convex optimization, indicating that if x satisfies this gradient condition, it's an optimal solution to the problem. (See figure below.)

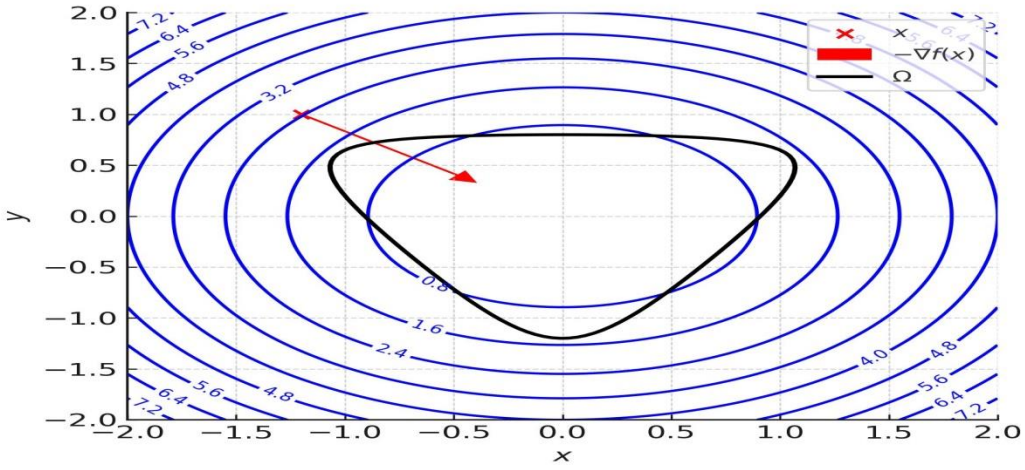


Figure 4: An illustration of the optimality condition for convex optimization

Sufficiency:

The sufficiency of the condition $\nabla f(x)^{\mathrm{T}}(y - x) \geq 0$ for all *y* in Ω in establishing *x* as an optimal point relies on the convexity off. This is demonstrated by combining it with the first-order

[15] Hiriart-Urruty, J.-B., & Lemaréchal, C. (2001). Fundamentals of convex analysis (p. 105). Springer.

characterization of convexity, $f(y) \geq f(x) + \nabla f(x)^{\mathrm{T}}(y - x)$ for all y in Ω, to conclude that $f(y) \geq f(x)$ for all y in Ω, hence proving x is optimal.

Necessity:

The necessity of the condition is independent of the convexity off. If x is optimal but there exists some y in Ω for which $\nabla f(x)(y - x) < 0$, considering $g(\alpha) = f\left(x + \alpha(y - x)\right)$ for α in [0, 1], it's shown that $g'(0) < 0$, leading to $f\left(x + \alpha(y - x)\right) < f(x)$ for some α in the interval (0, δ), contradicting the optimality of x.

Special Case:

- If $\Omega = \mathbb{R}^{\mathrm{n}}$, the condition simplifies to $\nabla f(x) = 0$, the familiar first-order condition for unconstrained optimality. This is because, in the absence of constraints, the only way to satisfy $\nabla f(x)^{\mathrm{T}}(y - x) \geq 0$ for all y is for the gradient itself to be zero.

- If x is in the interior of Ω and is optimal, then $\nabla f(x) = 0$ must hold. This follows from the ability to choose y in the direction of $-\nabla f(x)$ (i.e., $y = x - \alpha \nabla f(x)$ for small α) to contradict the assumption of optimality unless the gradient is zero.

Theorem 5. Let's consider an optimization challenge defined as follows:[16]

Minimize *f(x)*

Subject to the constraint $Ax = b$,

where the function f is convex and A belongs to the space of R^{m*n} matrices. The condition for x in R^n to be deemed optimal for this problem is that it must both satisfy the constraint (be feasible) and there must exist a vector μ in R^m such that the equation $\nabla f(x) = A^T\mu$ holds true. The foundation of this proof is rooted in the nature of convex problems, which dictates that a feasible point x achieves optimality if and only if the transpose of the gradient of f at x, applied to the difference (y - x), is non-negative for all y that meet the condition $Ay = b$.

Every y satisfying $Ay = b$ can be represented as $y = x + v$, where v resides in the null space of A, meaning $Av = 0$. Thus, x is considered optimal only if the transpose of the gradient of f at x, when applied to v, is non-negative for all v that fulfill $Av = 0$. Given that $Av = 0$ also holds true for $A(-v) = 0$, it implies that the transpose of the gradient of f at x, applied to v, must also be non-positive. Consequently, the criterion for optimality stipulates that the transpose of the gradient of f at x, applied to v, equals zero for every v that satisfies $Av = 0$. This implies that the gradient of f at x is orthogonal to the null space of A, which, according to linear algebra, corresponds to the row space of A or, equivalently, the column space of A^T. Therefore, there exists a vector μ in R^m such that $\nabla f(x) = A^T\mu$, completing the proof.

[16] Meyer, C. D. (2001). Matrix analysis and applied linear algebra (p. 318). SIAM.

Chapter III : 1-D Optimization Algorithms

3.1 Introduction

The one-dimensional (1-D) optimization problem refers to an objective func- tion with one variable. In practice, optimization problems with many vari- ables are complex, and rarely does one find a problem with a single variable. However, 1-D optimization algorithms form the basic building blocks for multivariable algorithms. As these algorithms form a subproblem of mul- tivariable optimization problems, numerous methods (or algorithms) have been reported in the literature, each with some unique advantage over the others. These algorithms are classified into gradient-based and non–gradient- based algorithms. Some popular algorithms are discussed in this chapter.
As an example, a single-variable objective function could be:

$$f(x) = 2x^2 - 2x + 8$$

This is an unconstrained optimization problem where x has to be deter- mined, which results in minimization of $f(x)$. If we have to restrict x within $a \leq x \leq b$, where a and b are real numbers, then it becomes a constrained optimization problem. If the function $f(x)$ is either continuously increasing or decreasing between two points a and b, then it is referred to as a *monotonic* function (see Figure 2.1). In a *unimodal* function, the function is monotonic on either side of its minimum point (x^*). The function $f(x) = 2x^2 - 2x + 8$ is plotted in Figure 2.2, in which we observe that $f(x)$ is a unimodal function. Using the property of the unimodal function that it continuously decreases or increases on either side of the minimum point, the single-variable search algorithms can be devised in such a way that they eliminate certain regions of the function where the minimum is not located.
In the next section, a test problem in a solar energy system is defined. Both gradient-based and direct search methods are discussed and tested for this problem. Subsequently, these solution techniques will also be tested on some more standard optimization problems. The performances of these methods are compared toward the end of the chapter.

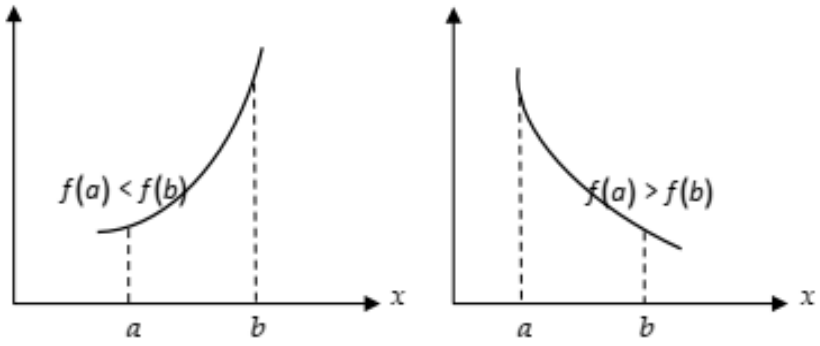


FIGURE 2.1: Monotonic increasing and decreasing functions.

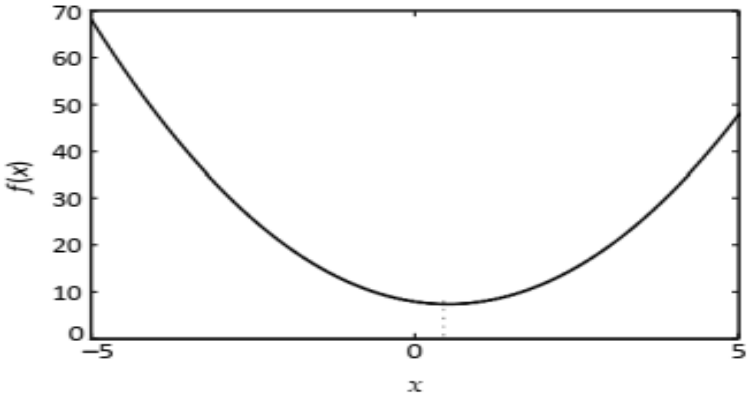


FIGURE 2.2: Unimodal function.

3.2 Test Problem

Before we discuss the optimization algorithms, let us set a problem on which we will be testing these algorithms. The solar energy problem is defined in Problem 8 of Chapter 1. In this cost minimization problem, the cost is a func- tion of the volume of the storage system and the surface area of the collector. The volume and surface area are functions of the design variable tempera- ture *T*. Let us rewrite the cost function in terms of *T* alone as:

$$U = \frac{204{,}165.5}{330-2T} + \frac{10{,}400}{T-20}$$

The variable *T* is restricted between 40°C and 90°C. The function *U* is plot- ted as a function of *T* in Figure 2.4. The minimum occurs at *T** = 55.08 and the minimum value of the function is *U** = 1225.166. Observe from the figure that the cost function is unimodal. A MATLAB® code, *exhaustive.m*, is used to plot the cost function by varying the design variable *T* from 40 to 90 in steps of 0.01. One may ask why, if this method is able to locate the minimum and is also simple, there is a need to discuss other algorithms. It may be noted that the number of function evaluations by this particular method is (90 – 40)/0.01 = 5000. For more complex problems, the time required for the function evaluation is at a premium and it may not be practical to evaluate the function so many times. This necessitates exploring new algorithms that require fewer function evaluations to reach the minimum of any function.
On executing this code, the output obtained is:

Minimum cost = 1225.17

Occurs at T = 55.08

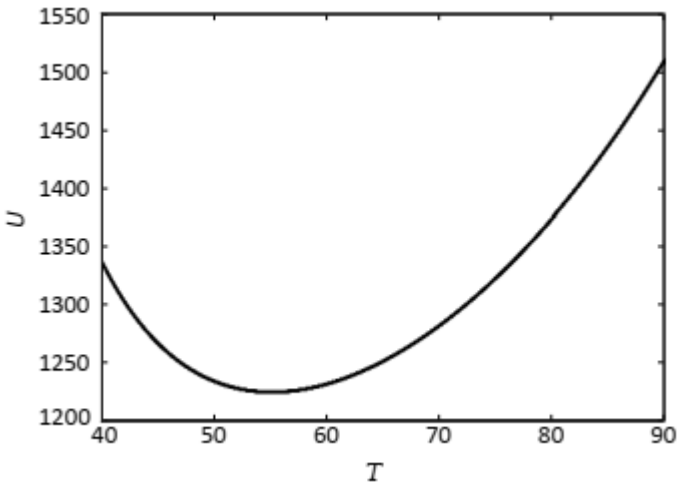


FIGURE 2.4: Cost function for the test problem.

3.3 Solution Techniques

As mentioned previously, the solution techniques for one-dimensional optimization problems can be classified into gradient-based and non–gradient- based algorithms. As the name suggests, gradient-based algorithms require derivative information. These methods find applications to problems in which derivatives can be calculated easily. In the search processes of these algorithms, the derivative of the function is driven to zero. The algorithm is terminated when the derivative of the function is very close to zero and the corresponding x is declared as the point ($\boldsymbol{x}^* = \boldsymbol{x}$) at which minimum of the function occurs. The following gradient-based methods are discussed in this section:

- Bisection method
- Newton–Raphson method
- Secant method
- Cubic polynomial fit

For certain types of optimization problems, the variable x may not be real, but can take only certain discrete values. Recall the pipe size problem dis- cussed in Chapter 1, where pipe size comes in some standard sizes such as 1, 2 inches, and so forth. For such discontinuous functions, gradient information will not be available at all points, and the search algorithm has to proceed using the function evaluations alone to arrive at the minimum of the function. The golden section method is a very effective solution technique for such problems and is discussed later in this section. The golden sec- tion method can also be applied to continuous functions. Some other direct search methods such as dichotomous search, the interval halving method, and the Fibonacci method are also briefly discussed.[17]

3.3.1 Bisection Method

In Chapter 1, we discussed that at the maximum or minimum of a function, $f'(x) = 0$. Because in these problems we are considering a unimodal function of minimization type, the condition that the gradient vanishes at the mini- mum point still holds. The gradient function changes sign near the optimum point. If $f'(x_1)$ and $f'(x_2)$ are the derivatives of a function computed at points x_1 and x_2, then the minimum of the function is located between x_1 and x_2 if:

$$f'(x1)f'(x2) < 0 \tag{2.2}$$

Based on this condition, certain regions of the search space can be elimi- nated. The algorithm is described in Table 2.1.

TABLE 2.1: Algorithm for the Bisection Method

Algorithm for the Bisection Method

Step 1: Given a, b, ε, and Δx

Step 2: Compute $\alpha = \frac{a+b}{2}$, $f'(a)$ and $f'(\alpha)$

If $f'(a)\,f'(\alpha) < 0$

then $b = \alpha$

else $a = \alpha$

If $|a - b| > \varepsilon$

then goto Step 2

else goto Step 3

Step 3: Converged. Print $x^* = a$, $f(x^*) = f(a)$

In this algorithm a and b are the bounds of the function, and Δx is used in the central

[17] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: The art of scientific computing (p. 452). Cambridge University Press.

difference formula for computing the derivative and ε is a small number required for terminating the algorithm when $|a - b| < \varepsilon$. See Figure 2.5, which gives physical insight into this method. The algorithm is coded in MATLAB (*bisection.m*). The objective function is coded in MATLAB file (*func.m*). Users can change the function in this file to minimize another objective function that may be of interest to them. In doing so, they also need to give appropriate bounds for the function, given by *a* and *b* in the main program (*bisection.m*).

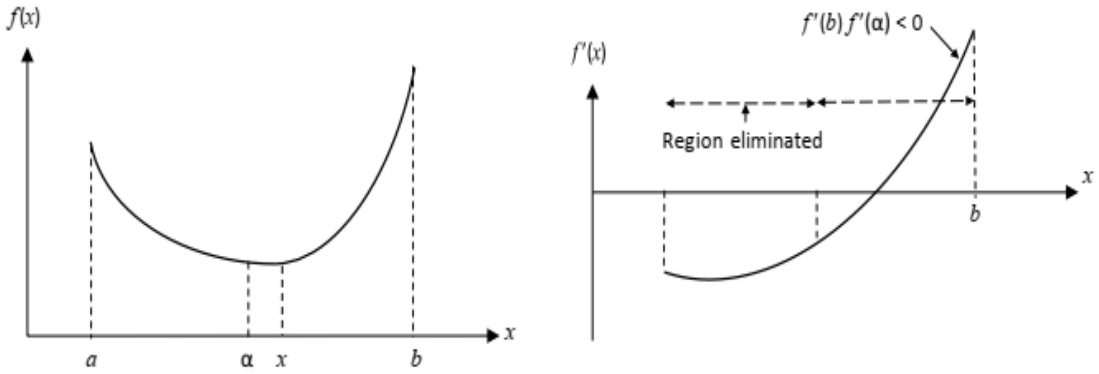


FIGURE 2.5: Bisection method.

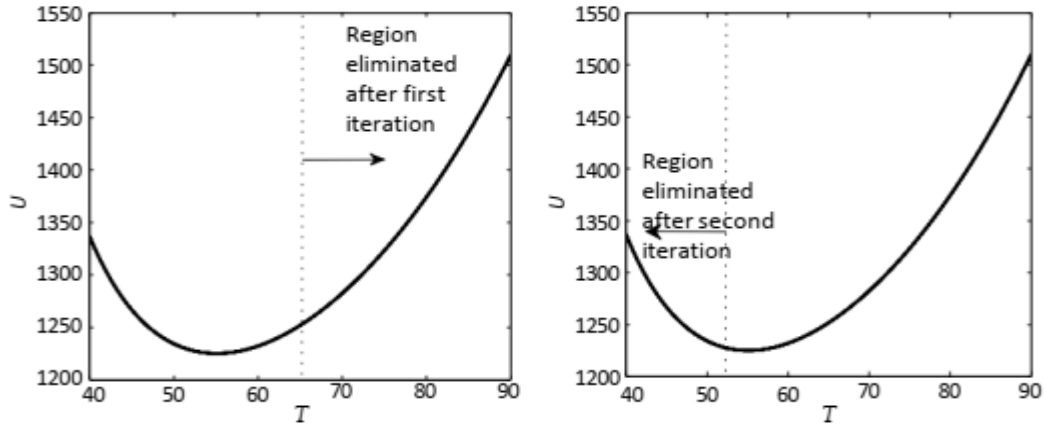


FIGURE 2.6 : Region elimination with iterations (bisection method).

On executing the code for the test problem, the output obtained is:

```
     a              b
-------------------------------------------------------------------------------
   40.000         90.000
   40.000         65.000
   52.500         65.000
   52.500         58.750
   52.500         55.625
   54.063         55.625
   54.844         55.625
   54.844         55.234
   55.039         55.234
   55.039         55.137
   55.039         55.088
   55.063         55.088
   55.076         55.088
-------------------------------------------------------------------------------
x* = 55.082  Minimum = 1225.166
Number of function calls = 52
```

The minimum obtained from this method matches very closely with the exhaustive search method. But the number of function evaluations in the bisection method is only 52 as compared to 5000 in the exhaustive search method. For this test problem, Figure 2.6 shows the regions that are eliminated in the first two iterations.

3.3.2 Newton–Raphson Method

Isaac Newton evaluated the root of an equation using a sequence of polynomials. The method in the present form was given by Joseph Raphson in 1960, with successive approximation to x given in an iteration form. The Newton–Raphson method is a root finding technique in which the root of the equation $f'(x) = 0$ is evaluated. Using the Taylor series, the function $f'(x)$ can be approximated as:

$$f'(xk) + f''(xk)\Delta x \tag{2.3}$$

where the gradient is approximated at point x_k. Setting Equation 2.3 to zero, the next approximation point can then be given as:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{2.4}$$

Figure 2.7 illustrates the steps of this method. The method shows quadratic convergence. That is, if x^* is the root of the equation, then:

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq c, \qquad c \geq 0 \tag{2.5}$$

The Newton–Raphson algorithm is described in Table 2.2.

The algorithm is coded in MATLAB (*newtonraphson.m*). On executing the code, the output obtained is:

x	f(x)	Deriv.	Second deriv.
45.000	1266.690	-9.551	1.449
51.590	1229.340	-2.485	0.800
54.697	1225.214	-0.249	0.650
55.079	1225.166	-0.003	0.636
55.084	1225.166	-0.000	0.635

Number of function calls = 25

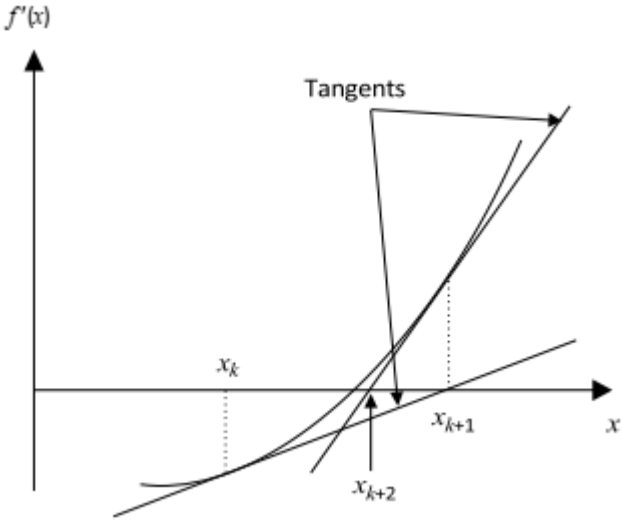


FIGURE 2.7: Newton–Raphson method.

TABLE 2.2: Algorithm for the Newton–Raphson Method

Algorithm for the Newton–Raphson Method

Step 1: Given x and Δx

Step 2: Compute, $f'(x)$ and $f''(x)$

Store, $xprev = x$

Update $x = xprev - \dfrac{f'(x)}{f''(x)}$

If $|x - xprev| > \varepsilon$

then goto Step 2

else goto Step 3

Step 3: Converged. Print $x^* = x$, $f(x^*) = f(x)$, $f'(x^*)$, $f''(x^*)$

The minimum obtained by this method is in agreement with the earlier methods. The number of function evaluations in this method is 25 as com- pared to those in the bisection method, for which 52 function evaluations were required. The Newton–Raphson method has the following disadvantages:

- The convergence is sensitive to the initial guess. For certain initial guesses, the method can also show divergent trends. For example (Dennis and Schnabel 1983), the solution to the function $\tan^{-1} x$ con- verges when the initial guess, $|x| < a$, diverges when $|x| > a$ and cycle indefinitely if the initial guess is taken as $|x| = a$, where $a = 1.3917452002707$.
- The convergence slows down when the gradient value is close to zero.
- The second derivative of the function should exist.

3.3.3 Secant Method

In the bisection method, the sign of the derivative was used to locate zero of $f'(x)$. In the secant method, both the magnitude and the sign of the derivative are used to locate the zero of $f'(x)$. The first step in the secant method is the same as in the bisection method, That is, if $f'(x_1)$ and $f'(x_2)$ are the derivatives of a function computed at point x_1 and x_2, then the minimum of the function is located between x_1 and x_2 if:

$$f'(x1)f'(x2) < 0 \tag{2.6}$$

Further, it is assumed that $f'(x)$ varies linearly between points x_1 and x_2. A secant line is drawn between the two points x_1 and x_2. The point α where the secant line crosses the x-axis is taken as the improved point in the next itera- tion (see Figure 2.8).
One of the points, x_1 or x_2, is then eliminated using the aforementioned derivative condition. Thus, either the (x_1, α) or the (α, x_2) region is retained:

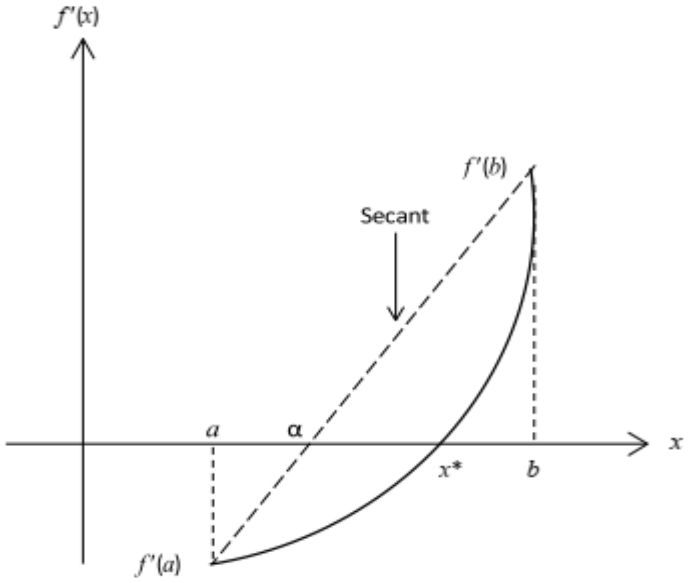


FIGURE 2.8: Secant method.

for the next iteration. The iteration continues until $f'(\alpha)$ is close to zero. The algorithm is coded in MATLAB (*secant.m*) and is described in Table 2.3.
On executing the code for the test problem, the output obtained is:

```
  Alpha           Deriv.
-------------------------------------------------------------------------
  65.000          5.072
  59.832          2.675
  57.436          1.402
  56.265          0.726
  55.680          0.373
  55.385          0.190
  55.237          0.097
  55.161          0.049
  55.123          0.025
  55.104          0.013
  55.094          0.006
  55.089          0.003
  55.086          0.002
-------------------------------------------------------------------------
x* = 55.085  Minimum = 1225.166
Number of function calls = 82
```

The secant method is able to locate the minimum of the function, but with a higher number of function evaluations as compared to other gradient- based methods.

TABLE 2.3: Algorithm for the Secant Method

Algorithm for the Secant Method

Step 1: Given a, b, ε, and Δx, flag = 0;

Step 2: Compute $\alpha = \frac{a+b}{2}$, $f'(a)$ and $f'(\alpha)$

If $f'(a)\,f'(\alpha) < 0$

then $b = \alpha$

set flag = 1(zero is bracketed)

else $a = \alpha$

If flag = 1

then goto Step 3

else goto Step 2

Step 3: Compute $\alpha = x_2 - \frac{f'(x_2)}{\left(f'(x_2) - f'(x_1)\right)/(x_2 - x_1)}$

If $f'(\alpha) > 0$

then $b = \alpha$

else $a = \alpha$

If $|f'(\alpha)| < \varepsilon$

then goto Step 4

else goto Step 3

Step 4: Converged. Print $x^* = \alpha$, $f(x^*) = f(\alpha)$

3.3.4 Cubic Polynomial Fit

In this method, the function $f(x)$ to be minimized is approximated by a cubic polynomial $P(x)$ as:

$$P(x) = a0 + a1x + a2x^2 + a3x^3 \tag{2.7}$$

If the function $f(x)$ is evaluated at four different points, then the polynomial coefficients a_0, a_1, a_2, and a_4 can be evaluated by solving four simultaneous linear equations. Alternatively, if the value of the function and its derivatives are available at two points, the polynomial coefficients can still be evaluated. Once a polynomial is approximated for the function, the minimum point can be evaluated using the polynomial coefficients.

The first step in this search method is to bracket the minimum of the func- tion between two points, x_1 and x_2, such that the following conditions hold:

$$f'(x1)f'(x2) < 0 \tag{2.8}$$

Using the information of $f(x_1)$, $f'(x_1)$, $f(x_2)$, and $f'(x_2)$, the minimum point of the approximating cubic polynomial can be given as:

$$\underline{x} = \begin{cases} x_2 & \text{if } \mu < 0 \\ x_2 - \mu(x_2 - x_1) & \text{if } 0 \le \mu \le 1 \\ x_1 & \text{if } \mu > 0 \end{cases} \tag{2.9}$$

$$\mu = \frac{f'(x_2) + \omega - z}{f'(x_2) - f'(x_1) + 2\omega} \tag{2.10}$$

$$z = \frac{3(f(x_1) - f(x_2))}{x_2 - x_1} + f'(x_1) + f'(x_2) \tag{2.11}$$

$$w = \frac{x_2 - x_1}{|x_2 - x_1|} + \sqrt{z^2 - f'(x_1) f'(x_2)} \tag{2.12}$$

The algorithm for this method is coded in MATLAB (*ubic.m*) and is described in Table 2.4.

On executing the code for the test problem, the output obtained is:

```
     a                b
-------------------------------------
   40.000          65.000
   54.109          65.000
   54.109          55.120
-------------------------------------
x* = 55.084  Minimum = 1225.166
Number of function calls = 28
```

This method is able to capture the minimum point of the function with the number of function evaluations comparable to that in the Newton–Raphson method.

TABLE 2.4: Algorithm for Cubic Polynomial Fit

Algorithm for Cubic Polynomial Fit
Step 1: Given x, ε, and Δx
Step 2: Compute $\alpha = \frac{a+b}{2}$, $f'(a)$ and $f'(\alpha)$
If $f'(a) f'(\alpha) < 0$
then $b = \alpha$
else $a = \alpha$
Step 3: Repeat Step 2 until $f'(a) f'(\alpha) < 0$
Step 4: Using $f(a)$, $f'(a)$, $f(b)$, $f'(b)$, compute μ, z, and w
Step 5: Compute $\bar{x}$
If $\lvert f'(\bar{x}) \rvert < \varepsilon$ goto Step 6
If $f'(a) f'(\bar{x}) < 0$
then $b = \bar{x}$
else $a = \bar{x}$
goto Step 4
Step 6: Converged. Print $x^* = \bar{x}$, $f(x^*) = f(\bar{x})$

3.3.5 Golden Section Method

Two numbers, p and q, are in a golden ratio if:

$$\frac{p+q}{p} = \frac{p}{q} = \tau \tag{2.13}$$

Equation 2.13 can be written as:

$$1 + \frac{q}{p} = \tau \tag{2.14}$$

or

$$1 + \frac{1}{\tau} = \tau \tag{2.15}$$

On solving the quadratic equation:

$$\tau^2 - \tau - 1 = 0 \tag{2.16}$$

we get:

$$\tau = \frac{1+\sqrt{5}}{2} = 1.618033 \tag{2.17}$$

τ is called the golden number, which has a significance in aesthetics (e.g., the Egyptian pyramids).
Gradient information was required in the search methods that were dis- cussed earlier. In the golden section method, the search is refined by elimi- nating certain regions based on function evaluations alone. No gradient computation is required in the golden section method. This method has two significant advantages over other region elimination techniques:

- Only one new function evaluation is required at each step.
- There is a constant reduction factor at each step.

