Republic of Iraq
Ministry of Higher Education & Scientific Research
University of Babylon
College of Education for Pure Sciences
Department of Mathematics

Linear Programming Design Optimization Coding In Python

A Thesis

Submitted to the Council of College of Education for Pure Sciences, University of Babylon in Partial Fulfillment of the Requirements for the Degree of Master in Education / Mathematics.

By

Nadia Ali Abbas Hussein

Supervised by

Assist Prof. Dr. Ahmed Sabah Al-Jilawi

2022 A.D. **1444 A.H.**

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنْكُمْ
وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ
وَاللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ

صَدَقَ اللهُ العَظِيمُ

المجادلة الاية 11

The Supervisor Certificate

I certify that this thesis entitled " Linear Programing Design Optimization Coding In Python" was prepared by the student " Nadia Ali Abbas Hussain " under my supervision at the University of Babylon, College of Education for Pure Sciences as a partial fulfillment of the requirement the Degree of Master in Education /Mathematics.

Signature: 

Name: Dr Ahmed Sabah Al-Jilawi

Scientific Grade: Assistant Professor

Date: / / 2022

According to the available recommendation, I forward this thesis for debate by the examining committee.

Signature: 

Name: Assist proof.Dr.Azal Jaafar Musa

Scientific Grade:Assist Professor

Address:Head of Mathematics Department

Date: / / 2022

Scientific Supervisor's Certification

This is to certify that I have read this thesis entitled " Linear Programing Design Optimization Coding In Python " and I found it is qualified for debate.

Signature: 

Name: Dr.Hussam Abid Ali Mohammed

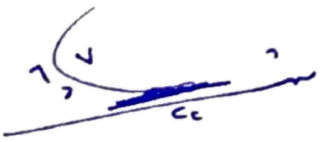
Title:Assist Professor

Address: College of Education for Pure Sciences, University of Karbala

Date: / / 2022

Scientific Supervisor's Certification

This is to certify that I have read this thesis entitled " Linear Programing Design Optimization Coding In Python " and I found it is qualified for debate.

Signature: 

Name: Dr.Arshed Adham Ahmad

Title: Assist Professor

Address:College of Education for Pure Sciences, University of Diyala

Date: / / 2022

Linguistic Supervisor's Certification

This is to certify that I have read this thesis entitled " Linear Programing Design Optimization Coding In Python" and I found it is qualified for debate.

Signature: 

Name: Dr.Aseel kadhim Alrikabi

Address: English Department, University of Babylon

Title: Assist Professor

Date: / / 2022

Examining Committee Certificate

We certify that we have read this thesis entitled " Linear Programing Design Optimization Coding In Python " as an examination committee, examined the student " Nadia Ali Abbas Hussain " in its contents and that in our opinion it meets the standard of a dissertation for the Degree of PhD in Education/ Mathematics.

Signature:
Name:
Title:
linear programing design optimization coding in python Date : / / 2022
Date : / / 2022
Chairman

Signature:
Name :
Title:
Member

Signature:
Name:
Title:
Date : / / 2022
Member

Signature:
Name :
Title:
Date : / / 2022
Member / Advisor

Approved by the Dean of College

Signature:
Name: Dr. Bahaa Hussein Salih Rabee
Title:Professor
Address: Dean of the College of Education for Pure Sciences
Date : / / 2022

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

Linear and nonlinear programming are basically a subfield of numerical optimization, linear and nonlinear optimization is a mathematical programming technique to find an optimal value for linear and nonlinear objective functions. The objective of this study is design coding which is an important tool for modeling numerical optimization fields. This study investigated and proposed a python program for general optimization modeling linear and nonlinear optimization. Our concept provides the basic foundation for learning and improving mathematical problem-solving abilities using Python Language. We provide new approaches and algorithms and created different mathematical modeling applications for optimization. Numerical and application results gave a good feasibility area from which the optimal solution was derived. This study developed design codes based on different mathematical models which included, Tyler's rounding technique, linear programming, nonlinear programming, double linear, nonlinear, Lagrange multiplier, Karush-Kuhn-Tucker(KKT) condition, and penalty (internal and external penalty), all of them provided efficient algorithms An open source packages that make mathematical programs to be described in the Python programming language.

Dedication

To whom God has entrusted with prestige and dignity.. To my role model in life, to the one who instilled in him the love of science, to the one who surrounded me with the wall of his tenderness .. To whom I carry his name with pride.. I hope that God will extend your age to see fruits whose harvest has come after a long wait, and your words will remain stars guided by them today, tomorrow and forever

(To My Dear Father)

To my angel in life.. To the meaning of love, to the meaning of tenderness and devotion.. To the grace of life The secret of existence.. To whom her prayer was the secret of my success and her tenderness is a surgical balm to the most expensive beloved

(My beloved Mom)

To those who are big and have to rely on them .. To a burning candle that illuminates the darkness of my life

(My Dear Sister and My Dear Brother)

to those in whose presence I gain boundless strength and love ...To those with whom I knew the meaning of life, to those who stood by me and supported me

(My Husband and Children)

Nadia Ali

Acknowledgements

Praise be to God Almighty, who gave me the grace and beauty of his talent and guided me to follow the path of knowledge and knowledge, and inspired me to study these great sciences and helped me in that.I would also like to thank the Prophet Muhammad and the Ahl al-Bayt (may God bless them and grant him peace). From which doors of Praise will we enter, which poetic verses will we pass, I was like a cloud of his tender that watered the Earth and made it green , I was and still am like a towering palm tree , infinitely thanking dad and mom who have watched my upbringing and education since my life began I also extend my sincere thanks and gratitude to Dr. Ahmed Sabah Al-Jilawi, who kindly accepted the supervision of the master's thesis, and who gave me his valuable time and his vast knowledge and experience, which was a great addition to the research work, as his guidance and advice was a beacon that I used in my entire research work, so I ask May God Almighty reward him with the best reward.

Table 1: **Notations used in the Thesis**

$f(x)$	The Objective function
$Minf(x)$	The Minimum of Objective Function
$Maxf(x)$	The Maximum of Objective Function
$g(x)$	The Inequality Constraint
$h(x)$	The Equality Constraint
Lpp	linear programming problem
Lp	linear programming
∇	gradient
C	The convex set
$\bigtriangledown f$	The Gradient of f
$\bigtriangledown^2 f$	The Hessian of f
KKT	Karush-Kuhn-Tucker Conditions
L	The Lagrangian Function
$P(x)$	The Penalty Function
$B(x)$	The Interior Penalty Function
$E(x)$	The Exterior Penalty Function
CP	constraint programming

CHAPTER 1

INTRODUCTION

1.1 Introduction

Optimization theory and methods are a relatively new branch of applications mathematics, computational mathematics and operations research with a wide range of applications in science, engineering, corporate management, military and space exploration. The subject is interested in finding the best solution to theoretically detailed difficulties, that is, the "optimal" solution to a real-world problem using the scientific methods, and the problem can be detected from different strategies as well as the tools used to investigate the criteria of the ideal state, thus creating typical problems, as well as the selection of a computational approach to the solution and the development of the theory of algorithmic convergence and numerical experiments with real-world challenges and typical problems although optimization may be due to the first days of maximum value cases [23]. In the late 1940s, Danzig was the first to make it a separate subject. He was given a well-known simple method of linear programming. After conjugated gradient techniques and quasi-Newtonian methods became popular in 1950, when these ideas were introduced, nonlinear programming grew . Modern optimization methods have become a vital tool for solving problems in a variety of situations [64]. Optimization is an activity to which no particular discipline belongs. Optimization is described as determining the best suitable solution in the theory of performing the optimization task in an optimal math problem studies have put a great deal of effort into trying to describe the properties of such an ideal solution [44].

1.2 Related Work

1. Optimization Problem
2. Linear Programming Problem
3. The General Linear programming Problem
4. Nonlinear Programming Problem
5. Mathematical Modeling
6. Design Coding
7. The relation between Optimization and Design Coding
8. Python language
9. Python Optimization Code
10. Classification of mathematical optimization models

1.2.1 Optimization Problem

In the function of the optimization problem, the target must be enlarged or reduced. Improved functionality is referred to as functionality. Function is a quantity like Cost, profit, efficiency, size, shape and weight. Developed with Various criteria to achieve improved efficiency in reducing the energy consumption needs of the user.Design variables are the variables in the objective function [56]. The following is a mathematical expression for the optimization problem:

$$(P)\begin{cases} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) < 0 \quad i = 1, 2, \ldots\ldots\ldots, m \\ & h_j(x) = 0 \quad j = 1, 2, \ldots\ldots\ldots, r \\ & x_1 < x < x_n \end{cases} \tag{1.1}$$

where x is a vector of n design variables given by

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

The goal of an optimization problem is to find the optimal solution out of all the possibilities [45]. Optimization problems can be divided into two categories, depending on whether the variables are continuous or discrete [35]:

1. The optimization problem with discrete variables is known as a discrete optimization, in which an object such as an integer, permutation or graph must be found from a countable set.

2. Continuous optimization refers to a problem with continuous variables in which the optimum value of a continuous function must be determined. These may involve limited and multimodal problems.

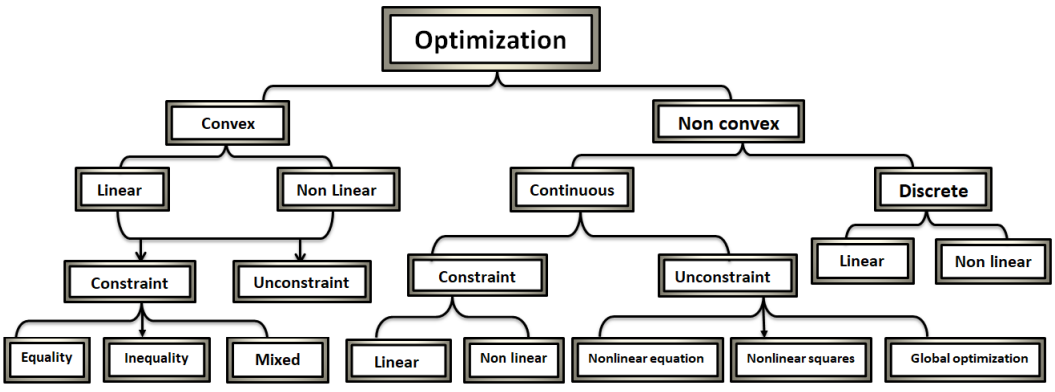


Figure 1.1: The diagram is described Classification of Optimization

1.2.2 Optimization Models

Models of Optimization is the key aspects of the business challenge are trying to solve and translated into an optimization model. The objective function, decision variables, and business constraints are the three components of the model [41]. We refer to the application of mathematical tools to solve issues based on particular features as optimization modeling.

- Programming in linear form (LP)
- Programming with mixed integers (MIP)
- Nonlinear programming is a type of programming that is used to solve (NLP)
- Programming with constraints is a type of constraint programming (CP)

Many fields of study employ optimization methods to find solutions that maximize or minimize certain study parameters, such as minimizing expenses in the manufacture of a thing or service, maximizing earnings, minimizing raw materials in the development of a good, or maximizing productivity

Optimization Model Works

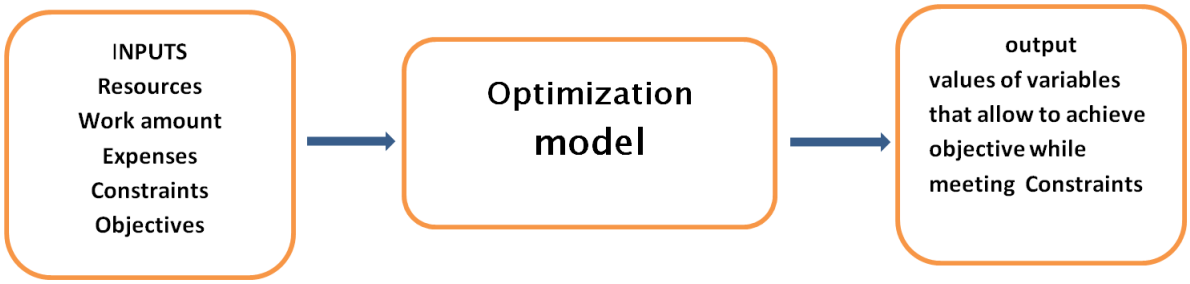


Figure 1.2: The diagram is described Inputs and outputs of the optimization model The output of the optimization model is the values the variables that allow you to reduce or increase the objective function while still meeting the constraints

1.2.3 Linear Programming Problem

Linear programming(LP) was created during World War II, when the need for a method to optimize resource efficiency was critical. New war-related initiatives needed

attention, causing resources to be stretched thin [31]. Programming was a word used in the military to describe operations such as schedule preparation.Efciently or optimally deploying George Dantzig is a member of the United States Congress.In 1947, the Air Force devised the Simplex technique of optimization to solve a problem.after using approach for tackling linear programming problems structures. The idea underlying it has been created by economists and sociologists [49].As well as considering its possible applications, linear programming. The expression being optimized in linear programming z is referred to as the objective function. Decision variables are the variables $x_1, x_2 \ldots\ldots x_n$ are called decision variables, and their values are subject to m constraints (every line ending with a bi, plus the non negativity constraint). A set of $x_1; x_2 \ldots x_n$ satisfying all the constraints is called a feasible point and the set of all such points is called the feasible region. The solution of the linear program must be a point $(x_1, x_2, \ldots.., x_n\ )$ in the feasible region, or else not all the constraints would be satisfied [48]. When aircraft crews are scheduled, or factory managers calculate the raw material mix that generates the most lucrative blend of output items, they are sometimes faced with difficulties with hundreds or thousands of variables and constraints [56]. Linear programming is an optimization problem in which the goal and constraints are expressed as a linear function of the design variables. The constraints may be equality, inequality, or both types. A linear function has the following mathematical features:

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

$$f(kx) = kf(x)$$

Where x and y are variables,and k is a scalar. A real linear programming problem (LPP) may include hundreds of design variables and constraints, necessitating the use of different solution strategies.

Linear Programming divided into Three Parts [46] There are three parts to linear programming:

1. A decision variable is a set of variables over which the decision-maker has direct influence.

2. The goal function is a linear function that mathematically determines the quantities to be maximized or minimized.

3. Constraints: a mathematical formulation of equalities or inequalities that represents the boundaries of the choice variables.

1.2.4 The General Linear Programming in Optimization

The linear programming problem structure may be condensed to the following form. [37].

$$\begin{cases} \text{Minimize} \quad c_1x_1 + c_2x_2 + ... + c_nx_n = z \\ \text{Subject to} \quad a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ \qquad\qquad\quad a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ \qquad\qquad\quad . \\ \qquad\qquad\quad . \\ \qquad\qquad\quad a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \\ \qquad\qquad\quad x_1, x_2, ..., x_n \geq 0 \end{cases} \tag{1.2}$$

The goal function in linear programming is the expression that is being optimized. The variables $x_1, x_2, ..., x_n$ are known as choice variables. A feasible point is a collection of $x_1, x_2, ..., x_n$ that fulfill all of the requirements, and the feasible region is the set of all such points. The linear program's solution must be a point $(x_1, x_2, ..., x_n)$ in the feasible area, otherwise all of the constraints will not be met.

1.2.5 Non-linear Programming Problem [42]

When the goal or constraint functions are not linear but not known to be convex, non-linear optimization (or nonlinear programming(NLP)) is employed to define the problem . Nonlinear problems can be categorized according to several properties. There are problems in which the objective and constraints are smooth functions, and there are nonsmooth problems in which the slope or value of a function may change abruptly. There are unconstrained problems, in which the aim is to minimize (or maximize) the objective function f(x) with no restrictions on the value of x, and there are constrained problems, in which the components of x must satisfy certain bounds or other more complex interrelationships.

1.2.6 Mathematical Modeling [52]

Models describe we beliefs about how the world functions. In mathematical modelling, we translate those beliefs into the language of mathematics . This has many advantages

1. Mathematics is a very precise language. This helps us to formulate ideas and identify underlying assumptions.
2. Mathematics is a concise language, with well-defined rules for manipulations.
3. Computers can be used to perform numerical calculations.

In mathematical modeling, there is a lot of room for compromise. The bulk of real-world interacting systems are just too complex to represent in their entirety. As a result, the first degree is to identify the system's most critical components. These will be included in the final product. The remainder will be left out of the model. Second degree is the quantity of mathematical knowledge required. Despite the fact that mathematics has the ability to verify broad conclusions, The form of equations utilized has a significant impact on the findings .

1.2.7 Design Coding

The fundamental structural design concept Using trial-and-error, find the optimal design by repeatedly adjusting the design variables. approach based on the designer's expertise, subject to structural criteria [60].How can we come up with the finest design?
If the present design isn't working, what's the best way to change the design variables? isn't it the best?
These are questions that optimization can solve:
In the conventional design process, the design variables are assigned arbitrary beginning (trial) values, structural analysis is used to evaluate the responses, and the if design criteria are not met, the variables are intuitively changed [33]. i.e., there is no standard approach or framework for making design changes. Furthermore, If all of the needs have been met and there are no more requirements to be met, the design process is often concluded .There is a concerted attempt to discover better solutions.

1.2.8 The Relation between Optimization and Design Coding

In the case of code-based design, where the design code determines the upper and lower limits of answers as well as the bounds of design variables, structural optimization offers the following advantages: If the problem is correctly framed such that the set of feasible designs is nonempty, a design fulfilling all the constraints (feasible design) may be identified automatically and efficiently while simultaneously reducing the objective function, such as total structural volume [33]. The optimization tool aids the designer's decision-making; it is not an automated design tool that gives structural engineers and designers a bad image. If the optimum design is not acceptable to the designer, the upperlevel characteristics of the structure, such as its geometry and material qualities, may be changed, or new constraints can be applied to the optimization problem to produce a more plausible optimal solution. As a result, rather than structural analysis, the designers may devote more time on the synthetic work. if decision-making

optimization techniques are employed effectively digram depicts the traditional design process and the optimization-based design approach.

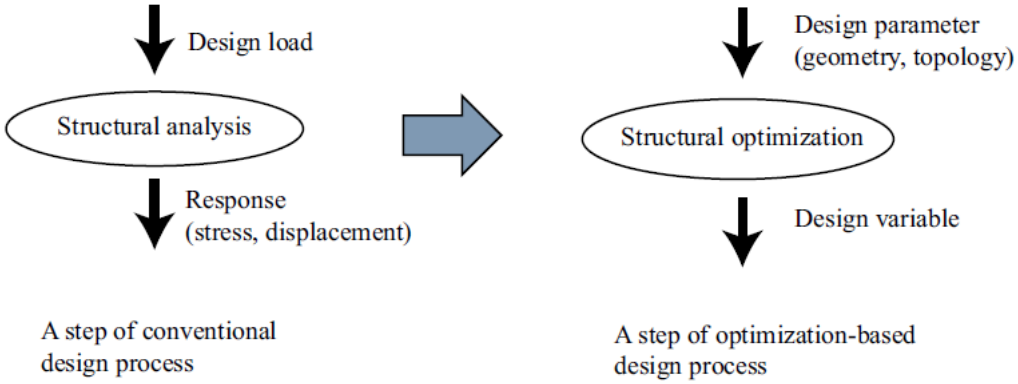


Figure 1.3: The diagram is described Illustration of conventional and optimization-based design processes.

1.2.9 Python language

Python is easy to learn high level programming language. Python is a flexible language that can be used to manage the performance of many programs in general, as well as to construct independent programs with graphical interfaces and in web applications. It can also be used as a scripting language to control the performance of many programs . Python is a programming language that may be used to create basic programs for novices as well as to complete huge projects simultaneously.Guido van Rossum created Python at the Institute of Mathematics and Informatics in Amsterdam in the late 1980s, and it was initially released in 1991 [62]. C is used to create the language kernel. Rossum called his language Python in honor of the iconic British show Monty Python and the Holy Grail.Python is the first language on the list of artificial intelligence and machine learning languages, due to its ease of learning and its ability to implement many artificial intelligence algorithms. One of the most popular Python libraries used in this field is the library (skipy),(NumPy),(Pandas) [11].For many data science applications, Python has become the de facto language franca. It combines the

flexibility of general-purpose programming languages with the simplicity of domain-specific scripting languages like as MATLAB or R. Data loading, visualization, analytics, natural language processing, image processing, and more are all supported by Python libraries. This comprehensive toolkit offers data scientists with a wide range of general- and special-purpose capabilities [67]. One of the most compelling features of Python is the ability to interact directly with the code through a terminal or other tools [54].

1.2.10 Python Optimization Code

The process of optimizing Python programs includes choosing the optimal solution from a variety of possibilities available to developers. Python is the most widely used, dynamic, and versatile programming language for web development [47]. Python is still the greatest and most relevant programming language for application developers, even in tasks like machine learning and data mining.

Python code optimization is a Technique to make the program work any action more efficiently and fast while using less lines of code, memory, or other resources while still producing the desired outputs. It's critical while doing a task that requires processing a high number of actions or data. As a result, changing and improving some inefficient code blocks and features might be quite beneficial: Improve the application's speed make the code more legible Error tracking and debugging are made easier. Save a significant amount of processing power [2].

1.2.11 Classification of Mathematical Optimization Models

The operator varies based on the item for which the model is being utilized.If the operator of a model employs linear relationships between the model's outputs and inputs, the model is said to be linear. These models are perhaps the most straightforward to analyze. This also brings us to the end of another key property of linear models: superposition. To put it another way, if the model response for two sets of inputs x_1 and x_2 is known and equals

y_1 and y_2, the model response for the total of inputs $x_1 + x_2$ will be precisely the sum of outputs $y_1 + y_2$. Non-linear models are those in which the operator employs non-linear output-input dependencies. When compared to linear models, they are more complicated to analyze and lack the superposition characteristic.The models may be classed as basic or sophisticated depending on the operator. The model is considered to be simple if the operator is a compact dependence of outputs from inputs (like a direct formula) [3]. The model is considered to be complicated if the model's operator needs extra operations to determine the relationships between outputs and inputs, as if it were described as a collection of differential equations.It's worth noting that complicated models may be reduced to simple if a set of equations can be solved analytically, either exactly or roughly with a reasonable degree of accuracy.The model is considered to be algorithmic if the model's operator is represented in algorithmic form, as a series of steps that link inputs to outputs.The model operator is used to classify mathematical models like the diagram below [47].