The algorithm is coded in MATLAB (*golden.m*) and is described in Table 2.5.

TABLE 2.5: Algorithm for the Golden Section Method

Algorithm for the Golden Section Method
Step 1: Given x, ε, and τ
Step 2: Compute $\alpha_1 = a(1-\tau) + b\tau$ $\alpha_2 = a\tau + b(1-\tau)$
Step 3: If $f(\alpha_1) > f(\alpha_2)$ then $a = \alpha_1$, $\alpha_1 = \alpha_2$, $\alpha_2 = a\tau + b(1-\tau)$ else $a = \alpha_2$, $\alpha_2 = \alpha_1$, $\alpha_1 = a(1-\tau) + b\tau$
Step 4: Repeat Step 3 until $\lvert f(\alpha_1) - f(\alpha_2)\rvert < \varepsilon$
Step 5: Converged. Print $x^* = \alpha_1$, $f(x^*) = f(\alpha_1)$

On executing the code for the test problem, output obtained is:

```
     a               b
-------------------------------------------------------------------------------
  40.000          90.000
  40.000          70.902
  40.000          59.098
  47.295          59.098
  51.803          59.098
  51.803          56.312
  53.526          56.312
  54.590          56.312
  54.590          55.654
  54.590          55.248
  54.841          55.248
  54.996          55.248
  54.996          55.152
  55.056          55.152
  55.056          55.115
  55.056          55.092
-------------------------------------------------------------------------------
x* = 55.077  Minimum = 1225.166
Number of function calls = 18
```

3.3.6 Other Methods

In addition to the golden section method, there are other direct search methods that can be used to solve the one-dimensional optimization problems, including

- Dichotomous search
- Interval halving method
- Fibonacci method

In the dichotomous search, a function is evaluated at two points, close to the center of the interval of uncertainty. Let these two points be x_a and x_b given by:

$$x_a = \frac{L}{2} - \frac{\delta}{2} \tag{2.17}$$

$$x_b = \frac{L}{2} + \frac{\delta}{2} \tag{2.18}$$

where δ is a small number and L is the region of uncertainty. Depending on the computed value of the function at these points, a certain region is eliminated. In Figure 2.9, the region toward the right-hand side of x_b is elimi- nated. In this method, the region of uncertainty after n function evaluations is given by:

$$\frac{L}{2^{n/2}} + \delta\left(1 - \frac{1}{2^{n/2}}\right) \tag{2.20}$$

In the interval halving method, half of the region of uncertainty is deleted in every iteration. The search space is divided into four equal parts and func- tion evaluation is carried out at x_1, x_2, and x_3. Again, a certain region gets eliminated based on the value of the functions computed at three points. In Figure 2.10, the region toward the right-hand side of x_2 is eliminated. In this method, the region of uncertainty after n function evaluations is given by:

$$\left(\frac{1}{2}\right)^{\frac{n-1}{2}} L \tag{2.21}$$

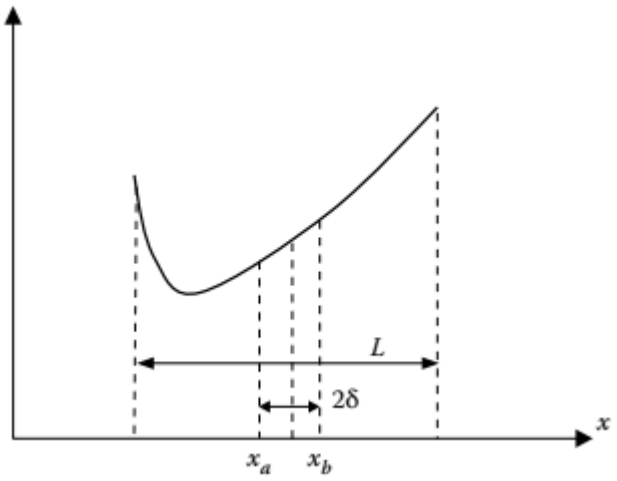
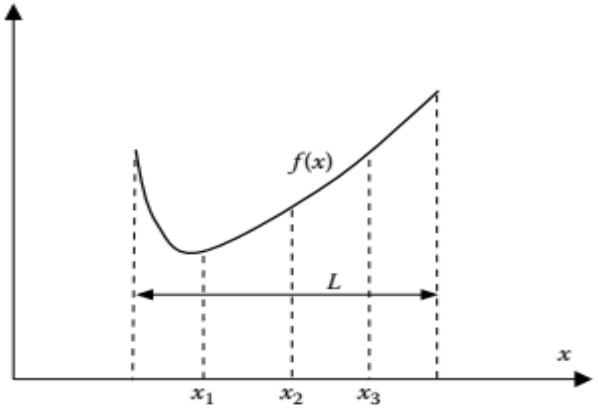


FIGURE 2.9 : Dichotomous search.



A Fibonacci sequence is given by:

$$F_n = F_{n-1} + F_{n-2} \tag{2.22}$$

Where:

$$F_0 = F_1 = 1 \tag{2.23}$$

In the Fibonacci method, the functions are evaluated at points:

$$x_a = a + L^* \tag{2.24}$$

$$x_b = b - L^* \tag{2.25}$$

where [a, b] define the region of uncertainty and L^* is given by:

$$L^* = \frac{F_{n-2}}{F_n} L \tag{2.26}$$

In this method n has to be defined before the start of the algorithm.

3.4 Comparison of Solution Methods

Having defined a number of solution methods to find the minimum of a function, it is natural to ask the question of which solution method to use for a given problem. The answer is quite straightforward: no single method can be used for all types of problems. Different methods may have to be tried for different problems.[18]

Let us evaluate the efficiency of each of the methods for the test case problem that we discussed in an earlier section. One way of defining efficiency of an optimization method could be to show how x approaches $\boldsymbol{x}^*$ with increasing iterations. Because the number of function evaluations in each iteration is dif- ferent for different methods, we can plot $|x - x^*|$ versus number of function evaluations for a meaningful comparison. Figure 2.11 shows this plot for differ- ent solution methods for the solar energy test problem. It is observed from this figure that the cubic polynomial fit and Newton–Raphson approach x^* with 25 number of function evaluations. The bisection and secant methods take a much larger number of function evaluations to reach the minimum. The golden sec- tion method takes a minimum number of function evaluations.

Let us further evaluate these methods for some well-known test problems (Philips et al. 1976; Reklaitis et al. 1983). Table 2.6 summarizes the number of function evaluations required by each of the methods in reaching the minimum of the function. The golden section, cubic polynomial fit, and Newton–Raphson methods perform well for all the test problems except for the function:

$$2(x-3)^2 + e^{0.5x^2} \qquad 0 \le x \le 100$$

[18] Burden, R. L., & Faires, J. D. (2011). Numerical analysis (p. 225). Cengage Learning.

which is highly skewed. The Newton–Raphson method requires a good ini- tial guess for convergence. It takes 275 function evaluations for convergence with an initial guess of $x = 5$. The method takes fewer function evaluations for convergence with $x < 5$. However, the method diverges for $x > 10$. The cubic polynomial fit did not converge for this particular function. The golden

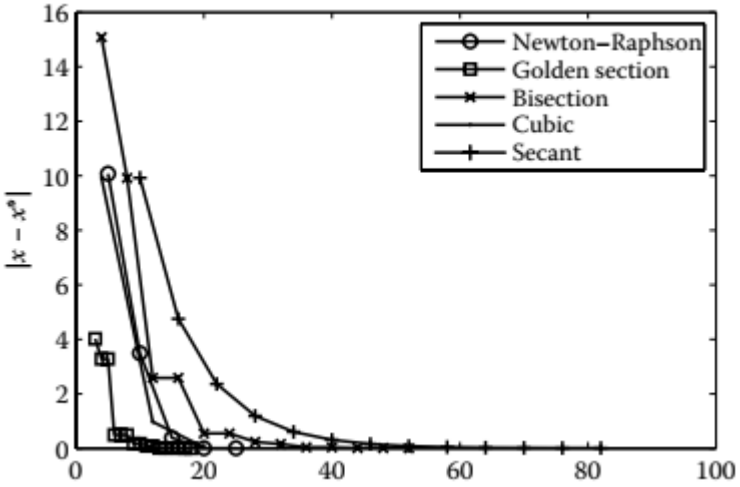


FIGURE 2.11: Comparing different solution methods.

TABLE 2.6: Comparing Different Solution Techniques for Different Problems

Comparing Different Solution Techniques for Different Problems

			Number of Function Evaluations				
Minimize	x^*	$f(x^*)$	**Golden**	**Bisection**	**Cubic**	**Newton**	**Secant**
$3x^4 + (x-1)^2$ $0 \le x \le 4$	0.451	0.426	16	36	36	35	346
$-4x \sin x$ $0 \le x \le \pi$	2.029	–7.28	14	36	24	20	32
$2(x-3)^2 + e^{0.5x^2}$ $0 \le x \le 100$	1.591	7.516	14	36	–	275	–
$3x^2 + \frac{12}{x^3} - 5$ $0.5 \le x \le 2.5$	1.431	5.238	14	32	28	20	604
$2x^2 + \frac{16}{x}$ $1 \le x \le 5$	1.587	15.12	12	36	28	25	70

and bisection methods converged for all the test functions. The solution to these problems is obtained by modifying the *func.m* routine and executing the code for the corresponding method.

❖ Chapter Highlights:

- The one-dimensional (1-D) optimization problems refer to an objective function that has one variable. 1-D optimization algorithms form the basic building blocks for the multivariable algorithms.
- If a function is either continuously increasing or decreasing between two points, then it is referred as a monotonic function.
- In a unimodal function, the function is monotonic on either side of its minimum point.
- The solution techniques for one-dimensional optimization problems can be classified into gradient-based and non–gradient-based algo- rithms. Some popular gradient-based

algorithms are bisection, cubic polynomial fit, secant, and Newton–Raphson methods. The golden sec- tion algorithm does not require derivative information of the function.

• The Newton–Raphson method requires the second derivative of the function, and convergence of this method is strongly dependent on a good initial guess.

• In the bisection method, the sign of the derivative is used to locate the zero of $f'(x)$. In the secant method, both magnitude and sign of the derivative are used to locate the zero of $f'(x)$.

• In the golden section method, the search is refined by eliminating certain regions based on function evaluations only. No gradient computation is required in the golden section method. This method derives its name from the number 1.61803, referred to as the golden number, which has significance in aesthetics.

❖ Formulae Chart:

- Newton–Raphson method:

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

- Secant method:

$$\propto = x_2 - \frac{f'(x_2)}{\left(f'(x_2) - f'(x_1)\right)/(x_2 - x_1)}$$

❖ Problems:

1. Maximizing Lift to Drag Ratio (L/D): For a lifting body, the lift to drag ratio (L/D) is given as a function of the angle of attack (α): $L = -0.004\alpha^2 + 0.16\alpha + 0.11$, where α lies between 0 and 35 degrees. The task is to find the α at which L/D is maximized. Use the Golden Section, Cubic Polynomial Fit, Bisection, and Secant methods to optimize the ratio.

2. Minimization of Various Functions: Apply the following methods to minimize the given functions:

a) $f(x) = 3e^x - x^3 + 5x$, for -3 ≤ x ≤ 3

b) $f(x) = -x^3 + 4x^2 - 3x + 5, for\ -2 \leq x \leq 2$

c) $f(x) = e^{x^2} - 2x^3, for\ -0.5 \leq x \leq 2$

d) $f(x) = 2x^2 + \frac{10}{x}, for\ 0 \leq x \leq 4$

3. Maximization of the Function: $f(x) = (1 - (1 / (1 + x^2))) / (1 + x^2), for\ 0 \leq x \leq 3.$
Find the maximum value of this function using optimization methods.

4. Maximization of the Exponential Function: $f(x) = 5x^2 - e^x, for\ 0 \leq x \leq 5.$ Use optimization techniques to find the maximum value.

5. Minimizing Logarithmic Function: $f(x) = \ln(\cos(x)\cos(x) + 1), for\ 0 \leq x \leq \frac{\pi}{2}.$ Minimize this function using the methods described.

6. Beam Strength Problem: The strength of a beam varies as the product of its breadth and the square of its depth. Find the dimensions of the strongest beam that can be cut from a circular log of diameter 1 meter.

7. Car Petrol Consumption Optimization: A car burns petrol at the rate of ($300 + x^3$) liters per 100 km, where x is the speed in km/h. The task is to find the steady speed that minimizes the total cost of a 600 km trip.

8. Swimmer Optimization Problem: A swimmer in the sea is at a distance of 5 km from the closest point C on the shore on a straight line. The house of the swimmer is on the shore at a distance of 7 km from point C. He can swim at a speed of 2 km/h and run at a speed of 6 km/h. At what spot on the shore should he land so that he reaches his house in the shortest possible time?

9. Aircraft Thrust Optimization: Given various parameters for an aircraft flying at 5 km altitude, find the velocity at which the thrust requirement is minimized using the given equation for thrust: $T = 1/2 * \rho * v^2$.

10. Plotting and Identifying Concave/Convex Regions: For the function $f(x) = x^4 + x^3 - x^2 - 5$, plot the function for $-2 \leq x \leq 2$, identify the concave and convex regions, and determine the local and global minima.

11. Maximizing the Demand Function: The consumer demand function is given by $f(x) = kx - p_1 * x^2 / p_2$, with constants $k = 90$, $p_1 = 10$, and $p_2 = 5$. Maximize this function.

12. Minimizing Function with Constraints: $f(x) = 2(-3) + x / 100, for\ 0.3 \leq x \leq 0.6$. Minimize this function using optimization techniques.

Chapter IV : Unconstrained Optimization

4.1 Introduction

The solution techniques for unconstrained optimization problems with mul- tiple variables are dealt in this chapter. In practice, optimization problems are constrained, and unconstrained optimization problems are few. One example of an unconstrained optimization problem is data fitting, where one fits a curve on the measured data. However, the algorithms presented in this chapter can be used to solve constrained optimization problems as well. This is done by suitably modifying the objective function, which includes a penalty term in case constraints are violated.

The solution methods for unconstrained optimization problems can be broadly classified into gradient-based and non–gradient-based search methods. As the name suggests, gradient-based methods require gradi- ent information in determining the search direction. The gradient-based methods discussed in this chapter are steepest descent, Davidon–Fletcher– Powell (DFP), Broyden–Fletcher–Goldfarb–Shanno (BFGS), Newton, and Levenberg–Marquardt methods. The search direction computed by these methods uses the gradient information, Hessian information, or a combina- tion of these two. Some methods also make an approximation of the Hessian matrix. Once the search direction is identified, one needs to evaluate how much to move in that direction so as to minimize the function. This is a one-dimensional problem. We will be using the golden section method, as discussed in Chapter 3, for solving the one-dimensional problem. The non– gradient-based method does not require derivatives or second derivative information in finding the search direction. The search direction is guided by the function evaluations as well as the search directions computed from earlier iterations. Powell's conjugate direction method, a non–gradient-based method, is elaborated in this chapter as it is much superior (shows quadratic convergence) to other non-gradient methods such as simplex and pattern search methods. The simplex method (Nelder–Mead algorithm) is also discussed in Section 3.4.9 on the direct search method. In the last section, Powell's method is used to solve a complicated motion design problem of a robot.

For a single-variable function, it was discussed earlier that the derivative of the function vanishes at the optimum and the second derivative of the function is greater than zero at the minimum of the function. The same can be extended to a multivariable function. The necessary conditions for x^* to be a minimum are that:

$$\nabla f(\boldsymbol{x}^*) = 0 \tag{3.1}$$

and $\boldsymbol{x}^T\ \boldsymbol{H}\boldsymbol{x}$ is positive definite ($\boldsymbol{x}^T\ \boldsymbol{H}\boldsymbol{x} > 0$). To ensure this, eigenvalues of $\boldsymbol{H}$ are to be positive. Consider a two-variable function

$$f(x) = x_1^2 + x_2^2 - 2x_1 \tag{3.2}$$

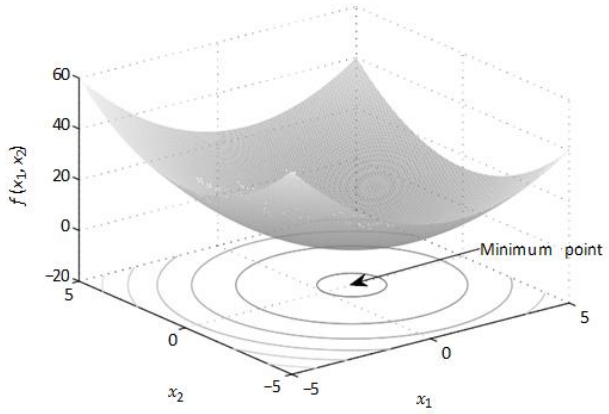


FIGURE 3.2: Surface-contour plot of the function.

The gradient is:

$$\nabla f(x) = \begin{bmatrix} 2x_1 - 2 \\ 2x_2 \end{bmatrix} \tag{3.3}$$

Equating the gradient to zero, the optimum is at (1, 0). For this function $\boldsymbol{x}^T \boldsymbol{H}\boldsymbol{x} > 0$. Hence, the point (1, 0) is the minimum of $f(\boldsymbol{x})$. The surface-contour plot of this function is shown in Figure 3.2.
For a two-variable function:

$$f(x) = x_1^2 - x_2^2 \tag{3.4}$$

the optimum is at (0, 0) from the first-order condition. Checking the second- order condition, we find that $\boldsymbol{x}^T \boldsymbol{H}\boldsymbol{x} = 0$. Therefore, the point (0, 0) represents saddle point (see Figure 3.3).

4.2 Unidirectional Search

The unidirectional search refers to minimizing the value of a multivariable function along a specified direction. For example, if x_i is the initial starting point of the design variables for minimizing a multivariable function and S_i:

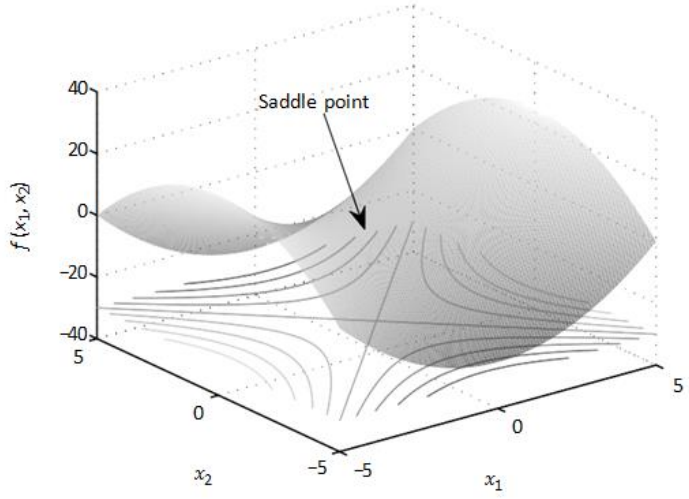


FIGURE 3.3: Surface-contour plot of the function with saddle point.

is the search direction, then we need to determine a scalar quantity α such that the function:

$$f(\alpha) = \boldsymbol{x}_i + \alpha \boldsymbol{S}_i \tag{3.5}$$

is minimized. The value of α at which this function reaches a minimum is given by α*. This is a one-dimensional optimization problem and we can use the golden section technique to minimize this function. The golden section method is modified to handle multivariable functions and the MATLAB® code *golden_funct1.m* is given.

Let us perform a unidirectional search on the Rosenbrock function[19] given by:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{3.6}$$

with different starting values of x and with different search directions. The results are summarized in Table 3.1. It is observed from this table that:

TABLE 3.1: Unidirectional Search for a Multivariable Function

x_i	$f(x_i)$	S_i	α^*	$f(\alpha^*)$
(3, 0.5)	7229	(2, 1)	−1.3731	88.45
(3, 0.5)	7229	(2, 3)	−1.1249	1181.7
(1, 1)	0	(2, 2)	0	0
(2, 2)	401	(1, 1)	−1	0

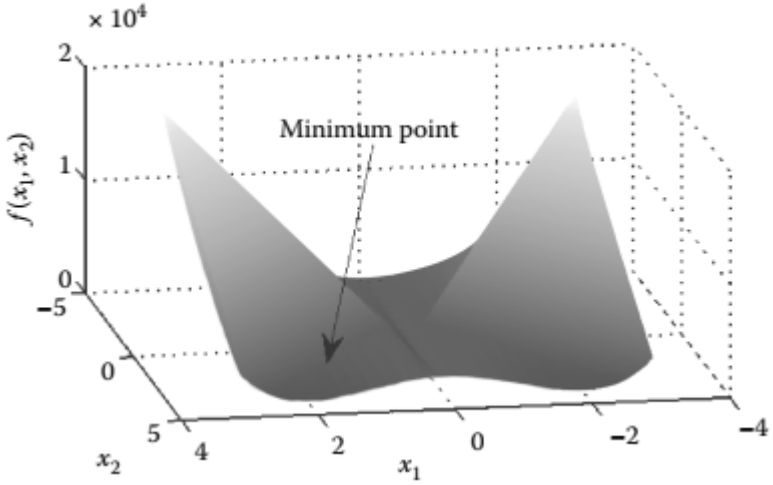


FIGURE 3.4: Rosenbrock function.

performing a linear search in the direction (2, 1) from the starting point (3, 0.5) results in $f(\alpha^*) = 88.45$ as compared to initial function value of 7229. This can be easily shown on the MATLAB command prompt as:

```
>> x = [3 0.5];
>> search = [2 1];
>> [alpha1,falpha1] = golden_funct1(x,search)

alpha1 =
 -1.3731

falpha1 =
 88.4501
```

The function has to be appropriately coded in *func_multivar.m*. Note that this function has a minimum at (1, 1) and the minimum value of the function is zero. If we are at minimum

[19] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: The art of scientific computing (p. 476). Cambridge University Press.

point, then any search direction should not improve the function value. It is the reason why search in the direction (2, 2) from the point (1, 1) results in $f(\alpha^*) = 0$ with $\alpha^* = 0$. Similarly, search in the direction (1, 1) from the point (2, 2) results in $f(\alpha^*) = 0$ with $\alpha^* = -1$. This function is plotted in Figure 3.4 and is constructed by executing the MATLAB code (*rosenbrock.m*).

4.3 Test Problem

Let us define a spring system as a test problem on which we will apply multi- variable optimization algorithms such as the steepest descent, DFP, BFGS, Newton, and Levenberg–Marquardt methods.

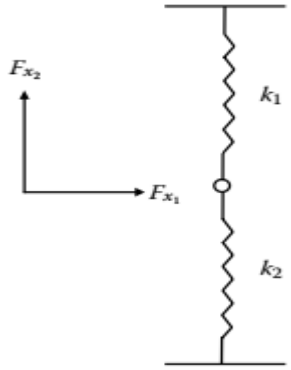


FIGURE 3.5: Spring system.

Consider two springs of unit length and with stiffness k_1 and k_2, joined at the origin. The other two ends of the springs are fixed on a wall (see Figure 3.5). On applying a force, the spring system will deflect to an equilibrium position, which we are inter- ested in determining. The potential of the spring system is given by:

$$U = k_1\left(\sqrt{x_1^2 + (x_2+1)^2} - 1\right)^2 + k_2\left(\sqrt{x_1^2 + (x_2-1)^2} - 1\right)^2 - (F_{x_1}x_1 + F_{x_2}x_2) \quad (3.7)$$

Where: $\left(F_{x_1}, F_{x_2}\right)$ is the force applied at the origin due to which it moves to a posi- tion (x_1, x_2). Assuming k_1 = 100 N/m, k_2 = 90 N/m, and $\left(F_{x_1}, F_{x_2}\right)$ = (20, 40), our aim is to evaluate (x_1, x_2) such that U is minimized.

A MATLAB code (*springsystem.m*) is used to find the minimum of the potential function by varying the design variables from –1 to 1 in steps of 0.01. On executing this code, the output obtained is:

```
Minimum Potential  = -9.6547

occurs at x1,x2    = 0.5000      0.1200
```

4.4 Solution Techniques

Similar to 1-D optimization algorithms, solution techniques for multivariable, unconstrained optimization problems can be grouped into gradient and non–gradient-based methods.[20] Gradient-based methods require derivative information of the function in constituting a search. The first and second derivatives can be computed using the central difference formula as given below:

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i+\Delta x_i) - f(x_i-\Delta x_i)}{2\Delta x_i} \quad (3.8)$$

[20] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: The art of scientific computing (p. 498). Cambridge University Press.

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{f(x_i+\Delta x_i)-2f(x_i)+f(x_i-\Delta x_i)}{\Delta x_i^2} \tag{3.9}$$

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \left[f\left(x_i + \Delta x_i, x_j + \Delta x_j\right) - f\left(x_i + \Delta x_i, x_j - \Delta x_j\right) - f\left(x_i - \Delta x_i, x_j + \Delta x_j\right) + f\left(x_i - \Delta x_i, x_j - \Delta x_j\right)\right]/(4\Delta x_i \Delta x_j) \tag{3.10}$$

The computation of first derivative requires two function evaluations with respect to each variable. So for an n variable problem, $2n$ function evaluations are required for computing the gradient vector. The computation of the Hessian matrix requires $O(n^2)$ function evaluations. Note that in the Hessian matrix:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i} \tag{3.11}$$

Alternatively, one can also compute the derivative of a function using com- plex variables as:

$$f'(x) = \frac{Imaginary[f(x+i\Delta x)/\Delta x]}{\Delta x} \tag{3.12}$$

The gradient-based methods such as steepest descent, DFP, BFGS, Newton, and Levenberg–Marquardt methods are discussed next followed by Powell's conjugate direction method, which is a direct search method. The efficiency of solution methods can be gauged by three criteria:

- Number of function evaluations.
- Computational time.
- Rate of convergence. By this we mean how fast the sequence $\boldsymbol{x}_i$, $\boldsymbol{x}_{i+1}$,… converges to $\boldsymbol{x}^*$. The rate of convergence is given by the parameter n in the equation.

$$\frac{\|\boldsymbol{x_{i+1}} - \boldsymbol{x}^*\|}{\|\boldsymbol{x_i} - \boldsymbol{x}^*\|^{\boldsymbol{n}}} \quad , \qquad \boldsymbol{c} \geq \boldsymbol{0} \quad , \quad \boldsymbol{n} \geq \boldsymbol{0} \tag{3.13}$$

- For $n = 1$ and $0 \leq c \leq 1$ the method is said to have *linear convergence*. For $n = 2$, the method is said to have *quadratic convergence*. When the rate of convergence is higher, the optimization method is better. A method is said to have *superlinear convergence* if:

$$\lim_{i\to\infty} \left(\frac{\|x_{i+1}-x^*\|}{\|x_i-x^*\|^n}\right) \leq c, \quad c \geq 0, \quad n \geq 0 \tag{3.14}$$

3.4.1 Steepest Descent Method

The search direction $\boldsymbol{S}_i$ that reduces the function value is a descent direction. It was discussed earlier that along the gradient direction, there is the maxi- mum change in the function value. Thus, along the negative gradient direc- tion, the function value decreases the most. The negative gradient direction is called the steepest descent direction. That is:

$$\boldsymbol{S_i} = -\nabla f(\boldsymbol{x_i}) \tag{3.15}$$

In successive iterations, the design variables can be updated using the equation

$$\boldsymbol{x_i} + 1 \;=\; \boldsymbol{x_i} \;-\; \alpha \nabla f(\boldsymbol{x_i}) \tag{3.16}$$

where α is a positive scalar parameter that can be determined using the line search algorithm such as the golden section method.

The steepest descent method ensures a reduction in the function value at every iteration. If the starting point is far away from the minimum, the gra- dient will be higher and the function reduction will be maximized in each iteration. Because the gradient value of the function changes and decreases to a small value near the optimum, the function reduction is uneven and the method becomes sluggish (slow convergence) near the minimum. The method can therefore be utilized as a starter for other gradient-based algo- rithms. The algorithm for the steepest descent method is described in Table 3.2 and a MATLAB code of its implementation is given in *steep_des.m*.

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2), following output is produced for the test problem. After the first iteration, the function value decreases from 1452.2619 to –2.704. Notice from the output that as the gradi- ent value decreases, the reduction in function value at each iteration also decreases. The steepest descent algorithm converges to the minimum of the test problem in 15 iterations.