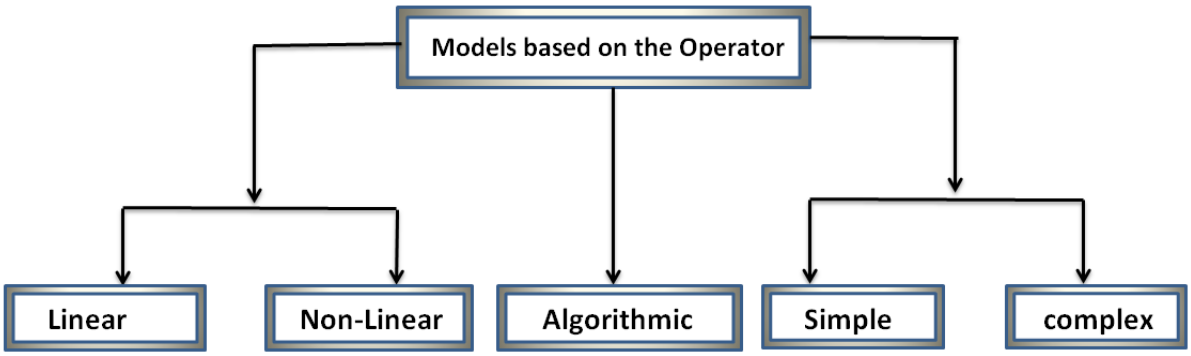


Figure 1.4: The digram shows the mathematical models based on the model parameters with the modelling purpose in mind

CHAPTER 2

MATHEMATICAL BACKGROUND

In this chapter, we discuss some of the mathematical concepts that will be needed later.

Definitions and Properties

This section, we provides definitions, notation, and necessary results of Euclidean space. Also, it presents some basic concepts and facts of real mathematical analysis , convexity ,optimality condition, kkt condition.

2.1 Linear Programming

The basic linear programming problem lends itself to a linear interpretation based on algebra , the basic problem model is: [49]:

$$\begin{cases} \min(\max)\ \langle\ c, y\rangle = \sum_{i=1}^{n} c_i y_i \\ s.t \qquad uy = b \rightarrow Equality \\ \qquad\quad uy\ \ \geq b \Longrightarrow inequility \\ \qquad\quad\ y \in R \end{cases} \tag{2.1}$$

The problem is equality

$$\begin{cases} \text{minimize,maximize} & c_1 y_1 + c_2 y_2 + \ldots\ldots\ldots\ldots + c_n y_n = z \\ \text{subject to} & u_{11} y_1 + u_{12} y_2 + \ldots\ldots\ldots + u_{1n} y_n = b_1 \\ & u_{21} y_1 + u_{22} y_2 + \ldots\ldots\ldots + u_{2n} y_n = b_2 \\ & \quad . \\ & u_{m1} y_1 + u_{m2} y_2 + \ldots\ldots\ldots + u_{mn} y_n = bm \\ & y_1, y_2, \ldots\ldots\ldots, y_n \geq 0 \end{cases} \tag{2.2}$$

The problem is inequality

$$\begin{cases} \text{minimize} & c_1y_1 + c_2y_2 + \ldots\ldots\ldots\ldots + c_ny_n = z \\ \text{subject to} & u_{11}y_1 + u_{12}y_2 + \ldots\ldots\ldots + u_{1n}y_n \geq b_1 \\ & u_{21}y_1 + u_{22}y_2 + \ldots\ldots\ldots + u_{2n}y_n \geq b_2 \\ & . \\ & u_{m1}y_1 + u_{m2}y_2 + \ldots\ldots\ldots + u_{mn}y_n \geq b_m \\ & y_1, y_2, \ldots\ldots\ldots, y_n \geq 0 \end{cases} \tag{2.3}$$

2.2 Nonlinear Programming (NLP)

Consider the non-linear programming problem below [8]:

$$\begin{cases} \min & f(x) \\ s.t & g_j(x) \leq 0 \quad j = 1, 2, \ldots, m \\ & b_i(x) = 0 \quad i = 1, 2, \ldots\ldots\ldots k \\ & x \geq 0 \end{cases} \tag{2.4}$$

Where f, $g_1 \ldots\ldots\ldots, g_m, b_1, \ldots\ldots\ldots, b_k$ are functions defined on R^n, x is a subset of R^n, and x is a vector of n components $x_1, \ldots\ldots, x_n$. The above problem must be solved for the values of the variables $x_1, \ldots\ldots, x_n$, that satisfies the constraints while minimizing the function f. The function f is usually called the objective function, or the criterion function. Each of the constraints $g_j(x) \leq 0$ for $j = 1, \ldots, \mathrm{m}$ is called an inequality constraint, and each of the constraints $b_i(x) = 0 \quad i = 1, 2, \ldots\ldots\ldots$k is called an equality constraint. Lower and upper boundaries on variables are often included in the set x, which, even if inferred by the other restrictions, may be beneficial in particular algorithms [28]. Alternatively, this collection might represent some carefully constructed restrictions that are emphasized for the optimization method to exploit, or it could represent some regional confinement or other complex constraints that are handled independently through an

unique mechanism. [29].

2.2.1 Vector Norm

This section introduces the Euclidean space, which means a fnite dimensional inner product in vector space h $\langle .,. \rangle$ and the Euclidean norm defned by:

$$\| x \| = \sqrt{\langle x, x \rangle}$$

for all vectors x in the vector space

Definition 2.2.1. (Inner Product)A function $\langle .,. \rangle : R^n \times R^n \to R$ is an inner product if [21]:

1. $\langle u, v \rangle > 0, \langle u, u \rangle = 0 \Leftrightarrow u = 0$ (positivity).
2. $\langle u, v \rangle = \langle v, u \rangle$ (symmetric).
3. $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$ (additivity).
4. $\langle \alpha u, v \rangle = \alpha \langle u, v \rangle$ for all $\alpha \in R$ (homogeneity).

The characteristics of additivity and homogeneity are likewise valid; that is [24],

$\langle u, y + z \rangle = \langle u, y \rangle + \langle u, z \rangle \ \langle u, r\ v \rangle = r\langle u, v \rangle$

Definition 2.2.2. (Norm) A norm $\|.\|$ on R^n is a scalar assignment function $\|x\|$ to every $x \in R^n$ and it has the following characteristics [15]:

(a) $\|x\| \geq 0, \forall x \in R^n$.

(b) $\|\alpha x\| = |\alpha| \|x\|$ for each and every scalar α and every $x \in R^n$.

(c) $\|x\| = 0 \leftrightarrow x = 0$.

(d) $\|x + y\| \leq \|x\| + \|y\|, \quad \forall x, y \in R^n$ (This is known as the triangle inequality.)

The Euclidean norm of a vector $x = (x_1, x_2, \ldots, x_n)$ is defined by

$$\|x\|_E = (x^*x)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2} \tag{2.5}$$

proposition 2.1(**Schwarz Inequality**) [18] For any two vectors x and y, we have

$$|\langle x, y\rangle| \leq \|x\|\|y\|$$

If and only if, equality holds

$$x = \alpha y$$

for some scalar α

Example 2.2.1. Calculate the Euclidean norm of the matrix

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

solution

$$\begin{aligned} \|B\| &= \sqrt{(-4)^2 + (3)^2 + (5)^2 + (2)^2 + (6)^2 + (1)^2} \\ &= \sqrt{91} \\ &= 9.539 \end{aligned}$$

Definition 2.2.3. (Holder's inequality) : if p > 1 and $\frac{1}{p} + \frac{1}{q} = 1$, the inequality

$$\sum_{i=1}^{n} |u_i v_i| \leq \left(\sum_{i=1}^{n} |u_i|^p\right)^{1/p} \left(\sum_{i=1}^{n} |v_i|^q\right)^{1/q} \tag{2.6}$$

For p = 2, it is the Cauchy-Schwarz inequality.

- The LP-triangular norm's inequality

$$\sum_{i=1}^{n} |u_i + v_i| \leq \left(\sum_{i=1}^{n} |u_i|^p \right)^{1/p} + \left(\sum_{i=1}^{n} |v_i|^q \right)^{1/q} \tag{2.7}$$

is called Minkowski's inequality For every $(x_1, \ldots, \mathrm{x}_n) \in \mathrm{E}$,

- we have the 1-norm [50] $\|x\|_1$ defined such that,

$$\|x\|_1 = Max(|x_1| + \cdots + |x_n|)$$

- we have the Euclidean norm [26] $\|x\|_2$, defined such that,

$$\|x\|_2 = \left(|x_1|^2 + \cdots + |x_n|^2\right)^{1/2}$$

- and the sup-norm $\|x\|_\infty$, defined such that,

$$\|x\|_\infty, = \max\left\{|x_i| \mid 1 \leq \mathrm{i} \leq \mathrm{n}\right\}.$$

More generally, we define the ℓp-norm (for p ≥ 1) by

$$\|x\|_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{1/p}$$

Example 2.2.2. let $\boldsymbol{A} = \begin{bmatrix} -8 & -9 \\ -1 & -1 \\ 7 & 4 \end{bmatrix}$ we have

$$\|A\|_1 = \max\{|-8| + |-1| + |7|, |-9| + |-1| + |4|\} = \max\{16, 14\} = 16$$

Solve by python code

Code 1 :$\|A\|_1$ **norm**

```
#from array import array
import numpy as np
a = np.random.randint(-7, 7, size=(3, 2))
n, m = a.shape
colsums = [-8 ,-1 ,7
          -9 ,-1,4 ]
for i in np.arange(m):
    v = np.sum(np.abs(a[:, i]))
    colsums.append(v)
r=np.max(colsums)
print(r)
Output
  ....sol....
           16
```

And

$$\|A\|_\infty = \max\{|-8|+|-9|, |-1|+|-1|, |7|+|4|\} = \max\{17, 2, 11\} = 17$$

Code 2 :$\|A\|_\infty$ **sup-norm**

```
#from array import array
import numpy as np
a = np.random.randint(-7, 7, size=(3, 2))
n, m = a.shape
rowsums = [-8,9
           -1,-1
```

```
            7,4]
for i in np.arange(n):
    v = np.sum(np.abs(a[i, :]))
    rowsums.append(v)
r=np.max(rowsums)
print(r)
Output
....sol....
     17
```

Example 2.2.3. Calculate the ℓ_1 and ℓ_2 Norm of a Vector A $= [-1, -2, 3, 4, 5]$?

Solution:

$$\ell_1 = |-1| + |-2| + |3| + |4| + |5| = 15$$

By using python code

Code 3 :ℓ_1 **norm**

```
from numpy import array
from numpy.linalg import norm
arr = array([-1, -2, 3, 4, 5])
norm_1 = norm(arr,1)
print(norm_1)
output
15.0
```

---

to find ℓ_2 norm

$\ell_2 = \sqrt{(-1)^2 + (-2)^2 + (3)^2 + (4)^2 + (5)^2} = 7.146$

Code 4 : $\ell 2$ norm

```
from numpy import array

from numpy.linalg import norm

arr = array([-1, -2, 3, 4, 5])

norm_2 = norm(arr)

print(norm_2)

output\\

...sol...\\

7.416198487095663
```

Definition 2.2.4. The Frobenius Norm [20] : Let A be any matrix n × m. The Frobenius norm denoted by $\|\|_F$ is defined:

$$\|A\|_F = \left(\sum_{j=1}^{m}\sum_{i=1}^{n}|a_{ij}|^2\right)^{1/2} = \sqrt{\mathrm{tr}\,(AA^*)} \tag{2.8}$$

Where a_ij are element of amatrix A

Example 2.2.4. Using the python code to find the Frobenius norm of the matrix A

$$A = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 7 \\ 2 & 9 & 3 \end{bmatrix}$$

Solution

$$\|A\|_F = \sqrt{|\ 1\ |^2 + |\ 2\ |^2 + |\ 5\ |^2 + |\ 3\ |^2 + |\ 4\ |^2 + |\ 7\ |^2 + |\ 2\ |^2 + |\ 9\ |^2 + |\ 3\ |^2} = \sqrt{198}$$

$$= 14.0712473$$

Code 5 :The Frobenius norm

```
from math import sqrt
row = 3
col = 3
# Function to return the Frobenius
# Norm of the given matrix
def frobeniusNorm(mat):
   # To store the sum of squares of the elements of the given matrix
   sumSq = 0
   for i in range(row):
      for j in range(col):
         sumSq += pow(mat[i][j], 2)
   # Return the square root of the sum of squares
   res = sqrt(sumSq)
   return round(res, 5)
# Driver code
mat = [ [ 1, 2,5 ], [ 3, 4,7 ],[2,9,3]]
print(frobeniusNorm(mat))
Output
...solution...
        14.07125
```

Definition 2.2.5. Matrix Norm [12]A matrix norm denoted by $|||$ on the space of square n $\times$ n matrices in $M_n(k)$ with k = R or k = C, is a norm on the vector space $M_n(k)$ with the additional property that is a norm on the space of square n $\times$ n matrices in $M_n(k)$ with the additional property that it is a norm on the vector space $M_n(k)$:

$$\|AB\| \leq \|A\|\|B\|$$

for all A, B $\in M_n(k)$

2.3 The Objective Function's Gradient Vector and Hessian Matrix

Definition 2.3.1. (Objective Function) [64] A_n objective function represents the model's principal goal, which is either minimization or maximization.

Definition 2.3.2. (Partial Derivative) [64] let$f : R^n \to R^n$ and $x \in R^n$ The partial derivative of f at x with respect to x is then calculated. x_i is defned as

$$\frac{df(x)}{dx_i} = \lim_{n \to 0} \frac{f(x + \eta\beta_i) - f(x)}{\eta} \tag{2.9}$$

Where β_i is ith vector of a unit.

Definition 2.3.3. Gradient [5] For a differentiable objective function $f(x) : R^n \to Rf(x)$ its gradient vector given by $\nabla f(x) : R^n \to R^n$, is established at a certain point x in the represents the vector in n-dimensional space of first order partial derivatives:

$$\nabla f(x) = \begin{pmatrix} \frac{df}{dx_1}x \\ . \\ \frac{df}{dx_n}x \end{pmatrix} \tag{2.10}$$

Definition 2.3.4. (Hessian Matrix) [6] For a second continuously differentiable function $f : R^n \to R$, its Hessian matrix given byH (f(x)) is defined at the point x in the

$n \times n$-dimensional space as the matrix of second order partial derivatives

$$\mathrm{Hf(x)} = \frac{d^2 f}{dx_i dx_j} = \begin{pmatrix} \frac{d^2 f}{dx_1^2} & \frac{d^2 f}{dx_1 dx_2} & \frac{d^2 f}{dx_1 dx_n} \\ \frac{d^2 f}{dx_2 dx_1} & \frac{d^2 f}{dx_2^2} & \cdots \frac{d^2 f}{dx_2 dx_n} \\ \vdots & & \\ \frac{d^2 f}{dx_n dx_1} & \frac{d^2 f}{dx_n dx_2} & \cdots \frac{d^2 f}{dx_n^2} \end{pmatrix} \tag{2.11}$$

One important relation that we will keep in mind is that the Hessian matrix is the Jacobean of the gradient vector of f(x), where the Jacobean matrix of a vector-valued function F(x) is the matrix of all its first order partial derivatives given by, $\mathrm{JF}(x) = \left(\frac{df}{dx_1} \ldots \frac{df}{dxn} \right)$. The relation is given as follow:

$$\mathrm{Hf(x)} = J(\nabla f(x))$$

Example 2.3.1. Find Jacobean to the function $f : R^2 \to R^2$ given by

$$f(x, y) = \begin{bmatrix} x^2 \cos(y) \\ 9\cos(x) + \sin(y) \end{bmatrix}$$

Solution: Let

$$f_1(x, y) = x^2 \cos(y)$$

and

$$f_2(x, y) = 9\cos(x) + \sin(y)$$

And the Jacobean matrix of f is

$$J_f(x,y) = \begin{bmatrix} \frac{df_1}{dx} & \frac{df_1}{dy} \\ \frac{df_2}{dx} & \frac{df_2}{dy} \end{bmatrix} = \begin{bmatrix} 2 * \mathrm{x} * \cos(\mathrm{y}) & -\mathrm{x}^2 * \sin(\mathrm{y}) \\ -9sin(x) & \cos(y) \end{bmatrix}$$

The Jacobean is computed using the python code

Code 6 :Jacobean matrix

```
import symengine
vars=symengine.symbols('x y') # Define x and y variables
f=symengine.sympify(['x**2 *cos(y)','9*cos(x)+sin(y)']) # Define function
J=symengine.zeros(len(f),len(vars))# Initialise Jacobean matrix
for i,fi in enumerate(f):
 for j, s in enumerate(vars):
  J[i,j]=symengine.diff(fi,s)
print(J)
```

output

solution

$$\begin{bmatrix} 2x\cos(y) & -x^2\sin(y) \\ -9sin(x) & \cos(y) \end{bmatrix}$$

Example 2.3.2. Let an objective function be f(x) $= 4x_2^5x_3^2 + 5x_1^2x_3 + x_1^2x_2^3$. We will find out the gradient vector ∇f(x) and the Hessian matrix Hf(x) at the point p $= (2, 1, 1)$?

Solution: The gradient vector is

$$\nabla f(x) = \begin{bmatrix} 10x_1x_3 + 2x_1x_2^3 \\ 20x_2^4x_3^2 + 3x_1^2x_2^2 \\ 8x_2^5x_3 + 5x_1^2 \end{bmatrix}$$

at $p(2, 1, 1)$

$$= \begin{bmatrix} 24 \\ 32 \\ 28 \end{bmatrix}$$

The Hessian matrix

$$Hf(x) = \begin{bmatrix} 10x_3 + 2x_2^3 & 6x_1x_2^2 & 10x_1 \\ 6x_1x_2^2 & 80x_2^3x_3^2 + 6x_1^2x_2 & 40x_2^4x_3 \\ 10x_1 & 40x_2^4x_3 & 8x_2^5 \end{bmatrix}$$

$$= \begin{bmatrix} 12 & 12 & 20 \\ 12 & 104 & 40 \\ 20 & 40 & 8 \end{bmatrix}$$

This example can be solved using the Python.

The auto grad library is needed, along with the numpy library. Autograd is mainly used

for gradient based optimization

Code 7 :gradient vector and Hessian matrix

```
import autograd.numpy as np
from autograd import grad, jacobean
p = np.array([2, 1, 1], dtype=float)
def f(x): # Objective function
    return 4*x[1]**5*x[2]**2+5*x[0]**2*x[2]+x[0]**2*x[1]**3
grad_f = grad(f) # gradient of the objective function
hessian_f = jacobean(grad_f) # Hessian of the objective function
print("gradient vector:",grad_f(p))
## gradient vector: [24. 32. 28.]
print("Hessian matrix:\n",hessian_f(p))


out put
solution
 gradient vector: [24, 32, 28.]


Hessian matrix:
 [[ 12.  12.  20.]
 [ 12. 104.  40.]
 [ 20.  40.   8.]]
```

Definition 2.3.5. (Directional Derivative of the Objective Function) In the direction, the directional derivative of $f(x)$ for a real valued objective function $f(x)$ and a feasible direction δ is given by [64]:

$$\frac{df}{d\delta} = \lim_{\delta \to 0} \frac{f(x + \delta\beta) - f(x)}{\delta} \tag{2.12}$$

Now x $\in R^n$ Let's have a look at the differential equation.

$$df(x) = \frac{df(x)}{dx_1}dx_1 + \ldots + \frac{df(x)}{dx_n}dx_n = \nabla^T f(x)dx = \langle \nabla f(x), dx \rangle \tag{2.13}$$

Where $\langle ., . \rangle$ represents the dot product of two matrices or vectors. Let's have a look at a function now $f^*(x) - f\left(x^* + \delta\beta\right)$, such that for a point x on the dotted line x^* in the direction n is given by

$$x(\delta) = x^* + \delta\beta \tag{2.14}$$

now, for a minuscule variation d_δ we have $dx = \beta d_\delta$ As a result, the difference at the position x in the given direction is $df^* = \nabla^\top f(x)\beta d\delta$ So, the directional derivative now can be written as:

$$\frac{df}{d\beta}(x) = \frac{d}{d\delta}(x + \delta\beta)\Big|_{\delta=0} = \nabla^T f(x)\beta \tag{2.15}$$

Example 2.3.3. Let an objective function be f(x) $= 2x_1x_2^2 + 3x_1x_2^3 + 4x_1x_3^2$ We will find out the gradient vector $\bigtriangledown f(x)$ at the point $p = (1, 2, 3)$ and then calculate the directional derivative in the direction $\delta = \left(\frac{1}{\sqrt{35}}, \frac{3}{\sqrt{35}}, \frac{5}{\sqrt{35}}\right)$ by using python code?

$$\nabla f(x) = \begin{bmatrix} 2x_2^2 + 3{x_2}^3 + 4x_3^2 \\ 4x_1x_2 + 9x_1x_1^2 \\ 8x_1x_3 \end{bmatrix}$$

$$p = (1, 2, 3)$$

$$\nabla f(x) = \begin{bmatrix} 68 \\ 44 \\ 24 \end{bmatrix}$$

$$\nabla f(x) = \begin{bmatrix} 68 \\ 44 \\ 24 \end{bmatrix} \left[\frac{1}{\sqrt{35}}, \frac{3}{\sqrt{35}}, \frac{5}{\sqrt{35}}\right]$$

$$= \frac{1}{\sqrt{35}}[68 + 132 + 120] = 54.0898$$

solution by using python code

Code 8 :Directional Derivative

```
import numpy as np
from autograd import grad
p = np.array([1, 2, 3], dtype=float)
delta = np.array([1, 3, 5], dtype=float)/np.sqrt(35)
def f(x):
    return 2*x[0]*x[1]**2+3*x[0]*x[1]**3+4*x[0]*x[2]**2
grad_f = grad(f)
print("directional derivative:", grad_f(p).dot(delta))
```

```
out put
solution
  directional derivative: 54.08987230262507
```

Definition 2.3.6. Decision Variables: [34] In an Optimization Problem, decision variables are variables whose values may change throughout the feasible set of alternatives to increse or decrease the main objective function's value.