TABLE 3.2: Algorithm for the Steepest Descent Method

Algorithm for the Steepest Descent Method
Step 1: Given x_i (starting value of design variable)
ε_1 (tolerance of function value from previous iteration)
ε_2 (tolerance on gradient value)
Δx (required for gradient computation)
Step 2: Compute $f(x_i)$ and $\nabla f(x_i)$ (function and gradient vector)
$S_i = -\nabla f(x_i)$ (search direction)
Minimize $f(x_{i+1})$ and determine α (use golden section method)
$x_{i+1} = x_i + \alpha S_i$ (update the design vector)
If $\lvert f(x_{i+1}) - f(x_i)\rvert > \varepsilon_1$ or $\lVert \nabla f(x_i)\rVert > \varepsilon_2$
then goto Step 2
else goto Step 3
Step 3: Converged. Print $x^* = x_{i+1}$, $f(x^*) = f(x_{i+1})$

Observe the sluggishness of the algorithm as it approaches the minimum point. The convergence history is shown pictorially in Figure 3.6 along with the function contours of the test problem. The function contours can be plotted using the MATLAB code *contour_testproblem.m*.

Initial function value = 1452.2619

No.	x-vector		f(x)	Deriv.
1	0.095	0.023	-2.704	1006.074
2	0.170	0.141	-5.278	37.036
3	0.318	0.048	-7.369	23.451
4	0.375	0.138	-8.773	26.656
5	0.448	0.092	-9.375	14.021
6	0.470	0.127	-9.583	10.071
7	0.491	0.114	-9.639	4.403
8	0.497	0.123	-9.652	2.509
9	0.501	0.120	-9.655	1.050
10	0.503	0.122	-9.656	0.554
11	0.504	0.122	-9.656	0.236
12	0.504	0.122	-9.656	0.125
13	0.504	0.122	-9.656	0.047
14	0.504	0.122	-9.656	0.027
15	0.504	0.122	-9.656	0.016

3.4.2 Newton's Method

The search direction in this method is based on the first and second deriva- tive information

and is given by

$$\boldsymbol{S}i \;=\; -[\boldsymbol{H}]^{-1}\nabla f(\boldsymbol{x}i) \tag{3.17}$$

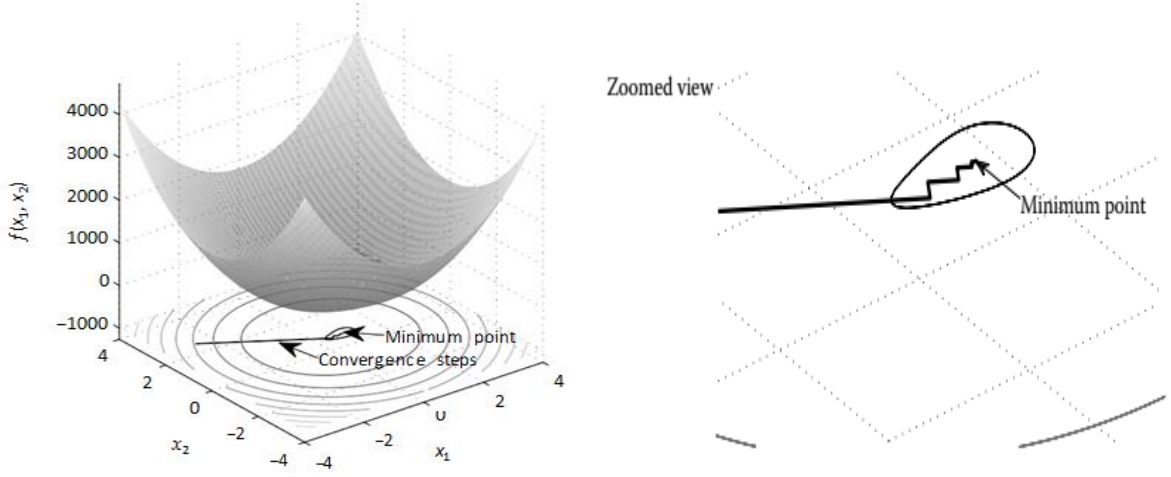


FIGURE 3.6: Function contours of the test problem and convergence history.

where [$\boldsymbol{H}$] is the Hessian matrix. If this matrix is positive definite, then $\boldsymbol{S}_i$ will be a descent direction. The same can be assumed true near the vicinity of the optimum point. However, if the initial starting point is far away from the optimum, the search direction may not always be descent. Often a restart is required with a different starting point to avoid this difficulty. Though the Newton's method is known for converging in a single iteration for a qua- dratic function, seldom do we find functions in practical problems that are quadratic. However, Newton's method is often used as a hybrid method in conjunction with other methods.

The algorithm for the Newton's method is described in Table 3.3 and a MATLAB code of its implementation is given in *newton.m*. A MATLAB code that computes Hessian matrix is given in *hessian.m*.

TABLE 3.3: Algorithm for Newton's Method

Step 1: Given x_i (starting value of design variable)
ε_1 (tolerance of function value from previous iteration)
ε_2 (tolerance on gradient value)
Δx (required for gradient computation)
Step 2: Compute $f(x_i)$, $\nabla f(x_i)$, and [H] (function, gradient, and Hessian)
$S_i = -[H]^{-1}\nabla f(x_i)$ (search direction)
$x_{i+1} = x_i + S_i$ (update the design vector)
If $\lvert f(x_{i+1}) - f(x_i)\rvert > \varepsilon_1$ or $\lVert\nabla f(x_i)\rVert > \varepsilon_2$
then goto Step 2
else goto Step 3
Step 3: Converged. Print $x^* = x_{i+1}$, $f(x^*) = f(x_{i+1})$

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2), the following output is displayed in the command window for the test problem. Note that in some iteration, the search direction is not a descent as the function value increases instead of monotonically decreasing. The method, however, con- verges to the minimum point.

Initial function value = 1452.2619

No.	x-vector		f(x)	Deriv.
1	-0.754	0.524	44.244	1006.074
2	-0.362	-0.010	8.398	116.281
3	0.094	0.125	-3.920	50.597
4	11.775	0.324	22007.14	21.420
5	1.042	0.093	14.533	4076.916
6	0.640	0.142	-8.479	102.745
7	0.524	0.122	-9.635	18.199
8	0.505	0.122	-9.656	2.213
9	0.504	0.122	-9.656	0.059
10	0.504	0.122	-9.656	0.000

Let us restart the method with $\boldsymbol{x}$ as (1, 1). The output is given below. If the starting value is closer to the minimum, the function value reduces mono- tonically in all the iterations and eventually converges to the minimum.

Initial function value = 92.7864

No.	x-vector		f(x)	Deriv.
1	0.818	0.041	-1.428	202.492
2	0.569	0.138	-9.386	56.085
3	0.510	0.122	-9.655	8.516
4	0.504	0.122	-9.656	0.602
5	0.504	0.122	-9.656	0.004

TABLE 3.4: Algorithm for Modified Newton's Method

Step 1: Given x_i (starting value of design variable)
ε_1 (tolerance of function value from previous iteration)
ε_2 (tolerance on gradient value)
Δx (required for gradient computation)

Step 2: Compute $f(x_i)$, $\nabla f(x_i)$, and $[H]$ (function, gradient, and Hessian)
$S_i = -[H]^{-1}\nabla f(x_i)$ (search direction)
Minimize $f(x_{i+1})$ and determine α (use golden section method)
$x_{i+1} = x_i + \alpha S_i$ (update the design vector)
If $\left|f(x_{i+1}) - f(x_i)\right| > \varepsilon_1$ or $\|\nabla f(x_i)\| > \varepsilon_2$
then goto Step 2
else goto Step 3

Step 3: Converged. Print $x^* = x_{i+1}$, $f(x^*) = f(x_{i+1})$

3.4.3 Modified Newton's Method

The method is similar to Newton's method with a modification that a unidi- rectional search is performed in the search direction $\boldsymbol{S}_i$ of the Newton method. The algorithm for the modified Newton method is described in Table 3.4 and a MATLAB code of its implementation is given in *modified_newton.m.*

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2), the following output is displayed in the command window for the test problem. For the same starting point, the modified Newton's method converges to the mini- mum point in just six iterations as compared to Newton's method, which converges in ten iterations.

Initial function value = 1452.2619

No.	x-vector		f(x)	Deriv.
1	0.006	0.025	-1.010	1006.074
2	0.498	0.026	-8.227	36.392
3	0.496	0.121	-9.653	29.839
4	0.504	0.122	-9.656	0.873
5	0.504	0.122	-9.656	0.018
6	0.504	0.122	-9.656	0.003

3.4.4 Levenberg–Marquardt Method

The advantage of the steepest descent method is that it reaches closer to the minimum of the function in a few iterations even when the starting guess is far away from the optimum. However, the method shows sluggishness near the optimum point. On the contrary, Newton's method shows a faster convergence if the starting guess is close to the minimum point. Newton's method may not converge if the starting point is far away from the optimum point.

The Levenberg–Marquardt method is a kind of hybrid method that com- bines the strength of both the steepest descent and Newton's methods. The search direction in this method is given by:

$$\boldsymbol{S_i} = -[\boldsymbol{H} + \lambda \boldsymbol{I}]^{-1} \nabla f(\boldsymbol{x_i}) \tag{3.18}$$

where $\boldsymbol{I}$ is an identity matrix and λ is a scalar that is set to a high value at the start of the algorithm. The value of λ is altered during every iteration depending on whether the function value is decreasing or not. If the function value decreases in the iteration, λ it decreases by a factor (less weightage on steepest descent direction). On the other hand, if the function value increases in the iteration, λ it increases by a factor (more weightage on steepest descent direction). The algorithm for the Levenberg–Marquardt method is described in Table 3.5 and a MATLAB code of its implementation is given in *levenbergmarquardt.m*.

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2), following output is displayed at the command window for the test problem.

Initial function value = 1452.2619

No.	x-vector		f(x)	Deriv.
1	-2.384	1.604	815.738	1006.074
2	-1.680	1.139	325.925	733.709
3	-1.104	0.705	102.059	429.113
4	-0.740	0.327	28.673	201.554
5	-0.444	0.133	8.324	86.884
6	-0.164	0.105	1.186	34.005
7	0.546	0.091	-9.390	20.542
8	0.508	0.122	-9.655	11.361
9	0.505	0.122	-9.656	0.409
10	0.504	0.122	-9.656	0.016

TABLE 3.5: Algorithm for the Levenberg–Marquardt Method

Step 1: Given x_i (starting value of design variable)
- ε_1 (tolerance of function value from previous iteration)
- ε_2 (tolerance on gradient value)
- Δx (required for gradient computation)

Step 2: Compute $f(x_i)$, $\nabla f(x_i)$, and $[H]$ (function, gradient, and Hessian)
- $S_i = -[H + \lambda I]^{-1} \nabla f(x_i)$ (search direction)
- $x_{i+1} = x_i + S_i$ (update the design vector)
- If $f(x_{i+1}) < f(x_i)$
  - then change the value of λ as λ/2
  - else change the value of λ as 2λ
- If $\left|f(x_{i+1}) - f(x_i)\right| > \varepsilon_1$ or $\|\nabla f(x_i)\| > \varepsilon_2$
  - then goto Step 2
  - else goto Step 3

Step 3: Converged. Print $x^* = x_{i+1}$, $f(x^*) = f(x_{i+1})$

3.4.5 Fletcher–Reeves Conjugate Gradient Method

The Levenberg–Marquardt method uses the strengths of both steepest descent and Newton's method for accelerating the convergence to reach the minimum of a function. The method is a second-order method, as it requires computation of the Hessian matrix. On the other hand, the conjugate gra- dient method is a first-order method, but shows the property of quadratic convergence and thus has a significant advantage over the second-order methods. Two directions, $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$, are said to be conjugate if:

$$S_1^T H S_2 = 0 \tag{3.19}$$

where $\boldsymbol{H}$ is a symmetric matrix. For example, orthogonal directions are con- jugate directions. In Figure 3.7, starting from point x_{1a}, the search direction S_1 results in the minimum point $x_a{}^*$. Similarly, starting from point x_{1b}, the search direction S_1 results in the minimum point $x^*{}_b$. The line joining $x_a{}^*$ and $x^*{}_b$ is the search direction S_2. Then, S_1 and S_2 are conjugate directions.

The steepest descent method was modified by Fletcher and Reeves in the conjugate gradient method. Starting with the search direction

$$\boldsymbol{S_1} = -\nabla f(\boldsymbol{x_1}) \tag{3.20}$$

the subsequent search direction is taken as a linear combination of $\boldsymbol{S}_1$ and $-\nabla f(\boldsymbol{x}_2)$. That is,

$$\boldsymbol{S_2} = -\nabla f(\boldsymbol{x_2}) + \alpha \boldsymbol{S_1} \tag{3.21}$$

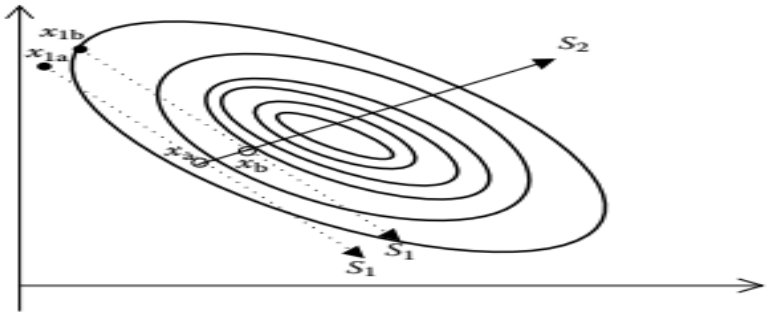


FIGURE 3.7: Conjugate directions.

Using the property $S_1^T H S_2 = 0$ of conjugate directions, α can be evaluated as:

$$\propto = \frac{\|\nabla f(x_{i+1})\|^2}{\|\nabla f(x_i)\|^2} \tag{3.22}$$

Starting with $\boldsymbol{S}1 = -\nabla f(\boldsymbol{x}1)$, the search direction in every iteration is calculated using the equation:

$$S_{i+1} = -\nabla f(x_i) + \frac{\|\nabla f(x_{i+1})\|^2}{\|\nabla f(x_i)\|^2} S_i \tag{3.23}$$

The algorithm for the conjugate gradient method is described in Table 3.6 and a MATLAB code of its implementation is given in *conjugate.m*.

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2), the following output is displayed at the command window in the test problem. The effi- ciency of conjugate gradient method can be seen from Figure 3.8, where it is compared with the first-order, steepest descent method.

TABLE 3.6: Algorithm for Fletcher–Reeves's Conjugate Gradient Method

Step 1: Given x_i (starting value of design variable)
ε_1 (tolerance of function value from previous iteration)
ε_2 (tolerance on gradient value)
Δx (required for gradient computation)

Step 2: Compute $f(x_i)$ $\nabla f(x_i)$ (function and gradient)
$S_i = -\nabla f(x_i)$ (search direction)
Minimize $f(x_{i+1})$ and determine α (use the golden section method)
$x_{i+1} = x_i + \alpha S_i$ (update the design vector)

Step 3: $$S_{i+1} = -\nabla f(x_{i+1}) + \frac{\|\nabla f(x_{i+1})\|^2}{\|\nabla f(x_i)\|^2} S_i$$
Minimize $f(x_{i+2})$ and determine α (use the golden section method)
$x_{i+2} = x_{i+1} + \alpha S_{i+1}$
Minimize $f(x_{i+2})$ and determine α (use the golden section method)
If $|f(x_{i+2}) - f(x_{i+1})| > \varepsilon_1$ or $\|\nabla f(x_{i+1})\| > \varepsilon_2$
then goto Step 3
else goto Step 4

Step 4: Converged. Print $x^* = x_{i+2}$, $f(x^*) = f(x_{i+2})$

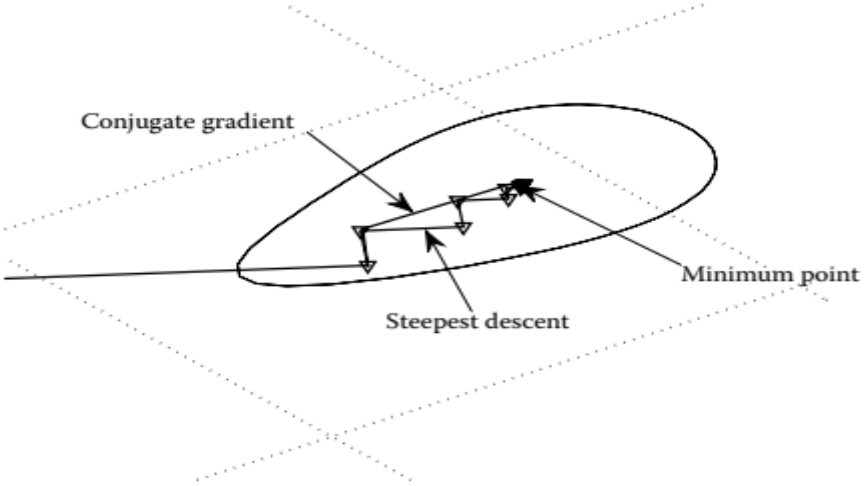


FIGURE 3.8: Convergencelot of conjugate gradient/steepest descent method.

The conjugate method does not show sluggishness in reaching the minimum point.

Initial function value = 1452.2619

No.	x-vector		f(x)	Deriv.
1	0.095	0.023	-2.704	1006.074
2	0.178	0.145	-5.404	37.036
3	0.507	0.136	-9.627	23.958
4	0.510	0.123	-9.655	4.239
5	0.505	0.121	-9.656	0.605
6	0.504	0.122	-9.656	0.340
7	0.504	0.122	-9.656	0.023

3.4.6 DFP Method

In the DFP method, the inverse of the Hessian is approximated by a matrix [$\boldsymbol{A}$] and the search direction is given by:

$$\boldsymbol{Si} = -[\boldsymbol{A}]\nabla f(\boldsymbol{xi}) \tag{3.24}$$

The information stored in the matrix [$\boldsymbol{A}$] is called as the metric and because it changes with every iteration, the DFP method is known as the variable metric method. Because this method uses first-order derivatives and has the property of quadratic convergence, it is referred to as a quasi-Newton method. The inverse of the Hessian matrix can be approximated as:

$$[A]_{i+1} = [A]_i + \frac{\Delta x \Delta x^T}{\Delta x^T \nabla g} - \frac{[A]_i \nabla g \nabla g^T [A]_i}{\nabla g^T [A]_i \nabla g} \tag{3.25}$$

Where:

$$\Delta x = \Delta x_i - \Delta x_{i-1} \tag{3.26}$$

$$\nabla g = \nabla g_i - \nabla g_{i-1} \tag{3.27}$$

The matrix [$\boldsymbol{A}$] is initialized to the identity matrix. The algorithm for the DFP method is described in Table 3.7 and a MATLAB code of its implementation is given in *dfp.m*.

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2) the following output is displayed in the command window for the test problem. Observe that in the second and the third iterations, search points are similar in this method and the conjugate gradient method, indicating that search directions were similar. In further iterations, however, the search direction is different. Further, on typing *inv*(*A*) in the MATLAB command prompt and then

TABLE 3.7 : Algorithm for the DFP Method

Step 1: Given x_i (starting value of design variable)
- ε_1 (tolerance of function value from previous iteration)
- ε_2 (tolerance on gradient value)
- Δx (required for gradient computation)
- [A] (initialize to identity matrix)

Step 2: Compute $f(x_i)$ and $\nabla f(x_i)$ (function and gradient vector)
- $S_i = -\nabla f(x_i)$ (search direction)
- $x_{i+1} = x_i + \alpha S_i$ (update the design vector)
- Minimize $f(x_{i+1})$ and determine α (use the golden section method)

Step 3: Compute Δx and ∇g

$$[A]_{i+1} = [A]_i + \frac{\Delta x \Delta x^T}{\Delta x^T \nabla g} - \frac{[A]_i \nabla g \nabla g^T [A]_i}{\nabla g^T [A]_i \nabla g}$$

$S_{i+1} = -[A]_{i+1} \nabla f(x_{i+1})$

$x_{i+2} = x_{i+1} + \alpha S_{i+1}$

Minimize $f(x_{i+2})$ and determine α (use the golden section method)

If $\left|f(x_{i+2}) - f(x_{i+1})\right| > \varepsilon_1$ or $\|\nabla f(x_{i+1})\| > \varepsilon_2$
- then goto Step 3
- else goto Step 4

Step 4: Converged. Print $x^* = x_{i+2}$, $f(x^*) = f(x_{i+2})$

printing the Hessian matrix at the converged value of $\boldsymbol{x}$, it is observed that [$\boldsymbol{A}$] approaches $[\boldsymbol{H}]^{-1}$.

```
        Initial function value = 1452.2619
No.       x-vector          f(x)          Deriv.
-------------------------------------------------
1     0.095       0.023     -2.704       1006.074
2     0.179       0.145     -5.418         37.036
3     0.508       0.145     -9.576         23.983
4     0.501       0.122     -9.656          7.004
5     0.504       0.122     -9.656          0.396
6     0.504       0.122     -9.656          0.053
7     0.504       0.122     -9.656          0.038
8     0.504       0.122     -9.656          0.028
9     0.504       0.122     -9.656          0.005
-------------------------------------------------
```

```
>> A
A =
    0.0091    0.0005
    0.0005    0.0033
>> inv(hessian(x,delx,n_of_var))
ans =
    0.0091    0.0005
    0.0005    0.0033
```

3.4.7 BFGS Method

In the BFGS method, the Hessian is approximated using the variable metric matrix [***A***] given by the equation:

$$[A]_{i+1} = [A]_i + \frac{g\nabla g^T}{\nabla g^T \Delta x} + \frac{\nabla f(x_i)\nabla f(x_i)^T}{\nabla f(x_i)^T S_i} \tag{3.28}$$

It is important to note that whereas the matrix [***A***] converges to the inverse of the Hessian in the DFP method, the matrix [***A***] converges to the Hessian itself in the BFGS method. As the BFGS method needs fewer restarts as com- pared to the DFP method, it is more popular than the DFP method. The algo- rithm for the BFGS method is described in Table 3.8 and a MATLAB code of its implementation is given in *BFGS.m*.

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2) the following output is displayed in the command window for the test problem. Again, it is observed that in the second and third iterations, search points are similar to this method as compared to DFP and the conjugate gradient methods, indicating that search directions were similar. Further, on typing A in the

TABLE 3.8: Algorithm for the BFGS Method

Step 1: Given x_i (starting value of design variable)
- ε_1 (tolerance of function value from previous iteration)
- ε_2 (tolerance on gradient value)
- Δx (required for gradient computation)
- $[A]$ (initialize to identity matrix)

Step 2: Compute $f(x_i)$ and $\nabla f(x_i)$ (function and gradient vector)
- $S_i = -\nabla f(x_i)$ (search direction)
- $x_{i+1} = x_i + \alpha S_i$ (update the design vector)
- Minimize $f(x_{i+1})$ and determine α (use golden section method)

Step 3: Compute Δx and ∇g

$$[A]_{i+1} = [A]_i + \frac{g\nabla g^T}{\nabla g^T \Delta x} + \frac{\nabla f(x_i)\nabla f(x_i)^T}{\nabla f(x_i)^T S_i}$$

$$S_{i+1} = -[[A]_{i+1}]^{-1}\nabla f(x_{i+1})$$

$$x_{i+2} = x_{i+1} + \alpha S_{i+1}$$

Minimize $f(x_{i+2})$ and determine α (use the golden section method)

If $\left|f(x_{i+2}) - f(x_{i+1})\right| > \varepsilon_1$ or $\|\nabla f(x_{i+1})\| > \varepsilon_2$
- then goto Step 3
- else goto Step 4

Step 4: Converged. Print $x^* = x_{i+2}$, $f(x^*) = f(x_{i+2})$

MATLAB command prompt and then printing the Hessian matrix at the converged value of $\boldsymbol{x}$, it is observed that [$\boldsymbol{A}$] approaches [$\boldsymbol{H}$].

```
        Initial function value = 1452.2619
No.       x-vector          f(x)          Deriv.
-------------------------------------------------
1     0.095       0.023     -2.704       1006.074
2     0.179       0.145     -5.418         37.036
3     0.508       0.145     -9.578         24.017
4     0.501       0.122     -9.655          6.900
5     0.504       0.122     -9.656          0.471
6     0.504       0.122     -9.656          0.077
7     0.504       0.122     -9.656          0.056
8     0.504       0.122     -9.656          0.040
9     0.504       0.122     -9.656          0.007
-------------------------------------------------
```

```
>> A
A =
    110.5001  -16.9997
    -16.9997  306.7238
>> hessian(x,delx,n_of_var)
ans =
    111.0981  -15.9640
    -15.9640  308.5603
```

3.4.8 Powell Method

The Powell method is a direct search method (no gradient computation is required) with the property of quadratic convergence. Previous search directions are stored in this method and they form a basis for the new search direction. The method makes a series of unidirectional searches along these search directions. The last search direction replaces the first one in the new iteration and the process is continued until the function value shows no improvement. A MATLAB code (*powell.m*) is written in which this method is implemented and the algorithm is described in Table 3.9.

On executing the code with a starting value of $\boldsymbol{x}$ as (–3, 2), following output is displayed at the command window for the test problem.

```
Initial function value = 1452.2619
No.       x-vector          f(x)
-------------------------------------
1     0.504       0.122     -9.656
2     0.505       0.122     -9.656
3     0.504       0.122     -9.656
4     0.504       0.122     -9.656
5     0.505       0.122     -9.656
-------------------------------------
```

TABLE 3.9: Algorithm for the Powell Method

Step 1: Given x_i (starting value of design variable)
ε (tolerance of function value from previous iteration)
S_i (linearly independent vectors)
$f(X_{prev}) = f(x_i)$
Step 2: $X = x_i + \alpha S_i$
Minimize $f(X)$ and determine α (use the golden section method)
Step 3: Set $Y = X$, $i = 1$
do
Minimize $f(X)$ and determine α (use the golden section method)
$X = X + \alpha S_i$
$i = i + 1$
while $i <$ (number of variable) + 1
If $|f(X) - f(X_{prev})| < \varepsilon$
then goto Step 4
else continue
$S_i = X - Y$
$X = X + \alpha S_i$
$f(X_{prev}) = f(X)$
goto Step 3
Step 4: Converged. Print $x^* = X$, $f(x^*) = f(X)$

3.4.9 Nelder–Mead Algorithm

Simplex refers to a geometric figure formed by $n + 1$ points in an n dimension space. For example, in a two-dimensional space, the figure formed is a trian- gle. The Nelder–Mead algorithm is a direct search method and uses function information alone (no gradient computation is required) to move from one iteration to another. The objective function is computed at each vertex of the simplex. Using this information, the simplex is moved in the search space. Again, the objective function is computed at each vertex of the simplex. The process of moving the simplex is continued until the optimum value of the function is reached. Three basic operations are required to move the simplex in the search space: reflection, contraction, and expansion.

In an optimization problem with two dimensions, the simplex will be a triangle, whose vertices are given by (say) $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$. Of these, let the worst value of the objective function be at $\boldsymbol{x}_3 = \boldsymbol{x}_{\text{worst}}$. If the point $\boldsymbol{x}_{\text{worst}}$ is reflected on the opposite face of the triangle, the objective function value is expected to decrease. Let the new reflected point be designated as $\boldsymbol{x}_r$. The new simplex (see Figure 3.9) is given by the vertices $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_r$. The centroid point $\boldsymbol{x}_c$ is computed using all the points but with the exclusion of $\boldsymbol{x}_{\text{worst}}$. That is,

$$x_c = \frac{1}{n}\sum_{\substack{i=1 \\ i\neq worst}}^{n+1} x_i \tag{3.29}$$

The reflected point is computed as:

$$\boldsymbol{x}_r = \boldsymbol{x}_c + \alpha(\boldsymbol{x}_c - \boldsymbol{x}_{worst}) \tag{3.30}$$

where α is a predefined constant. Typically, α = 1 is taken in the simulations. If the reflected value does not show improvement, the second worst value is taken and the process as discussed earlier is repeated.

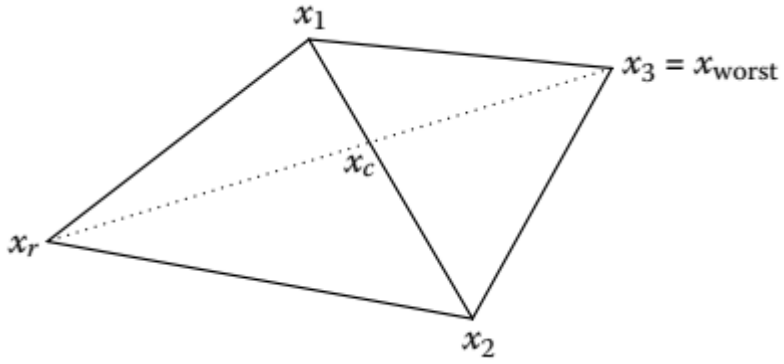


FIGURE 3.9: Reflection operation.

Sometimes reflection can lead to cycling with no improvement in the objective function value. Under such conditions, a contraction operation is performed.

If $\boldsymbol{x}_r$ results in a new minimum point, then it is possible to further expand the new simplex (see Figure 3.10) in the hope of further reducing the objec- tive function value. The expanded point is computed as:

$$\boldsymbol{xe} = \boldsymbol{xc} + \gamma(\boldsymbol{xc} - \boldsymbol{xworst}) \tag{3.31}$$

where γ is a predefined constant. Typically, γ = 2 is taken in the simulations. If $\boldsymbol{x}_e$ results in the new minimum point, it replaces the $\boldsymbol{x}_{\text{worst}}$ point. Else, $\boldsymbol{x}_r$ replaces the $\boldsymbol{x}_{\text{worst}}$ point.

The contraction operation is used when it is certain that the reflected point is better than the second worst point ($\boldsymbol{x}_{\text{second worst}}$). The contracted point is computed as

$$\boldsymbol{xcontr} = \boldsymbol{xc} + \rho(\boldsymbol{xc} - \boldsymbol{xworst}) \tag{3.32}$$

where ρ is a predefined constant. Typically, ρ = −0.5 is taken in the simulations. The preceding operations are continued until the standard deviation of the functions computed at the vertices of the simplex becomes less than ε.
That is,

$$\sum_{i=1}^{n+1} \frac{[f(x_i)-f(x_c)]^2}{n+1} \leq \varepsilon \tag{3.33}$$

The Nelder–Mead algorithm is described in Table 3.10 and a MATLAB code (*neldermead.m*) is written in which this method is implemented.

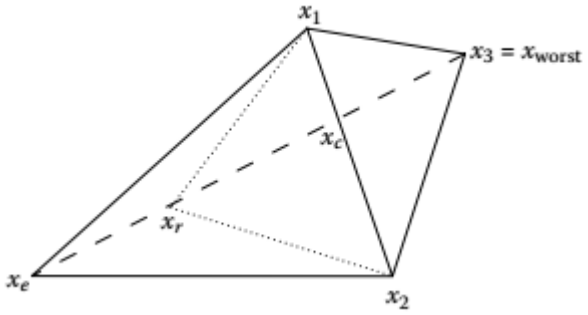


FIGURE 3.10: Expansion operation.

TABLE 3.10: Nelder–Mead Algorithm

Step 1: Given x_i (randomly select starting value of design variables)
$\alpha, \gamma, \rho, \sigma, \varepsilon$ (value of constants)
Compute $f(x_i)$, $f(x_{best}) \leq \ldots \leq f(x_{second\ worst}) \leq f(x_{worst})$

Step 2: Compute the centroid as

$$x_c = \frac{1}{n} \sum_{\substack{i=1 \\ i \neq worst}}^{n+1} x_i$$

Step 3: Reflection
$x_r = x_c + \alpha(x_c - x_{worst})$
If $f(x_{best}) \leq \ldots \leq f(x_r) \leq f(x_{second\ worst})$ then replace x_{worst} with x_r and goto Step 1

Step 3: Expansion
If $f(x_r) \leq f(x_{best})$ then
$x_e = x_c + \gamma(x_c - x_{worst})$
If $f(x_e) \leq f(x_r)$ then replace x_{worst} with x_e and goto Step 1
Else
replace x_{worst} with x_r
Else
goto Step 5

Step 4: Contraction
$x_{contr} = x_c + \rho(x_c - x_{worst})$
If $f(x_{contr}) \leq f(x_{worst})$ then replace x_{worst} with x_c and goto Step 1

Step 5: If

$$\sum_{i=1}^{n+1} \frac{\left[f(x_i) - f(x_c)\right]^2}{n+1} \leq \varepsilon$$

then converged,
else
goto Step 1

On executing the code with a random value of ***x***, the following output is displayed at the command window for the test problem.

Iteration	Deviation	f(x)
1	72.2666	-0.733
2	36.7907	-0.733
3	6.8845	-0.733
4	9.7186	-8.203
5	5.0965	-8.203
6	3.8714	-8.426
7	1.3655	-8.426
8	0.7944	-9.351
9	0.6497	-9.509
10	0.2242	-9.509
11	0.1083	-9.509
12	0.1068	-9.641
13	0.0794	-9.641
14	0.0299	-9.641
15	0.0173	-9.641
16	0.0126	-9.653
17	0.0079	-9.653
18	0.0034	-9.654
19	0.0025	-9.654
20	0.0021	-9.656
21	0.0011	-9.656
22	0.0003	-9.656
23	0.0004	-9.656
24	0.0003	-9.656

```
xc =
    0.5028    0.1219
```

4.5 Additional Test Functions

Different solution techniques were applied to the test problem on the spring system in the previous section. In this section, some additional test problems such as Rosenbrock's function, Wood's function, quadratic function, and so forth are taken, on which different solution methods will be tested. The performance of each method is compared in terms of the computational time. The MATLAB functions *tic* and *toc* can be used to estimate the computational time.

3.5.1 Rosenbrock Function

The two-variable function is given by:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{3.34}$$

The minimum of this "banana valley" function is zero (see Figure 3.11 where the minimum is marked with *) and occurs at (1, 1). Different solution methods are applied from the same starting point (–1.5, 1.5) and their performances are summarized in Table 3.11. All methods are able to track the minimum of the function. The steepest descent method takes a maximum computational time as compared to all other methods. The computational time required by other methods is comparable. The convergence history of the steepest descent method is plotted in Figure 3.12 and marked with °. Because of the particular nature of the problem, the method dwells in the region with a low gradient value. The Nelder–Mead method is not compared here as it uses more than one starting point.

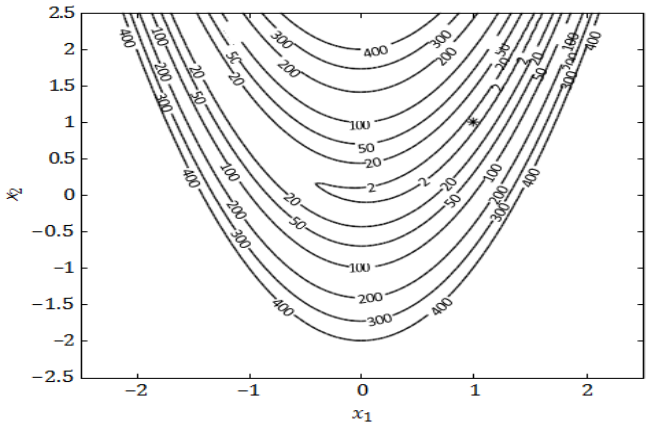


FIGURE 3.11: Contours of Rosenbrock function.

TABLE 3.11: Performance Comparison of Different Solution Methods for Rosenbrock's Function

Method	Computational Time (ms)
Steepest descent	49.7
Newton	8.04
Modified Newton	11.9
Marquardt	9.4
Conjugate gradient	18.8
DFP	11.23
BFGS	10.34
Powell	10.52

3.5.2 Quadratic Function

The two-variable function is given by:

$$f(\boldsymbol{x}) = (1 - x1)^2 + (2 - x2)^2 \tag{3.35}$$

The minimum of this function is zero (see Figure 3.13, where the minimum is marked with *) and occurs at (1, 2). Different solution methods are applied from a starting point (2, –3) and their performances are summarized in Table 3.12. All methods are able to track the minimum of the function. The conjugate gradient method takes minimum computational time compared to other solution methods.

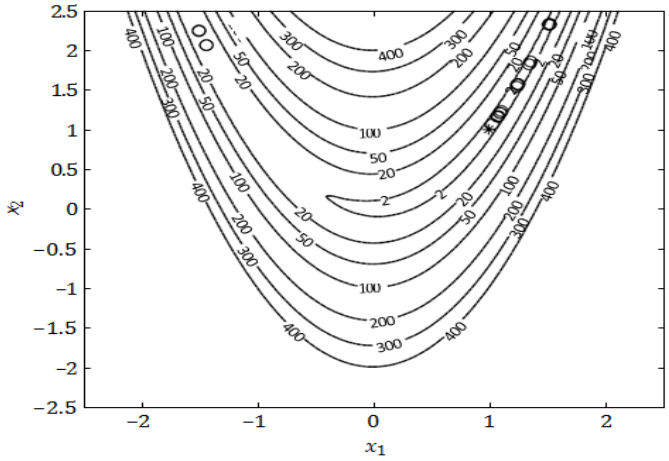


FIGURE 3.12: Behavior of steepest descent method on Rosenbrock function.

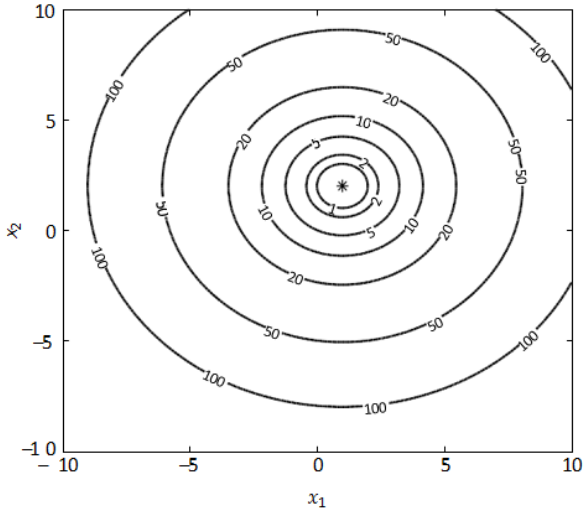


FIGURE 3.13: Contours of a quadratic function.

TABLE 3.12: Performance Comparison of Different Solution Methods for a Quadratic Function

Method	Computational Time (ms)
Steepest descent	6.06
Newton	7.5
Modified Newton	10.38
Marquardt	9.72
Conjugate gradient	5.79
DFP	7.69
BFGS	7.45
Powell	7.26

3.5.3 Nonlinear Function

The two-variable function is given by

$$f(x) = 4x_1^2 - 4x_1x_2 + 3x_2^2 + x_1 \tag{3.36}$$

The minimum of this function is –0.09375 (see Figure 3.14, where the mini- mum is marked with *) and occurs at (–3/16, –1/8). Different solution meth- ods are applied from a starting point (4, 3) and their performances are summarized in Table 3.13. All methods are able to track the minimum of the function. The conjugate gradient method takes minimum computational time compared to other solution methods.

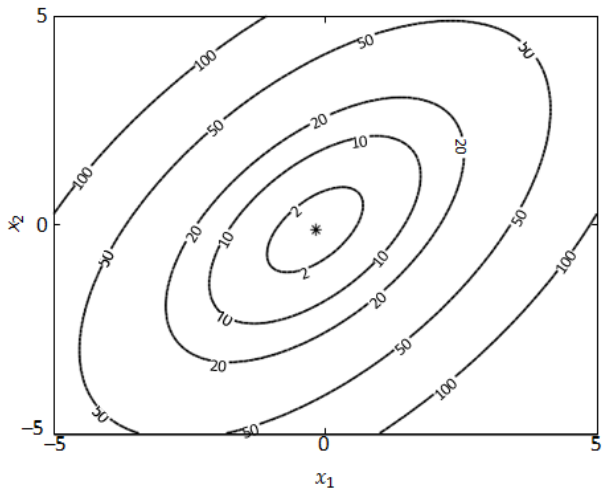


FIGURE 3.14: Contours of a nonlinear function.

TABLE 3.13: Performance Comparison of Different Solution Methods for a Nonlinear Function

Method	Computational Time (ms)
Steepest descent	11.19
Newton	7.67
Modified Newton	10.51
Marquardt	10.0
Conjugate gradient	6.27
DFP	7.85
BFGS	7.70
Powell	8.32

3.5.4 Wood's Function

The two-variable function is given by:

$$f(x) = \frac{1}{10}\left(12 + x_1^2 + \frac{1+x_1^2}{x_1^2} + \frac{100+x_1^2+x_2^2}{(x_1x_2)^4}\right) \tag{3.37}$$

The minimum of this function is 1.744 (see Figure 3.15, where the minimum is marked with *) and occurs at (1.743, 2.03). Different solution methods are applied from a starting point (0.5, 0.5) and their performances are summa- rized in Table 3.14. All methods are able to track the minimum of the func- tion. The conjugate gradient method takes minimum computational time compared to other solution methods.

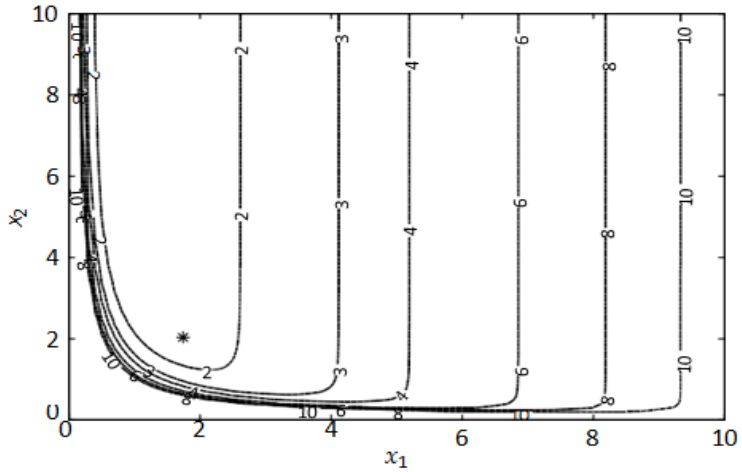


FIGURE 3.15: Contours of Wood's function.

TABLE 3.14: Performance Comparison of Different Solution Methods for Wood's Function

Method	Computational Time (ms)
Steepest descent	7.16
Newton	9.46
Modified Newton	12.1
Marquardt	10.6
Conjugate gradient	6.22
DFP	9.54
BFGS	8.33
Powell	12.75

❖ Chapter Highlights:

- The unidirectional search refers to minimizing the value of a multi- variable function along a specified direction.
- Solution techniques for multivariable, unconstrained optimization problems can be grouped into gradient- and non–gradient-based methods.
- The negative gradient direction is addressed as the steepest descent direction.
- The steepest descent method ensures a reduction in the function value at every iteration. If the starting point is far away from the minimum, the gradient will be higher and function reduction will be maximum in each iteration. Because the gradient value of the function decreases near the optimum, the method becomes sluggish (slow convergence) near the minimum.
- Newton's method requires computation of the Hessian matrix, which is computationally expensive. Newton's method is known for converging in one iteration for a quadratic

function. The method requires a restart if the starting point is far away from optimum.

• In the modified Newton method, a line search is performed in the search direction computed by the Newton method.

• The Levenberg–Marquardt method is a sort of hybrid method that combines the strength of both the steepest descent and Newton methods.

• The conjugate gradient method is a first-order method, but shows the property of quadratic convergence and thus has a significant advantage over the second-order methods.

• DFP and BFGS methods are called the variable metric methods.

• It is important to note that whereas the matrix [$\boldsymbol{A}$] converges to the inverse of the Hessian in the DFP method, it converges to the Hessian itself in the BFGS method.

• The Powell method is a direct search method (no gradient computa- tion is required) with the property of quadratic convergence.

• In the Nelder–Mead algorithm, the simplex is moved using reflec- tion, expansion, and contraction.

❖ **Formulae Chart:**

- Necessary conditions for minimum of a function:

$$\nabla f(\boldsymbol{x}*) = 0$$
$$\nabla^2 f(\boldsymbol{x}*) \geq 0$$

- Unidirectional search:

$$f(\alpha) = \boldsymbol{x}i + \alpha \boldsymbol{S}i$$

- Search direction in steepest descent method:

$$\boldsymbol{S}i = -\nabla f(\boldsymbol{x}i)$$

- Search direction in the Newton method:

$$\boldsymbol{S}i = -[\boldsymbol{H}]^{-1}\nabla f(\boldsymbol{x}i)$$

Search direction in the Levenberg–Marquardt method:

$$\boldsymbol{S}i = -[\boldsymbol{H} + \lambda \boldsymbol{I}]^{-1}\nabla f(\boldsymbol{x}i)$$

- Search direction in the conjugate gradient method:

$$S_{i+1} = -\nabla f(x_i) + \frac{\|\nabla f(x_{i+1})\|^2}{\|\nabla f(x_i)\|^2} S_i$$

- Search direction in the DFP method:

$$S_i = -[A]\nabla f(x_i)$$
$$[A]_{i+1} = [A]_i + \frac{\Delta x \Delta x^T}{\Delta x^T \nabla g} - \frac{[A]_i \nabla g \nabla g^T [A]_i}{\nabla g^T [A]_i \nabla g}$$

- Search direction in the BFGS method:

$$\boldsymbol{S}i = -[\boldsymbol{A}]^{-1}\nabla f(\boldsymbol{x}i)$$
$$[A]_{i+1} = [A]_i + \frac{\mathrm{g}\nabla \mathrm{g}^T}{\nabla g^T \Delta \mathrm{x}} + \frac{\nabla f(x_i)\nabla f(x_i)^T}{\nabla f(x_i)^T S_i}$$

❖ **Problems:**

1. Find the steepest descent direction for the function:

$$f(x) = x^2 + 3x + 2x^2$$

at the point (1,2).

2. Minimize the function:

$$f(x) = 10{,}000x_1x_2 + e^{-x_1} + e^{-x_2} - 2.0001$$

From a starting value of (2,2) using the BFGS, DFP, and steepest descent methods.

3. Minimize the function:

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_2^2 + x_1 - 7)^2$$

From a starting value of (2,3) using the following methods:

- Steepest descent
- Newton's method
- Modified Newton's method
- Levenberg–Marquardt
- DFP
- BFGS
- Powell
- Nelder–Mead

4. Show that in the DFP method, the variable metric [A] approaches the inverse of the Hessian matrix for the following function:

$$f(x) = x_1^2 + 3x_1x_2 + 5x_2^2$$

5. Show that in the BFGS method, the variable metric [A] approaches the Hessian matrix for the following function:

$$f(x) = x_1^2 + 3x_1x_2 + 5x_2^2$$

Take the starting value as (1,1).

6. Minimize the function using the DFP method with a starting value of (1,1):

$$f(x) = e^{(x_1^2 + x_2^2)} + x_1 + x_2 - 3 - \sin\big(3(x_1 + x_2)\big)$$

7. Minimize the function:

$$f(x) = 100(x_3 - 100)^2 + 100\left(\sqrt{x_1^2 + x_2^2 - 1}\right)^2 + x_3^2$$

where:

$$2\pi\theta = \tan^{-1}\left(\frac{x_2}{x_1}\right), \qquad x_1 > 0$$

and

$$2\pi\theta = \pi + \tan^{-1}\left(\frac{x_2}{x_1}\right), \qquad x_1 < 0$$

Take the starting value as (−1,0,0).

8. Instead of using the central difference formula for computing the derivative of a function, use the complex variable formula:

$$f'(x) = \frac{Imaginary\,[f(x + i\Delta x)]}{\Delta x}$$

The MATLAB code grad_vec.m can be modified as:

```matlab
function deriv = grad_vec_complex(x, delx, n_of_var)
xvec = x;
h = 1e-14;
for j = 1:length(x)
    xvec = x;
    c = complex(xvec(j), h);
    xvec(j) = c;
    deriv(j) = imag(func_multivar(xvec) / h);
end
end
```

Now use the steepest descent method to optimize the test function given in the main text.

9. Compare the accuracy of the derivative computation using the central difference formula and the complex variable formula against the analytical value of the derivative of the test function:

$$f(x) = \sin(x) + \ln(x)$$

at ***x=0.1***

10. Use the line search algorithm to minimize the function:

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_2^2 + x_1 - 7)^2$$

Starting from different initial points and different search directions:

- Starting point (1,1) and search direction (2,4).
- Starting point (0,0) and search direction (1,2).
- Starting point (3,2) and search direction (1,1).

11. Minimize the function:

$$f(x) = x_1^2 + 8x_2^2$$

From the starting point (1,2) using the steepest descent method. Observe the sluggishness of this method. Again, solve the function by the conjugate gradient method and compare the performance with the steepest descent method.

12. A manufacturing firm wants to divide its resources suitably between capital x_1 and labor x_2 so as to maximize the profit function given by:

$$f(x) = p(\ln(1 + x_1) + \ln(1 + x_2)) - wx_2 - vx_1$$

where ***p*** is the unit price of the product, ***w*** is the wage rate of labor, and ***v*** is the unit cost of capital.

- By computing the gradient vector of the above function with respect to x_1 and x_2, and then equating it to zero, compute the design variables x_1 and x_2 as a function of ***p***, ***v***, and ***w***.

- Using the second-order condition, check whether the solution corresponds to a maximum of the function.
- Compute numerical values of x_1 and x_2 by assuming suitable values of p, v, and w (where $p > w, v$).
- Starting with an initial guess of (0,0) and using the values of p, v, and w as assumed, find the maximum of the function using the steepest descent method. Compare the values of x_1 and x_2 with those obtained from the previous step.

13. Minimize the potential energy function for a two-bar unsymmetrical shallow truss ((Figure 3.17) using DFP and BFGS methods.

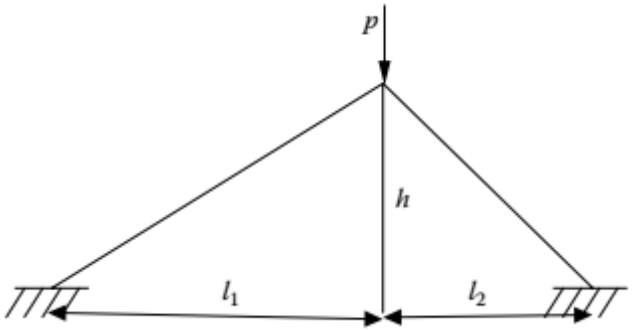


FIGURE 3.17: Two-bar truss.

The energy function is given as:

$$f(x) = \frac{1}{2} m_\gamma \left(-\alpha_1 x_1 + \frac{1}{2} x_1^2 + x_2\right)^2 + \frac{1}{2}\left(-\alpha_1 x_1 + \frac{1}{2} x_1^2 - \frac{x_2}{\gamma}\right)\gamma^4 - \rho\gamma x_1$$

where m, γ, α, and ρ are nondimensional quantities.

- Take m=5, γ=4, α=0.02, and ρ=0.00002.
- Starting with an initial guess of (0,0), minimize the function using DFP and BFGS methods.

Chapter V : Constrained Optimization

5.1 Introduction

Most problems in structural optimization must be formulated as constrained minimization problems. In a typical structural design problem the objective function is a fairly simple function of the design variables (e.g., weight), but the design has to satisfy a host of stress, displacement, buckling, and frequency constraints. These constraints are usually complex functions of the design variables available only from an analysis of a finite element model of the structure. This chapter offers a review of methods that are commonly used to solve such constrained problems.

The methods described in this chapter are for use when the computational cost of evaluating the objective function and constraints is small or moderate. In these methods the objective function or constraints these are calculated exactly (e.g., by a finite element program) whenever they are required by the optimization algorithm. This approach can require hundreds of evaluations of objective function and constraints, and is not practical for problems where a single evaluation is computationally expensive. For these more expensive problems we go through an intermediate stage of constructing approximations for the objective function and constraints, or at least for the more expensive functions. The optimization is then performed on the approximate problem. This approximation process is described in the next chapter.

The basic problem that we consider in this chapter is the minimization of a function subject to equality and inequality constraints

$$\begin{aligned} &\boldsymbol{minimize} \quad f(\boldsymbol{x}) \\ &\boldsymbol{such\ that}\ h_i(\mathbf{x}) = 0, g_j(\mathbf{x}) \geq 0, \qquad i = 1,\ldots,ne\,, j = 1,\ldots,ng\,. \end{aligned} \tag{5.1}$$

The constraints divide the design space into two domains, the feasible domain where the constraints are satisfied, and the infeasible domain where at least one of the constraints is violated. In most practical problems the minimum is found on the boundary between the feasible and infeasible domains, that is at a point where $g_j(\mathbf{x}) = 0$ for at least one j. Otherwise, the inequality constraints may be removed without altering the solution. In most structural optimization problems the inequality constraints prescribe limits on sizes, stresses, displacements, etc. These limits have great impact on the design, so that typically several of the inequality constraints are active at the minimum.

While the methods described in this section are powerful, they can often perform poorly when design variables and constraints are scaled improperly. To prevent ill-conditioning, all

the design variables should have similar magnitudes, and all constraints should have similar values when they are at similar levels of criticality. A common practice is to normalize constraints such that $g(\mathbf{x}) = 0.1$ correspond to a ten percent margin in a response quantity. For example, if the constraint is an upper limit σ_a on a stress measure σ, then the constraint may be written as

$$g = 1 - \frac{\sigma}{\sigma_a} \geq 0 \tag{5.2}$$

Some of the numerical techniques offered in this chapter for the solution of constrained nonlinear optimization problems are not able to handle equality constraints, but are limited to inequality constraints. In such instances it is possible to replace the equality constraint of the form $h_i(\mathbf{x}) = 0$ with two inequality constraints $h_i(\mathbf{x}) \leq 0$ and $h_i(\mathbf{x}) \geq 0$. However, it is usually undesirable to increase the number of constraints. For problems with large numbers of inequality constraints, it is possible to construct an equivalent constraint to replace them. One of the ways to replace a family of inequality constraints ($g_i(\mathbf{x}) \geq 0, i = 1...m$) by an equivalent constraint is to use the Kreisselmeier-Steinhauser function [1] (*KS*-function) defined as

$$KS[g_i(x)] = -\frac{1}{\rho}\ln[\textstyle\sum_i e^{-\rho g_{i(x)}}] \tag{5.3}$$

where ρ is a parameter which determines the closeness of the *KS*-function to the smallest inequality $min[gi(\boldsymbol{x})]$. For any positive value of the ρ, the *KS*-function is always more negative than the most negative constraint, forming a lower bound envelope to the inequalities. As the value of ρ is increased the *KS*-functions conforms with the minimum value of the functions more closely. The value of the *KS*-function is always bounded by

$$g_{min} \leq KS[g_i(x)] \leq g_{min} - \frac{\ln(m)}{\rho} \tag{5.4}$$

For an equality constraint represented by a pair of inequalities, $h_i(\mathbf{x}) \leq 0$ and $-h_i(\mathbf{x}) \leq 0$, the solution is at a point where both inequalities are active, $h_i(\mathbf{x}) = -h_i(\mathbf{x}) = 0$, Figure 5.1 . Sobieski [2] shows that for a *KS*-function defined by such a positive and negative pair of h_i, the gradient of the *KS*-function at the solution point $h_i(\mathbf{x}) = 0$ vanishes regardless of the ρ value, and its value approaches to zero as the value of ρ tends to infinity, Figure 5.1 . Indeed, from Eq. (5.4) at $\mathbf{x}$ where $h_i = 0$, the *KS*-function has the property:

$$0 \geq KS(h, -h) \geq -\frac{\ln(2)}{\rho} \tag{5.5}$$

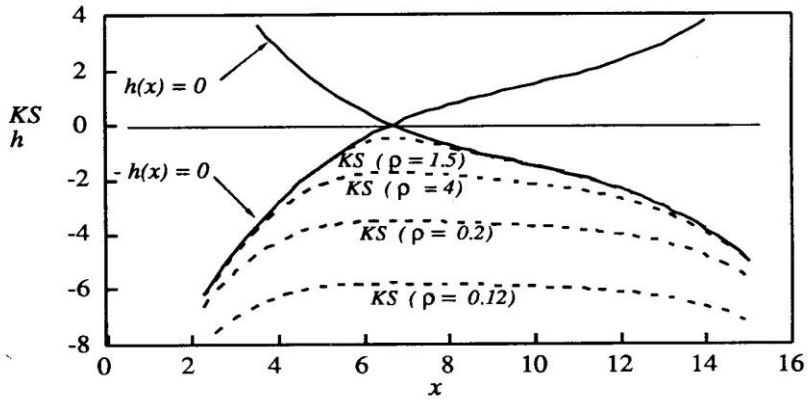


Figure 5.1 Kreisselmeier-Steinhauser function for replacing h(**x**) = 0.

Consequently, an optimization problem

$$\boldsymbol{minimize} \quad f(\boldsymbol{x})$$
$$\boldsymbol{such\ that} \quad hk(\boldsymbol{x}) = 0, k = 1, \ldots, ne\,, \tag{5.6}$$

may be reformulated as:

$$\boldsymbol{minimize} \quad f(\boldsymbol{x})$$
$$\boldsymbol{such\ that} \quad KS\left(h_1, -h_1, h_2, -h_2, \ldots, h_{n_e}, -h_{n_e}\right) \geq -\epsilon \tag{5.7}$$

where ϵ is a small tolerance.