Definition 2.3.7. (Constrained Optimization) Consider the following problem of restricted optimization:

$$\begin{cases} \max f(x_1, \ldots\ldots, x_n) \\ Or \\ \min f(x_1, \ldots\ldots, x_n) \\ \text{subject to} \quad b_1(x_1, \ldots, x_n) \leq z_1, \ldots, b_i(x_1, \ldots., x_n) \leq z_i \\ g_1(x_1, \ldots\ldots, x_n) = h_1, \ldots\ldots\ldots g_1(x_1, \ldots., x_n) = h_j \end{cases} \tag{2.16}$$

Where $f(x)$ is the objective function, $b(x)$ is an inequality constraint, and $g(x)$ is an equality constraint, and f, b, and g are all differentiable functions with continuous derivatives [70].

Definition 2.3.8. (Unconstraint Optimization) Unconstrained optimization problems are openly present in many actual situations. If the variables have inherent restrictions, it's usually acceptable to ignore them and assume that they have no effect on the best solution [70]. Unconstrained optimization problems are often reformulated constrained optimization problems in which the constraints are replaced with penalty terms in the objective function that avoid constraint breaches. We minimize an objective function that relies on actual variables without any limitations on their values

in unconstrained optimization. The mathematical formula is:

$$\begin{cases} \text{minimize} \quad f(x) \\ x \in R \end{cases} \tag{2.17}$$

Definition 2.3.9. (Feasible Region) [24] A viable area is a collection of locations that meet all of the restrictions. The collection List all feasible optimization problem points (sets of values for the choice variables) that satisfy the Problem's criteria, which may include integer restrictions, inequalities, and equalities, is known as a feasible region, feasible set, search space, or solution space. i.e If a point x is in R^n and meets the restrictions of the problem, it is feasible.

$$\begin{cases} \text{minimize} & f(x) \\ \text{subject to} & b_i(x) = 0 \quad i = 1, 2, \ldots\ldots k \\ & g_j(x) \leq 0 \quad j = 1, 2, \ldots, m \\ & x \in R^n \end{cases} \tag{2.18}$$

The set F of all feasible points defines the feasible region of the optimization problem:

$$\text{S} = \{x \in R^n \mid b_i(x) = 0 \text{ for } i = 1, 2, \ldots, k, g_j(x) \leq 0 \text{ for } j = 1, 2, \ldots, m\}$$

Example 2.3.4. Finding the Feasible Region by using python code

$$\begin{cases} \text{Max} \quad \text{z} = 10\text{x} + 10\text{y} \\ \text{s.t} \quad 5\text{x} + 8\text{y} \leq 200 \\ \qquad 25\text{x} - 10\text{y} \geq 250 \\ \qquad \text{x} + \text{y} \geq 150 \\ \qquad \text{x}, \text{y} \geq 0 \end{cases}$$

Solution

Code 9 :Feasible Region

```
import matplotlib.pyplot
from matplotlib.pyplot import *
import numpy
from numpy import arange
figure()
x= arange (-100,200,50)
y= range (-100,200,50)
y1=(200-5*x)/8
y2=(250-25*x)/10
y3=150.0-x
xlim(-100,200)
ylim(-100,200)
hlines(0,-100,200,color='k')
vlines(0,-100,200,color='k')
grid(True)
xlabel('x-axis')
ylabel('y-axis')
plot(x,y1,color='r')
plot(x,y2,color='g')
plot(x,y3,color='m')
title ('objective function z=10*x+10*y with the following constraints')
legend (['5*x+8*y<=200','25*x-10*y>=250','x+y>=150'])
x=[0.0,40.0,150.0,0.0]
y=[25.0,0.0,0.0,150.0]
fill(x,y)
show()
```

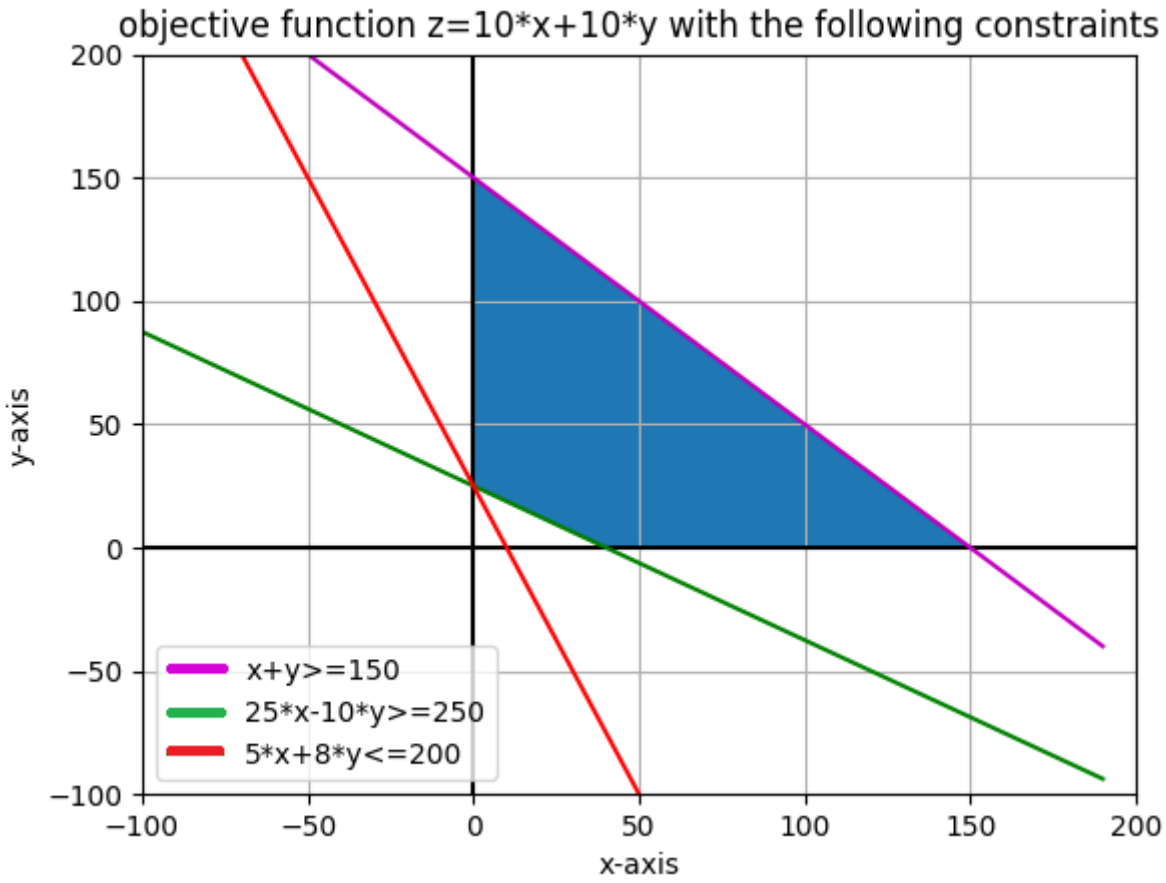


Figure 2.1: The blue area represents feasible Region and The violet, green, and red lines represent region constraints and intersect with the x-axis and y-axis

Definition 2.3.10. (Infeasible Region) [14] If no solution exists that fulfills all of the requirements, LP is infeasible. As a result, the LP is infeasible if no feasible solution can be found. Note that a genuine operation must adhere to the restrictions of reality; hence, infeasibility almost always reveals a flaw. It might be due to a mistake in setting some of the limitations or incorrect data quantities.If there is no optimum The problem is either infeasible or unbounded for every linear program in standard form. A basic practicable solution exists if a viable solution exists. In the existence of an optimal solution, there is a fundamental feasible solution that is also a perfect solution.

Example 2.3.5. Find Infeasible solution by using graphical method:

$$\begin{cases} \text{maximum.} & z = 6x - 4y \\ \text{s.t} & 2x + 4y \leq 4 \\ & 4x + 8y \geq 16 \\ & x, y \geq 0 \end{cases}$$

Solution:

The one constraint $2x + 4y \leq 4$

$$2x + 4y = 4$$

$$\text{if } x = 0$$

$$2(0) + 4y = 4 \rightarrow y = 1$$

$$\text{if } y = 0$$

$$2x + 4(0) = 4 \Rightarrow x = 2$$

The two constraint

$$4x + 8y \geq 16$$

$$4x + 8y = 16$$

$$\text{if } x = 0$$

$$4(0) + 8y = 16 \Rightarrow y = 2$$

$$\text{if } y = 0$$

$$4x + 8(0) = 16 \Longrightarrow x = 4$$

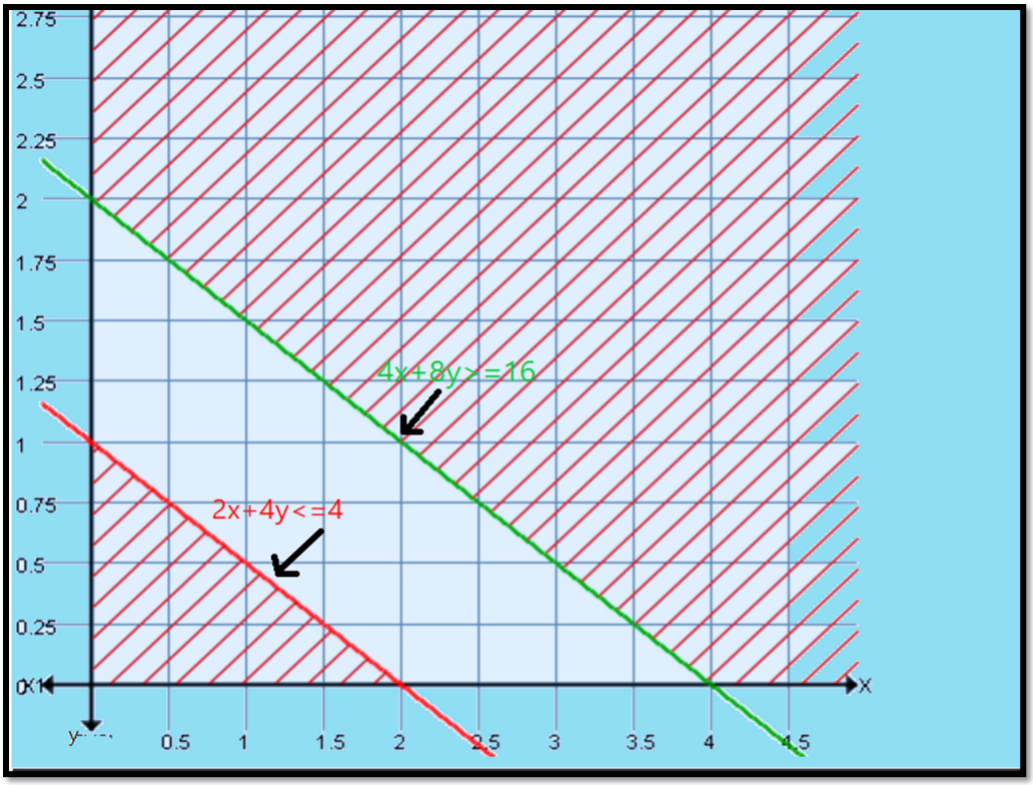


Figure 2.2: Infeasible Region

2.4 Convex Set And Convex Function

This section explains what key words related to a linear program's viable region

Definition 2.4.1. Convex Set [4] If given any two points x_1 and x_2 in x, any convex combination of these two points is also in x, then the set $x \in R$ is a convex set. That is correct :

$$\lambda x_1 + (1 - \lambda)x_2 \in x \tag{2.19}$$

for any

$$\lambda \in [0, 1]$$

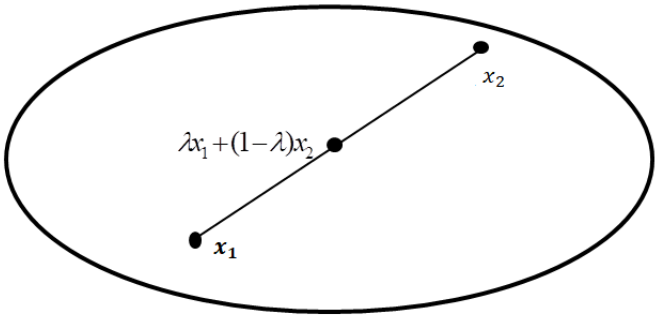


Figure 2.3: convex set

Example 2.4.1. (convex and non-convex sets) The following may be proven using the definition of a convex set:

(a) The collection R^n is a convex collection.

(b) empty set is a convex set .

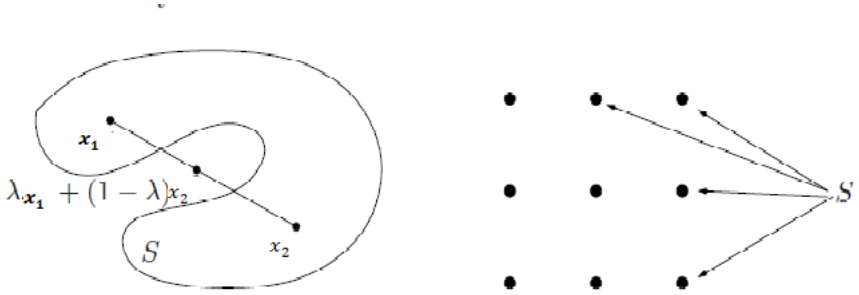


Figure 2.4: two non –convex sets

in figure 2.4 a line segment is broken, any two points in x are included in x, and the set x is convex. If any section of the line segment is not in X, it is deemed nonconvex. figure 2.3 2.4 is convex and non-convex sets

The feasible region of a linear program is always a convex set [69]. The axes depict a two-dimensional area. In linear programming, this area is non-convex, and therefore cannot arise because [49]

$$y \leq m \times x + h$$

for

$$cx \leq d$$

but

$$y \leq b$$

for

$$d < x < e$$

The constraint $y \leq m \times x + h$ doesn't hold when $d < x < e$ and the constraint $y \leq b$ doesn't hold when $0 \leq x \leq d$ In linear programming, conditional restrictions like this don't exist. It's impossible for a condition that is valid in one zone but not in another.

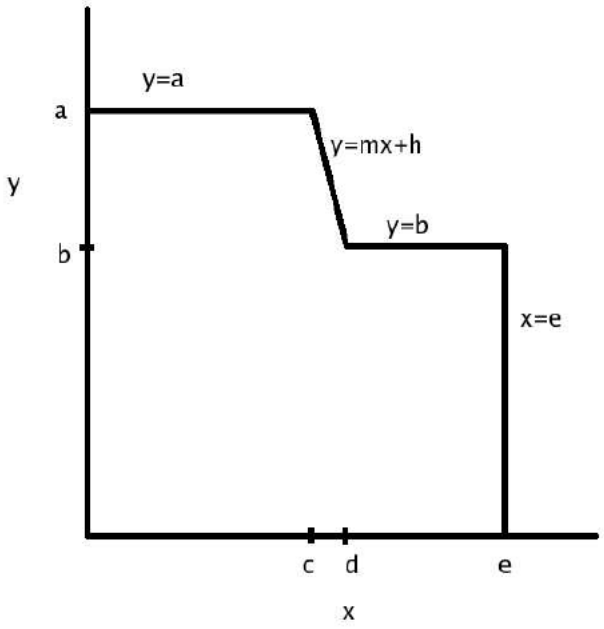


Figure 2.5: A non-convex (and therefore invalid) feasible region

Definition 2.4.2. (Convex Hull) [8]The hull is convex. In R^n, let S be any arbitrary set. The collection of all convex combinations of S is known as the convex hull of S , or

conv(S). To put it another way,$X \in conv(S)$ if and only if X may be shown as :

$$\begin{cases} x = \sum_{j=1}^{k} \lambda_j x_j \\ \mathbf{x} = \sum_{j=1}^{k} \lambda_j = 1 \\ \lambda_j \geq 0 \quad \text{for } j = 1, \ldots\ldots\ldots, k \end{cases} \tag{2.20}$$

where k is a positive integer and $x_1, \ldots\ldots x_k \in S$.

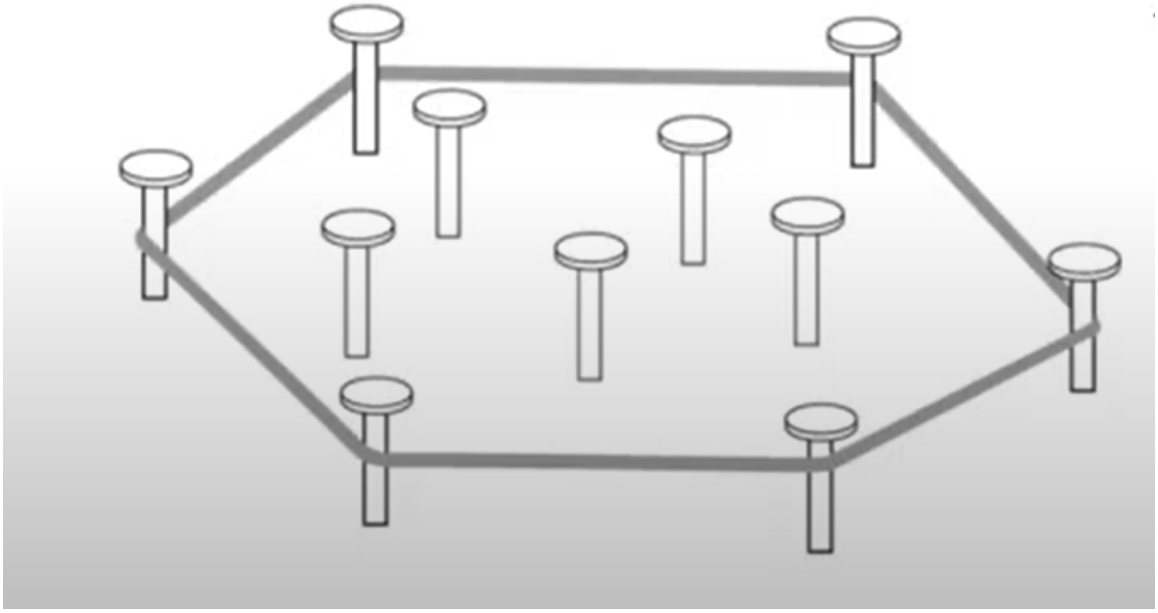


Figure 2.6: Convex Hull

By using python code to find the convex hull

Code 10 :convex hull

```
import numpy as np
from scipy.spatial
import ConvexHull, convex_hull_plot_2d
rng = np.random.default_rng()
points = rng.random((30, 2))   # 30 random points in 2D
```

```
hull = ConvexHull(points)
import matplotlib.pyplot as plt
plt.plot(points[:,0], points[:,1], 'o')
for simplex in hull.simplices:
    plt.plot(points[simplex, 0], points[simplex, 1], 'k')
    plt.plot(points[hull.vertices, 0], points[hull.vertices, 1])

    plt.show()
```

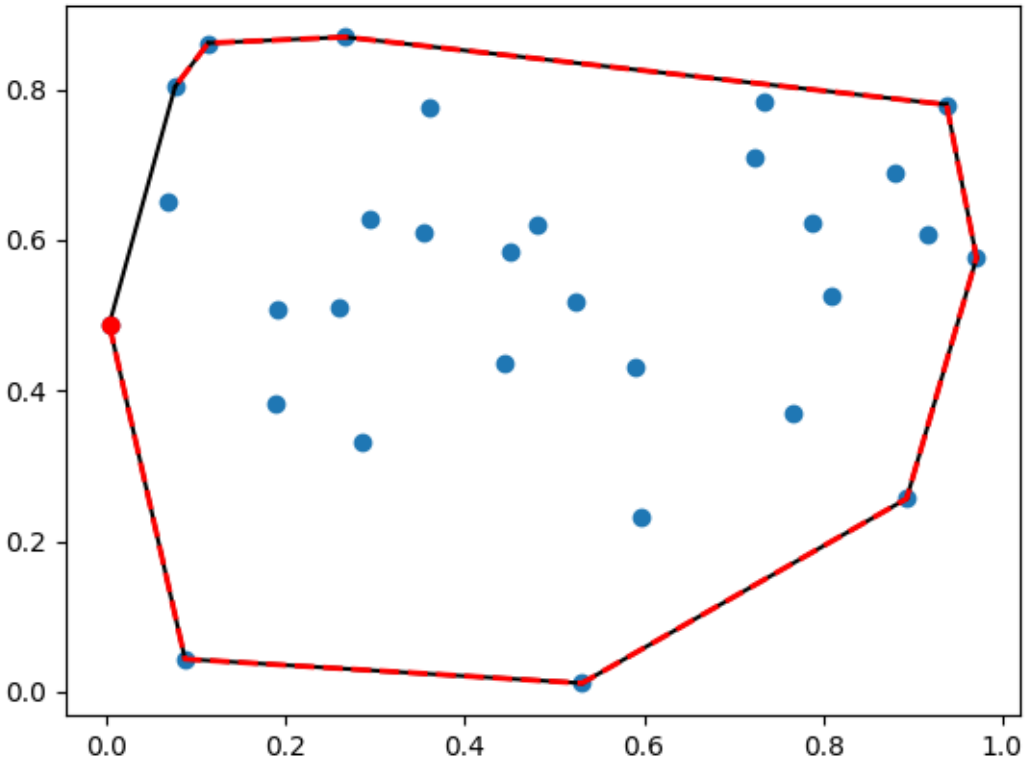


Figure 2.7: convex hull

Definition 2.4.3. (Convex Function) [4] let $S \subseteq R^n$ A function $f : R^n \to R \bigcup \{+\infty.\}$

is convex at $y \in$ S if : x $\in$ S, $\lambda \in (0, 1)$ then

$$\lambda x + (1 - \lambda)y \in S \rightarrow f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \tag{2.21}$$

The function is also a convex function if $f''(x) > 0$ The function f is convex on S if it is convex at every $y \in S_3$

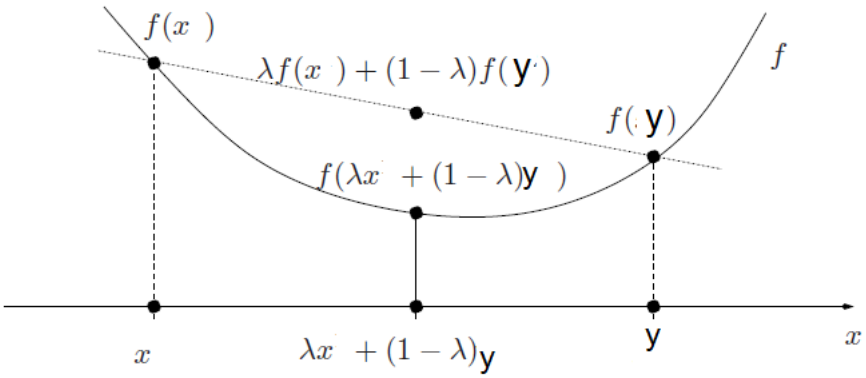


Figure 2.8: A strictly convex function

Example 2.4.2. The function $f(x) = x^2, f''(x) = 2x$ As a result, f is a convex function. With a high convexity constant of 2 , it is likewise strongly convex (and so strictly convex).

Python may be used to prove that function f(x) is convex.

Code 11 : Convex Function $f(x) = x^2$

```
# plot a convex target function
from numpy import arange
from matplotlib import pyplot
# objective function
def objective(x):
    return ( x)^ 2
```

```
# define range
r_min, r_max = -10.0, 10.0

# prepare inputs
inputs = arange(r_min, r_max, 1.5)

# compute targets
targets = [objective(x) for x in inputs]

# plot inputs vs target
pyplot.plot(inputs, targets, '--')
pyplot.show()
```

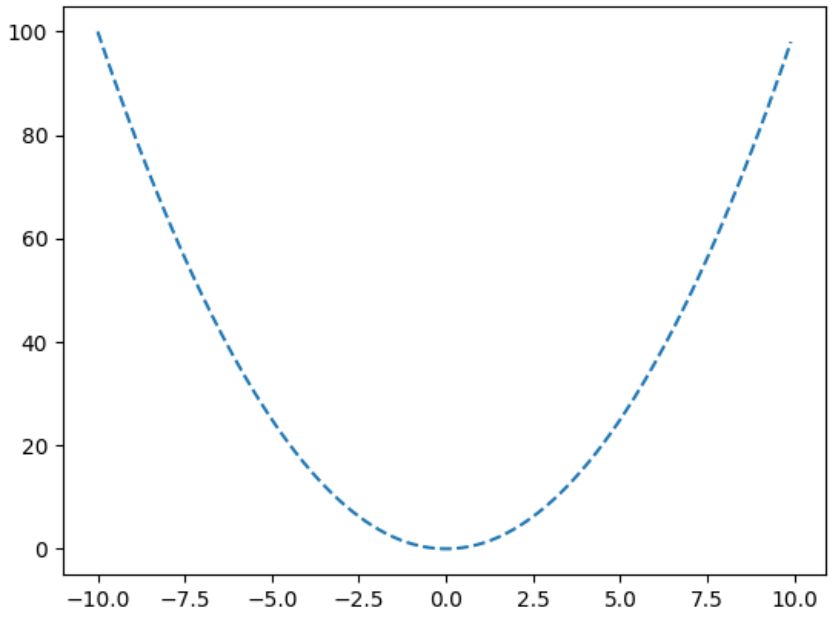


Figure 2.9: $f(x) = x^2$Convex Function

Example 2.4.3. The exponential function $f(x) = e^x$ is convex. It is also strictly convex, since $f''(x) = e^x > 0$, but it is not strongly convex since the second derivative can be arbitrarily close to zero. More generally, the function $g(x) = e^{f(x)}$ is logarithmically convex if f is a convex function.

Code 12 :**Convex Function** $f(x) = e^x$

```
# plot a convex target function
from math import exp
from numpy import arange
from matplotlib import pyplot

# objective function

def objective(x):
    return exp(x)

# define range
r_min, r_max = -10.0, 10.0
# prepare inputs
inputs = arange(r_min, r_max, 1.5)
# compute targets
targets = [objective(x) for x in inputs]
# plot inputs vs target
pyplot.plot(inputs, targets, '--')
pyplot.show()
```

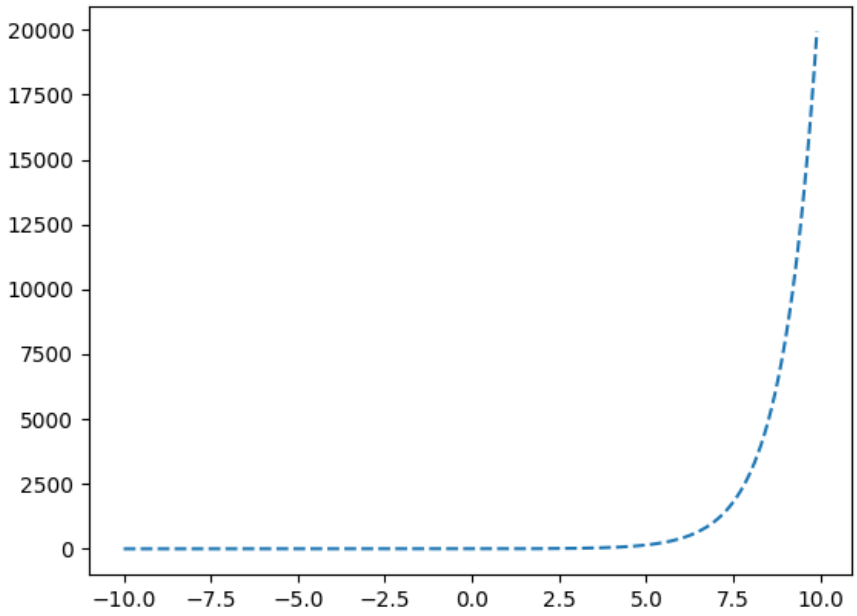


Figure 2.10: $f(x) = e^x$ Convex Function

Example 2.4.4. Is $f(x) = cos(x)$ convex or concave ?

solution by using Python which got $cos(x)$ that is convex.at $[1, 5]$ and concave at $[6, 7]$

Code 13 :f(x)=cos(x)

```
import numpy as np
import matplotlib.pyplot as plt

# plotting cos(x)
x = np.linspace(1, 9)

# points on the x axis
f = np.cos(x)

# Objective function
plt.plot(x, f, color=(1, 0, 1))
```

```
plt.grid()
plt.xlabel('$x$')
plt.ylabel('$\cos_x$')
# Co n v e x i t y / Co n c a v i t y
a = 2
b = 7
lamda = 0.4

c = lamda * a + (1 - lamda) * b
f_a = np.cos(a)
f_b = np.cos(b)
f_c = np.cos(c)
f_c_hat = lamda * f_a + (1 - lamda) * f_b

# Plot commands
plt.plot([a, a], [0, f_a], color=(0, 0, 0), label='$f(a) $')
plt.plot([b, b], [0, f_b], color=(0, 0, 1), label="$ f(b) $")
plt.plot([c,c],[0,f_c_hat],color=(1,1,0),label="$\lambda a +(1-\lambda) b$")
plt.plot([c,c],[0,f_c_hat],color=(1,0,0),label="$\lambda f(a)+(1-\lambda)f(b)$")
plt.plot([a, b], [f_a, f_b], color=(0, 1, 0))
plt.show()
```

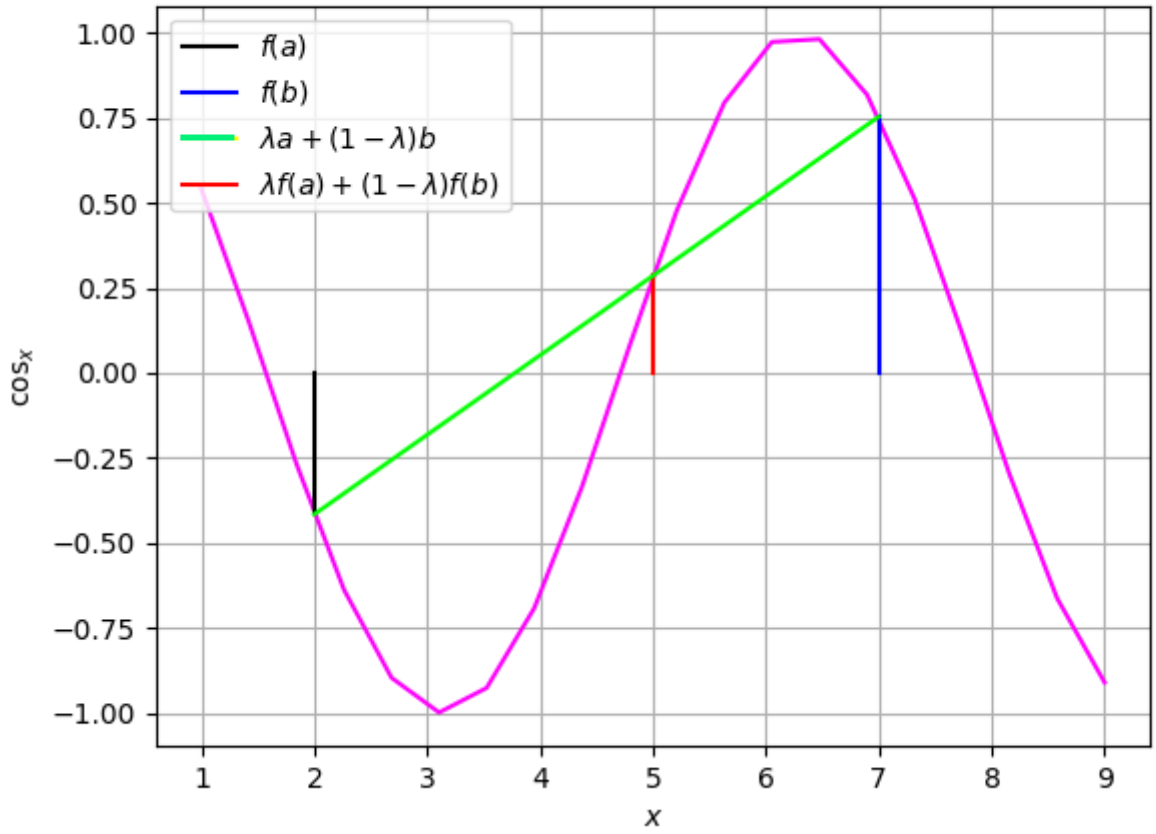


Figure 2.11: cos(x) convex function

Example 2.4.5. :is afunction $(x-2)x(x+2)^2$ convex or non convex ?

Solution: by using python code

Code 14 :Non-Convex

```
# plot a non-convex univariate function


from numpy import arange

from matplotlib import pyplot


# objective function

def objective(x):
```

```
    return (x - 2.0) * x * (x + 2.0)**2.0

# define range
r_min, r_max = -3.0, 2
# prepare inputs
inputs = arange(r_min, r_max, 1)

# compute targets
targets = [objective(x) for x in inputs]
# plot inputs vs target

pyplot.plot(inputs, targets, '--')
pyplot.show()
```

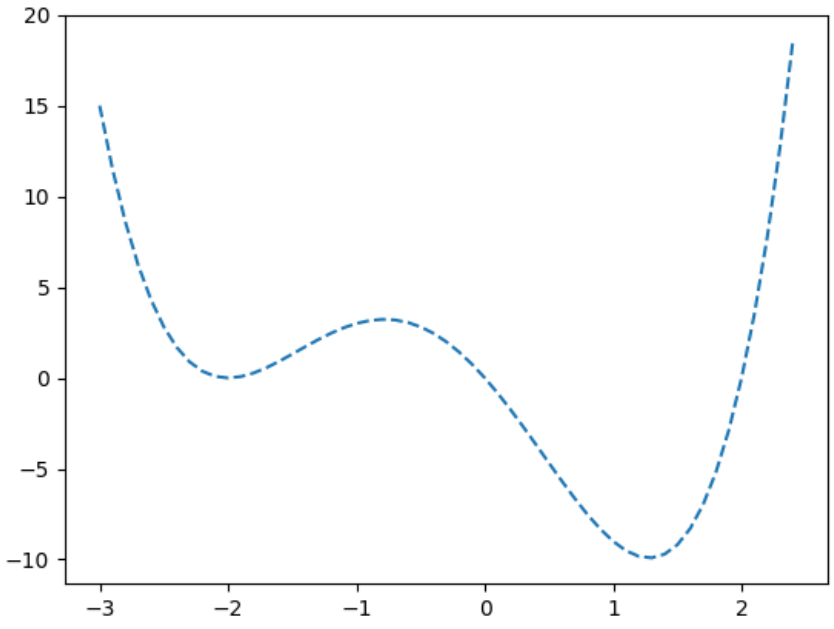


Figure 2.12: Non Convex Function

Definition 2.4.4. (Epigraph) Let C $\subseteq R^n$ be a nonempty convex set. The epigraph of a function $f : C \to R$, denoted by $epi(f)$, is a subset of R^{n+1} defined by [22]:

$$\text{epi}(f) = \{(x,t) \mid f(x) \leq t, x \in C, t \in R\} \tag{2.22}$$

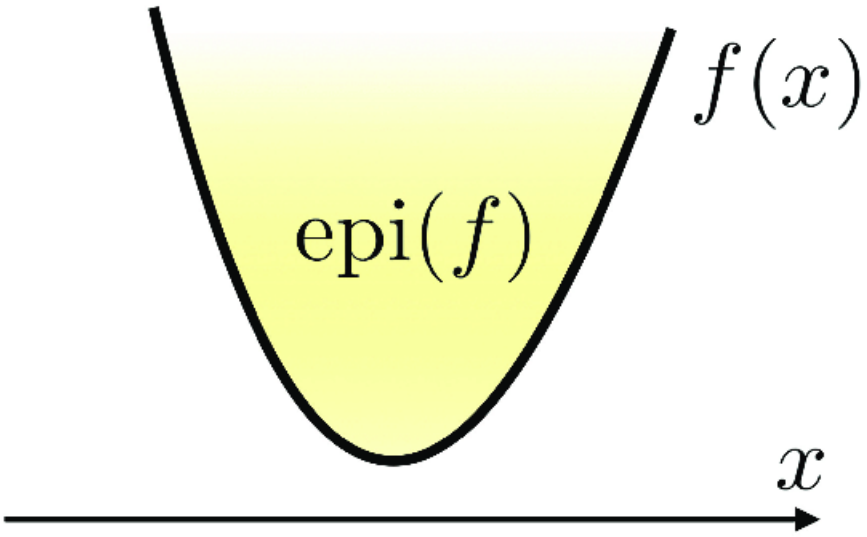


Figure 2.13: A convex function and its epigraph

The hypo graph of f, shown as $hyp(f)$, is a subset of R^{n+1} defined by

$$hyp(f) = \{(x,t) \mid f(x) \geq t, x \in C, t \in R\} \tag{2.23}$$

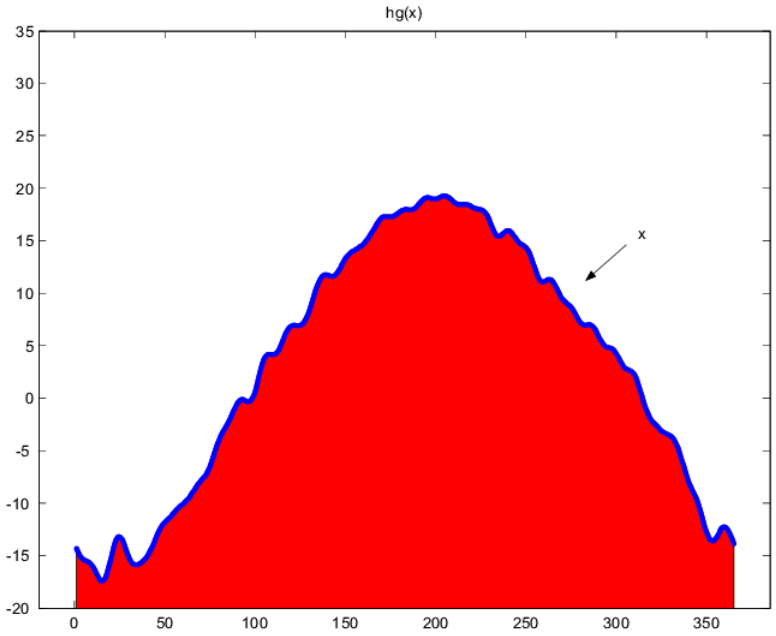


Figure 2.14: Hypo graph

Theorem 2.4.1. *Let's pretend* $S \subseteq R^n$ *is a convex set. Then* $f : R^n \to R \cup \{+\infty\}$ *is convex on* S *if and only if its epigraph confined to* S *is a convex set in* R^{n+1} *[4].*

Definition 2.4.5. (Feasible Space) [39] The feasible space or feasible area is comprised of viable options. The solution is optimum if the objective function value of a feasible solution equals the minimal value z that may occupy the feasible area.

Definition 2.4.6. (Argument) [1] Let $c \subseteq R^n$ be a nonempty set. If f : C $\to$ R, the argument of the minimum is the set of elements in c that achieve the global minimum in S, which is defned by argmin :

$$\arg\min_{x \in s} f(x) = \{x \in C \mid f(y) \geq f(x), \forall y \in c\} \tag{2.24}$$

Definition 2.4.7. (Global Minimum) [4] Assume $x^* \in$ S. If f reaches its lowest value over S at x, we say that is a global minimum of f over S. To put it another way, $x^* \in$ S is a global minimum of f over S if and only if :

$$\mathrm{f}\left(x^{*}\right) \leq \mathrm{f}(\mathrm{x}), \mathrm{x} \in \mathrm{S}$$

holds. A point x^{*} is called a strict global minimum if $\mathrm{f}\left(x^{*}\right) < \mathrm{f}(\mathrm{x})$ for all $\mathrm{x} \in R^{n}$, with $\mathrm{x} \neq x^{*}$

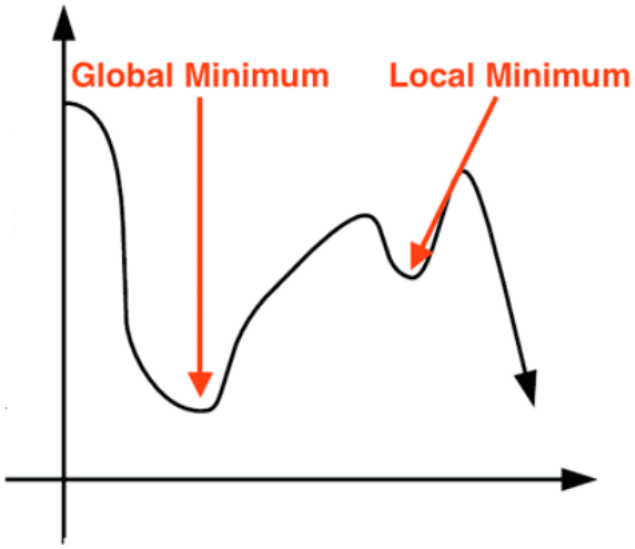


Figure 2.15: local and global minimum

Definition 2.4.8. (Local Minimum) [4] Let's say $x^{*} \in \mathrm{S}$ which call it a local minimum of f over S if there is a tiny enough ball intersected with S around that is a globally optimal solution in that smaller set.