5.2 The Kuhn-Tucker conditions

5.2.1 *General Case*

In general, problem (5.1) may have several local minima. Only under special circumstances are sure of the existence of single global minimum. The necessary conditions for a minimum of the constrained problem are obtained by using the Lagrange multiplier method. We start by considering the special case of equality constraints only. Using the Lagrange multiplier technique, we define the Lagrangian function:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{j=1}^{n_e} \lambda_j h_j(x) \tag{5.1.1}$$

jwhere λ_j are unknown Lagrange multipliers. The necessary conditions for a stationary point are:

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial f}{\partial x_i} - \sum_{j=1}^{n_e} \lambda_j \frac{\partial h_j}{\partial x_i} = 0, \qquad i = 1, \ldots \ldots, n \tag{5.1.2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_j} = h_j(x) = 0, \qquad j = 1, \ldots, n_e\,. \tag{5.1.3}$$

These conditions, however, apply only at a *regular point*, that is at a point where the gradients of the constraints are linearly independent. If we have constraint gradients that are linearly dependent, it means that we can remove some constraints without affecting the solution. At a regular point, Eqs. (5.1.2) and (5.1.3) represent $n + n_e$ equations for the n_e Lagrange multipliers and the n coordinates of the stationary point.

The situation is somewhat more complicated when inequality constraints are present. To be able to apply the Lagrange multiplier method we first transform the inequality constraints to equality constraints by adding slack variables. That is, the inequality constraints are written as:

$$g_j(\boldsymbol{x}) - t_j^2 = 0, \quad j = 1, \ldots, n_g, \tag{5.1.4}$$

where t_j is a slack variable which measures how far the jth constraint is from being critical. We can now form a Lagrangian function

$$\mathcal{L}(x, t, \lambda) = f - \sum_{j=1}^{n_g} \lambda_j \left(g_j - t_j^2\right) \tag{5.1.5}$$

Differentiating the Lagrangian function with respect to **x**, λ and **t** we obtain:

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial f}{\partial x_i} - \sum_{j=1}^{n_g} \lambda_j \frac{\partial g_j}{\partial x_i} = 0, \qquad i = 1, \ldots \ldots, n \tag{5.1.6}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_j} = -g_j + t_j^2 = 0, \qquad j = 1, \ldots., n_g, \tag{5.1.7}$$

$$\frac{\partial \mathcal{L}}{\partial t_j} = 2\lambda_j t_j = 0, \qquad j = 1, \ldots., n_g \tag{5.1.8}$$

Equations (5.1.7) and (5.1.8) imply that when an inequality constraint is not critical (so that the corresponding slack variable is non-zero) then the Lagrange multiplier associated with the constraint is zero. Equations (5.1.6) to (5.1.8) are the necessary conditions for a stationary regular point. Note that for inequality constraints a regular point is one where the gradients of the *active* constraints are linearly independent. These conditions are modified slightly to yield the necessary conditions for a minimum and are known as the Kuhn-Tucker conditions. The Kuhn-Tucker conditions may be summarized as follows:

A point **x** is a local minimum of an inequality constrained problem only if a set of nonnegative λ_j's may be found such that:

1. Equation (5.1.6) is satisfied
2. The corresponding λ_j is zero if a constraint is not active.

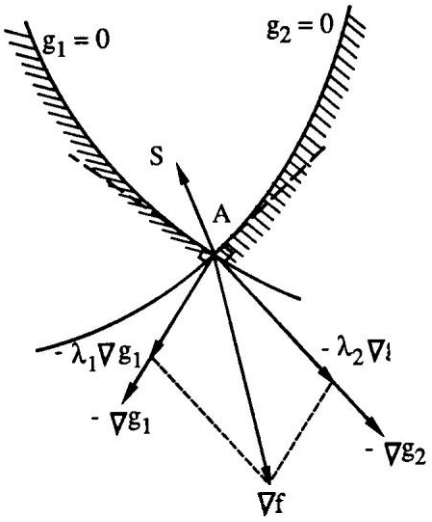


Figure 5.1.1 A geometrical interpretation of Kuhn-Tucker condition for the case of two constraints.

A geometrical interpretation of the Kuhn-Tucker conditions is illustrated in Fig. (5.1.1) for the case of two constraints. ∇g_1 and ∇g_2 denote the gradients of the two constraints which are orthogonal to the respective constraint surfaces. The vector **s** shows a typical feasible direction which does not lead immediately to any constraint violation. For the two-constraint case Eq. (5.1.6) may be written as

$$-\nabla f = -(\lambda 1 \nabla g 1 + \lambda 2 \nabla g 2) \quad . (5.1.9)$$

Assume that we want to determine whether point A is a minimum or not. To improve the design we need to proceed from point A in a direction **s** that is usable and feasible. For the direction to be usable, a small move along this direction should decrease the objective function. To be feasible, **s** should form an obtuse angle with $-\nabla g_1$ and $-\nabla g_2$. To be a direction of decreasing f it must form an acute angle with $-\nabla f$. Clearly from Figure (5.1.1), any vector which forms an acute angle with $-\nabla f$ will also form and acute angle with either $-\nabla g_1$ or $-\nabla g_2$. Thus the Kuhn-Tucker conditions mean that no feasible design with reduced objective function is to be found in the neighborhood of A. Mathematically, the condition that a direction **s** be feasible is written as

$$\boldsymbol{s}^T \nabla g j \geq 0, j \in IA\,, \quad (5.1.10)$$

where I_A is the set of active constraints Equality in Eq. (5.1.10) is permitted only for linear or concave constraints (see Section 5.1.2 for definition of concavity). The condition for a usable direction (one that decreases the objective function) is

$$\boldsymbol{s}T\ \nabla f\ <\ 0\,. \tag{5.1.11}$$

Multiplying Eq. (5.1.6) by s_i and summing over i we obtain:

$$s^T\nabla f = \sum_{j=1}^{n_g} \lambda_j s^T \nabla g_j \tag{5.1.12}$$

In view of Eqs. (5.1.10) and (5.1.11), Eq. (5.1.12) is impossible if the λ_j's are positive.

If the Kuhn-Tucker conditions are satisfied at a point it is impossible to find a direction with a negative slope for the objective function that does not violate the constraints. In some cases, though, it is possible to move in a direction which is tangent to the active constraints and perpendicular to the gradient (that is, has zero slope), that is

$$\boldsymbol{s^T}\nabla f\ =\ \boldsymbol{s^T}\nabla gj\ =\ 0, j\ \in\ I_A\,. \tag{5.1.13}$$

The effect of such a move on the objective function and constraints can be determined only from higher derivatives. In some cases a move in this direction could reduce the objective function without violating the constraints even though the Kuhn-Tucker conditions are met. Therefore, the Kuhn-Tucker conditions are necessary but not sufficient for optimality.

The Kuhn-Tucker conditions are sufficient when the number of active constraints is equal to the number of design variables. In this case Eq. (5.1.13) cannot be satisfied with $\mathbf{s}\ \neq\ 0$ because ∇g_j includes n linearly independent directions (in n dimensional space a vector cannot be orthogonal to n linearly independent vectors).

When the number of active constraints is not equal to the number of design variables sufficient conditions for optimality require the second derivatives of the objective function and constraints. A sufficient condition for optimality is that the Hessian matrix of the Lagrangian function is positive definite in the subspace tangent to the active constraints. If we take, for example, the case of equality constraints, the Hessian matrix of the Lagrangian is:

$$\nabla^2\mathcal{L} = \nabla^2 f - \textstyle\sum_{j=1}^{n_e} \lambda_j \nabla^2 h_j \tag{5.1.14}$$

The sufficient condition for optimality is that:

$$\boldsymbol{s}T\ (\nabla 2L)\boldsymbol{s}\ >\ 0, \quad \text{for all } \mathbf{s} \text{ for which } \quad \boldsymbol{s}T\ \nabla hj\ =\ 0, j\ =\ 1\ldots, ne\,. \tag{5.1.15}$$

When inequality constraints are present, the vector **s** also needs to be orthogonal to the gradients of the active constraints with positive Lagrange multipliers. For active constraints with zero Lagrange multipliers, **s** must satisfy:

$$\boldsymbol{s}T\ \nabla gj\ \geq\ 0, \quad \text{when } gj\ =\ 0\ and\ \lambda j\ =\ 0\,. \tag{5.1.16}$$

Example 5.1.1

Find the minimum of:

$$f = -x_1^3 - 2x_2^2 + 10x_1 - 6 - 2x_2^3$$

subject to:

$$g1\ =\ 10\ -\ x1x2\ \geq\ 0,$$
$$g2\ =\ x1\ \geq\ 0,$$
$$g3\ =\ 10\ -\ x2\ \geq\ 0\,.$$

The Kuhn-Tucker conditions are

$$-\ 3x21\ +\ 10\ +\ \lambda 1x2\ -\ \lambda 2\ =\ 0,$$

$$-4x2 - 6x22 + \lambda 1x1 + \lambda 3 = 0.$$

We have to check for all possibilities of active constraints.

The simplest case is when no constraints are active, $\lambda_1 = \lambda_2 = \lambda_3 = 0$. We get $x_1 = 1.826$, $x_2 = 0$, $f = 6.17$. The Hessian matrix of the Lagrangian,

$$\nabla^2 \mathcal{L} = \begin{bmatrix} -6x_1 & \lambda_1 \\ \lambda_1 & -4 - 12x_2 \end{bmatrix}$$

is clearly negative definite, so that this point is a maximum. We next assume that the first constraint is active, $x_1x_2 = 10$, so that $x_1 \neq 0$ and g_2 is inactive and therefore $\lambda_2 = 0$. We have two possibilities for the third constraint. If it is active we get $x_1 = 1$, $x_2 = 10$, $\lambda_1 = -0.7$, and $\lambda_3 = 639.3$, so that this point is neither a minimum nor a maximum. If the third constraint is not active $\lambda_3 = 0$ and we obtain the following three equations

$$-3x_1^2 + 10 + \lambda_1 x_2 = 0,$$

$$-4x_2 - 6x_2^2 + \lambda_1 x_1 = 0,$$

$$x_1 x_2 = 10.$$

The only solution for these equations that satisfies the constraints on x_1 and x_2 is

$$x_1 = 3.847, \qquad x_2 = 2.599, \qquad \lambda_1 = 13.24, \qquad f = -73.08.$$

This point satisfies the Kuhn-Tucker conditions for a minimum. However, the Hessian of the Lagrangian at that point

$$\nabla^2 \mathcal{L} = \begin{bmatrix} -23.08 & 13.24 \\ 13.24 & -35.19 \end{bmatrix}$$

is negative definite, so that it cannot satisfy the sufficiency condition. In fact, an examination of the function f at neighboring points along $x_1x_2 = 10$ reveals that the point is not a minimum.

Next we consider the possibility that g_1 is not active, so that $\lambda_1 = 0$, and

$$-3x_1^2 + 10 - \lambda_2 = 0,$$

$$-4x_2 - 6x_2^2 + \lambda_3 = 0.$$

We have already considered the possibility of both λ's being zero, so we need to consider only three possibilities of one of these Lagrange multipliers being nonzero, or both being nonzero. The first case is $\lambda_2 \neq 0$, $\lambda_3 = 0$, then $g_2 = 0$ and we get $x_1 = 0$, $x_2 = 0$, $\lambda_2 = 10$, and $f = -6$, or $x_1 = 0$, $x_2 = -2/3$, $\lambda_2 = 10$, and $f = -6.99$. Both points satisfy the Kuhn-Tucker conditions for a minimum, but not the sufficiency condition. In fact, the vectors tangent to the active constraints ($x_1 = 0$ is the only one) have the form $\mathbf{s}^T = (0,a)$, and it is easy to check that $\mathbf{s}^T \nabla^2 \mathrm{L} \mathbf{s} < 0$. It is also easy to check that these points are indeed no minima by reducing x_2 slightly.

The next case is $\lambda_2 = 0$, $\lambda_3 \neq 0$, so that $g_3 = 0$. We get $x_1 = 1.826$, $x_2 = 10$, $\lambda_3 = 640$ and $f = -2194$. this point satisfies the Kuhn-Tucker conditions, but it is not a minimum either. It is easy to check that $\nabla^2 \mathrm{L}$ is negative definite in this case so that the sufficiency condition could not be satisfied. Finally, we consider the case $x_1 = 0$, $x_2 = 10$, $\lambda_2 = 10$, $\lambda_3 = 640$, $f = -2206$. Now the Kuhn-Tucker conditions are satisfied, and the number of active constraints is equal to the number of design variables, so that this point is a minimum.

5.2.2 *Convex Problems*

There is a class of problems, namely convex problems, for which the Kuhn-Tucker conditions are not only necessary but also sufficient for a global minimum. To define convex problems we need the notions of convexity for a set of points and for a function. A set of points *S* is convex whenever the entire line segment connecting two points that are in *S* is also in *S*. That is

$$\text{if } \boldsymbol{x_1}, \boldsymbol{x_2} \in S, \text{ then } \alpha \boldsymbol{x_1} + (1-\alpha)\boldsymbol{x_2} \in S, \quad 0 < \alpha < 1 . \tag{5.1.17}$$

A function is convex if

$$f[\alpha \boldsymbol{x_2} + (1-\alpha)\boldsymbol{x}1] \leq \alpha f(\boldsymbol{x_2}) + (1-\alpha) f(\boldsymbol{x_1}), \quad 0 < \alpha < 1 . \tag{5.1.18}$$

This is shown pictorially for a function of a single variable in Figure (5.1.2). The straight segment connecting any two points on the curve must lie above the curve. Alternatively we note that the second derivative of *f* is non-negative $f''(x) \geq 0$. It can be shown that a function of *n* variables is convex if its matrix of second derivatives is positive semi-definite.

A convex optimization problem has a convex objective function and a convex feasible domain. It can be shown that the feasible domain is convex if all the inequality constraints g_j are *concave* (that is, $-g_j$ are convex) and the equality constraints are linear. A convex optimization problem has only one minimum, and the Kuhn-Tucker conditions are sufficient to establish it. Most optimization problems encountered in practice cannot be shown to be convex. However, the theory of convex programming is still very important in structural optimization, as we often approximate optimization problems by a series of convex approximations (see Chapter 9). The simplest such approximation is a linear approximation for the objective function and constraints– this produces a linear programming problem.

Figure 5.1.2 Convex function.

Example 5.1.2

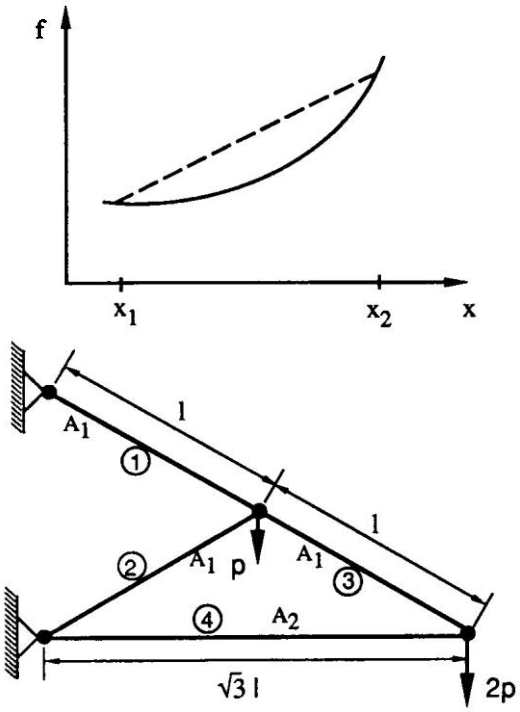


Figure 5.1.3 Four bar statically determinate truss.

Consider the minimum weight design of the four bar truss shown in Figure (5.1.3). For the sake of simplicity we assume that members 1 through 3 have the same area A_1 and member 4 has an area A_2. The constraints are limits on the stresses in the members and on the vertical

displacement at the right end of the truss. Under the specified loading the member forces and the vertical displacement δ at the end are found to be:

$$f_1 = 5p, \quad f_2 = -p, \quad f_3 = 4p, \quad f_4 = -2\sqrt{3p},$$

$$\delta = \frac{6pl}{E}\left(\frac{3}{A_1} + \frac{\sqrt{3}}{A_2}\right)$$

We assume the allowable stresses in tension and compression to be $8.74\times10^{-4}E$ and $4.83 \times 10^{-4}E$, respectively, and limit the vertical displacement to be no greater than $3\times10^{-3}l$. The minimum weight design subject to stress and displacement constraints can be formulated in terms of nondimensional design variables:

$$x_1 = 10^{-3}\frac{A_1 E}{p}, \quad x_2 = 10^{-3}\frac{A_2 E}{p}$$

as

$$\begin{aligned} \textbf{minimize} \quad & f = 3x_1 + \sqrt{3x_2} \\ \textbf{subject to} \quad & g_1 = 3 - \frac{18}{x_1} - \frac{6\sqrt{3}}{x_2} \geq 0 \\ & g_2 = x_1 - 5.73 \geq 0, \\ & g_3 = x_2 - 7.17 \geq 0 \end{aligned}$$

The Kuhn-Tucker conditions are:

$$\frac{\partial f}{\partial x_i} - \sum_{j=1}^{3} \lambda_j \frac{\partial g_j}{\partial x_i} = 0, \qquad i = 1,2$$

or

$$3 - \frac{18}{x_1^2}\lambda_1 - \lambda_2 = 0$$

$$\sqrt{3} - \frac{6\sqrt{3}}{x_2^2} - \lambda_1 - \lambda_3 = 0$$

Consider first the possibility that $\lambda_1 = 0$. Then clearly $\lambda_2 = 3$, $\lambda_3 = 3$ so that $g_2 = 0$ and $g_3 = 0$, and then $x_1 = 5.73$, $x_2 = 7.17$, $g_1 = -1.59$, so that this solution is not feasible. We conclude that $\lambda_1 \neq 0$, and the first constraint must be active at the minimum. Consider now the possibility that $\lambda_2 = \lambda_3 = 0$. We have the two Kuhn-Tucker equations and the equation $g_1 = 0$ for the unknowns λ_1, x_1, x_2. The solution is:

$$x_1 = x_2 = 9.464, \quad \lambda_1 = 14.93, \qquad f = 44.78\,.$$

The Kuhn-Tucker conditions for a minimum are satisfied. If the problem is convex the Kuhn-Tucker conditions are sufficient to guarantee that this point is the global minimum. The objective function and the constraint functions g_2 and g_3 are linear, so that we need to check only g_1. For convexity g_1 has to be concave or $-g_1$ convex; this holds if the second derivative matrix $-\mathbf{A}_1$ of $-g_1$ is positive semi-definite

$$-A_1 = \begin{bmatrix} 36/x_1^3 & 0 \\ 0 & 12\sqrt{3}/x_2^3 \end{bmatrix}$$

Clearly, for $x_1 > 0$ and $x_2 > 0$, $-\mathbf{A}_1$ is positive definite so that the minimum that we found is a global minimum.

5.3 Quadratic Programming Problems:

One of the simplest form of nonlinear constrained optimization problems is in the form of *Quadratic Programming* (QP) problem. A general QP problem has a quadratic objective

function with linear equality and inequality constraints. For the sake of simplicity we consider only an inequality problem with n_g constraints stated as:

$$\begin{aligned} \textbf{minimize} \quad & f(x) = c^T x + \tfrac{1}{2} x^T Q x \\ \textbf{such that} \quad & \boldsymbol{Ax} \geq \boldsymbol{b}, \\ & xi \geq 0, i = 1, \ldots, n\,. \end{aligned} \tag{5.2.1}$$

The linear constraints form a convex feasible domain. If the objective function is also convex, then we have a convex optimization problem in which, as discussed in the previous section, the Kuhn-Tucker conditions become sufficient for the optimality of the problem. Hence, having a positive semi-definite or positive definite **Q** matrix assures a global minimum for the solution of the problem, if one exists. For many optimization problems the quadratic form $\mathbf{x}^T\mathbf{Qx}$ is either positive definite or positive semi-definite. Therefore, one of the methods for solving QP problems relies on solving the Kuhn-Tucker conditions.

We start by writing the Lagrange function for the Problem (5.2.1)

$$\mathcal{L}(x, \lambda, \mu, t, s) = c^T x + \tfrac{1}{2} x^T Q x - \lambda^T \left(Ax - \left\{t_j^2\right\} - b\right) - \mu^T (x - \{s_i^2\}) \tag{5.2.2}$$

where λ and μ are the vectors of Lagrange multipliers for the inequality constraints and the nonnegativity constraints, respectively, and $\{t_j^2\}$ and $\{s_i^2\}$ are the vectors of positive slack variables for the same. The necessary conditions for a stationary point are obtained by differentiating the Lagrangian with respect to the **x**,***λ*,*μ*,*t***, and **s**,

$$\frac{\partial \mathcal{L}}{\partial x} = c - Qx - A^T\lambda - \mu = 0, \tag{5.2.3}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = Ax - \left\{t_j^2\right\} - b = 0, \tag{5.2.4}$$

$$\frac{\partial \mathcal{L}}{\partial \mu} = x - \{s_i^2\} = 0, \tag{5.2.5}$$

$$\frac{\partial \mathcal{L}}{\partial t_j} = 2\lambda_j t_j = 0, \quad j = 1, \ldots, n_g, \tag{5.2.6}$$

$$\frac{\partial \mathcal{L}}{\partial s_i} = 2\mu_i s_i = 0, \quad i = 1, \ldots, n. \tag{5.2.7}$$

where n_g is the number of inequality constraints, and n is the number of design variables. We define a new vector $\{q_j\} = \{t^2_j\}$, $j = 1,\ldots,n_g$ ($\mathbf{q} \geq 0$). After multiplying Eqs. (5.2.6) and (5.2.7) by $\{t_j\}$ and $\{s_i\}$, respectively, and eliminating $\{s_i\}$ from the last equation by using Eq. (5.2.5), we can rewrite the Kuhn-Tucker conditions

$$\boldsymbol{Qx} + \boldsymbol{A^T}\lambda + \mu = \boldsymbol{c}, \tag{5.2.8}$$

$$\boldsymbol{Ax} - \boldsymbol{q} = \boldsymbol{b}, \tag{5.2.9}$$

$$\boldsymbol{\lambda_j q_j} = \mathbf{0}, \quad \boldsymbol{j} = \mathbf{1}, \ldots, \boldsymbol{ng}\,, \tag{5.2.10}$$

$$\mu_{\boldsymbol{i}} \boldsymbol{x_i} = \mathbf{0}, \quad \boldsymbol{i} = \mathbf{1}, \ldots, \boldsymbol{n}, \tag{5.2.11}$$

$$\boldsymbol{x} \geq \mathbf{0}, \lambda \geq \mathbf{0}, and\ \mu \geq \mathbf{0}\,. \tag{5.2.12}$$

Equations (5.2.8) and (5.2.9) form a set of $n+n_g$ linear equations for the solution of unknowns x_i,λ_j,μ_i, and q_j which also need to satisfy Eqs. (5.2.10) and (5.2.11). Despite the nonlinearity of the Eqs. (5.2.10) and (5.2.11), this problem can be solved as proposed by Wolfe [3] by using the procedure described in 3.6.3 for generating a basic feasible solution through the use of artificial variables. Introducing a set of artificial variables, y_i, minimized, $i = 1,...,n$, we define an artificial cost function to be

$$minimize \quad \sum_{i=1}^{n} y_i \tag{5.2.13}$$

$$subject\ to \quad Qx + A^T\lambda + \mu + y = c, \tag{5.2.14}$$

$$Ax - q = b, \tag{5.2.15}$$

$$x \geq 0, \quad \lambda \geq 0, \quad \mu \geq 0, \quad and \quad y \geq 0 \tag{5.2.16}$$

Equations (5.2.13) through (5.2.16) can be solved by using the standard simplex method with the additional requirement that (5.2.10) and (5.2.11) be satisfied. These requirements can be implemented during the simplex algorithm by simply enforcing that the variables λ_j and q_j (and μ_i and x_i) not be included in the basic solution simultaneously. That is, we restrict a non-basic variable μ_i from entering the basis if the corresponding x_i is already among the basic variables.

Other methods for solving the quadratic programming problem are also available, and the reader is referred to Gill et al. ([4], pp. 177–180) for additional details.

5.4 Computing the Lagrange Multipliers:

As may be seen from example 5.1.1, trying to find the minimum directly from the Kuhn-Tucker conditions may be difficult because we need to consider many combinations of active and inactive constraints, and this would in general involve the solution of highly nonlinear equations. The Kuhn-Tucker conditions are, however, often used to check whether a candidate minimum point satisfies the necessary conditions. In such a case we need to calculate the Lagrange multipliers (also called the Kuhn-Tucker multipliers) at a given point **x**. As we will see in the next section, we may also want to calculate the Lagrange multipliers for the purpose of estimating the sensitivity of the optimum solution to small changes in the problem definition. To calculate the Lagrange multipliers we start by writing Eq. (5.1.6) in matrix notation as:

$$\nabla f - \boldsymbol{N}\lambda = 0, \tag{5.3.1}$$

where the matrix **N** is defined by :

$$n_{ij} = \frac{\partial g_j}{\partial x_i}, \qquad j = 1, \dots, r, \quad and \quad i = 1, \dots, n \tag{5.3.2}$$

We consider only the active constraints and associated lagrange multipliers, and assume that there are *r* of them.

Typically, the number, *r*, of active constraints is less than *n*, so that with *n* equations in terms of *r* unknowns, Eq. (5.3.1) is an overdetermined system. We assume that the gradients of the constraints are linearly independent so that **N** has rank *r*. If the Kuhn-Tucker conditions are satisfied the equations are consistent and we have an exact solution. We could therefore use a subset of *r* equations to solve for the Lagrange multipliers. However, this approach may be

susceptible to amplification of errors. Instead we can use a least-squares approach to solve the equations. We define a residual vector **u:**

$$\boldsymbol{u} = \boldsymbol{N}\lambda - \nabla f \tag{5.3.3}$$

A least squares solution of Eq. (5.3.1) will minimize the square of the Euclidean norm of the residual with respect to λ:

$$\|u\|^2 = (N\lambda - \nabla f)^T(N\lambda - \nabla f) = \lambda^T N^T N\lambda - 2\lambda^T N^T \nabla f + \nabla f^T \nabla f \tag{5.3.4}$$

To minimize k**u**k^2 we differentiate it with respect to each one of the Lagrange multipliers and get:

$$2\boldsymbol{N^T}\,\boldsymbol{N}\lambda - 2\boldsymbol{N^T}\,\nabla f = 0, \tag{5.3.5}$$

Or:

$$\lambda = (\boldsymbol{N^T N})^{-1}\boldsymbol{N^T}\,\nabla f\,. \tag{5.3.6}$$

This is the best solution in the least square sense. However, if the Kuhn-Tucker conditions are satisfied it should be the exact solution of Eq. (5.3.1). Substituting from Eq. (5.3.6) into Eq. (5.3.1) we obtain:

$$\boldsymbol{P}\nabla f = 0, \tag{5.3.7}$$

Where:

$$\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{N}(\boldsymbol{N^Y N})^{-1}\boldsymbol{N^T}\,. \tag{5.3.8}$$

P is called the projection matrix. It will be shown in Section 5.5 that it projects a vector into the subspace tangent to the active constraints. Equation (5.3.7) implies that for the Kuhn-Tucker conditions to be satisfied the gradient of the objective function has to be orthogonal to that subspace.

In practice Eq. (5.3.6) is no longer popular for the calculation of the Lagrange multipliers. One reason is that the method is ill-conditioned and another is that it is not efficient. An efficient and better conditioned method for least squares calculations is based on the QR factorization of the matrix **N**. The QR factorization of the matrix **N** consists of an $r \times r$ upper triangular matrix **R** and an $n \times n$ orthogonal matrix **Q** such that:

$$QN = \begin{pmatrix} Q_1 N \\ Q_2 N \end{pmatrix} = \begin{pmatrix} R \\ 0 \end{pmatrix} \tag{5.3.9}$$

Here $\mathbf{Q}_1$ is a matrix consisting of the first r rows of **Q**, $\mathbf{Q}_2$ includes the last $n - r$ rows of **Q**, and the zero represents an $(n - r) \times r$ zero matrix (for details of the QR factorization see most texts on numerical analysis, e.g., Dahlquist and Bjorck [5]). Because **Q** is an orthogonal matrix, the Euclidean norm of **Qu** is the same as that of **u**, or:

$$\|u\|^2 = \|Qu\|^2 = \|QN\lambda - Q\nabla f\|^2 = \left\|\begin{pmatrix} R \\ 0 \end{pmatrix}\lambda - Q\nabla f\right\|^2 = \left\|\begin{pmatrix} R\lambda - Q_1\nabla f \\ -Q_2\nabla f \end{pmatrix}\right\|^2 \tag{5.3.10}$$

From this form it can be seen that k**u**k^2 is minimized by choosing λ so that

$$\boldsymbol{R}\lambda = \boldsymbol{Q_1}\nabla f\,. \tag{5.3.11}$$

The last $n - r$ rows of the matrix **Q** denoted $\mathbf{Q}_2$ are also important in the following. They are orthogonal vectors which span the null space of $\mathbf{N}^T$. That is $\mathbf{N}^T$ times each one of these vectors is zero.