In other words, if $x^{*} \in S$ is a local minimum of f over S,

$$\exists\, \varepsilon > 0$$

such that

$$\mathrm{f}\left(x^{*}\right) \leq \mathrm{f}(\mathrm{x}), \mathrm{x} \in \mathrm{S} \cap \mathrm{B}\varepsilon\left(x^{*}\right) \tag{2.25}$$

We say that $x^* \in$ S is a strict local minimum of f over S if, the inequality holds that [59]:

$$\mathrm{f}\left(x^*\right) < \mathrm{f(x)}, \mathrm{x} \in \mathrm{S} \cap \mathrm{B}\varepsilon\left(x^*\right) \tag{2.26}$$

Example 2.4.6. Find local and global minimum to the function $f(x) = 5sin(x)$

Code 15 :local and global minimum $f(x) = 5sin(x)$

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
x=np.arange(-10,10,2.5)
def f(x):
 return 5*np.sin(x)
#Global optimization
grid =(-10,10,2.5)
xmin_global=optimize.brute(f,(grid,))
print("Global minima found %s" % xmin_global)
#Constrain optimization
xmin_local=optimize.fminbound(f,0,10)
print("Local minimum found %s" % xmin_local)
fig=plt.figure(fig)
ax=fig.add_subplot(111)
#Plot the function
ax.plot(x,f(x),'b',label="f(x)")
#Plot the minima
xmins=np.array([xmin_global[0],xmin_local])
ax.plot(xmins,f(xmins),'go',label="Minima")
# Decorate the figure
ax.legend(loc='best')
```

```
ax.set_xlabel('x')
ax.set_ylabel('f(x)')
ax.axhline(0,color='gray')
plt.show()
output
solution
Global minima found␣[4.71239258]]
Local minimum found 4.712388884507233
```

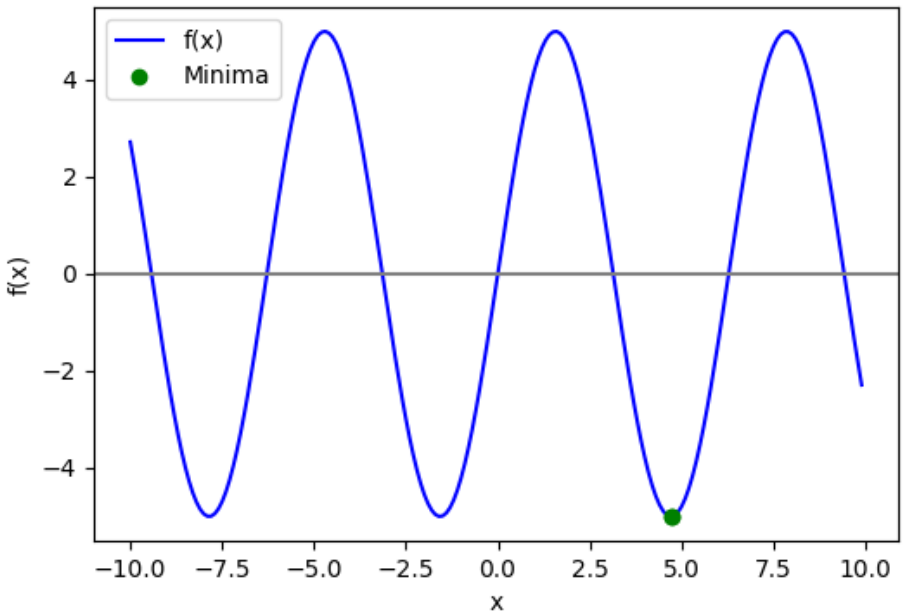


Figure 2.16: The green dot represents the local and global minimum

Theorem 2.4.2. *To prove the following theorem, let x and x^* be two points in $\mathbf{R}^J$. A point z on the line segment $[x^*; x]$ connecting x with the x^* is therefore defined as follows:*

- ❇ *x^* is a global minimizer of $f(x)$ if $(x - x^*) \cdot H(z)\,(x - x^*) \geq 0$ for all x and for all z in $[x^*; x]$;*
- ❇ *x^* is a strict global minimizer of $f(x)$ if $(x - x^*)\,.H(z)\,(x - x^*) > 0$ for all $x \neq x^*$ and for all z in $[x^*; x]$;*

❇ x^* *is a global maximizer of* $f(x)$ *if* $(x - x^*) \cdot H(z)\,(x - x^*) \leq 0$ *for all* x *and for all* z *in* $[x^*; x]$;

❇ x^* *is a strict global maximizer of* $f(x)$ *if* $(x - x^*) \cdot H(z)\,(x - x^*) < 0$ *for all* $x \neq x^*$ *and for all* z *in* $[x^*; x]$.

2.5 Smooth and Non-smooth Functions

Definition 2.5.1. (Smooth Function) [38] Every point on a smooth function has a distinct first derivative (slope or gradient). A smooth function of one variable may be represented graphically as a single continuous line with no sudden bends or interruptions.

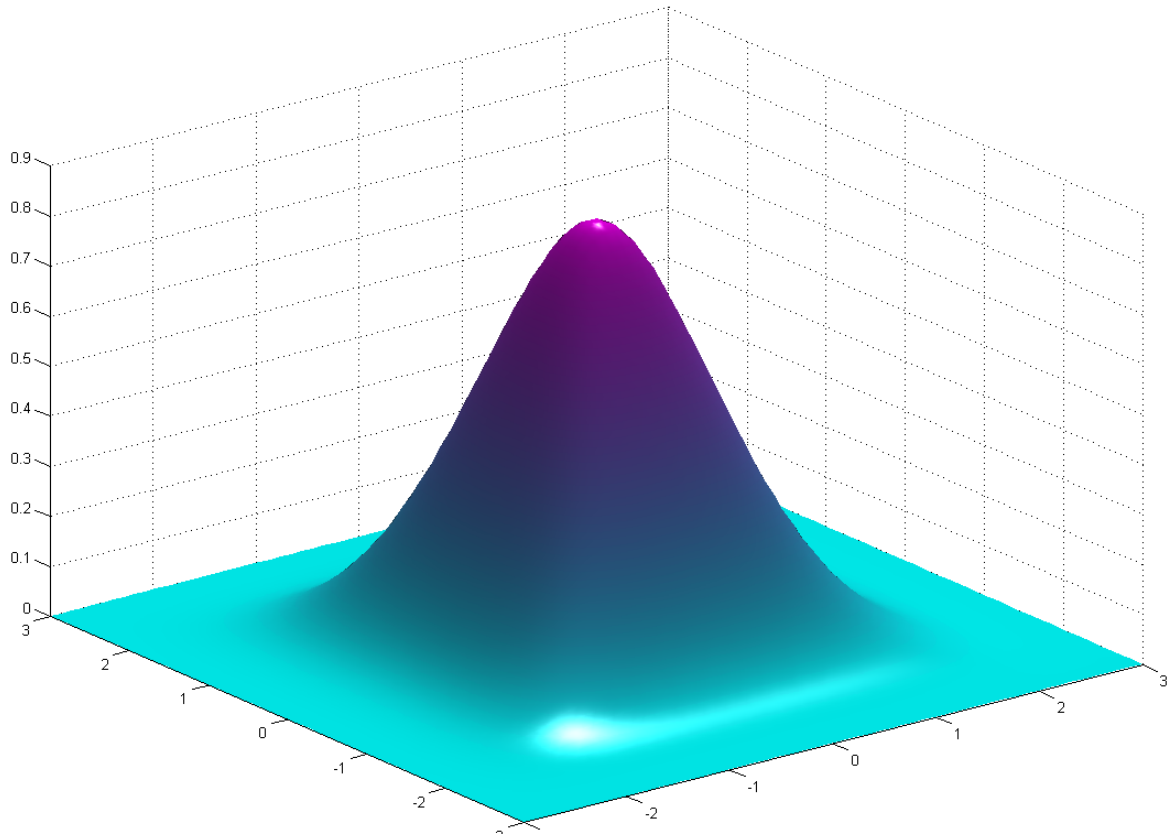


Figure 2.17: Smooth Functions

Example 2.5.1. by using python code find smooth of the function $f(x) = x^2$

solution

Code 16 :Smooth Function $f(x) = x^2$

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-1.5, 1.5)
# A smooth function
plt.figure
plt.plot(x, np.sqrt( x**2), linewidth=2)
plt.text(-1, 0, '$f(x)$', size=20)
plt.ylim(ymin=-.2)
plt.axis('oN')
plt.tight_layout()
plt.ylim(ymin=-.2)
plt.axis('oN')
plt.tight_layout()
plt.show()
```

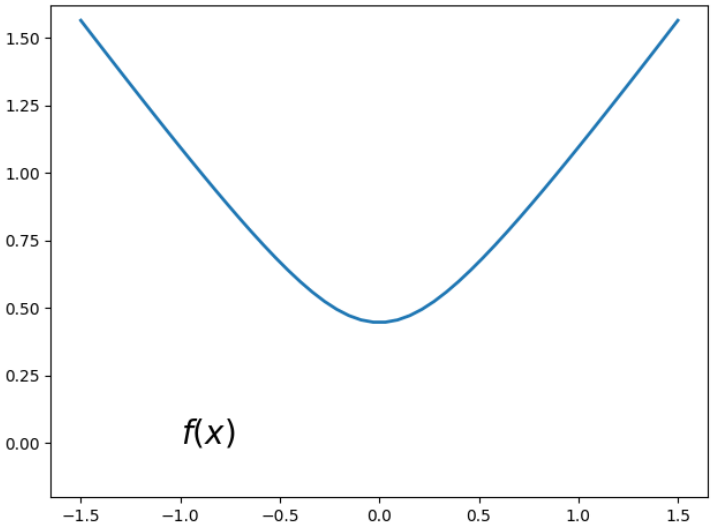


Figure 2.18: Smooth Function

Definition 2.5.2. (Non-Smooth Function) [38] Non-differentiable and discontinuous functions are examples of non-smooth functions. Non-differentiable functions have first derivatives with undefined areas. Non-differentiable function graphs may feature sharp bends .The absolute value of a variable .

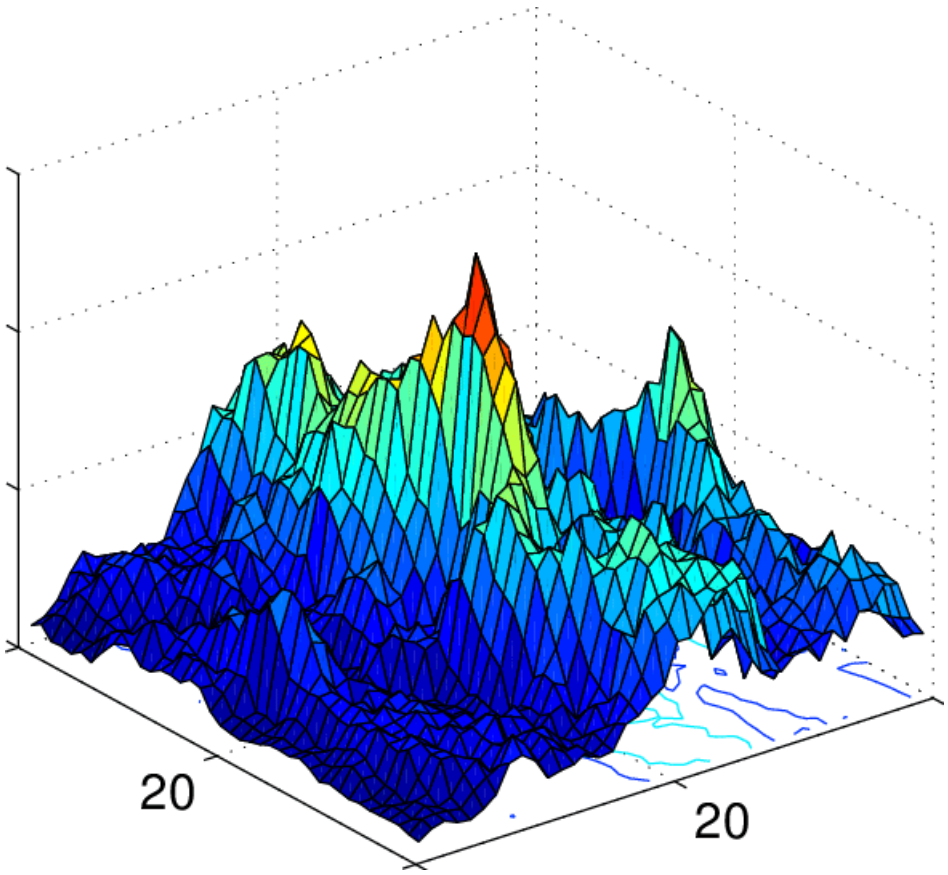


Figure 2.19: Non-Smooth Function

Example 2.5.2. by using python code find Non-smooth of the function $f(x) =\mid x \mid$?

solution

Code 17 :Non-Smooth Function $f(x) =\mid x \mid$

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-1.5, 1.5)
```

```
# A non-smooth function
plt.figure
plt.plot(x, np.abs(x))
plt.text(-1, 0, '$f(x)$', size=20)
plt.ylim(ymin=-.2)
plt.axis('on')
plt.tight_layout()
plt.show()
```

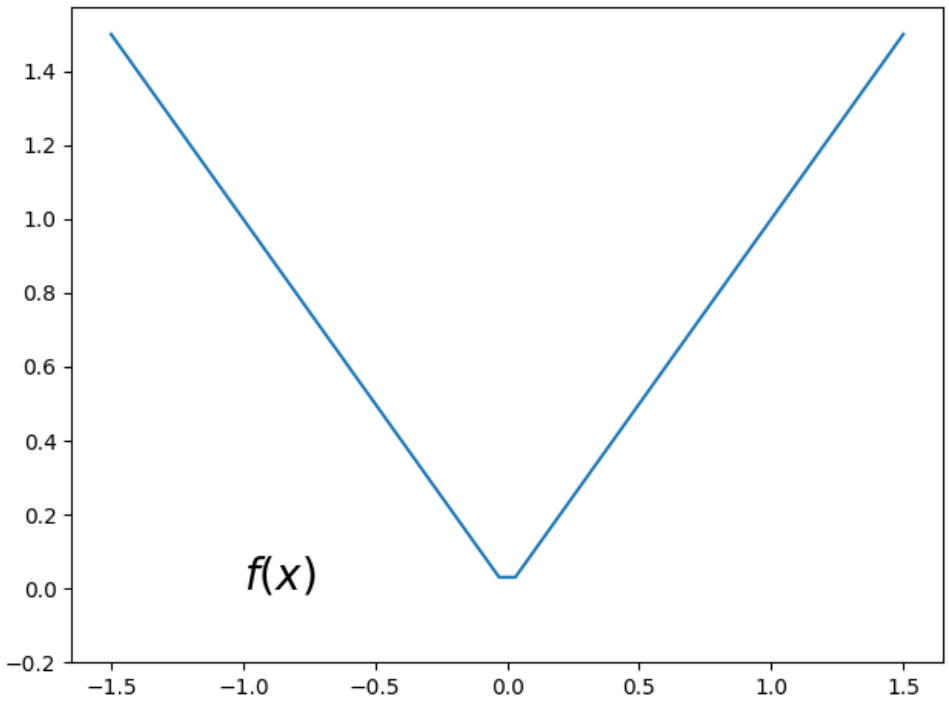


Figure 2.20: Non-Smooth Function

Definition 2.5.3. (Eigenvector and Eigenvalue) [68] Let A be an $n \times n$ matrix.

1. A_n Eigenvector of A is a nonzero vector v in R_n such that $Ay = \lambda$v, for some scalar λ.

2. A_n Eigenvalue of A is a scalar λ such that the equation Av $= \lambda$v has a nontrivial solution.

If Av $= \lambda$v for vA $= 0$, we say that λ is the eigenvalue for v, and that v is an eigenvector

for λ.

Steps to Find Eigenvalues of a Matrix

The following procedures must be taken in order to obtain the eigenvalues of a matrix:

Step 1: Double-check that matri A is a square matrix. Define the identity matrix I of the same order as well.

Step 2: Calculate the A- λI matrix, where λ is a scalar number.

Step 3: Equality the determinant of matrix $\mathrm{A} - \lambda \mathrm{I}$ with zero.

Step 4: Using the equation above, find all the possible values of λ, which are the matrix A's requisite eigenvalues.

Example 2.5.3. by using python Find the matrix's Eigenvalues and related Eigenvectors:

$$\begin{bmatrix} 7 & 0 & -3 \\ -9 & -2 & 3 \\ 18 & 0 & -8 \end{bmatrix}$$

Code 18 :Eigenvector and Eigenvalue

```
#importing numpy library
import numpy as np

# create numpy 2d-array
m = np.array([[7, 0, -3],
              [-9,-2,3],
              [18, 0, -8]])
print("Printing the Original square array:\n",m)
```

```
# finding eigenvalues and eigenvectors
w, v = np.linalg.eig(m)
# printing eigen values
print("Printing the Eigen values of the given square array:\n", w)
# printing eigen vectors
print("Printing Right eigenvectors of the given square array:\n", v)

output
...solution...
Printing the Eigen values of the given square array:
 [-2.  1. -2.]
Printing Right eigenvectors of the given square array:
 [[ 0.          0.40824829  0.28867513]
 [ 1.         -0.40824829 -0.40824829]
 [ 0.          0.81649658  0.8660254 ]]
```

Theorem 2.5.1. *Taylor Expansion Let $f : R^n \to \mathrm{R}$ be continuously differentiable.Then, for all $x_1, x_2 \in \mathrm{R^n}$, there is an $\beta \in [0, 1]$, such that [55]:*

$$f(x_2) = f(x_1) + \nabla f(\beta x_1 + (1-\beta)\mathrm{x_2})^T((x_2 - x_1)) \tag{2.27}$$

Furthermore, if f is twice continuously differentiable, then, for all $x_1, x_2 \in R^n$, there is $\beta \in [0, 1]$, such that

$$\begin{aligned} f(x_2) = f(x_1) + \nabla f(x_1)^T (x_2 - x_1) + \frac{1}{2}(x_2 - x_1)^T \nabla^2 f(\beta x_1) . \\ +(1-\beta)x_2)(x_2 - x_1) \end{aligned} \tag{2.28}$$

In addition, if $\mathrm{x}, u \in R^n$ and $\lambda \in \mathrm{R}$, we have:

$$\mathrm{f}(x + \lambda u) = \mathrm{f(x)} + \lambda u^T \nabla \mathrm{f}(x) + \frac{\lambda^2}{2} u^T \nabla^2 f(x) u + o\left(\lambda^2\right) \tag{2.29}$$

as

$$\lambda \to 0$$

In the following theorems, we give the optimality conditions for unconstrained optimization.

Definition 2.5.4. (Symmetric matrix) [14] A matrix that is square If $A = A^T, A$ is said to be symmetric. Symmetric matrices feature a number of unique characteristics, notably in terms of eigenvalues and eigenvectors.

Definition 2.5.5. (positive semidefinite) [66]

A symmetric $n \times n$ matrix A is called positive definite denoted by $\succ$ if $x'Ax \succ 0$ for all nonzero $x \in R^n, x \neq 0$ is define

$$x'Ax = \sum_{ij} A_{ij}x_i x_j \geq 0 \; for \; all \; x \in \mathbb{R}^n \tag{2.30}$$

Furthermore,we let A denote the set of symmetric positive semidefinite matrices:

$$A = \{X \in A \mid X \succeq 0\}.$$

Theorem 2.5.2. *All A positive definite matrix's eigenvalues are positive.*

Example 2.5.4. We use a Python script to compute the eigenvalues and check whether of the matrices below is positive definite, positive semi-definite, or negative definite

$$M = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Code 19 :Eigenvalue and Positive Definite

```
import numpy as np
M = np.array(([2, -1, 0], [-1, 2, -1], [0, -1, 2]), dtype=float)
eigs = np.linalg.eigvals(M)
print("The eigenvalues of M:", eigs)
if (np.all(eigs>0)):
 print("M is positive definite")
elif (np.all(eigs>=0)):
 print("M is positive semi-definite")
else:
 print("M is negative definite")
output
...solution...
The eigenvalues of M: [3.41421356 2.         0.58578644]
M is positive definite
```

Theorem 2.5.3. ***(First-order necessary conditions).(FONC)*** *[32] Assume that x is a local minimizer of the unconstrained optimization problem and that the objective function f in $x \in S$ open neighborhood is continuously differentiable . Then that is correct.*

$$\nabla \mathrm{f}(x) = 0$$

where $\nabla f(x) = \left[\frac{dfx}{dx_1}, \frac{dfx}{dx_2}, \ldots\ldots, \frac{dfx}{dx_n}\right]^T$ *signifies the objective's gradient function f at x and the superscript T denotes the transpose.*

Theorem 2.5.4. ***(Second-order necessary conditions)(SONC)*** *[32] Assume that x^* is the unconstrained optimization Problem's local minimizer. In the region of x^*, if the objective function f is twice continuously differentiable, then*

1- $\nabla f\left(x^*\right) = 0$

2- The Hessian $\nabla^2 f(x^*)$ is positive semi definite

Theorem 2.5.5. ***(Second-order sufficient conditions)(SOSC)*** *[32] Let x^* be a plausible unconstrained optimization problem solution [64]. Assume that in the vicinity of x, the objective function f is twice continuously differentiable, that*

1- $\bigtriangledown f(x) = 0$

2- $\bigtriangledown^2 f(x^*)$ *is positive definite*

3- x^* *is a strict local minimizer*

Definition 2.5.6. (descent direction) [66] If a direction d^k meets the following conditions, it is termed a descent direction of the objective function f at x_k A direction is called a descent direction of the objective function f at x_k if it satisfies :

$$(d_k)^T \nabla f\left(x^k\right) < 0$$

Definition 2.5.7. (stationary point) [4] A point $x^* \in R^n$ is called a stationary point if $\nabla f(x^*) = 0$ any local minimizer is a stationary point

Definition 2.5.8. Lagrangian [7] The Lagrangian function $L : R^n \times R^e \to R$ is represented by

$$L(\mathbf{x}, \lambda) = J(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i c_i(x) \tag{2.31}$$

where λ is the vector of the Lagrange multipliers $\lambda i, i = 1, \ldots, n$. Equivalently,

$$L(\mathbf{x}, \lambda) = J(\mathbf{x}) + \lambda \, \mathrm{Tc}(\mathbf{x}) \tag{2.32}$$

Definition 2.5.9. Karush Kuhu-Tucker(KKTconditions) [22] The KKT conditions (for a problem with differentiable variables) are the four criteria listed below. f_i, g_i:

1- Primal constraints: $f_i(x) \leq 0, \quad i = 1, \cdots, \text{I}$ and $g_i(x) = 0 \quad i = 1, \ldots, E.$

2- Complementary slackness: $\beta_i f_i(x) = 0, i = 1, \cdots, I$

3- Dual constraints: $\beta \geq 0.$

4- The Lagrangian gradient with respect to x vanishes:

$$\nabla f_0(x) + \sum_{i=1}^{I} \beta_i \nabla f_0(x) + \sum_{i=1}^{E} \gamma_i \nabla g_i(x) = 0 \tag{2.33}$$

Definition 2.5.10. continuous [40] A function f is said to be continuous at a point a if the following statements hold:

1.the function f is defined at a

2.the limit $\lim x \to a f(x)$ exists.

3.the limit is equal to $f(a), \lim x \to a f(x) = f(a)$

Example 2.5.5. Using Python code, we draw the continuous function

$$f(x) = \begin{cases} x^2 & \text{if } x \leq x_0 \\ ax + b & \text{otherwise} \end{cases}$$

solution

Code 20:continuous function

```
from symfit import parameters, variables, Fit, Piecewise, Eq, Model
import numpy as np
import matplotlib.pyplot as plt
x, y = variables('x, y')
a, b, x0 = parameters('a, b, x0')
y1 = x**2
y2 = a * x + b
model = Model({y: Piecewise((y1, x <= x0), (y2, x > x0))})
# As a constraint, we demand equality between the two models at the point x0
```