Example 5.3.1

Check whether the point (−2,−2,4) is a local minimum of the problem:

$$f = x_1 + x_2 + x_3,$$
$$g_1 = 8 - x_1^2 - x_2^2 \geq 0,$$
$$g_2 = x_3 - 4 \geq 0,$$
$$g_3 = x_2 + 8 \geq 0\,.$$

Only the first two constraints are critical at (−2,−2,4)

$$\frac{\partial g_1}{\partial x_1} = -2x_1 = 4, \quad \frac{\partial g_1}{\partial x_2} = -2x_2 = 4, \quad \frac{\partial g_1}{\partial x_3} = 0$$
$$\frac{\partial g_2}{\partial x_1} = 0, \quad \frac{\partial g_2}{\partial x_2} = 0, \quad \frac{\partial g_2}{\partial x_3} = 1,$$
$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial x_3} = 1.$$

So,

$$N = \begin{bmatrix} 4 & 0 \\ 4 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \nabla f = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix},$$
$$N^T N = \begin{bmatrix} 32 & 0 \\ 0 & 1 \end{bmatrix}, \quad N^T \nabla f = \begin{Bmatrix} 8 \\ 1 \end{Bmatrix},$$
$$\lambda = (N^T N)^{-1} N^T \nabla f = \begin{Bmatrix} 1/4 \\ 1 \end{Bmatrix},$$

Also,

$$[I - N(N^T N)^{-1} N^T] \nabla f = 0.$$

Equation (5.3.7) is satisfied, and all the Lagrange multipliers are positive, so the Kuhn-Tucker conditions for a minimum are satisfied.

5.5 Sensitivity of Optimum Solution to Problem Parameters

The Lagrange multipliers are not only useful for checking optimality, but they also provide information about the sensitivity of the optimal solution to problem parameters. In this role they are extremely valuable in practical applications. In most engineering design optimization problems we have a host of parameters such as material properties, dimensions and load levels that are fixed during the optimization. We often need the sensitivity of the optimum solution to these problem parameters, either because we do not know them accurately, or because we have some freedom to change them if we find that they have a large effect on the optimum design.

We assume now that the objective function and constraints depend on a parameter *p* so that the optimization problem is defined as

$$\textbf{minimize} \qquad f(\boldsymbol{x}, p)$$
$$\textbf{such that} \qquad g_j(\boldsymbol{x}, p) \geq 0, \quad j = 1, \ldots, ng\,. \qquad (5.4.1)$$

The solution of the problem is denoted $\mathbf{x}^*(p)$ and the corresponding objective function $f^*(p) = f(\mathbf{x}^*(p),p)$. We want to find the derivatives of $\mathbf{x}^*$ and f^* with respect to p. The equations that govern the optimum solution are the Kuhn-Tucker conditions, Eq. (5.3.1), and the set of active constraints:

$$\boldsymbol{g_a} = 0. \tag{5.4.2}$$

where $\mathbf{g}_a$ denotes the vector of r active constraint functions. Equations (5.3.1) and (5.4.2) are satisfied by $\mathbf{x}^*(p)$ for all values of p that do not change the set of active constraints. Therefore, the derivatives of these equations with respect to p are zero, provided we consider the implicit dependence of $\mathbf{x}$ and λ on p. Differentiating Eq. (5.3.1) and (5.4.2) with respect to p we obtain:

$$(A - Z)\frac{dx^*}{dp} - N\frac{d\lambda}{dp} + \frac{\partial}{\partial p}(\nabla f) - \left(\frac{\partial N}{\partial p}\right)\lambda = 0, \tag{5.4.3}$$

$$N^T\frac{dx^*}{dp} + \frac{\partial g_a}{\partial p} = 0, \tag{5.4.4}$$

where $\mathbf{A}$ is the Hessian matrix of the objective function f, $a_{ij} = \partial^2 f/\partial x_i \partial x_j$, and $\mathbf{Z}$ is a matrix whose elements are:

$$z_{kl} = \sum_j \frac{\partial^2 g_j}{\partial x_k \partial x_l}\lambda_j \tag{5.4.5}$$

Equations (5.4.3) and (5.4.4) are a system of simultaneous equations for the derivatives of the design variables and of the Lagrange multipliers. Different special cases of this system are discussed by Sobieski et al. [6].

Often we do not need the derivatives of the design variables or of the Lagrange multipliers, but only the derivatives of the objective function. In this case the sensitivity analysis can be greatly simplified. We can write:

$$\frac{df}{dp} = \frac{\partial f}{\partial p} + \sum_{l=1}^{n}\frac{\partial f}{\partial x_l}\frac{dx_l^*}{dp} = \frac{\partial f}{\partial p} + (\nabla f)^T\frac{dx^*}{dp} \tag{5.4.6}$$

Using Eq. (5.3.1) and (5.4.4) we get:

$$\frac{df}{dp} = \frac{\partial f}{\partial p} - \lambda^T\frac{\partial g_a}{dp}. \tag{5.4.7}$$

Equation (5.4.7) shows that the Lagrange multipliers are a measure of the effect of a change in the constraints on the objective function. Consider, for example, a constraint of the form $g_j(\mathbf{x}) = G_j(\mathbf{x}) - p \geq 0$. By increasing p we make the constraint more difficult to satisfy. Assume that many constraints are critical, but that p affects only this single constraint. We see that $\partial g_j/\partial p = -1$, and from Eq. (5.4.7) $df/dp = \lambda_j$, that is λ_j is the 'marginal price' that we pay in terms of an increase in the objective function for making g_j more difficult to satisfy.

The interpretation of Lagrange multipliers as the marginal prices of the constraints also explains why at the optimum all the Lagrange multipliers have to be non-negative. A negative Lagrange multiplier would indicate that we can reduce the objective function by making a constraint more difficult to satisfy— an absurdity.

Example 5.4.1

Consider the optimization problem:

$$f = x_1 + x_2 + x_3,$$
$$g_1 = p - x_1^2 - x_2^2 \geq 0,$$
$$g_2 = x_3 - 4 \geq 0,$$
$$g_3 = x_2 + p \geq 0\,.$$

This problem was analyzed for $p = 8$ in Example 5.3.1, and the optimal solution was found to be (−2,−2,4). We want to find the derivative of this optimal solution with respect to p. At the optimal point we have $f = 0$ and $\lambda^T = (0.25,1.0)$, with the:

first two constraints being critical. We can calculate the derivative of the objective function from Eq. (5.4.7)

$$\frac{\partial f}{\partial p} = 0, \quad \frac{\partial g_a}{\partial p} = \begin{Bmatrix}1\\0\end{Bmatrix},$$

So,

$$\frac{df}{dp} = -0.25$$

To calculate the derivatives of the design variables and constraints we need to set up Eqs. (5.4.3) and (5.4.4). We get:

$$A = 0, \qquad \frac{\partial \nabla f}{\partial p} = 0, \qquad \frac{\partial N}{\partial p} = 0$$

Only g_1 has nonzero second derivatives $\frac{\partial^2 g_1}{\partial x_1^2} = \frac{\partial^2 g_1}{\partial x_2^2} = -2$ so from Eq. (5.4.5):

$$z11 = -2\lambda_1 = -0.5, \qquad z22 = -2\lambda_1 = -0.5, \qquad Z = \begin{bmatrix}-0.5 & 0 & 0\\ 0 & -0.5 & 0\\ 0 & 0 & 0\end{bmatrix}.$$

With **N** from Example 5.3.1, Eq. (5.4.3) gives us:

$$\begin{cases}0.5\dot{x}_1 - 4\dot{\lambda}_1 = 0,\\ 0.5\dot{x}_2 - 4\dot{\lambda}_1 = 0,\\ \dot{\lambda}_2 = 0,\end{cases}$$

where a dot denotes derivative with respect to p. From Eq. (5.4.4) we get:

$$\begin{cases}4\dot{x}_1 + 4\dot{x}_2 + 1 = 0,\\ \dot{x}_3 = 0,\end{cases}$$

The solution of these five coupled equations is:

$$\dot{x}_1 = \dot{x}_2 = -0.125, \quad \dot{x}_3 = 0, \quad \dot{\lambda}_1 = -0.0156, \quad \dot{\lambda}_2 = 0.$$

We can check the derivatives of the objective function and design variables by changing p from 8 to 9 and re-optimizing. It is easy to check that we get $x_1 = x_2 = -2.121$, $x_3 = 4$, $f = -0.242$. These values compare well with linear extrapolation based on the derivatives which gives $x_1 = x_2 = -2.125$, $x_3 = 4$, $f = -0.25$.

5.6 Gradient Projection and Reduced Gradient Methods

Rosen's gradient projection method is based on projecting the search direction into the subspace tangent to the active constraints. Let us first examine the method for the case of linear constraints [7]. We define the constrained problem as:

$$\textbf{Minimize} \quad f(\mathbf{x})$$

$$\textbf{such that.} \quad g_j(x) = \sum_{i=1}^{n} a_{ji}x_i - b_j \geq 0, \qquad j = 1, \ldots, n_{g.} \tag{5.5.1}$$

In vector form

$$g_j = \boldsymbol{a_j^T x} - b_j \geq 0\,. \tag{5.5.2}$$

If we select only the *r* active constraints ($j \in I_A$), we may write the constraint equations as:

$$\boldsymbol{g_a} = \boldsymbol{N^T x} - \boldsymbol{b} = 0, \tag{5.5.3}$$

where $\mathbf{g}_a$ is the vector of active constraints and the columns of the matrix **N** are the gradients of these constraints. The basic assumption of the gradient projection method is that **x** lies in the subspace tangent to the active constraints. If

$$\boldsymbol{x_{i+1}} = \boldsymbol{x_i} + \alpha \boldsymbol{s}, \tag{5.5.4}$$

and both $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ satisfy Eq. (5.5.3), then:

$$\boldsymbol{N^T s} = 0\,. \; minimize \quad s^T \nabla f \;\; such\ that \quad N^T s = 0, \quad and \quad s^T s = 1. \tag{5.5.5}$$

If we want the steepest descent direction satisfying Eq. (5.5.5), we can pose the problem as:

$$\begin{aligned} minimize \quad & s^T \nabla f \\ such\ that \quad & N^T s = 0, \\ and \quad & s^T s = 1. \end{aligned}$$

That is, we want to find the direction with the most negative directional derivative which satisfies Eq. (5.5.5). We use Lagrange multipliers λ and μ to form the Lagrangian:

$$\boldsymbol{L(s, \lambda}, \mu) = \boldsymbol{s^T} \nabla f - \boldsymbol{s^T N \lambda} - \mu(\boldsymbol{s^T s} - 1) \tag{5.5.7}$$

The condition for L to be stationary is:

$$\frac{\partial \mathcal{L}}{\partial s} = \nabla f - N\lambda - 2\mu s = 0. \tag{5.5.8}$$

Premultiplying Eq. (5.5.8) by $\mathbf{N}^T$ and using Eq. (5.5.5) we obtain:

$$\boldsymbol{NT}\, \nabla f - \boldsymbol{NT\, N\lambda} = 0, \tag{5.5.9}$$

Or,

$$\lambda = (\boldsymbol{N^T N}) - 1\boldsymbol{N^T}\, \nabla f\,. \tag{5.5.10}$$

So that from Eq. (5.5.8)

$$s = \frac{1}{2\mu}[I - N(N^T N)^{-1} N^T]\nabla f = \frac{1}{2\mu} P \nabla f \tag{5.5.11}$$

P is the projection matrix defined in Eq. (5.3.8). The factor of $1/2\mu$ is not significant because **s** defines only the direction of search, so in general we use $\mathbf{s} = -\mathbf{P}\nabla f$. To show that **P** indeed has the projection property, we need to prove that if **w** is an arbitrary vector, then **Pw** is in the subspace tangent to the active constraints, that is **Pw** satisfies:

$$\boldsymbol{N^T P w} = 0\,. \tag{5.5.12}$$

We can easily verify this by using the definition of **P**.

Equation (5.3.8) which defines the projection matrix **P** does not provide the most efficient way for calculating it. Instead it can be shown that

$$P = Q_2^T Q_2 \tag{5.5.13}$$

where the matrix $\mathbf{Q}_2$ consists of the last $n - r$ rows of the **Q** factor in the QR factorization of **N** (see Eq. (5.3.9)).

A version of the gradient projection method known as the *generalized reduced gradient* method was developed by Abadie and Carpentier [8]. As a first step we select r linearly independent rows of **N**, denote their transpose as $\mathbf{N}_1$ and partition $\mathbf{N}^T$ as:

$$\boldsymbol{N^T} = [\boldsymbol{N_1 \& N_2}]. \tag{5.5.14}$$

Next we consider Eq. (5.5.5) for the components s_i of the direction vector. The r equations corresponding to $\mathbf{N}_1$ are then used to eliminate r components of **s** and obtain a reduced order problem for the direction vector.

Once we have identified $\mathbf{N}_1$ we can easily obtain $\mathbf{Q}_2$ which is given as:

$$Q_2^T = \begin{bmatrix} -N_1^{-1}N_2 \\ I \end{bmatrix} \tag{5.5.15}$$

Equation (5.5.15) can be verified by checking that $\mathbf{N}^T\mathbf{Q}_2^T = 0$, so that $\mathbf{Q}_2\mathbf{N} = 0$, which is the requirement that $\mathbf{Q}_2$ has to satisfy (see discussion following Eq. (5.3.11)).

After obtaining **s** from Eq. (5.5.11) we can continue the search with a one dimensional minimization, Eq. (5.5.4), unless $\mathbf{s} = 0$. When $\mathbf{s} = 0$ Eq. (5.3.7) indicates that the Kuhn-Tucker conditions may be satisfied. We then calculate the Lagrange multipliers from Eq. (5.3.6) or Eq. (5.3.11). If all the components of λ are nonnegative, the Kuhn-Tucker conditions are indeed satisfied and the optimization can be terminated. If some of the Lagrange multipliers are negative, it is an indication that while no progress is possible with the current set of active constraints, it may be possible to proceed by removing some of the constraints associated with negative Lagrange multipliers. A common strategy is to remove the constraint associated with the most negative Lagrange multiplier and repeat the calculation of **P** and **s**. If **s** is now non-zero, a one-dimensional search may be started. If **s** remains zero and there are still negative Lagrange multipliers, we remove another constraint until all Lagrange multipliers become positive and we satisfy the Kuhn-Tucker conditions.

After a search direction has been determined, a one dimensional search must be carried out to determine the value of α in Eq. (5.5.4). Unlike the unconstrained case, there is an upper limit on α set by the inactive constraints. As α increases, some of them may become active and then violated. Substituting $\mathbf{x} = \mathbf{x}_i + \alpha\mathbf{s}$ into Eq. (5.5.2) we obtain:

$$g_j = a_j^T(x_i + \alpha s) - b_j \geq 0 \tag{5.5.16}$$

Or

$$\alpha \leq -\frac{a_j^T x_i - b_j}{a_j^T s} = -\frac{g_i(x_i)}{a_j^T s}. \tag{5.5.17}$$

Equation (5.5.17) is valid if $\mathbf{a}^T_j\,\mathbf{s} < 0$. Otherwise, there is no upper limit on α due to the j^{th} constraint. From Eq. (5.5.17) we get a different α, say α_j for each constraint. The upper limit on α is the minimum:

$$\bar{\alpha} = \min_{\alpha_j>0,\quad j\ni I_A} \alpha_j \,. \tag{5.5.18}$$

At the end of the move, new constraints may become active, so that the set of active constraints may need to be updated before the next move is undertaken.

The version of the gradient projection method presented so far is an extension of the steepest descent method. Like the steepest descent method, it may have slow convergence. The method may be extended to correspond to Newton or quasi-Newton methods. In the unconstrained case, these methods use a search direction defined as

$$\boldsymbol{s} \;=\; -\boldsymbol{B}\nabla f \,, \tag{5.5.19}$$

where **B** is the inverse of the Hessian matrix of f or an approximation thereof. The direction that corresponds to such a method in the subspace tangent to the active constraints can be shown [4] to be

$$s = -Q_2^T(Q_2^T A_L Q_2)^{-1} Q_2 \nabla f \tag{5.5.20}$$

where $\mathbf{A}_L$ is the Hessian of the Lagrangian function or an approximation thereof.

The gradient projection method has been generalized by Rosen to nonlinear constraints [9]. The method is based on linearizing the constraints about $\mathbf{x}_i$ so that:

$$\boldsymbol{N} \;=\; [\nabla g_1(\boldsymbol{x_i}), \nabla g_2(\boldsymbol{x_i}), \ldots, \nabla g_r(\boldsymbol{x_i})] \,. \tag{5.5.21}$$

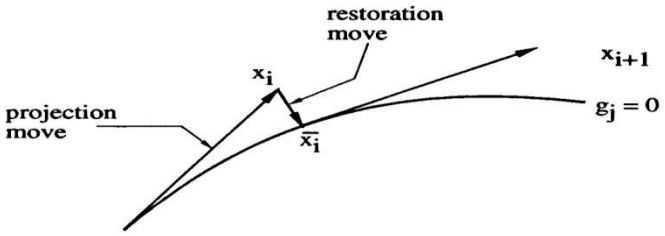


Figure 5.5.1 Projection and restoration moves.

The main difficulty caused by the nonlinearity of the constraints is that the one-dimensional search typically moves away from the constraint boundary. This is because we move in the tangent subspace which no longer follows exactly the constraint boundaries. After the one-dimensional search is over, Rosen prescribes a restoration move to bring **x** back to the constraint boundaries, see Figure 5.5.1.

To obtain the equation for the restoration move, we note that instead of Eq.(5.5.2) we now use the linear approximation:

$$g_j \approx g_j(x_i) + \nabla g_j^T(\bar{x}_i - x_i) \tag{5.5.22}$$

We want to find a correction $\bar{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}}$ in the tangent subspace (i.e. $\mathbf{P}(\bar{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}}) = 0$) that would reduce g_j to zero. It is easy to check that:

$$\bar{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}} = -\boldsymbol{N}(\boldsymbol{N}^T\boldsymbol{N})^{-1}\boldsymbol{g}_{\boldsymbol{a}}(\boldsymbol{x}_{\boldsymbol{i}}), \tag{5.5.23}$$

is the desired correction, where $\mathbf{g}_a$ is the vector of active constraints. Equation (5.5.23) is based on a linear approximation, and may therefore have to be applied repeatedly until $\mathbf{g}_a$ is small enough.

In addition to the need for a restoration move, the nonlinearity of the constraints requires the re-evaluation of **N** at each point. It also complicates the choice of an upper limit for α which guarantees that we will not violate the presently inactive constraints. Haug and Arora [10] suggest a procedure which is better suited for the nonlinear case. The first advantage of their procedure is that it does not require a one-dimensional search. Instead, α in Eq. (5.5.4) is determined by specifying a desired specified reduction γ in the objective function. That is, we specify

$$f(\boldsymbol{x_i}) \; - \; f(\boldsymbol{x_i} + 1) \; \approx \; \gamma f(\boldsymbol{x_i}) \tag{5.5.24}$$

Using a linear approximation with Eq. (5.5.4) we get:

$$\alpha^* = -\frac{\gamma f(x_i)}{s^T \nabla f} \tag{5.5.25}$$

The second feature of Haug and Arora's procedure is the combination of the projection and the restoration moves as

$$\boldsymbol{x_{i+1}} \; = \; \boldsymbol{x_i} \; + \; \alpha^* \boldsymbol{s} \; - \; \boldsymbol{N}(\boldsymbol{N^T N})^{-1} \boldsymbol{g_a} \, , \tag{5.5.26}$$

where Eqs. (5.5.4), (5.5.23) and (5.5.25) are used.

Example 5.5.1

Use the gradient projection method to solve the following problem

$$\begin{aligned} \textbf{minimize} \quad & f = x_1^2 + x_2^2 + x_3^2 + x_4^2 - 2x_1 - 3x_4 \\ \textbf{subject to} \quad & g_1 \; = \; 2x_1 \; + \; x_2 \; + \; x_3 \; + \; 4x_4 \; - \; 7 \; \geq \; 0, \\ & g_2 = x_1 + x_2 + x_3^2 + x_4 - 5.1 \geq 0, \\ & x_i \; \geq \; 0, \quad i \; = \; 1,\ldots,4 \, . \end{aligned}$$

Assume that as a result of previous moves we start at the point $x_0^T = (2,2,1,0)$, $f(\boldsymbol{x}_0) = 5.0$, where the nonlinear constraint g_2 is slightly violated. The first constraint is active as well as the constraint on x_4. We start with a combined projection and restoration move, with a target improvement of 10% in the objective function. At $\boldsymbol{x}_0$:

$$N = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 0 \\ 4 & 1 & 1 \end{bmatrix}, \qquad N^T N = \begin{bmatrix} 22 & 9 & 4 \\ 9 & 7 & 1 \\ 4 & 1 & 1 \end{bmatrix},$$

$$(N^T N)^{-1} = \frac{1}{11} \begin{bmatrix} 6 & -5 & -19 \\ -5 & 6 & 14 \\ -19 & 14 & 73 \end{bmatrix},$$

$$p = I - N(N^T N)^{-1} N^T = \frac{1}{11} \begin{bmatrix} 1 & -3 & 1 & 0 \\ -3 & 9 & -3 & 0 \\ 1 & -3 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \nabla f = \begin{Bmatrix} 2 \\ 4 \\ 2 \\ -3 \end{Bmatrix}$$

The projection move direction is $\mathbf{s} = -\mathbf{P}\nabla f = [8/11, -24/11, 8/11, 0]^T$. Since the magnitude of a direction vector is unimportant we scale **s** to $\mathbf{s}^T = [1, -3, 1, 0]$. For a 10% improvement in the objective function $\gamma = 0.1$ and from Eq. (5.5.25)

$$\alpha^* = -\frac{0.1f}{s^T \nabla f} = -\frac{0.1 \times 5}{-8} = 0.0625 \qquad \mathbf{g}_a^T = (0, -0.1, 0)$$

For the correction move we need the vector $\mathbf{g}_a$ of constraint values, $g_a^T = (0, -0.1, 0)$, so the correction is:

$$-N(N^T N)^{-1} g_a = \frac{-1}{110}\begin{Bmatrix} 4 \\ -1 \\ -7 \\ 0 \end{Bmatrix}$$

Combining the projection and restoration moves, Eq. (5.5.26)

$$x_1 = \begin{Bmatrix} 2 \\ 2 \\ 1 \\ 0 \end{Bmatrix} + 0.0625\begin{Bmatrix} 1 \\ -3 \\ 1 \\ 0 \end{Bmatrix} - \frac{1}{110}\begin{Bmatrix} 4 \\ -1 \\ -7 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 2.026 \\ 1.822 \\ 1.126 \\ 0 \end{Bmatrix}$$

we get $f(\mathbf{x}_1) = 4.64$, $g_1(\mathbf{x}_1) = 0$, $g_2(\mathbf{x}_1) = 0.016$. Note that instead of 10% reduction we got only 7% due to the nonlinearity of the objective function. However, we did satisfy the nonlinear constraint.

5.7 The Feasible Directions Method

The feasible directions method [11] has the opposite philosophy to that of the gradient projection method. Instead of following the constraint boundaries, we try to stay as far away as possible from them. The typical iteration of the feasible direction method starts at the boundary of the feasible domain (unconstrained minimization techniques are used to generate a direction if no constraint is active).

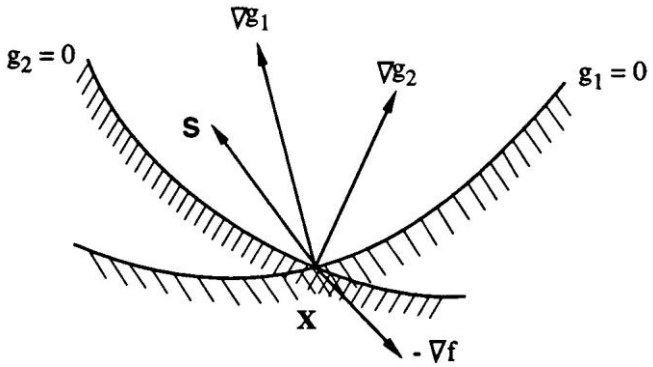


Figure 5.6.1 Selection of search direction using the feasible directions method.

Consider Figure 5.6.1. As a result of a previous move the design is at point **x** and we look for a direction **s** which keeps **x** in the feasible domain and improves the objective function. A vector **s** is defined as a feasible direction if at least a small step can be taken along it that does not immediately leave the feasible domain. If the constraints are smooth, this is satisfied if:

$$\boldsymbol{s}^T \nabla g_j > 0, j \in I_A , \tag{5.6.1}$$

where I_A is the set of critical constraints at **x**. The direction **s** is called a usable direction at the point **x** if in addition:

$$\boldsymbol{s}^T \nabla f = \boldsymbol{s}^T \boldsymbol{g} < 0 . \tag{5.6.2}$$

That is, **s** is a direction which reduces the objective function.

Among all possible choices of usable feasible directions we seek the direction which is best in some sense. We have two criteria for selecting a direction. On the one hand we want to reduce the objective function as much as possible. On the other hand we want to keep away from the constraint boundary as much as possible. A compromise is defined by the following maximization problem:

$$\begin{aligned} &maximize && \beta \\ &such\ that && -s^T\nabla g_j + \theta_j\beta \le 0, \qquad j \in I_A, \\ & && s^T\nabla f + \beta \le 0, \qquad \theta_j \ge 0, \\ & && |s_i| \le 1. \end{aligned} \tag{5.6.3}$$

The θ_j are positive numbers called "push-off" factors because their magnitude determines how far **x** will move from the constraint boundaries. A value of $\theta_j = 0$ will result in a move tangent to the boundary of the the *j*th constraint, and so may be appropriate for a linear constraint. A large value of θ_j will result in a large angle between the constraint boundary and the move direction, and so is appropriate for a highly nonlinear constraint.

The optimization problem defined by Eq. (5.6.3) is linear and can be solved using the simplex algorithm. If $\beta_{max} > 0$, we have found a usable feasible direction. If we get $\beta_{max} = 0$ it can be shown that the Kuhn-Tucker conditions are satisfied.

Once a direction of search has been found, the choice of step length is typically based on a prescribed reduction in the objective function (using Eq. (5.5.25)). If at the end of the step no constraints are active, we continue in the same direction as long as $\mathbf{s}^T\nabla f$ is negative. We start the next iteration when **x** hits the constraint boundaries, or use a direction based on unconstrained technique if **x** is inside the feasible domain. Finally, if some constraints are violated after the initial step we make **x** retreat based on the value of the violated constraints. The method of feasible directions is implemented in the popular CONMIN program [12].

Example 5.6.1

Consider the four bar truss of Example 5.1.2. The problem of finding the minimum weight design subject to stress and displacement constraints was formulated as:

$$\begin{aligned} &\boldsymbol{minimize} && f = 3x_1 + \sqrt{3x_2} \\ &\boldsymbol{subject\ to} && g_1 = 3 - \frac{18}{x_1} - \frac{6\sqrt{3}}{x_2} \ge 0 \\ & && g_2 = x_1 - 5.73 \ge 0, \\ & && g_3 = x_2 - 7.17 \ge 0, \end{aligned}$$

where the x_i are non-dimensional areas:

$$x_i = \frac{A_i E}{1000P}, \qquad i = 1,2$$

The first constraint represents a limit on the vertical displacement, and the other two constraints represent stress constraints.

Assume that we start the search at the intersection of $g_1 = 0$ and $g_3 = 0$ where $X_0^T = (11.61,7.17)$ and $f = 47.25$. The gradient of the objective function and two

active constraints are

$$\nabla f = \begin{Bmatrix} 3 \\ \sqrt{3} \end{Bmatrix}, \qquad \nabla g_1 = \begin{Bmatrix} 0.1335 \\ 0.2021 \end{Bmatrix}, \qquad \nabla g_3 = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}.$$

Selecting $\theta_1 = \theta_2 = 1$, we find that Eq. (5.6.3) becomes

$$\begin{aligned}
&\textbf{maximize} && \beta \\
&\textbf{subject to} && -0.1335s_1 - 0.2021s_2 + \beta \le 0 \\
& && -s_2 + \beta \le 0, \\
& && 3s_1 + \sqrt{3}s_2 + \beta \le 0, \\
& && -1 \le s_1 \le 1, \\
& && -1 \le s_2 \le 1.
\end{aligned}$$

The solution of this linear program is $s_1 = -0.6172$, $s_2 = 1$, and we now need to execute the one dimensional search:

$$x_1 = \begin{Bmatrix} 11.61 \\ 7.17 \end{Bmatrix} + \alpha \begin{Bmatrix} -0.6172 \\ 1 \end{Bmatrix}$$

Because the objective function is linear, this direction will remain a descent direction indefinitely, and α will be limited only by the constraints. The requirement that g_2 is not violated will lead to $\alpha = 9.527$, $x_1 = 5.73$, $x_2 = 16.7$ which violates g_1. We see that because g_1 is nonlinear, even though we start the search by moving away from it we still bump into it again (see Figure 5.6.2). It can be easily checked that for $\alpha > 5.385$ we violate g_1. So we take $\alpha = 5.385$ and obtain $x_1 = 8.29$, $x_2 = 12.56$, $f = 46.62$.