```
# to do this, we substitute x -> x0 and demand equality using ‘Eq‘
constraints = [
    Eq(y1.subs({x: x0}), y2.subs({x: x0}))]
xdata = np.linspace(-4, 4.)
ydata = model(x=xdata, a=0.0, b=1.0, x0=1.0).y
np.random
ydata = np.random.normal(ydata, 0.5)  # add noise
fit = Fit(model, x=xdata, y=ydata, constraints=constraints)
fit_result = fit.execute()
print(fit_result)
plt.scatter(xdata, ydata)
plt.plot(xdata, model(x=xdata, **fit_result.params).y)
plt.show()
```

output solution

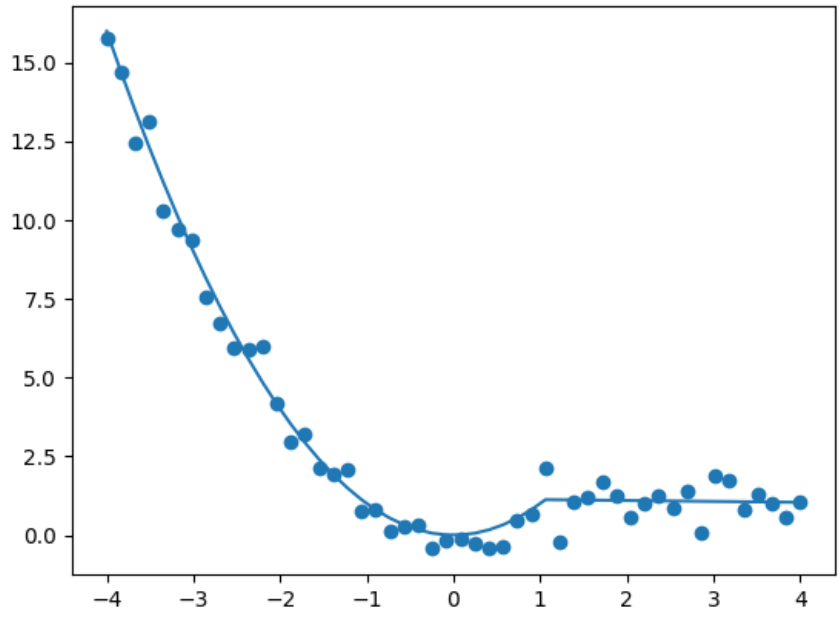


Figure 2.21: continuous function

CHAPTER 3

DESIGN OPTIMIZATION CODING

Introduction

In this chapter which develop a new mathematical model by using python language and also we improving theoretical convergent properties we will focus on several sports models to find the optimal solution based on new design algorithm by using python code. Some of the mathematical models is proposed in this chapter Also we will discuss the applications in our daily.

3.1 The Linear Programming

Linear programming is an optimization technique for solving problems when the objective function and constraints are represented as linear functions of the choice variables . In a linear programming problem, the constraint equations might be equalities or inequalities. Economists initially noticed the linear programming kind of optimization problem in the 1930s while creating approaches for the best resource allocation. During World War II, the United States Air Force looked for more efficient resource allocation methods and resorted to linear programming [2]. In 1947, group member George B. Dantzig developed the general linear programming problem and created the simplex technique of solution which is mean is a method for manually solving linear programming models employing slack variables, tableaus, and pivot variables to find the best solution to an optimization problem [61]. Simplex tableau is used to execute row operations on the linear programming model and to ensure that it is optimum . This is a huge step forward in the adoption of linear programming. After then, there was a lot of progress in both theoretical research and practical applications of linear programming. Among all of their contributions, Kuhn and Tucker's theoretical contributions had a significant effect on the development of duality theory in LP [13].

An L.P. program is an optimization program of the form :

$$\begin{cases} \min & c^T x \\ \text{s.t} & Ax \geq b \\ & Ax = b \\ & x \geq 0 \end{cases} \tag{3.1}$$

Linear programming is an optimization problem in which the goal and constraints are expressed as a linear function of the design variables. The limitations may be the equality, inequality, or both types .

Applications for linear programming may be found all over the place. Linear programming is on applied personal and professional level. When we wish to go from home to work as quickly as possible, we employ linear programming.

GEKKO [9]

The GEKKO Python package is a high-level abstraction of mathematical optimization problems .solves large-scale mixed-integer and differential algebraic equations with nonlinear programming solvers . Modes of operation include machine learning, data reconciliation, real-time optimization, dynamic simulation, and nonlinear model predictive control. In addition, the package solves Linear programming (LP), Quadratic programming (QP), Quadratically constrained quadratic program (QCQP), Nonlinear programming (NLP), Mixed integer programming (MIP), and Mixed integer linear programming (MILP). GEKKO is available in Python and installed with pip from PyPI of the Python Software Foundation.

3.1.1 Applications of linear programming problems

Several designs of algorithms are used Mathematical model of Ecominc, this example will explain about the optimization in the field of economics.The goal is to find the optimal solution

Example 3.1.1. The profit maximization problem faced by a furniture manufacturer The company uses timber and labor to produce the tables and chairs that the unit profit for the tables is 6, and the unit profit for the chairs 8. There is 300 foot board (bf) of lumber, and 110 working hours available. It takes 30 bf and 5 hours to make a table, 20 bf and 10 hours to make a chair. This table contains the information for the LP problem.

Resouree	$Table(x_1)$	$Chair(x_2)$	*Available*
wood	30	20	300
Labor	5	10	110
Unit proft	6dolar	8dolar	

solution:

$$\begin{cases} \max\limits_{x_1,x_2} z = 6x_1 + 8x_2 \\ \text{s.t} \quad 30x_1 + 20x_2 \leq 300 \\ \qquad\quad 5x_1 + 10x_2 \leq 110 \\ \qquad\quad x_1, x_2 \geq 0 \end{cases}$$

constraint 1 $30x_1 + 20x_2 \leq 300$ if $x_1 = 0 \rightarrow x_2 = 15 \Rightarrow (0, 15)$

$x_2 = 0 \rightarrow x_1 = 10 \Rightarrow (10, 0)$

constraint 2 $5x_1 + 10x_2 \leq 110$ if $x_1 = 0 \rightarrow x_2 = 11 \Rightarrow (0, 11)$

$x_2 = 0 \rightarrow x_1 = 22 \Rightarrow (22, 0)$

To find optimal solution by using python code

Code 21 :linear programming

```
# import the library pulp as p
import pulp as p

# Create a LP Maximization problem
Lp_prob = p.LpProblem('Problem', p.LpMaximize)

# Create problem Variables
x_1 = p.LpVariable("x_1", lowBound = 0) # Create a variable x_1 >= 0
x_2 = p.LpVariable("x_2", lowBound = 0) # Create a variable x_2 >= 0

# Objective Function
Lp_prob += 6 * x_1 + 8 * x_2

# Constraints:
Lp_prob += 30 * x_1 + 20 * x_2 <= 300
Lp_prob += 5*x_1 + 10 * x_2 <= 110

# Display the problem
print(Lp_prob)

status = Lp_prob.solve() # Solver
print(p.LpStatus[status]) # The solution status

# Printing the final solution
print(p.value(x_1), p.value(x_2), p.value(Lp_prob.objective))
output
```

```
x1=4.0

x2=9.0

f(x)=96.0

max proft

x1=22 ,x2=0 f(x)=132
```

---

we used the graphical methode to fined optimal solution by using python language

---

Code 22 :linear programming graphical

```
import numpy as np

import matplotlib.pyplot as plt

import inline as inline

# Construct lines

# x > 0

x = np.linspace(0, 2.5, 20)

# y >= 0

y1 = (300-30*x)/20

y2 = (110-5*x)/10

# Make plot

plt.plot(x, y1, '$y\geq0')

plt.plot(x, y2, '$y\geq0')

plt.xlim((0, 20))

plt.ylim((0, 20))

plt.xlabel('x-axises')

plt.ylabel('y-axises')

plt.show()
```

---

out put

solution

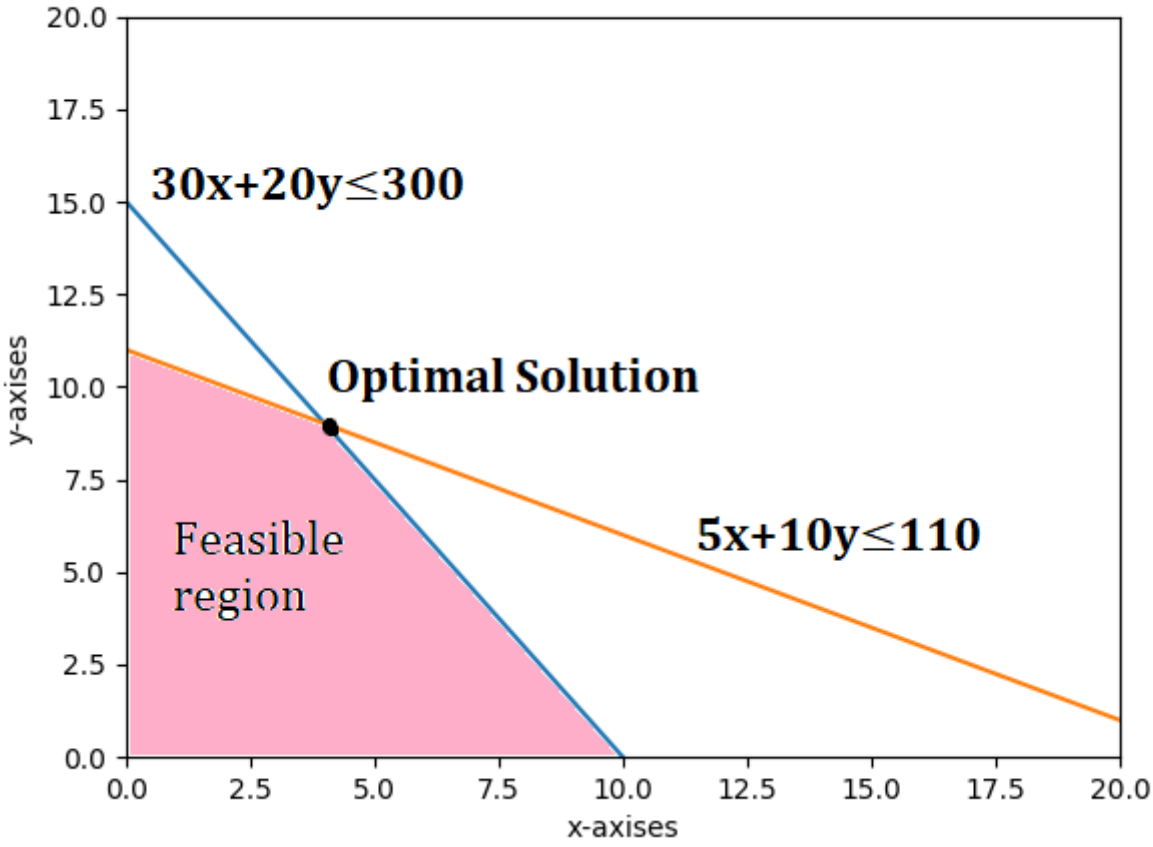


Figure 3.1: find optimal solution

Example 3.1.2. A health-food business would like to create a high-potassium blend of dried fruit in the form of a box of 10 fruit bars. It decides to use dried apricots, which have 407mg of potassium per serving, and dried dates, which have 271mg of potassium per serving The company can purchase its fruit through in bulk for a reasonable price. Dried apricots cost \$9.99 /lb. (about 3 servings) and dried dates cost \$7.99/ lb. (about 4 servings). The company would like the box of bars to have at least the recommended daily potassium intake of about 4700mg, but would like to keep it under twice the recommended daily intake. In order to minimize cost, how many servings of each dried fruit should go into the box of bars?

Solution We begin by defining the variables. Let,

$x =$ of servings of dried apricots

$y =$ of servings of dried dates

We next work on the objective function. For apricots, there are 3 servings in one pound. This means that the cost per serving is $\$9.99/3 = \$3.33$. The cost for x servings would thus be $3.33x$. For dates, there are 4 servings per pound. This means that the cost per serving is \$7.99/4 \$2.00. The cost for y servings would thus be $2.00y$. The total cost for apricots and dates would be

$$C = 3.33x + 2.00y$$

We have two major constraints (in addition to the constraints that $x \geq 0$ and $y \geq 0$, given that negative servings cannot be used): - Product must contain at least 4700mg of potassium - Product should contain no more than $4700 \times 2 = 9400$mg of potassium Mathematically, - There are $407x$mg of potassium in x servings of apricots and $271y$mg of potassium in y servings of dates. The sum should be greater than or equal to 4700mg of potassium, or $407x + 271y \geq 4700$ The same sum should be less than or equal to 9400mg of potassium, or $407x + 271y \leq 9400$. Thus we have,

$$\begin{cases} \text{Objective Function} : C = 3.33x + 2.00y \\ \text{Subject To Constraints} : 407x + 271y \geqslant 4700 \\ \qquad\qquad\qquad\qquad\qquad\quad 407x + 271y \leqslant 9400 \\ \qquad\qquad\qquad\qquad\qquad\quad x \geq 0, y \geq 0 \end{cases}$$

constraint1 $1407x + 271y = 4700$

x	y	*point*
0	17.3	$(0, 17.3)$
11.5	0	$(11.5, 0)$

constraint2 $407x + 271y = 9400$

x	y	*point*
0	23.1	$(0, 34.686)$
34.7	0	$(23.095, 0)$

by using python language to find minimize cost

Code 23 :linear programming

```
from gekko import GEKKO
m = GEKKO()
x = m.Var(lb=0, ub=50) # Product 1
y = m.Var(lb=0, ub=50) # Product 2
m.Minimize(3.33*x+2.00*y) # Profit function
m.Equation(407*x+271*y>=4700) # Units of A
m.Equation(407*x+271*y<=9400) # Units of B
m.solve(disp=False)
p1 = x.value[0]; p2 = y.value[0]
print ('Product 1 (x): ' + str(p1))
print ('Product 2 (y): ' + str(p2))
print ('Profit : ' + str(3.33*p1+2.00*p2))
output
Profit : 34.68634
```

we used the graphical methode to find optimal solution by using python language

Code 24 :linear programming

```
import numpy as np
import matplotlib.pyplot as plt
Minimize: z = 3.33x + 2y
Subject to: 407x + 271y >= 4700 or y1 = (4700 - 407x)/271 BLUE
        407x + 271y <= 9400 or y2 = (9400 - 407x)/271 ORANGE
x = np.linspace(0, 30)
y1 = (4700 - 407*x)/271
y2 = (9400 - 407*x)/271
plt.plot(x, y1)
plt.plot(x, y2)
plt.xlim((0, 40))
plt.ylim((0, 40))
plt.xlabel(r'$x-axises $')
plt.ylabel(r'$y-axises$')
plt.fill_between(x, y1, y2, color='Pink')
plt.show()
```

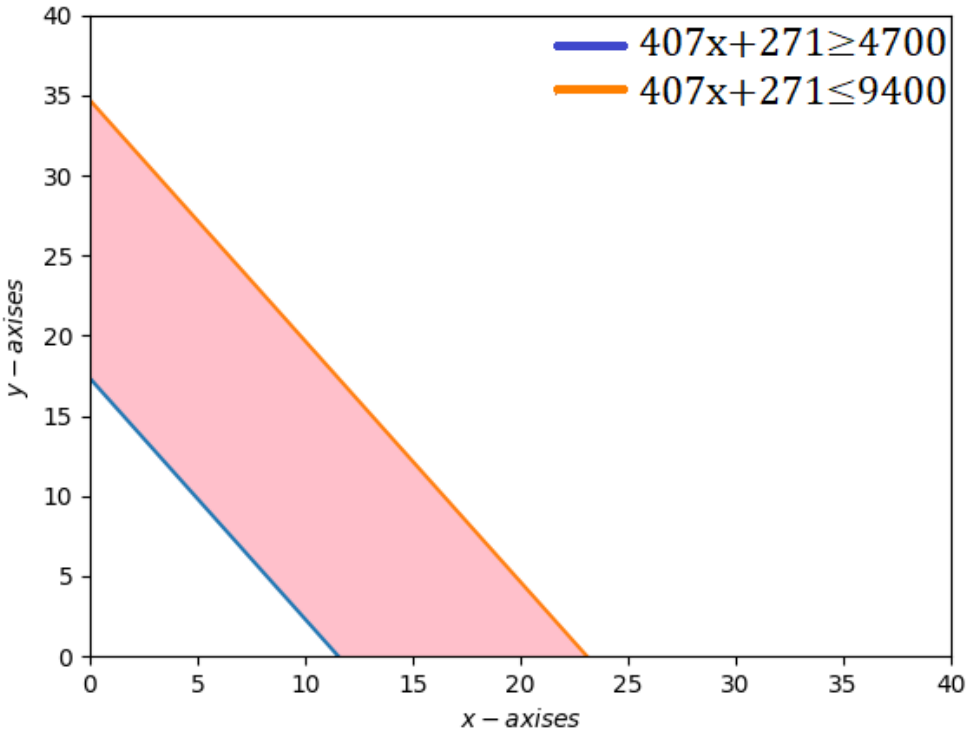


Figure 3.2: The Result of Above Problem Graphically

3.1.2 Dual Linear Programming Problem

Every linear programming problem, referred to as the primal, is accompanied by a second linear programming problem, referred to as the dual. These two situations have a lot of similarities and are quite intriguing. If you know the best answer to one, you can easily find the best solution to the other. It makes no difference whether problem is called the primordial since the dual of a dual is the primal. The answer to a linear programming problem may be achieved by solving either the primal or the dual, according to these features a linear programming problem has a dual, which is another (unique) linear program that is related with it [57]. In general, duality theory is concerned with the study of the relationship between two related linear programming problems, one of which is a maximization (minimization) problem and the other is a minimization (maximization) problem. Furthermore, the presence of an optimum

solution to one of these two problems ensures the existence of an ideal solution to the other, resulting in their extreme values being identical. There is no point to tackle both problems separately when the best solution to one of them may be obtained from the optimal solution to its twin [58] .

the primal linear program:

$$\begin{cases} \text{minimize} & z = c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{cases} \tag{3.2}$$

where A $\in R^{\mathrm{m}\times\mathrm{n}}$, b $\in R^{\mathrm{m}}$, and $c \in R^n$ and its dual linear program:

$$\begin{cases} \text{maximize} & w = b^T y \\ \text{subject to} & A^T \mathrm{y} \leq \mathrm{c} \\ & y \geq 0 \end{cases} \tag{3.3}$$

The goal is to find the optimal solution

Example 3.1.3. Find the dual of the following linear programming problem:

$$\begin{cases} \text{maximize} & z = 30x_1 - 10x_2 \\ \text{subject to} & -8x_1 - 2x_2 \leq 10 \\ & 5x_1 - 4x_2 \leq -9 \\ & x_1, x_2 \geq 0 \end{cases}$$

The dual of the linear programming problem is:

$$\begin{cases} \text{minimize} & z = 10u_1 - 9u_2 \\ \text{subject to} & -8u_1 + 5u_2 \geq 30 \\ & -2u_1 - 4u_2 \geq -10 \\ & u_1, u_2 \geq 0 \end{cases}$$

Example 3.1.4. Find the optimal solution of the primal problem and find the optimal solution of the dual problem by using python language :

$$\begin{cases} \max & 3x_1 + 4x_2 \\ \text{Subject to} & 2x_1 + 5x_2 \leq 30 \\ & 4x_1 + 2x_2 \leq 20 \\ & x_1, x_2 \geq 0 \end{cases}$$

Solution:the primal problem

Code 25 :optimal solution of primal problem

```
# import the library pulp as p
import pulp as p

# Create a LP Maximization problem
Lp_prob = p.LpProblem('Problem', p.LpMaximize)

# Create problem Variables
x_1 = p.LpVariable("x_1", lowBound = 0) # Create a variable x_1 >= 0
x_2 = p.LpVariable("x_2", lowBound = 0) # Create a variable x_2 >= 0

# Objective Function
```

```
Lp_prob += 3 * x_1 + 4 * x_2

# Constraints:
Lp_prob += 2 * x_1 + 5 * x_2 <= 30
Lp_prob += 4*x_1 + 2 * x_2 <= 20

# Display the problem
print(Lp_prob)

status = Lp_prob.solve() # Solver
print(p.LpStatus[status]) # The solution status

# Printing the final solution
print(p.value(x_1), p.value(x_2), p.value(Lp_prob.objective))
output
Optimal objective function 27.5
x1=2.5
x2=5.0
```

---

the dual problem:

$$\begin{cases} \min \quad 30y_1 + 20y_2 \\ \text{Subject to} \quad 2y_1 + 4y_2 \geq 3 \\ \qquad\qquad 5y_1 + 2y_2 \geq 4 \\ \qquad\qquad y_1, y_2 \geq 0 \end{cases}$$

The Dual Problem of the Primal Poroblem

Code 26 :Dual problem

```
 # import the library pulp as D
import pulp as D
# Create a dual LP Minimization problem
Dual_Lp_prob = D.LpProblem('Problem', D.LpMinimize)
# Create problem Variables
y_1 = D.LpVariable("y_1", lowBound = 0) # Create a variable x >= 0
y_2 = D.LpVariable("y_2", lowBound = 0) # Create a variable y >= 0
# Objective Function
Dual_Lp_prob += 30 * y_1 + 20 * y_2
# Constraints:
Dual_Lp_prob += 2*y_1 + 4 * y_2 >= 3
Dual_Lp_prob += 5*y_1 + 2*y_2 >= 4
# Display the problem
print(Dual_Lp_prob )
status = Dual_Lp_prob .solve() # Solver
print(D.LpStatus[status]) # The solution status
# Printing the final solution
print(D.value(y_1), D.value(y_2), D.value(Dual_Lp_prob.objective))
   Output
 Optimal objective 27.5
 y1=0.625
 y2=0.4375
```

3.1.3 Python implementation of the Simplex Algorithm [19]

The simplex technique is a linear programming procedure that may be used to find the best solution to a problem.When there are inequality constraints on a linear

optimization problem, this approach is used. The simplex method is a technique for methodically assessing vertices as viable solutions. The restrictions of a graph may be used to solve certain basic optimization problems. The Simplex Algorithm is implemented in Python.

Example 3.1.5. By use python code find Current function value

$$\begin{cases} \text{maximum} & -10R - 14T - 24M \\ \text{Subject to} & 19R + 29T + 36M \leq 91802 \\ & 14R + 23T + 17M \leq 42002 \\ & R \geq 0, T \geq 0, M \geq 0 \end{cases}$$

Code 27:Simplex method