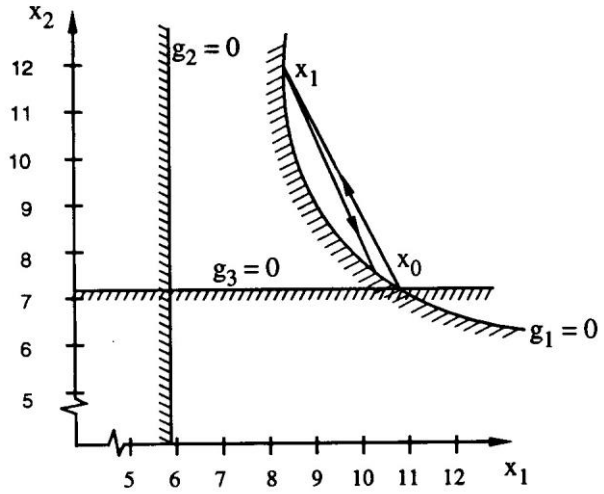


Figure 5.6.2 Feasible direction solution of 4 bar truss example.

For the next iteration we have only one active constraint:

$$\nabla g_1 = \begin{Bmatrix} 0.2619 \\ 0.0659 \end{Bmatrix} \qquad \nabla f = \begin{Bmatrix} 3 \\ \sqrt{3} \end{Bmatrix}$$

The linear program for obtaining **s** is:

$$\begin{aligned}
&\textbf{maximize} && \beta \\
&\textbf{subject to} && -0.2619s_1 - 0.0659s_2 + \beta \le 0 \\
& && 3s_1 + \sqrt{3}s_2 + \beta \le 0 \\
& && -1 \le s_1 \le 1, \\
& && -1 \le s_2 \le 1.
\end{aligned}$$

The solution of the linear program is $s_1 = 0.5512$, $s_2 = -1$, so that the onedimensional search is:

$$x = \begin{Bmatrix} 8.29 \\ 12.56 \end{Bmatrix} + \alpha \begin{Bmatrix} 0.5512 \\ -1 \end{Bmatrix}$$

Again α is limited only by the constraints. The lower limit on x_2 dictates $\alpha \leq 5.35$. However, the constraint g_1 is again more critical. It can be verified that for $\alpha > 4.957$ it is violated, so we take $\alpha = 4.957$, $x_1 = 11.02$, $x_2 = 7.60$, $f = 46.22$. The optimum design found in Example 5.1.2 is $x_1 = x_2 = 9.464$, $f = 44.78$. The design space and the two iterations are shown in Figure 5.6.2.

5.8 Penalty Function Methods

When the energy crisis erupted in the middle seventies, the United States Congress passed legislation intended to reduce the fuel consumption of American cars. The target was an average fuel consumption of 27.5 miles per gallon for new cars in 1985. Rather than simply legislate this limit Congress took a gradual approach, with a different limit set each year to bring up the average from about 14 miles per gallon to the target value. Thus the limit was set at 26 for 1984, 25 for 1983, 24 for 1982, and so on. Furthermore, the limit was not absolute, but there was a fine of $50 per 0.1 miles per gallon violation per car.

This approach to constraining the automobile companies to produce fuel efficient cars has two important aspects. First, by legislating a penalty proportional to the violation rather than an absolute limit, the government allowed the auto companies more flexibility. That meant they could follow a time schedule that approximated the government schedule without having to adhere to it rigidly. Second, the gradual approach made enforcement easier politically. Had the government simply set the ultimate limit for 1985 only, nobody would have paid attention to the law in the 1970's. Then as 1985 moved closer there would have been a rush to develop fuel efficient cars. The hurried effort could mean both non-optimal car designs and political pressure to delay the enforcement of the law.

The fuel efficiency law is an example in which constraints on behavior or economic activities are imposed via penalties whose magnitude depends on the degree of violation of the constraints. It is no wonder that this simple and appealing approach has found application in constrained optimization. Instead of applying constraints we replace them by penalties which depend on the degree of constraint violations. This approach is attractive because it replaces a constrained optimization problem by an unconstrained one.

The penalties associated with constraint violation have to be high enough so that the constraints are only slightly violated. However, just as there are political problems associated with imposing abrupt high penalties in real life, so there are numerical difficulties associated with such a practice in numerical optimization. For this reason we opt for a gradual approach where we start with small penalties and increase them gradually.

5.7.1 Exterior Penalty Function:

The exterior penalty function associates a penalty with a violation of a constraint. The term 'exterior' refers to the fact that penalties are applied only in the exterior of the feasible domain. The most common exterior penalty function is one which associates a penalty which is proportional to the square of a violation. That is, the constrained minimization problem, Eq. (5.1)

Minimize $f(x)$

Is replaced by: Such that $h_i(x) = 0, \qquad i = 1, \dots, n_e,$ (5.7.1)

$$g_i(x) \geq 0, \qquad j = 1, \dots, n_g,$$

Minimize $\phi(x, r) = f(x) + r \sum_{i=1}^{n_e} h_i^2(x) + r \sum_{j=1}^{n_g} < -g_j >^2$ (5.7.2)

$$r = r_1, r_2, \dots, \qquad r_i \to \infty,$$

where $< a >$ denote the positive part of a or $\max(a,0)$. The inequality terms are treated differently from the equality terms because the penalty applies only for constraint violation. The positive multiplier r controls the magnitude of the penalty terms. It may seem logical to choose a very high value of r to ensure that no constraints are violated. However, as noted before, this approach leads to numerical difficulties illustrated later in an example. Instead the minimization is started with a relatively small value of r, and then r is gradually increased. A typical value for r_{i+1}/r_i is 5. A typical plot of $\phi(\mathbf{x}, r)$ as a function of r is shown in Figure 5.7.1 for a simple example.

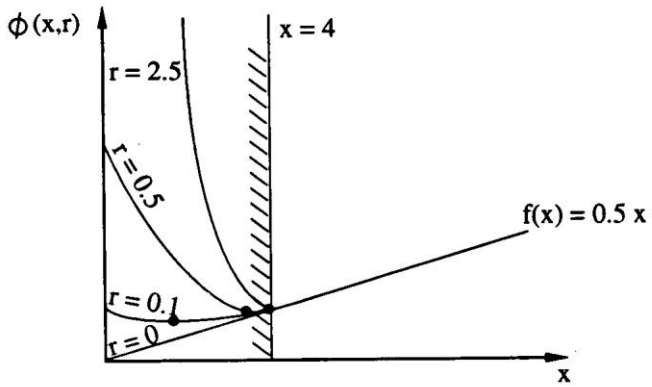


Figure 5.7.1 Exterior penalty function for $f = 0.5x$ subject to $x - 4 \geq 0$.

We see that as r is increased, the minimum of ϕ moves closer to the constraint boundary. However, the curvature of ϕ near the minimum also increases. It is the high values of the curvature associated with large values of r which often lead to numerical difficulties. By using a sequence of values of r, we use the minima obtained for small values of r as starting points for the search with higher r values. Thus the ill-conditioning associated with the large curvature is counterbalanced by the availability of a good starting point.

Based on the type of constraint normalization given by Eq. (5.2) we can select a reasonable starting value for the penalty multiplier r. A rule of thumb is that one should start with the total penalty being about equal to the objective function for typical constraint violation of 50% of the response limits. In most optimization problems the total number of active constraints is about the same as or just slightly lower than the number of design variables. Assuming we start with one quarter of the eventual active constraints being violated by about 50% (or $g = -0.5$) then we have

$$f(x_0) \approx r_0 \frac{n}{4}(0.5)^2, \qquad or \;\; r_0 = 16 \frac{f(x_0)}{n} \tag{5.7.3}$$

It is also important to obtain a good starting point for restarting the optimization as r is increased. The minimum of the optimization for the previous value of r is a reasonable starting point, but one can do better. Fiacco and McCormick [13] show that the position of the minimum of $\phi(\mathbf{x},r)$ has the asymptotic form:

$$\boldsymbol{x}^*(r) = \boldsymbol{a} + \frac{\boldsymbol{b}}{r}, \quad as\, r \to \infty \tag{5.7.4}$$

Once the optimum has been found for two values of r, say r_{i-1}, and r_i, the vectors **a** and **b** may be estimated, and the value of $\mathbf{x}^*(r)$ predicted for subsequent values of r. It is easy to check that in order to satisfy Eq. (5.7.4), **a** and **b** are given as:

$$a = \frac{cx^*(r_{i-1}) - x^*(r_i)}{c-1}, \tag{5.7.5}$$

$$b = [x^*(r_{i-1}) - a]r_{i-1},$$

where

$$c = ri - 1/ri \tag{5.7.6}$$

In addition to predicting a good value of the design variables for restarting the optimization for the next value of r, Eq. (5.7.4) provides us with a useful convergence criterion, namely:

$$\|\mathbf{x}^* - \mathbf{a}\| \leq \epsilon_1, \tag{5.7.7}$$

where **a** is estimated from the last two values of r, and ϵ_1 is a specified tolerance chosen to be small compared to a typical value of $\|x\|$.

A second convergence criterion is based on the magnitude of the penalty terms, which, as shown in Example 5.7.1, go to zero as r goes to infinity. Therefore, a reasonable convergence criterion is:

$$\left|\frac{\phi - f}{f}\right| \leq \epsilon_2 \tag{5.7.8}$$

Finally, a criterion based on the change in the value of the objective function at the minimum f^* is also used:

$$\left|\frac{f^*(r_i) - f^*(r_{i-1})}{f^*(r_i)}\right| \leq 0 \tag{5.7.9}$$

A typical value for ϵ_2 or ϵ_3 is 0.001.

Example 5.7.1 $f = x_1^2 + 10x_2^2$

Minimize $f = x_1^2 + 10x_2^2$ such that $x_1 + x_2 = 4$. We have,

$$\phi = x_1^2 + 10x_2^2 + r(4 - x_1 - x_2)^2$$

The gradient $\nabla\phi$ is given as:

$$g = \begin{Bmatrix} 2x_1(1+r) + 2rx_2 - 8r \\ 2x_2(10+r) + 2rx_1 - 8r \end{Bmatrix}$$

Setting the gradient to zero we obtain:

$$x_1 = \frac{40r}{10 + 11r}, \qquad x_2 = \frac{4r}{10 + 11r}$$

The solution as a function of r is shown in Table 5.7.1.

Table 5.7.1 Minimization of ϕ for different penalty multipliers.

r	x_1	x_2	f	ϕ
1	1.905	0.1905	3.992	7.619
10	3.333	0.3333	12.220	13.333
100	3.604	0.3604	14.288	14.144
1000	3.633	0.3633	14.518	14.532

It can be seen that as r is increased the solution converges to the exact solution of x^T = (3.636,0.3636), f = 14.54. The convergence is indicated by the shrinking difference between the objective function and the augmented function ϕ. The Hessian of ϕ is given as:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{2+2r} & \boldsymbol{2r} \\ \boldsymbol{2r} & \boldsymbol{20+2r} \end{bmatrix}$$

As r increases this matrix becomes more and more ill-conditioned, as all four components become approximately 2r. This ill-conditioning of the Hessian matrix for large values of r often occurs when the exterior penalty function is used, and can cause numerical difficulties for large problems.

We can use Table 5.7.1 to test the extrapolation procedure, Eq. (5.7.4). For example, with the values of r = 1 and r = 10, Eq. (5.7.5) gives:

$$a = \frac{0.1x^*(1) - x^*(10)}{-0.9} = \begin{Bmatrix} 3.492 \\ 0.3492 \end{Bmatrix}$$

$$b = x^*(1) - a = \begin{Bmatrix} -0.159 \\ -0.0159 \end{Bmatrix}$$

We can now use Eq. (5.7.4) to find a starting point for the optimization for r = 100 to get:

$$\boldsymbol{a} + \boldsymbol{b}/100 = (3.490,0.3490)T,$$

which is substantially closer to $\mathbf{x}^*(100) = (3.604,0.3604)^T$ than to $\mathbf{x}^*(10) = (3.333, 0.3333)^T$.

5.7.2 Interior and Extended Interior Penalty Functions:

With the exterior penalty function, constraints contribute penalty terms only when they are violated. As a result, the design typically moves in the infeasible domain. If the minimization is terminated before r becomes very large (for example, because of shortage of computer resources) the resulting designs may be useless. When only inequality constraints are present, it is possible to define an interior penalty function that keeps the design in the feasible domain. The common form of the interior penalty method replaces the inequality constrained problem:

$$\textbf{Minimize} \quad f(\boldsymbol{x})$$

$$\textbf{such that} \quad gj(\boldsymbol{x}) \geq 0, j = 1, \ldots, ng, \tag{5.7.10}$$

by,

$$minimize \quad \phi(x,r) = f(x) + r\sum_{j=1}^{n_g} {}^{1}/_{g_j(x)}, \tag{5.7.11}$$

$$r = r_1, r_2, \ldots, \qquad r_i \to 0, \quad r_i > 0$$

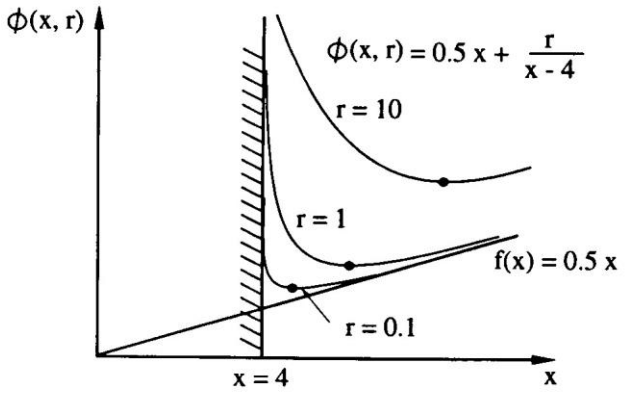


Figure 5.7.2 Interior penalty function for $f(x) = 0.5x$ subject to $x - 4 \geq 0$.

The penalty term is proportional to $1/g_j$ and becomes infinitely large at the boundary of the feasible domain creating a barrier there (interior penalty function methods are sometimes called barrier methods). It is assumed that the search is confined to the feasible domain. Otherwise, the penalty becomes negative which does not make any sense. Figure 5.7.2 shows the application of the interior penalty function to the simple example used for the exterior penalty function in Figure 5.7.1. Besides the inverse penalty function defined in Eq. (5.7.11), there has been some use of a logarithmic interior penalty function:

$$\phi(x,r) = f(x) - r\sum_{j=1}^{n_g} \log\big(g_j(x)\big) \tag{5.7.12}$$

While the interior penalty function has the advantage over the exterior one in that it produces a series of feasible designs, it also requires a feasible starting point. Unfortunately, it is often difficult to find such a feasible starting design. Also, because of the use of approximation (see Chapter 6), it is quite common for the optimization process to stray occasionally into the infeasible domain. For these reasons it may be advantageous to use a combination of interior and exterior penalty functions called an extended interior penalty function. An example is the quadratic extended interior penalty function of Haftka and Starnes [14]:

$$\phi(x,r) = f(x) - r\sum_{j=1}^{n_g} \log\Big(g_j(x)\Big) \tag{5.7.13}$$

$$r = r_1, r_2, \ldots, \qquad r_i \to 0,$$

Where,

$$p\big(g_j\big) = \begin{cases} 1/g_j & for\ g_j \geq g_0 \\ 1/g_0\left[3 - 3\big(g_j/g_0\big) + \big(g_j/g_0\big)^2\right] & for\ g_i < g_0 \end{cases} \tag{5.7.14}$$

It is easy to check that $p(g_j)$ has continuity up to second derivatives. The transition parameter g_0 which defines the boundary between the interior and exterior parts of the penalty terms must be chosen so that the penalty associated with the constraint, $rp(g_j)$, becomes infinite for negative g_j as r tends to zero. This results in the requirement that

$$r/g_0^3 \to \infty \qquad as\ r\ \to\ 0\,. \tag{5.7.15}$$

This can be achieved by selecting g_0 as:

$$g_0\ =\ cr^{1/2}\,, \tag{5.7.16}$$

where c is a constant.

It is also possible to include equality constraints with interior and extended interior penalty functions. For example, the interior penalty function Eq. (5.7.11) is augmented as:

$$\phi(x,r) = f(x) + r\sum_{j=1}^{n_g} 1/g_j(x) + r^{-1/2}\sum_{i=1}^{n_e} h_i^2(x) \qquad (5.7.17)$$

$$r = r_1, r_2, \ldots, \qquad r_i \to 0.$$

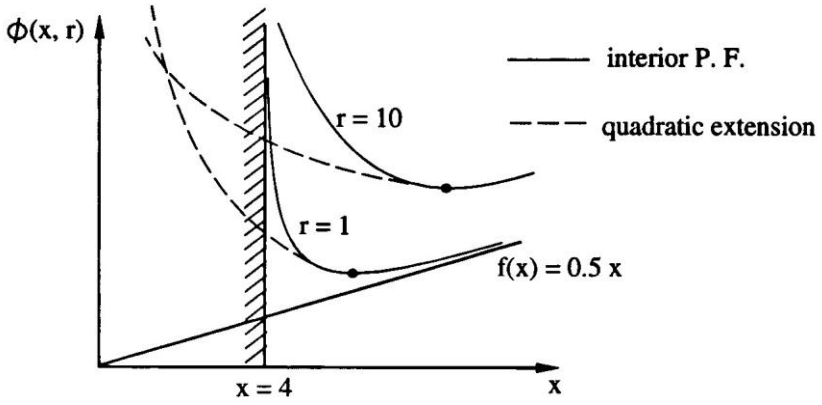


Figure 5.7.3 Extended interior penalty function for f(x) = 0.5x subject to g(x) = x − 4 ≥ 0.

The considerations for the choice of an initial value of *r* are similar to those for the exterior penalty function. A reasonable choice for the interior penalty function would require that *n*/4 active constraints at $g = 0.5$ (that is 50% margin for properly normalized constraints) would result in a total penalty equal to the objective function.

Using Eq. (5.7.11) we obtain: $f(\mathbf{x}) = \frac{n}{4}\frac{r}{0.5}$

$$f(x) = \frac{n}{4}\frac{r}{0.5} \quad , \quad or \;\; r = 2f(\boldsymbol{x})/n\,.$$

For the extended interior penalty function it is more reasonable to assume that the *n*/4 constraints are critical ($g = 0$), so that from Eq. (5.7.13)

$$f(x) = r\frac{n}{4}\frac{3}{g_0} \;\; , \;\; or \;\; r = \frac{4}{3}g_0 f(x)/n.$$

A reasonable starting value for g_0 is 0.1. As for the exterior penalty function, it is possible to obtain an expression for the asymptotic (as $r \to 0$) coordinates of the minimum of ϕas [10].

$$\boldsymbol{x}^*(r) = \boldsymbol{a} + \boldsymbol{b}r^{1/2}, \qquad as\, r \to 0, \qquad (5.7.18)$$

and

$$f^*(r) = a + br^{1/2}, \qquad as\, r \to 0\,.$$

a, **b**, *a* and *b* may be estimated once the minimization has been carried out for two values of *r*. For example, the estimates for **a** and **b** are:

$$a = \frac{c^{1/2}x^*(r_{i-1}) - x^*(r_i)}{c^{1/2} - 1}, \qquad (5.7.19)$$

$$b = \frac{x^*(r_{i-1}) - a}{r_{i-1}^{1/2}},$$

where $c = r_i/r_{i-1}$. As in the case of exterior penalty function, these expressions may be used for convergence tests and extrapolation.

5.7.3 Unconstrained Minimization with Penalty Functions:

Penalty functions convert a constrained minimization problem into an unconstrained one. It may seem that we should now use the best available methods for unconstrained minimization, such as quasi-Newton methods. This may not necessarily be the case. The penalty terms cause the function $\phi$to have large curvatures near the constraint boundary even if the curvatures of the objective function and constraints are small. This effect permits an inexpensive approximate calculation of the Hessian matrix, so that we can use Newton's method without incurring the high cost of calculating second derivatives of constraints. This may be more attractive than using quasi-Newton methods (where the Hessian is also approximated on the basis of first derivatives) because a good approximation is obtained with a single analysis rather than with the n moves typically required for a quasi-Newton method. Consider, for example, an exterior penalty function applied to equality constraints:

$$\phi(x,r) = f(x) + r\sum_{i=1}^{n_e} h_i^2(x) \qquad (5.7.20)$$

The second derivatives of ϕ are given as:

$$\frac{\partial^2\phi}{\partial x_k \partial x_l} = \frac{\partial^2 f}{\partial x_k \partial x_l} + r\sum_{i=1}^{n_e} 2\left(\frac{\partial h_i}{\partial x_k}\frac{\partial h_i}{\partial x_l} + h_i\frac{\partial^2 h_i}{\partial x_k \partial x_l}\right) \quad (5.7.21)$$

Because of the equality constraint, h_i is close to zero, especially for the later stages of the optimization (large r), and we can neglect the last term in Eq. (5.7.21). For large values of r we can also neglect the first term, so that we can calculate second derivatives of ϕbased on first derivatives of the constraints. The availability of inexpensive second derivatives permits the use of Newton's method where the number of iterations is typically independent of the number of design variables. Quasi-Newton and conjugate gradient methods, on the other hand, require a number of iterations proportional to the number of design variables. Thus the use of Newton's method becomes attractive when the number of design variables is large. The application of Newton's method with the above approximation of second derivatives is known as the Gauss-Newton method.

For the interior penalty function we have a similar situation. The augmented objective function $\phi$is given as:

$$\phi(x,r) = f(x) + r\sum_{k=1}^{n_g} 1/g_j(x), \qquad (5.7.22)$$

And the second derivatives are:

$$\frac{\partial^2\phi}{\partial x_k \partial x_l} = \frac{\partial^2 f}{\partial x_k \partial x_l} + r\sum_{i=1}^{n_e}\frac{1}{g_j^3}\left(\frac{\partial g_j}{\partial x_k}\frac{\partial g_j}{\partial x_l} - g_j\frac{\partial^2 g_j}{\partial x_k \partial x_l}\right) \qquad (5.7.23)$$

Now the argument for neglecting the first and last terms in Eq. (5.7.23) is somewhat lengthier. First we observe that because of the $1/g_j^3$ term, the second derivatives are dominated by the critical constraints (g_j small). For these constraints the last term in Eq. (5.7.23) is negligible compared to the first-derivative term because g_j is small. Finally, from Eq. (5.7.18) it can be shown that r/g_j^3 goes to infinity for active constraints as r goes to zero, so that the first term in

Eq. (5.7.23) can be neglected compared to the second. The same argument can also be used for extended interior penalty functions [14].

The power of the Gauss-Newton method is shown in [14] for a high- aspect-ratio wing made of composite materials (see Figure 5.7.4) designed subject to stress and displacement constraints.

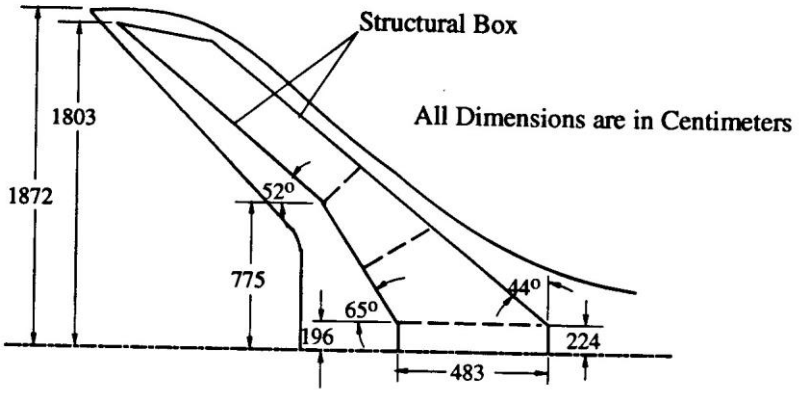


Figure 5.7.4 Aerodynamic planform and structural box for high-aspect ratio wing, from [14].

Table 5.7.2 Results of high-aspect-ratio wing study

Number of design variables	CDC 6600 CPU time sec	Total number of unconstrained minimizations	Total number of analyses	Final mass, kg
13	142	4	21	887.3
25	217	4	19	869.1
32	293	5	22	661.7
50	460	5	25	658.2
74	777	5	28	648.6
146	1708	5	26	513.0

The structural box of the wing was modeled with a finite element model with 67 nodes and 290 finite elements. The number of design variables controlling the thickness of the various elements was varied from 13 to 146. The effect of the number of design variables on the number of iterations (analyses) is shown in Table 5.7.2. It is seen that the number of iterations per unconstrained minimization is almost constant (about five). With a quasi-Newton method that number may be expected to be similar to the number of design variables.

Because of the sharp curvature of ϕ near the constraint boundary, it may also be appropriate to use specialized line searches with penalty functions [15].

5.7.4 Integer Programming with Penalty Functions:

An extension of the penalty function approach has been implemented by Shin et al. [16] for problems with discrete-valued design variables. The extension is based on introduction of additional penalty terms into the augmented-objective function $\phi(\mathbf{x},r)$ to reflect the requirement that the design variables take discrete values,

$$x_i \in X_i = \{d_{i1}, d_{i2}, \ldots, d_{il}\}, i \in I_d\,, \tag{5.7.24}$$

where I_d is the set of design variables that can take only discrete values, and X_i is the set of allowable discrete values. Note that several variables may have the same allowable set of

discrete values. In this case the augmented objective function which includes the penalty terms due to constraints and the non-discrete values of the design variables is defined as:

$$\phi(x,r,s) = f(x) + r\sum_{j=1}^{n_g} p(g_j) + s\sum_{i\epsilon I_d}\psi_d(x_i), \qquad (5.7.25)$$

where s is a penalty multiplier for non-discrete values of the design variables, and $\psi_d(x_i)$ the penalty term for non-discrete values of the ith design variable. Different forms for the discrete penalty function are possible. The penalty terms $\psi_d(x_i)$ are assumed to take the following sine-function form in Ref. [16],

$$\psi_d(x_i) = \frac{1}{2}\left(\sin\frac{2\pi\left[x_i - \frac{1}{4}(d_{i(j+1)} + 3d_{ij})\right]}{d_{i(j+1)} - d_{ij}} + 1\right), \qquad d_{ij} \le x_i \le d_{i(j+1)} \quad (5.7.25)$$

While penalizing the non-discrete valued design variables, the functions $\psi_d(x_i)$ assure the continuity of the first derivatives of the augmented function at the discrete values of the design variables. The response surfaces generated by Eq. (5.7.25) are determined according to the values of the penalty multipliers r and s. In contrast to the multiplier r, which initially has a large value and decreases as we move from one iteration to another, the value of the multiplier s is initially zero and increases gradually.

One of the important factors in the application of the proposed method is to determine when to activate s, and how fast to increase it to obtain discrete optimum design. Clearly, if the initial value of s is too big and introduced too early in the design process, the design variables will be trapped away from the global minimum, resulting in a sub-optimal solution. To avoid this problem, the multiplier s has to be activated after optimization of several response surfaces which include only constraint penalty terms. In fact, since sometimes the optimum design with discrete values is in the neighborhood of the continuous optimum, it may be desirable not to activate the penalty for the non-discrete design variables until reasonable convergence to the continuous solution is achieved. This is especially true for problems in which the intervals between discrete values are very small.

A criterion for the activation of the non-discrete penalty multiplier s is the same as the convergence criterion of Eq. (5.7.6), that is:

$$\left|\frac{\phi - f}{f}\right| \le \epsilon_c \qquad (5.7.27)$$

A typical value for $_c$ is 0.01. The magnitude of the non-discrete penalty multiplier, s, at the first discrete iteration is calculated such that the penalty associated with the discrete-valued design variables that are not at their allowed values is of the order of 10 percent of the constraint penalty.

$$s \approx 0.1rp(g)\,. \qquad (5.7.28)$$

As the iteration for discrete optimization proceeds, the non-discrete penalty multiplier for the new iteration is increased by a factor of the order of 10. It is also important to decide how to control the penalty multiplier for the constraints, r, during the discrete optimization process. If r is decreased for each discrete optimization iteration as in the continuous optimization process, the design can be stalled due to high penalties for constraint violation. Thus, it is suggested that the penalty multiplier r be frozen at the end of the continuous optimization process. However, the nearest discrete solution at this response surface may not be a feasible design, in which case the design must move away from the continuous

optimum by moving back to the previous response surface. This can be achieved by increasing the penalty multiplier, *r*, by a factor of 10.