```
import numpy as np
import scipy as sp
# Get matrices
c = [-10, -14, -24]
A = [[19, 29, 36], [14, 23, 17]]
b = [91802, 42002]
# define the upper bound and the lower bound
R = (0, None)
T = (0, None)
M = (0, None)
# Implementing the Simplex Algorithm
from scipy.optimize import linprog
# Solve the problem by Simplex method in Optimization
res = linprog(c, A_ub=A, b_ub=b,  bounds=(R, T, M), method='simplex',
 options={"disp": True})  # linear programming p[roblem
```

```
print(res) # print results
```

Out put Solution

Current function value: -59296.941176

Iterations: 2

message: 'Optimization terminated successfully.'

slack: array([2856.58823529, 0.])

success: True

x: array([0. , 0. , 2470.70588235])

3.2 Non-linear programming [51]

Nonlinear programming (NLP) is a mathematical term that refers to the method of solving an optimization problem in which some of the constraints or the objective functions are nonlinear. The computation of the extrema (maximam, minimam, or stationary points) of an objective function over a collection of unknown real variables, subject to the unknown of a system of equalities and inequalities, generally known as constraints, is an optimization problem. It is a branch of mathematical optimization that deals with non-linear problems. Let's use positive integers for n, m, and p. Allow X to be a subset of R^n, and f, gi, and hj to be real-valued functions on X for each $i \in \{1, 2, \ldots\ldots m\}$ and each $j \in \{1, 2, \ldots, p\}$, with at least one of f, g_i, and $h_j$being nonlinear. An optimization problem of the type is a nonlinear minimization problem.

$$\begin{cases} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0 \quad \text{for each} \quad i \in \{1, 2, \ldots\ldots m\} \\ & h_j(x) = 0 \quad \text{for each} \quad j \in \{1, 2, \ldots, p\} \\ & x \in X \end{cases} \tag{3.4}$$

A nonlinear maximization problem is defined in a similar way. Application of non linear

Example 3.2.1. Find the optimal solution of the problem

$$\begin{cases} \min & {x_1}^2{x_4}^3\left({x_1}^2+x_2+{x_3}^5\right)+{x_2}^4{x_3}^3 \\ \text{subject to} & {x_1}^5+{x_2}^3{x_3}^2+{x_4}^2\geq 73 \\ & 3{x_1}^3+{x_2}^2+{x_3}^2+{x_4}^2\geq 90 \\ & 1\leq x_1,x_2,x_3,x_4\leq 5 \end{cases}$$

Solution : by use python code

Code 28 : Non Linear programming

```
from gekko import GEKKO
m = GEKKO(remote=False)
# create variables
x1,x2,x3,x4 = m.Array(m.Var,4,lb=1,ub=5)
# initial values
x1.value = 1; x2.value = 5; x3.value = 5; x4.value = 1
m.Equation(x1**5+x2**3*x3**2+x4**2>=73)
m.Equation(3*x1**3+x2**2+x3**2+x4**2>=90)
m.Minimize(x1**2*x4**3*(x1**2+x2+x3**5)+x2**4*x3**3)
m.solve(disp=False)
print('Results')
print('x1: ' + str(x1.value[0]))
print('x2: ' + str(x2.value[0]))
print('x3: ' + str(x3.value[0]))
print('x4: ' + str(x4.value[0]))
     optimal
x1: 3.0723171242
x2: 1.0000047111
x3: 1.0000012022
```

```
x4: 1.0000001697

f(x1,x2,x3,x4)=108.974576662 which is the optimal solution
```

Example 3.2.2. Find the optimal solution of the Non-Linear problem :

$$\begin{cases} \text{minimize} & (-8+x_1)^2+(-2+x_2)^2 \\ \text{subject to:} & -\frac{1}{3}x_1 - x_2 \geq -4.5 \\ & -0.1x_1^2 + x_2 \geq 0 \\ & 0 \leq x_1 \leq 200 \\ & 0 \leq x_2 \leq 200 \\ & \forall i = 1,2 : x_i \in \mathbb{Z} \end{cases}$$

Solution :by use python code

Code 29 :**Non-linear programming**

```
from gekko import GEKKO

m = GEKKO()

x1,x2 = m.Array(m.Var,2,lb=0,ub=200)

x1.value = 0; x2.value = 200


m.Equation(-1/3*x1 -x2 >=-4.5)

m.Equation(-0.1*x1**2-x2>=0)

m.Minimize((-8+x1)**2 +(-2+x2)**2)

m.solve()

print(x1.value,x2.value)

out put

optimal solution  67.9282874221397
```

3.2.1 Lagrange Multiplier Method

The technique of Lagrange multipliers is an approach for finding the local maxima and minima of a function subject to equality requirements in mathematical optimization (subject to the condition that one or more equations have to be satisfied exactly by the chosen values of the variables). It is named after Joseph-Louis Lagrange, a mathematician [43]

$$\begin{cases} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0 \quad \text{for each} \quad i \in \{1, 2, \ldots\ldots m\} \\ & x \in X \end{cases} \tag{3.5}$$

The primary concept is to turn a limited problem into a form that can still be tested using the derivative test of an unconstrained problem. The link between the function's gradient and the constraints' gradients leads to the Lagrangian function, which is a reformulation of the original problem [10]:

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

λ is Parameter that control the movement of the function and put it in the desired direction

Example 3.2.3. Use Lagrangian method to solve constraint optimization problems

$$\begin{cases} \min & 5x_1^2 + 4x_2 \\ \text{such to} & f_1(x) = x_1 - 9x_2 + 4 \geq 0 \\ & f_2(x) = x_1 - 4x_2 - 5 \geq 0 \\ & x_1, x_2 >= 0 \end{cases}$$

solution: The Lagrangian

$$\mathrm{L}(\mathrm{x},\lambda) = \mathrm{f}(\mathrm{x}) + \lambda_1 f_1(x) + \lambda_2 f_2(x)$$

$$= 5x_1^2 + 4x_2 + \lambda_1 (x_1 - 9x_2 + 4) + \lambda_2 (x_1 - 4x_2 - 5)$$

$$\nabla L(x,\lambda) = \begin{pmatrix} 10x_1 + \lambda_1 + \lambda_2 \\ 4 - 9\lambda_1 - 4\lambda_2 \\ x_1 - 9x_2 + 4 \\ x_1 - 4x_2 - 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ -4 \\ 5 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} 12.2 \\ 1.8 \\ 96.8 \\ -218.8 \end{pmatrix}$$

Code 30 :Lagrange

```
import numpy as np
import matplotlib.pyplot as plt
A = np.matrix('10,0,1,1;,0,0,-9,-4;,1,-9,0,0;,1,-4,0,0')
b= np.matrix('0,;,4,;,-4;,5')
xs= np.array(np.linalg . inv(A)*b)
r=np.sqrt(5*xs[0]**2+4*xs[1])
print(xs)
print(r)
```

out put solution

$x_1 = [12.2]$

$x_2 = [1.8]$

$\lambda_1 = [96.8]$

$\lambda_2 = [-218.8]$

$f(x_1, x_2, \lambda_1, \lambda_2) = [27.41167634]$

3.2.2 Lagrange Duality

The term "Lagrangian duality theory" refers to a method for determining a bound or solving one optimization problem (the primal problem) by examining another optimization problem (the dual problem). More precisely, if the primal issue is more difficult to solve than the dual problem, the solution to the dual problem might offer a limit to the fundamental problem or the same optimum solution as the primal problem. The problem must be convex and meet a constraint condition in the latter instance. The original primal linear program is the dual of a dual linear program, according to the duality theory [25].

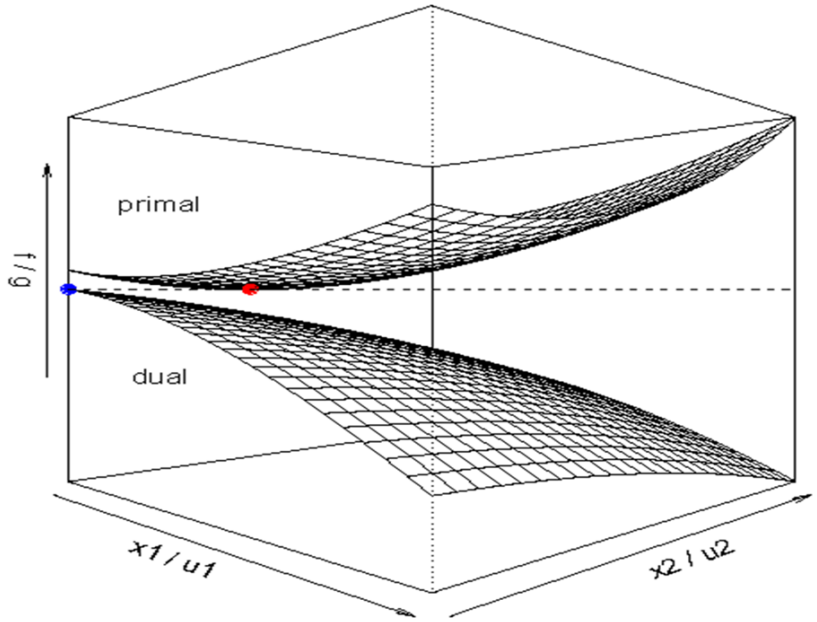


Figure 3.3: explain Lagrange Duality is bounded

The following is the conventional form of the optimization problem:

$$P\begin{cases} \text{minmize} & f_0(x) \\ \text{subject to} & h_j(x) = 0 \quad j = 1, \ldots\ldots, k \\ & g_i(x) \leq 0 \quad i = 1, \ldots\ldots\ldots, n \\ & x \in X \end{cases} \tag{3.6}$$

The Lagrangian function is defined as follows:

$$L(x, \mu, \delta) = f_0(x) + \sum_{j=1}^{k} \mu h_j(x) + \sum_{i=1}^{n} \delta g_i(x) \tag{3.7}$$

Where $\mu \in R^k$ and $\delta \in R^n$ are the dual variables (Lagrange multipliers). Then we define the Lagrangian dual function to be :

$$\emptyset(\mu, \delta) = \min_{x \in X} L(x, \mu, \delta) \tag{3.8}$$

The ideal value of the Lagrangian dual problem is denoted by d^*, the optimal value of the primal problem is denoted by p^*, and x^* is an optimal solution to the primal problem (P). With dual variables, the Lagrange dual problem (μ, δ) is given by:

$$D\begin{cases} \underset{x}{\text{maximize}} & \emptyset(\mu, \delta) \\ \text{subject to} & \delta \geq 0 \end{cases} \tag{3.9}$$

Example 3.2.4. Find the Lagrange dual of the problem :

$$\begin{cases} \text{minimize} \quad {x_1}^2 + {x_2}^2 \\ \text{subject to} \quad x_1 + x_2 \geq 4 \\ \qquad\qquad x_1; x_2 \geq 0 \end{cases}$$

let $\left\{x \in R^2 \mid x_1, x_2 \geq 0\right\} = R^2$

The Lagrangian is:

$$L(x, \lambda) = {x_1}^2 + {x_2}^2 + \lambda(4 - x_1 - x_2)$$

The Lagrangian dual function:

$$\begin{aligned} \theta(\lambda) &= \min_{x \in X} \left\{x_1^2 + x_2^2 + \lambda(4 - x_1 - x_2)\right\} \\ &= 4\lambda + \min_{x \in X} \left\{{x_1^2 + x_2}^2 - \lambda x_1 - \lambda x_2)\right\} \end{aligned}$$

For a fixed value of $\lambda \geq 0$, the minimum of $L(x, \lambda)$ over x $\in$ X is attained at
$x_1(\lambda) = \frac{\lambda}{2}, x_2(\lambda) = \frac{\lambda}{2}$

$$\rightarrow L(x(\lambda), \lambda) = 4\lambda - \frac{\lambda^2}{2} \quad \forall \lambda \geq 0$$

The dual function is concave and differentiable To maximize the value of the dual function:

$$\frac{dL}{d\lambda} = 4 - \lambda = 0$$

This implies $\lambda^* = 4, \theta\left(\lambda^*\right) = 8$

$$x\left(\lambda^*\right) = x^* = (2, 2)$$

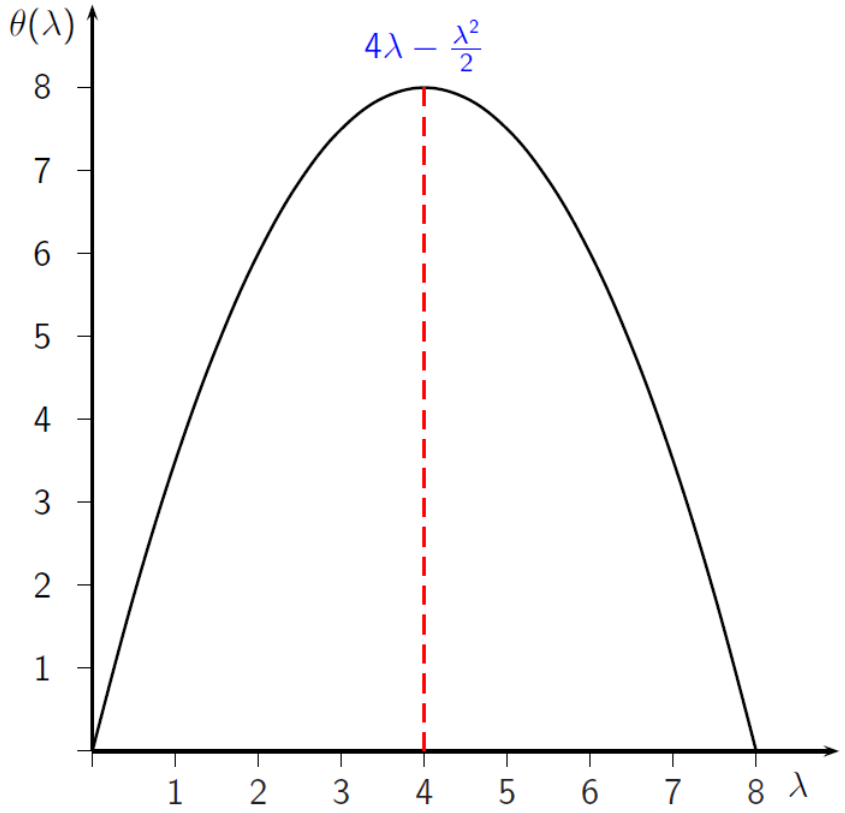


Figure 3.4: descrip the dual function

Example 3.2.5. Find The Optimal Solution by Using Python Code Of The non linear Problem :

$$(p)\begin{cases}\min\limits_{x_1,x_2} & (x_1-3)^2+(x_2+1)^2\\ \text{subject to} & x_1+x_2-1.5\leq 0\\ & 0\leq x_1\leq 1, -2\leq x_2\leq 1\end{cases}.$$

Code 31 :Prime of Lagrange

```
import inline as inline
import matplotlib as matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```

```
xr1 = np.linspace(0, 1)
xr2 = 1.5 - xr1
xr3 = (xr1 - 3)**2 + (xr2+1 )**2
xp3 = np.linspace(0, 16)
XP1, XP3 = np.meshgrid(xr1, xp3)
XP2 = -XP1 + 1.5
fig = plt.figure()
ax = Axes3D(fig)
ax.set_ylim(-2, 1)
x1 = np.linspace(0, 1)
x2 = np.linspace(-2, 1)
X1, X2 = np.meshgrid(x1, x2)
X3 = (X1 -3)**2 + (X2+1)**2
ax.plot_surface(XP1, XP2, XP3, cmap='gray', cstride=100,
rstride=100, edgecolor='b', alpha=0.15)
ax.plot_surface(X1, X2, X3, cmap='plasma', cstride=3,
rstride=3, edgecolor='r', alpha=0.5)
ax.plot(xr1, xr2, color='b')
ax.plot(xr1, xr2, xr3, color='k')
plt.show()
```

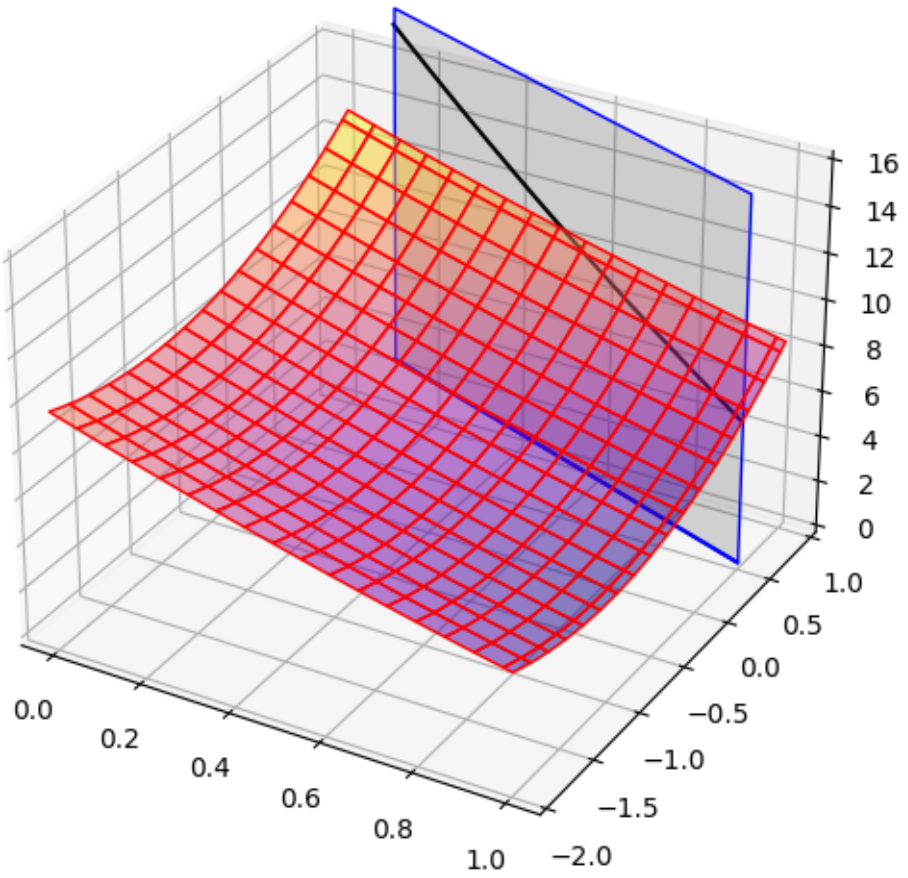


Figure 3.5: Primal Lagrange

To Find Lagrangian duality

The Lagrangian function that connects the objective function with its restrictions is

$$\mathcal{L} = (x_1 - 3)^2 + (x_2 + 1)^2 + \lambda\,(x_1 + x_2 - 1.5)$$

Compute the function derivatives differentiation of them results in

$$\frac{\partial \mathcal{L}_1}{\partial x_1} = 2x_1 + \lambda - 6, \quad \frac{\partial \mathcal{L}_2}{\partial x_2} = 2x_2 + \lambda + 2.$$

We test the gradients at the extremes to conduct the minimization of the Lagrangian in

each variable; the minimal point is denoted by x^*

$$\frac{\partial \mathcal{L}_1(0,\lambda)_1}{\partial x_1} = \lambda - 6 \geq 0 \qquad x_1^* = 0, \text{ if } \lambda \geq 6$$

$$\frac{\partial \mathcal{L}_1(1,\lambda)_1}{\partial x_1} = \lambda - 4 \leq 0 \qquad x_1^* = 1, \text{ if } 0 \leq \lambda \leq 4$$

$$\frac{\partial \mathcal{L}_1\left(x_1,\lambda\right)_1}{\partial x_1} = 2x_1 + \lambda - 6 = 0 \quad x_1^* = 3 - \frac{\lambda}{2}, \text{ if } 4 \leq \lambda \leq 6$$

and for $\mathcal{L}_2$,

$$\frac{\partial \mathcal{L}_2(-2,\lambda)_1}{\partial x_2} = \lambda - 2 \geq 0 \quad x_1^* = -2, \text{ if } \lambda \geq 2$$

$$\frac{\partial \mathcal{L}_2(1,\lambda)_1}{\partial x_2} = \lambda + 4 \leq 0 \quad \text{ never satisfied because } \lambda \geq 0$$

$$\frac{\partial \mathcal{L}_2\left(x_2,\lambda\right)_1}{\partial x_2} = 2x_2 + \lambda + 2 = 0 \quad x_2^* = -1 - \frac{\lambda}{2}, \text{ if } 0 \leq \lambda \geq 2.$$

The dual objective functions is the minimum of the Lagrangian function,

$$\phi(\lambda) = \min_{\boldsymbol{x} \in \chi} \mathcal{L} = \min_{\boldsymbol{x} \in \chi} \sum_{j=1}^{n} \mathcal{L}_j$$

or, the sum of the Lagrangian minimum on each variable, $\min \mathcal{L}_j$.

$$\phi(\lambda) = \left(x_1^* - 3\right)^2 + \lambda x_1^* + \left(x_2^* + 1\right)^2 + \lambda x_2^* - \frac{3}{2}\lambda$$

where x_1^* and x_2^* represent the points where the function is minimum. This can be divided

for each λ interval,

$$\begin{cases} -\frac{\lambda^2}{4} - \frac{3}{2}\lambda + 4, & 0 \leq \lambda \leq 2 \\ -\frac{5}{2}\lambda + 5, & 2 \leq \lambda \leq 4 \\ -\frac{\lambda^2}{4} - \frac{\lambda}{2} + 1, & 4 \leq \lambda \leq 6 \\ -\frac{7}{2}\lambda + 10, & \lambda \geq 6 \end{cases}$$