The solution process for the discrete optimization is terminated if the design variables are sufficiently close to the prescribed discrete values. The convergence criterion for discrete optimization is:

$$\max_{i\in I_i}\left\{min\left\{\frac{|x_i-d_{ij}|}{d_{i(j+1)}-d_{ij}},\frac{|x_i-d_{i(j+1)}|}{d_{i(j+1)}-d_{ij}}\right\}\right\}\leq\epsilon_d, \qquad (5.7.29)$$

where a typical value of the convergence tolerance ϵ_d is 0.001.

5.9 Multiplier Methods

Multiplier methods combine the use of Lagrange multipliers with penalty functions. When only Lagrange multipliers are employed the optimum is a stationary point rather than a minimum of the Lagrangian function. When only penalty functions are employed we have a minimum but also ill-conditioning. By using both we may hope to get an unconstrained problem where the function to be minimized does not suffer from ill-conditioning. A good survey of multiplier methods was conducted by:

Bertsekas [17]. We study first the use of multiplier methods for equality constrained problems.

$$\begin{aligned} &\boldsymbol{Minimize} && f(\boldsymbol{x}) \qquad (5.8.1)\\ &\boldsymbol{such\ that} && h_j(x)=0, \quad j=1,\dots,n_e. \end{aligned}$$

We define the augmented Lagrangian function:

$$\mathcal{L}(x,\lambda,r)=f(x)-\sum_{j=1}^{n_e}\lambda_j h_j(x)+r\sum_{j=1}^{n_e}h_j^2(x) \qquad (5.8.2)$$

If all the Lagrange multipliers are set to zero, we get the usual exterior penalty function. On the other hand, if we use the correct values of the Lagrange multipliers, λ^*_j, it can be shown that we get the correct minimum of problem (5.8.1) for any positive value of *r*. Then there is no need to use the large value of *r* required for the exterior penalty function. Of course, we do not know what are the correct values of the Lagrange multipliers.

Multiplier methods are based on estimating the Lagrange multipliers. When the estimates are good, it is possible to approach the optimum without using large *r* values. The value of *r* needs to be only large enough so that L has a minimum rather than a stationary point at the optimum. To obtain an estimate for the Lagrange multipliers we compare the stationarity conditions for L,

$$\frac{\partial\mathcal{L}}{\partial x_i}=\frac{\partial f}{\partial x_i}-\sum_{j=1}^{n_e}\left(\lambda_j-2rh_j\right)\frac{\partial h_j}{\partial x_i}=0 \qquad (5.8.3)$$

with the exact conditions for the Lagrange multipliers

$$\frac{\partial f}{\partial x_i}-\sum_{j=1}^{n_e}\lambda_j^*\frac{\partial h_j}{\partial x_i}=0 \qquad (5.8.4)$$

Comparing Eqs. (5.8.3) and (5.8.4) we expect that

$$\lambda_j-2rh_j\rightarrow\lambda_j^*, \qquad (5.8.5)$$

as the minimum is approached. Based on this relation, Hestenes [18] suggested using Eq. (5.8.5) as an estimate for λ_j^*. That is:

$$\lambda_j^{(k+1)} = \lambda_j^{(k)} - 2r^{(k)}h_j^{(k)} \tag{5.8.6}$$

where *k* is an iteration number.

Example 5.8.1

We repeat Example 5.7.1 using Hestenes' multiplier method.

$$f(x) = x_1^2 + 10x_2^2,$$
$$h(x) = x_1 + x_2 - 4 = 0.$$

The augmented Lagrangian is:

$$.\mathcal{L} = x_1^2 + 10x_2^2 - \lambda(x_1 + x_2 - 4) + r(x_1 + x_2 - 4)^2$$

To find the stationary points of the augmented Lagrangian we differentiate with respect to x_1 and x_2 to get:

$$2x1 - \lambda + 2r(x1 + x2 - 4) = 0,$$
$$20x2 - \lambda + 2r(x1 + x2 - 4) = 0,$$

which yield;

$$x_1 = 10x_2 = \frac{5\lambda + 40r}{10 + 11r}$$

We want to compare the results with those of Example 5.7.1, so we start with the same initial *r* value $r_0 = 1$, the initial estimate of $\lambda = 0$ and get:

$$\boldsymbol{x_1} = (1.905,0.1905)^T, \qquad h = -1.905.$$

So, using Eq. (5.8.6) we estimate $\lambda^{(1)}$ as

$$\lambda^{(1)} = -2 \times 1 \times (-1.905) = 3.81.$$

We next repeat the optimization with $r^{(1)} = 10$, $\lambda^{(1)} = 3.81$ and get:

$$\boldsymbol{x_2} = (3.492,0.3492)^T, \qquad h = -0.1587.$$

For the same value of *r*, we obtained in Example 5.7.1 $\mathbf{x}_2 = (3.333,0.3333)^T$, so that we are now closer to the exact solution of $\mathbf{x} = (3.636,0,3636)^T$. Now we estimate a new λ from Eq. (5.8.6):

$$\lambda^{(2)} = 3.81 - 2 \times 10 \times (-0.1587) = 6.984.$$

For the next iteration we may, for example, fix the value of *r* at 10 and change only λ. For $\lambda = 6.984$ we obtain:

$$\boldsymbol{x_3} = (3.624,0.3624), \qquad h = -0.0136,$$

which shows that good convergence can be obtained without increasing *r*.

There are several ways to extend the multiplier method to deal with inequality constraints. The formulation below is based on Fletcher's work [19]. The constrained problem that we examine is:

$$minimize \quad f(x)$$
$$such\ that \quad g_j(x) \geq 0, \qquad j = 1, \ldots, n_g. \tag{5.8.7}$$

The augmented Lagrangian function is:

$$\mathcal{L}(x, \lambda, r) = f(x) + r \sum_{j=1}^{n_g} \langle \frac{\lambda_j}{2r} - g_j \rangle^2 \tag{5.8.8}$$

where $< a >= max(a,0)$. The condition of stationarity of L is:

$$\frac{\partial f}{\partial x_i} - 2r \sum_{j=1}^{n_g} \langle \frac{\lambda_j}{2r} - g_j \rangle \frac{\partial g_j}{\partial x_i} = 0 \tag{5.8.9}$$

The exact stationarity condition is:

$$\frac{\partial f}{\partial x_i} - \sum_{j=1}^{n_g} \lambda_j^* \frac{\partial g_j}{\partial x_i} = 0 \tag{5.8.10}$$

where it is also required that $\lambda_j^* g_j = 0$. Comparing Eqs (5.8.9) and (5.8.10) we expect an estimate for λ^*_j of the form:

$$\lambda_j^* = max(\lambda_j - 2rgj, 0)\,. \tag{5.8.11}$$

5.10 Projected Lagrangian Methods (Sequential Quadratic Programming):

The addition of penalty terms to the Lagrangian function by multiplier methods converts the optimum from a stationary point of the Lagrangian function to a minimum point of the augmented Lagrangian. Projected Lagrangian methods achieve the same result by a different method. They are based on a theorem that states that the optimum is a minimum of the Lagrangian function in the subspace of vectors orthogonal to the gradients of the active constraints (the tangent subspace). Projected Lagrangian methods employ a quadratic approximation to the Lagrangian in this subspace. The direction seeking algorithm is more complex than for the methods considered so far. It requires the solution of a quadratic programming problem, that is an optimization problem with a quadratic objective function and linear constraints. Projected Lagrangian methods are part of a class of methods known as sequential quadratic programming (SQP)methods. The extra work associated with the solution of the quadratic programming direction seeking problem is often rewarded by faster convergence.

The present discussion is a simplified version of Powell's projected Lagrangian method [20]. In particular we consider only the case of inequality constraints:

$$\boldsymbol{minimize}\ f(\boldsymbol{x})$$
$$\boldsymbol{such\ that}\ g_j(\boldsymbol{x}) \geq 0, \quad j = 1, \ldots, ng\,. \tag{5.9.1}$$

Assume that at the i^{th} iteration the design is at $\mathbf{x}_i$, and we seek a move direction **s**. The direction **s** is the solution of the following quadratic programming problem:

$$\textbf{Minimize} \qquad \phi(s) = f(x_i) + s^T g(x_i) + \frac{1}{2} s^T A(x_i, \lambda_i) s$$
$$\boldsymbol{Such\ that} \qquad g_j(\boldsymbol{x_i}) + \boldsymbol{s^T} \nabla g_j(\boldsymbol{x_i}) \geq 0, j = 1, \ldots, n_g\,, \tag{5.9.2}$$

where **g** is the gradient of *f*, and **A** is a positive definite approximation to the Hessian of the Lagrangian function discussed below. This quadratic programming problem can be solved

by a variety of methods which take advantage of its special nature. The solution of the quadratic programming problem yields **s** and λ_{i+1}. We then have:

$$x_{i+1} = x_i + \alpha s, \tag{5.9.3}$$

where α is found by minimizing the function:

$$\psi(\alpha) = f(x) + \sum_{j=1}^{n_g} \mu_j \left|\min(0, g_j(x))\right|, \tag{5.9.4}$$

and the μ_j are equal to the absolute values of the Lagrange multipliers for the first iteration, i.e.

$$\mu_j = max\left[\left|\lambda_j^{(i)}, \frac{1}{2}\left(\mu_j^{(i-1)} + \left|\lambda_j^{(i-1)}\right|\right)\right|\right] \tag{5.9.5}$$

with the superscript *i* denoting iteration number. The matrix **A** is initialized to some positive definite matrix (e.g the identity matrix) and then updated using a BFGS type equation (see Chapter 4).

$$A_{new} = A - \frac{A\Delta x \Delta x^T A}{\Delta x^T A \Delta x} + \frac{\Delta l \Delta l^T}{\Delta x^T \Delta x}, \tag{5.9.6}$$

where

$$\Delta x = x_{i+1} - x_i\,, \quad \Delta l = \nabla_x \mathcal{L}(x_{i+1}, \lambda_i) - \nabla_x \mathcal{L}(x_i\,, \lambda_i), \tag{5.9.7}$$

where L is the Lagrangian function and ∇_x denotes the gradient of the Lagrangian function with respect to **x**. To guarantee the positive definiteness of **A**, Δ**l** is modified if $\Delta\mathbf{x}^T\,\Delta\mathbf{l} \leq 0.2\Delta\mathbf{x}^T\,\mathbf{A}\Delta\mathbf{x}$ and replaced by:

$$\Delta l' = \theta \Delta l + (1-\theta) A \Delta x, \tag{5.9.8}$$

Where

$$\theta = \frac{0.8\Delta x^T A \Delta x}{\Delta x^T A \Delta x - \Delta x^T \Delta l}\,. \tag{5.9.9}$$

Example 5.9.1:

Consider the four bar truss of Example 5.1.2. The problem of finding the minimum weight design subject to stress and displacement constraints was formulated as

$$\textbf{Minimize} \qquad f = 3x_1 + \sqrt{3}x_2$$

$$\textbf{subject to}\,, \quad g_1 = 3 - \frac{18}{x_1} - \frac{6\sqrt{3}}{x_2} \geq 0$$

$$g_2 = x_1 - 5.73 \geq 0,$$

$$g_3 = x_2 - 7.17 \geq 0.$$

Assume that we start the search at the intersection of $g_1 = 0$ and $g_3 = 0$ where $x_1 = 11.61$, $x_2 = 7.17$ and $f = 47.25$. The gradient of the objective function and two active constraints are:

$$\nabla f = \begin{Bmatrix} 3 \\ \sqrt{3} \end{Bmatrix}, \quad \nabla g_1 = \begin{Bmatrix} 0.1335 \\ 0.2021 \end{Bmatrix}, \quad \nabla g_3 = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \quad N = \begin{bmatrix} 0.1335 & 0 \\ 0.2021 & 1 \end{bmatrix}$$

We start with **A** set to the unit matrix so that:

$$\phi(s) = 47.25 + 3s_1 + \sqrt{3}s_2 + 0.5s_1^2 + 0.5s_2^2$$

and the linearized constraints are:

$$g_1(s) = 0.1335s_1 + 0.2021s_2 \geq 0,$$
$$g_2(s) = 5.88 + s_1 \geq 0,$$
$$g_3(s) = s_2 \geq 0.$$

We solve this quadratic programming problem directly with the use of the KuhnTucker conditions:

$$3 + s_1 - 0.1335\lambda_1 - \lambda_2 = 0,$$
$$\sqrt{3} + s_2 - 0.2021\lambda_1 - \lambda_3 = 0.$$

A consideration of all possibilities for active constraints shows that the optimum is obtained when only g_1 is active, so that $\lambda_2 = \lambda_3 = 0$ and $\lambda_1 = 12.8$, $s_1 = -1.29$, $s_2 = 0.855$. The next design is:

$$x_1 = \begin{Bmatrix} 11.61 \\ 7.17 \end{Bmatrix} + \alpha \begin{Bmatrix} -1.29 \\ 0.855 \end{Bmatrix}$$

where α is found by minimizing $\psi(\alpha)$ of Eq. (5.9.4). For the first iteration $\mu_j = |\lambda_j|$ so,

$$\psi = 3(11.61 - 1.29\alpha) + \sqrt{3}(7.17 + 0.855\alpha) + 12.8\left|3 - \frac{18}{11.61 - 1.29\alpha} - \frac{6\sqrt{3}}{7.17 + 0.855\alpha}\right|$$

By changing α systematically we find that ψ is a minimum near $\alpha = 2.2$, so that:

$$\boldsymbol{x_1} = (8.77, 9.05)^T, f(\boldsymbol{x_1}) = 41.98, g_1(\boldsymbol{x_2}) = -0.201 .$$

To update **A** we need $\Delta\mathbf{x}$ and $\Delta\mathbf{l}$. We have:

$$\mathcal{L} = 3x_1 + \sqrt{3}x_2 - 12.8\left(3 - 18/x_1 + 6\sqrt{3}/x_2\right),$$

so that,

$$\nabla_x \mathcal{L} = \left(3 - 230.4/x_1^2,\ \sqrt{3} - 133.0/x_2^2\right)^T,$$

and

$$\Delta x = x_1 - x_0 = \begin{Bmatrix} -2.84 \\ 1.88 \end{Bmatrix}, \quad \Delta l = \nabla_x \mathcal{L}(x_1) - \nabla_x \mathcal{L}(x_0) = \begin{Bmatrix} -1.31 \\ 0.963 \end{Bmatrix}$$

With **A** being the identity matrix we have $\Delta\mathbf{x}^T\mathbf{A}\Delta\mathbf{x} = 11.6$, $\Delta\mathbf{x}^T\Delta\mathbf{l} = 5.53$. Because $\Delta\mathbf{x}^T\Delta\mathbf{l} > 0.2\Delta\mathbf{x}^T\mathbf{A}\Delta\mathbf{x}$ we can use Eq. (5.9.5) to update **A:**

$$A_{new} = I - \frac{\Delta x \Delta x^T}{\Delta x^T \Delta x} + \frac{\Delta l \Delta l^T}{\Delta x^T \Delta x} = \begin{bmatrix} 0.453 & 0.352 \\ 0.352 & 0.775 \end{bmatrix}$$

For the second iteration:

$$\phi(s) = 41.98 + 3s_1 + \sqrt{3}s_2 + 0.5(0.453s_1^2 + 0.775s_2^2 + 0.704s_1s_2)$$
$$g_1(s) = -0.201 + 0.234s_1 + 0.127s_2 \geq 0,$$
$$g_2(s) = 3.04 + s_1 \geq 0,$$
$$g_3(s) = 1.88 + s_2 \geq 0.$$

We can again solve the quadratic programming directly with the use of the KuhnTucker conditions:

$$3 + 0.453s_1 + 0.352s_2 - 0.234\lambda_1 - \lambda_2 = 0,$$
$$\sqrt{3} + 0.352s_1 + 0.775s_2 - 0.127\lambda_1 - \lambda_3 = 0.$$

The solution is:

$$\lambda_1 = 14.31, \qquad \lambda_2 = \lambda_3 = 0, \qquad s_1 = 1.059, \qquad s_2 = -0.376 .$$

The one dimensional search seeks to minimize:

$$\psi(\alpha) = f(\alpha) + \mu_1 g_1(\alpha),$$

where:

$$\mu_1 = \max\left(\lambda_1, \frac{1}{2}\left(|\lambda_1| + \mu_1^{old}\right)\right) = 14.31$$

The one-dimensional search yields approximately $\alpha = 0.5$, so that:

$$\boldsymbol{x_2} = (9.30, 8.86)^T, \quad f(\boldsymbol{x_2}) = 43.25, \quad g_1(\boldsymbol{x_2}) = -0.108,$$

so that we have made good progress towards the optimum $\mathbf{x}^* = (9.46, 9.46)^T$.

❖ Chapter Highlights:

- A point is regular if the gradient of active inequality and all equality constraints are linearly independent.
- The optimality conditions for constrained optimization problems are frequently referred to as *Karush–Kuhn–Tucker* (KKT) conditions. KKT conditions are necessary but not sufficient for optimality.
- The Lagrange multiplier provides information on the sensitivity of the objective function with respect to the sensitivity of the righthand side of the constraint equation.
- A constrained optimization problem can be converted to an unconstrained problem by penalizing the objective function when constraints are violated. Such methods are termed *penalty function methods* and are very easy to implement.
- The motivation of using the penalty function method is to solve the constrained optimization problem using algorithms for unconstrained problems.
- The augmented Lagrange multiplier (ALM) method combines both Lagrange multiplier and penalty function methods.
- The sequential quadratic programming (SQP) method approximates the objective function to a quadratic form and linearizes the constraints in each iteration.
- The method of feasible directions ensures meeting the constraints in every iteration.
- In Rosen's gradient projection method, the search direction (negative of the gradient of the objective function) is projected into the subspace tangent of the active constraints.

❖ Formulae Chart:

• Lagrange Function:

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

• Optimality Condition:

$$-\nabla f(x) = \sum_{j=1}^{r} \lambda_j \nabla h_j(x) + \sum_{i=1}^{m} \mu_i \nabla g_i(x)$$

• Penalty Function:

$$f(x) = f(x) + r_k \sum_{j=1}^{r} h_j^2(x) + r_k \sum_{i=1}^{m} \langle g_i(x) \rangle^2$$

$$\langle g_i(x) \rangle = max[0, g_i(x)]$$

• Augmented Lagrangian Function:

$$A(x,\lambda,\beta,r_k) = f(x) + \sum_{j=1}^{r}\lambda_j h_j(x) + \sum_{i=1}^{m}\beta_i\alpha_i + r_k\sum_{j=1}^{r}h_j^2(x) + r_k\sum_{i=1}^{m}\alpha_i^2\,\langle g_i(x)\rangle$$

$$= max[0, g_i(x)]$$

$$\alpha_i = max\left\{g_i(x), \frac{-\beta_i}{2r_k}\right\}$$

• Quadratic Problem:

$$Q = \Delta x^T \nabla f(x) + \frac{1}{2}\Delta x^T \nabla^2 L \Delta x$$

Subject to

$$h_j(x) + \nabla h_j(x)^T \Delta x = 0$$

$$g_i(x) + \nabla g_i(x)^T \Delta x = 0$$

• Rosen's Gradient Projection Method:

$$P = I - N(N^T N)^{-1} N^T$$

$$\bar{x} = x + \alpha s - N(N^T N)^{-1} g_i(x)$$

$$\alpha = -\frac{\gamma f(x)}{S^T \nabla f(x)}$$

❖ **Problems:**

1. Check the nature of the stationary points of the constrained problem:

$$\begin{aligned} \textbf{minimize} \quad & f(x) = x_1^2 + 4x_2^2 + 9x_3^2 \\ \textbf{such that} \quad & x_1 + 2x_2 + 3x_3 \geq 30, \\ & x_2 x_3 \geq 2, \\ & x_3 \geq 4, \\ & x_1 x_2 \geq 0. \end{aligned}$$

2. For the problem:

$$f(\mathbf{x}) = 3x_1^2 - 2x_1 - 5x_2^2 + 30x_2$$

$$\begin{aligned} \textbf{minimize} \quad & f(x) = 3x_1^2 - 2x_1 - 5x_2^2 + 30x_2 \\ \textbf{such that} \quad & 2x_1 + 3x_2 \geq 8, \\ & 3x_1 + 2x_2 \leq 15, \\ & x_2 \leq 5. \end{aligned}$$

Check for a minimum at the following points: (a) (5/3, 5.00) (b) (1/3, 5.00) (c) (3.97,1.55).

1. Calculate the derivative of the solution of Example 5.1.2 with respect to a change inthe allowable displacement. First use the Lagrange multiplier to obtain the derivative of the objective function, and then calculate the derivatives of the design variables and Lagrange multipliers and verify the derivative of the objective function. Finally, estimate from the derivatives of the solution how much we can change the allowable displacement without changing the set of active constraints.
2. Solve for the minimum of problem 1 using the gradient projection method fromthe point (17, 1/2, 4).
3. Complete two additional moves in Example 5.5.2.

4. Find a feasible usable direction for problem 1 at the point (17, 1/2, 4).
5. Use an exterior penalty function to solve Example 5.1.2.
6. Use an interior penalty function to solve Example 5.1.2.
7. Consider the design of a box of maximum volume such that the surface area isequal to S and there is one face with an area of $S/4$. Use the method of multipliers to solve this problem, employing three design variables.
8. Complete two more iterations in Example 5.9.1.

References:

1. Burghes, D. N., & Wood, A. D. (1980). Mathematical models in the social, management and life sciences. Hemstead, UK: Ellis Horwood.
2. 18. Fletcher, R., & Powell, M. J. D. (1963). A rapidly convergent descent method for minimization. The Computer Journal, 6, 163–168.
3. A.L. Tits. Lecture notes on optimal control. University of Maryland, 2013.
4. Abadie, J., and Carpentier, J., "Generalization of the Wolfe Reduced Gradient Method for Nonlinear Constraints", in: Optimization (R. Fletcher, ed.), pp. 37– 49, Academic Press, 1969.
5. Agnew, R. P. (1960). Differential equations. New York: McGraw-Hill.
6. Andreas, A., & Wu-Shey, L. (2007). Practical optimization: Algorithms and engineering applications. New York: Springer Science+Business Media.
7. Arora, R. K., & Pradeep, K. (2003). Aerodynamic shape optimization of a re-entry capsule. In AIAA Atmospheric Flight Mechanics Conference and Exhibit (AIAA-5394-2003, Texas).
8. Ashok, D. B., & Tirupathi, R. C. (2011). Optimization concepts and applications in engineering. Cambridge, UK: Cambridge University Press.
9. Bellman, R. (1953). An introduction to the theory of dynamic programming. RAND Corp. Report.
10. Bertsekas, D.P., "Multiplier Methods: A Survey," Automatica, 12, pp. 133–145, 1976.
11. Betts, J. T. (1998). Survey of numerical methods for trajectory optimization. Guidance, Control and Dynamics, 21(2), 193–207.
12. Byrd, R. H., Gilbert, J. C., & Nocedal, J. (2000). A trust region method based on interior point techniques for nonlinear programming. Mathematical Programming, 89(1), 149–185.
13. Byrd, R. H., Schnabel, R. B., & Shultz, G. A. (1988). Approximate solution of the trust region problem by minimization over two-dimensional subspaces. Mathematical Programming, 40(3), 247–263.
14. Cauchy, A. L. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. Comptes Rendus de l'Académie des Sciences, 25, 536–538.
15. Coleman, T. F., & Verma, A. (2001). A preconditioned conjugate gradient approach to linear equality constrained minimization. Computational Optimization and Applications, 20(1), 61–72.
16. Colville, A. R. (1968). A comparative study on nonlinear programming codes. Report 320-2949, IBM New York Scientific Centre.
17. Dahlquist, G., and Bjorck, A., Numerical Methods, Prentice Hall, 1974.
18. Dantzig, G. B. (1949). Programming of interdependent activities: II mathematical model. Econometrica, 17(3), 200–211.
19. Dantzig, G. B. (1990). The diet problem. Interfaces, 20(4), 43–47.
20. Deb, K. (1995). Optimization for engineering design: Algorithms and examples. Upper Saddle River, NJ: Prentice Hall.
21. Dennis, J. E., & Schnabel, R. B. (1983). Numerical methods for unconstrained optimization and nonlinear equations. Upper Saddle River, NJ: Prentice Hall.
22. Epperson, J. F. (2010). An introduction to numerical methods and analysis. Hoboken, NJ: John Wiley & Sons.
23. Fiacco, V., and McCormick, G.P., Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley, New York, 1968.

24. Fletcher, R., & Reeves, C. M. (1964). Function minimization by conjugate gradients. The Computer Journal, 7(2), 149–154.
25. Fletcher, R., "An Ideal Penalty Function for Constrained Optimization," Journal of the Institute of Mathematics and its Applications, 15, pp.319–342, 1975.
26. Gill, P.E., Murray, W., and Wright, M.H., Practical Optimization, Academic Press, 1981.
27. Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Society, 64, 275–278.
28. Griva, I., Nash, S. G., & Sofer, A. (2009). Linear and nonlinear optimization. Philadelphia: SIAM.
29. Haftka, R.T., and Starnes, J.H., Jr., "Applications of a Quadratic Extended Interior Penalty Function forStructural Optimization", AIAA Journal, 14 (6), pp.718–724, 1976.
30. Hancock, H. (1917). Theory of maxima and minima. Boston: Ginn and Company.
31. Haug, E.J., and Arora, J.S., Applied Optimal Design: Mechanical and Structural Systems, John Wiley, New York, 1979.
32. Hestenes, M.R., "Multiplier and Gradient Methods," Journal of Optimization Theory and Applications, 4 (5), pp. 303–320, 1969.
33. Jaluria, Y. (2008). Design and optimization of thermal systems. Boca Raton, FL: CRC Press.
34. Kantorovich, L. V. (1939). Mathematical methods of organizing and planning production. Leningrad State University.
35. Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints. M.Sc. Dissertation. Illinois: University of Chicago.
36. King, J. R. (1975). Production, planning and control: An introduction to quantitative methods. Oxford: Pergamon Press.
37. Kreisselmeier, G., and Steinhauser, R., "Systematic Control Design by Optimizing a Vector Performance Index,"Proceedings of IFAC Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 113-117, 1979.
38. Kuhn, H. W., & Tucker, A. W. (1951). Nonlinear programming. Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, 481–492.
39. Marquardt, D. (1963). An algorithm for least squares estimation of nonlinear parameters. SIAM Journal of Applied Mathematics, 11(2), 431–441.
40. Moe, J., "Penalty Function Methods in Optimum Structural Design—Theory and Applications", in: Optimum Structural Design (Gallagher and Zienkiewicz, eds.), pp. 143–177, John Wiley, 1973.
41. Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. Computer Journal, 7(2), 308–313.
42. Nocedal, J., & Wright, S. J. (2006). Numerical optimization. Springer Series in Operations Research. New York: Springer Science+Business Media.
43. Powell, M.J.D., "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations", Proceedings of the 1977 Dundee Conference on Numerical Analysis, Lecture Notes in Mathematics, Vol. 630, pp. 144–157, Springer-Verlag, Berlin, 1978.
44. Rao, S. S. (2009). Engineering optimization: Theory and practice. Hoboken, NJ: John Wiley & Sons.
45. Reklaitis, G. V., Ravindran, A., & Ragsdell, K. M. (1983). Engineering optimization: Methods and applications. New York: John Wiley & Sons.
46. Rosen, J.B., "The Gradient Projection Method for Nonlinear Programming— Part I: Linear Constraints", The

Society for Industrial and Appl. Mech. Journal, 8 (1), pp. 181– 217, 1960.

47. Rosen, J.B., "The Gradient Projection Method for Nonlinear Programming—Part II: Nonlinear Constraints", The Society for Industrial and Appl. Mech. Journal, 9 (4), pp. 514–532, 1961.
48. S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge University Press, http://stanford.edu/ boyd/cvxbook/, 2004.
49. Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. Mathematics of Computation, 24, 647–656.
50. Shin, D.K, Gu¨rdal, Z., and Griffin, O. H. Jr., "A Penalty Approach for Nonlinear Optimization with Discrete Design Variables," Engineering Optimization, 16, pp. 29–42, 1990.
51. Sobieszczanski-Sobieski, J., "A Technique for Locating Function Roots and for Satisfying Equality Constraints in Optimization," NASA TM-104037, NASA LaRC, 1991.
52. Sobieszczanski-Sobieski, J., Barthelemy, J.F., and Riley, K.M., "Sensitivity of Optimum Solutions of Problem Parameters", AIAA Journal, 20 (9), pp. 1291– 1299, 1982.
53. Vanderplaats, G.N., "CONMIN—A Fortran Program for Constrained Function Minimization", NASA TM X-62282, 1973.
54. Venkataraman, P. (2009). Applied optimization with MATLAB programming. Hoboken, NJ: John Wiley & Sons.
55. Wolfe, P.. "The Simplex Method for Quadratic Programming," Econometrica, 27 (3), pp. 382–398, 1959.
56. Zoutendijk, G., Methods of Feasible Directions, Elsevier, Amsterdam, 1960.