The dual problem

Code 32 :dual of lagrange

```
import inline as inline
import matplotlib as matplotlib

import matplotlib.pyplot as plt
import numpy as np

lambd = np.linspace(0, 10)



def phi(lambd):
   return ((0 <= lambd <=2)*(-lambd**2/4 -3/2*lambd + 4) +
            (2 < lambd <=4)*(-5*lambd/2 + 5) +
           (4 < lambd <= 6)*(-lambd**2/4 - lambd/2 +1) +
            (lambd >= 6)*(-7*lambd/2 +10))
p = []
for i in lambd:
    p.append(phi(i))
```

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.set_xlabel(r'$\lambda$')
ax.set_ylabel(r'$\phi$')

ax.plot(lambd, p, '-k')
plt.show()
```

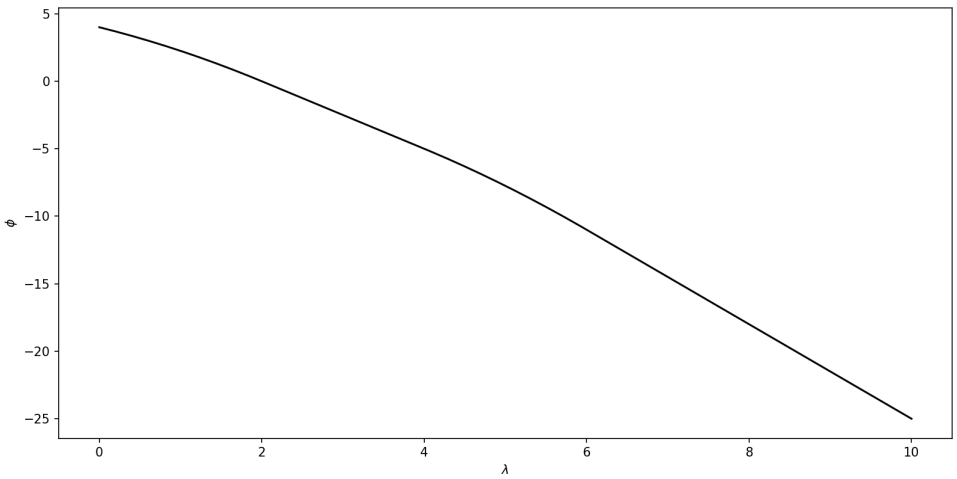


Figure 3.6: represent Lagrange Duality is bounded

3.2.3 Karush-Kuhn-Tucker(kkt)

In mathematical optimization, the Karush–Kuhn–Tucker (KKT) conditions, also known as the Kuhn–Tucker conditions, are first derivative tests (sometimes called first-order necessary conditions) for a solution in nonlinear programming to be optimal, provided that some regularity conditions are satisfied [63].

Example 3.2.6. For the following problem, locate all of the KKT point's ?

$$\begin{cases} \text{maximize} & x_1 + x_2 \\ \text{subject to} & x_1 + 2x_2 \leq 4 \\ & 2x_1 + x_2 \leq 6 \\ & x_1 x_2 \geq 0 \end{cases}$$

solution

First, express the problem in the conventional format that the KKT theory requires:

$$\begin{aligned} \text{maximize} \quad f_0(x_1, x_2) &= x_1 + x_2 \\ f_1(x_1, x_2) &= x_1 + 2x_2 - 4 \\ f_2(x_1, x_2) &= 2x_1 + x_2 - 6 \\ f_3(x_1, x_2) &= -x_1 \leq 0 \\ f_4(x_1, x_2) &= -x_2 \leq 0 \end{aligned}$$

Keep in mind the following when writing kkt:

$$\nabla \mathrm{f}_0 = \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} \quad \nabla f_1 = \begin{bmatrix} 1 \\ \\ 2 \end{bmatrix} \quad \nabla f_2 = \begin{bmatrix} 2 \\ \\ 1 \end{bmatrix} \quad \nabla f_3 = \begin{bmatrix} -1 \\ \\ 0 \end{bmatrix} \quad \nabla f_4 = \begin{bmatrix} 0 \\ \\ -1 \end{bmatrix}$$

$$\mathrm{L}\left((x_1, x_2), (\lambda_1, \lambda_2)\right) = x_1 + x_2 + \lambda_1 (x_1 + 2x_2 - 4) + \lambda_2 (2x_1 + x_2 - 6)$$

We can now write the KKT conditions for this problem as

1 (Primal Feasibility)

$$x_1 + 2x_2 \leq 4$$

$$2x_1 + x_2 \leq 6$$

$$x_1, x_2 \geq 0$$

2. (Dual Feasibility)

$$\begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} - \lambda_1 \begin{bmatrix} 1 \\ \\ 2 \end{bmatrix} - \lambda_2 \begin{bmatrix} 2 \\ \\ 1 \end{bmatrix} - \lambda_3 \begin{bmatrix} -1 \\ \\ 0 \end{bmatrix} - \lambda_4 \begin{bmatrix} -1 \\ \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix}$$

$$\lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0$$

3. Complementary Slackness

$$\lambda_1(x_1 + 2x_2 - 4) = 0$$

$$\lambda_2(2x_1 + x_2 - 6) = 0$$

$$\lambda_3(-x_1) = 0$$

$$\lambda_4(-x_2) = 0$$

4. (Stationarity of the Lagrangian) $= \nabla \mathrm{f}((x_1, x_2), (\lambda_1, \lambda_2, \lambda_3, \lambda_4))$

$$\lambda_1 + 2\lambda_2 - \lambda_3 = 1$$

$$2\lambda_1 + \lambda_2 - \lambda_4 = 1$$

$$\lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0$$

Since $\lambda_3, \lambda_4 \geq 0$, they act like surplus variables and we can write the Dual Feasibility

$$\lambda_1 + 2\lambda_2 \geq 1$$

$$2\lambda_1 + \lambda_2 \geq 1$$

$$\lambda_1, \lambda_2 \geq 0$$

It would now suffice to find values for $x_1, x_2, \lambda_1, \lambda_2, \lambda_3,$ and λ_4 the satisfy the KKT conditions and we could solve the linear programming problem P.

$$(x_1 + 2x_2 = 4) * -2$$

$$2x_1 + x_2 = 6$$

$$-3x_2 = -2$$

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

$$x_1 = \frac{8}{3}$$

we must have $\lambda_3 = \lambda_4 = 0$ because $x_1 > 0$ and $x_2 > 0$ at optimality and complementary slackness requires that: $\lambda_3(-x_1) = 0$ and $\lambda_4(-x2) = 0$ at optimality. This leaves λ_1 and λ_2

We know these must satisfy the equations. Therefore, we see that when $\lambda 1 = \lambda 2 = 1\backslash 3$, is satisfied. Thus: we have written

$$\begin{bmatrix} 1 \\ \\ 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ \\ 2 \end{bmatrix} - \frac{1}{3}\begin{bmatrix} 2 \\ \\ 1 \end{bmatrix}$$

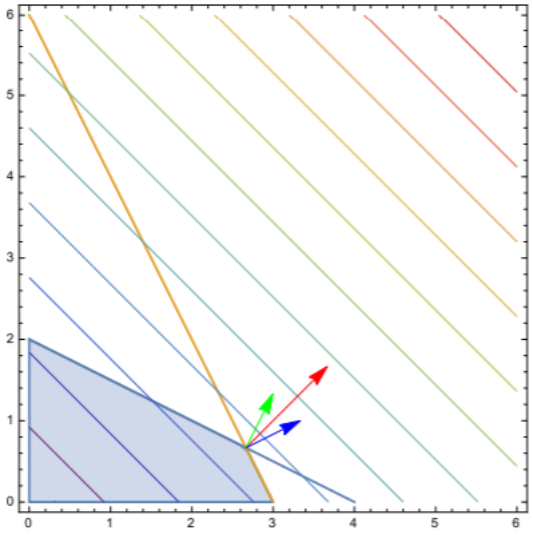


Figure 3.7: The objective function's slopes and the binding constraints at optimality and their geometric relation are illustrated

by use python code to solve example

Code 33:kkt condition

```
  from sympy import *
x1,x2,k,n = symbols('x1,x2,k,n')
f = x1+x2
g = x1+2*x2-4
h = 2*x1+x2-6
# I will construct the Lagrange equation
L = f+k*g+n*h
# Finding Derivatives, Building a KKT Case
dx1 = diff(L, x1) #Find the partial derivative of x1
print("dx1=",dx1)
dx2 = diff(L,x2)
#Find the partial derivative of x2
print("dx2=",dx2)
dk = diff(L,k) # partial derivative of k
print("dk=",dk)
dn = diff(L,n) # partial derivative of n
print("dn=",dn)
# Looking for a different solution
m= solve([dx1,dx2,dk,dn],[x1,x2,k,n])
print(m)
# Reset variables
x1=m[x1]
x2=m[x2]
k=m[k]
n=m[n]
# Calculate the value of the equation
f = x1+x2
```

```
print("The maximum value of the equation is:",f)


out put solution
dx1= k + 2*n + 1
dx2= 2*k + n + 1
dk= x1 + 2*x2 - 4
dn= 2*x1 + x2 - 6
{k: -1/3, n: -1/3, x1: 8/3, x2: 2/3}
The maximum value of the equation is: 10/3
```

Example 3.2.7. Use kkt condition to find the optimal solution of non-linear programming problem:

$$\begin{cases} f(x) = 10 - 9x_1 + (x_1)^2 + (x_2)^2 + x_1x_2 \\ g(x) = 5x_1 + 2x_2 - 12 \end{cases}$$

Solution :

Code 34:kkt condition

```
  from sympy import *
x1,x2,k = symbols('x1,x2,k')
f = 10-9*x1+(x1)**2+(x2)**2+x1*x2
g = 5*x1+2*x2-12
# I will construct the Lagrange equation
L=f-k*g
# Finding Derivatives, Building a KKT Case
dx1 = diff(L, x1) #Find the partial derivative of x1
print("dx1=",dx1)
dx2 = diff(L,x2)
#Find the partial derivative of x2
```

```
print("dx2=",dx2)
dk = diff(L,k) # partial derivative of k
print("dk=",dk)
# Looking for a different solution
m= solve([dx1,dx2,dk],[x1,x2,k])
print(m)
# Reset variables
x1=m[x1]
x2=m[x2]
k=m[k]
# Calculate the value of the equation
f = 10-9*x1+(x1)**2+(x2)**2+x1*x2
print("The minimum value of the equation is:",f)

out put solution
dx1= -5*k + 2*x1 + x2 - 9
dx2= -2*k + x1 + 2*x2
dk= -5*x1 - 2*x2 + 12
{x1: 66/19, x2: -51/19, k: -18/19}
The minimum value of the equation is: -215/19
```

3.3 Approximation Method

For minimizing a convex function $f : R^n \to R$ over a convex set X, approximation approaches are based on substituting f and X by approximations $\mathrm{F_k}$ and X_k respectively [17], at each iteration k, and finding

$$x \in \arg\min_{x \in X_k} F_k(x)$$

F_{k+1} and X_{k+1} are formed in the following iteration by refining the approximation, which is based on the new point x_{k+1} and perhaps on the previous points Xk, . . . , xo. Of course, this strategy only makes sense if the approximation problems are easier than the original. There are several approximation techniques available, each with its own set of goals and applications [16].

3.3.1 Approximations with Taylor Series [36]

A Taylor series approximationis when a number is represented as a polynomial that has a very similar value to the number in a neighborhood around a given x value:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^3(a)}{3!}(x-a)^3 + o\left(h^3\right)$$

Taylor series are incredibly useful tools for approximating functions that would otherwise be difficult to calculate, as well as assessing infinite sums and integrals. **The steps to approximate values are as follow :**

1.Compare the operation on the number in question to a function.

2.Assign a to a value that makes $f(a)$ simple to calculate.

3. Select x to make $f(x)$ the estimated number.

Example 3.3.1. by using Python to display the $cos(x)$ function together with the Taylor series approximations of the first, third, fifth, and seventh

Solution: Code python Approximations with Taylor Series

Code 35:Taylor Series Approximat of analytic cos(x)

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
x = np.linspace(-np.pi, np.pi, 200)
y = np.zeros(len(x))
labels = ['Third Order', 'fouth Order', 'Fifth Order', 'Seventh Order']
plt.figure(figsize = (10,8))
for n, label in zip(range(4), labels):
    y = y + ((-1)**n * (x)**(2*n+1)) / np.math.factorial(2*n+1)
    plt.plot(x,y, label = label)

plt.plot(x, np.cos(x), 'k', label = 'Analytic')
plt.grid()
plt.title('Taylor Series Approximations of Various Orders')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

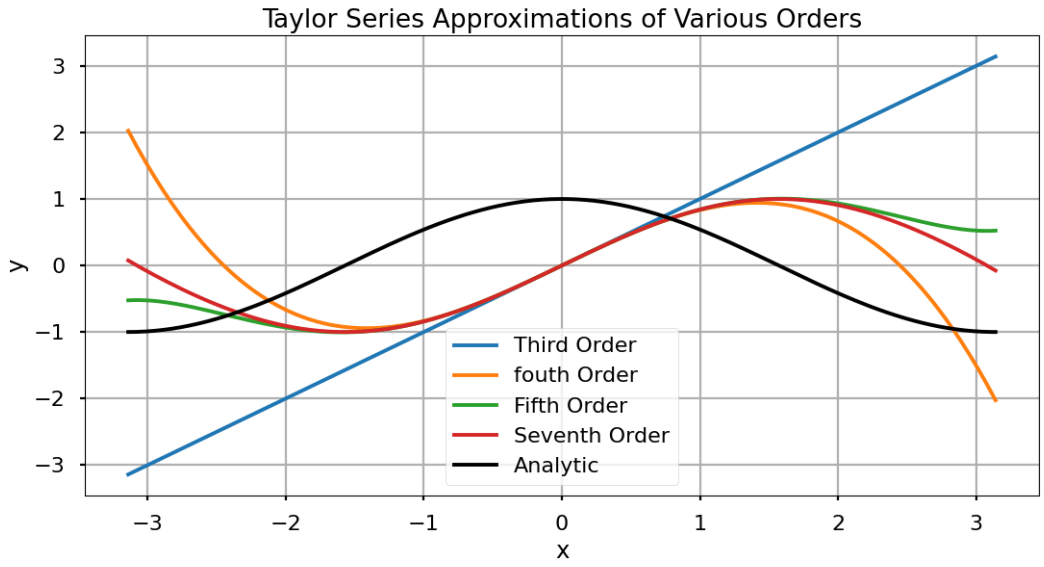


Figure 3.8: The black line represents the analytic, the blue line represents the first order of the Taylor Series, the orange line represents the third order of Taylor, the green line represents the fifth order of the Taylor Series, and The red line represents the seventh order of Taylor Series

Example 3.3.2. by using Python to display the sin(x) function together with the Taylor series approximations?

Solution:

Code 36:Taylor Series Approximat of function sin(x)

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import approximate_taylor_polynomial
x = np.linspace(-10.0, 10.0, num=100)
plt.plot(x, np.sin(x), label="sin curve")
for degree in np.arange(1, 15, step=2):
    sin_taylor = approximate_taylor_polynomial(np.sin, 0, degree, 1,
                                               order=degree + 2)
    plt.plot(x, sin_taylor(x), label=f"degree={degree}")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left',
```

```
            borderaxespad=0.0, shadow=True)
plt.tight_layout()
plt.axis([-10, 10, -10, 10])
plt.show()
```

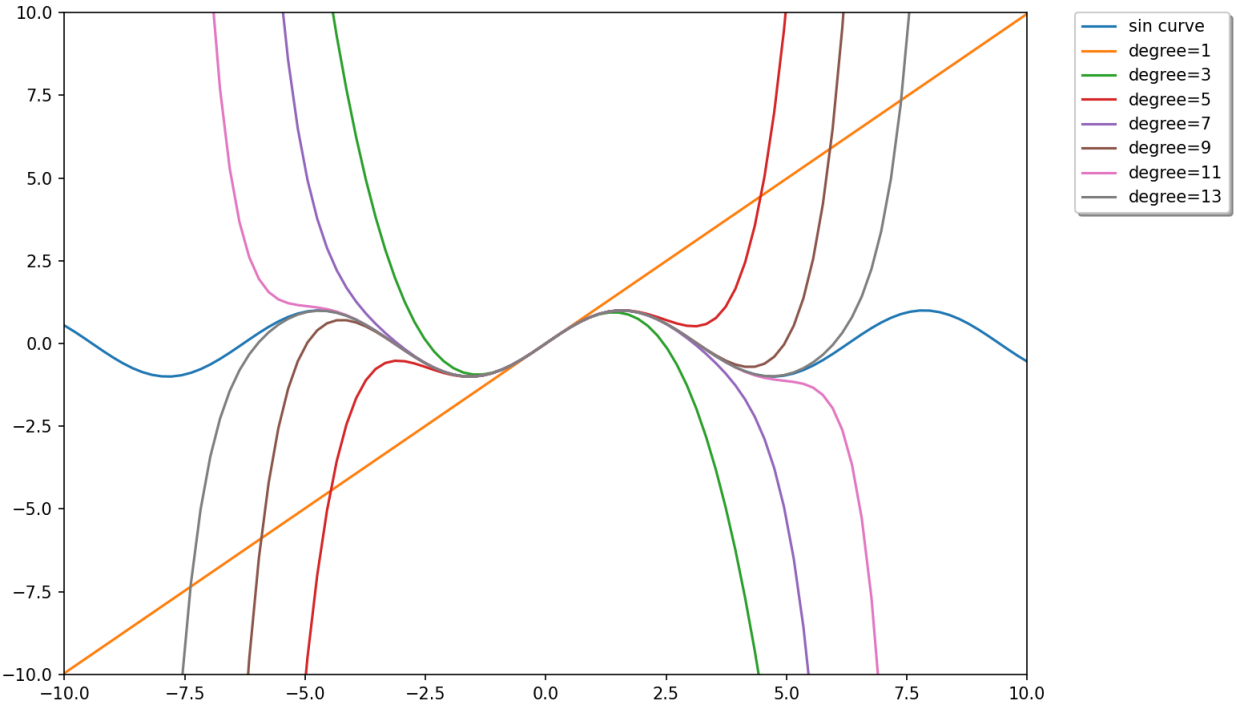


Figure 3.9: sin(x)

Example 3.3.3. by using Python to display the exp(x) function together with the Taylor series approximations?

Solution:

Code 37:Taylor Series Approximat of function exp(x)

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import approximate_taylor_polynomial
x = np.linspace(-10.0, 10.0, num=100)
plt.plot(x, np.exp(x), label="exp curve")
```

```
for degree in np.arange(1, 15, step=2):
    exp_taylor = approximate_taylor_polynomial(np.exp, 0, degree, 1,
                                               order=degree + 2)
    plt.plot(x, exp_taylor(x), label=f"degree={degree}")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left',
           borderaxespad=0.0, shadow=True)
plt.tight_layout()
plt.axis([-10, 10, -10, 10])
plt.show()
```

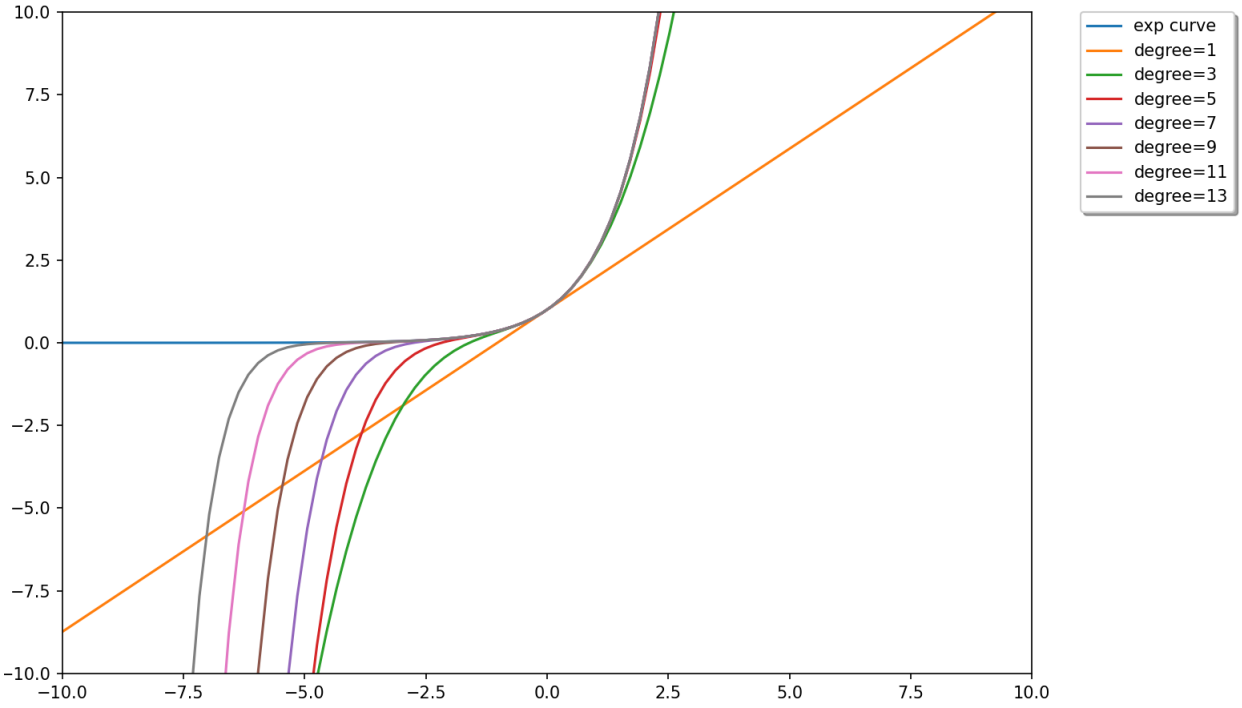


Figure 3.10: exp(x)

3.3.2 Penalty Function Method

The penalty optimization problem is transformed into different formulations, and numerical solutions are found by solving a series of unconstrained minimization problems [61]. For decades, penalty functions have been used in the literature on

restricted optimization. Exterior penalty functions, which punish infeasible solutions, and interior penalty functions, which penalize feasible solutions, are the two primary kinds of penalty functions [53]. The primary principle behind interior penalty functions is that an optimum solution necessitates the existence of an active (i.e., tight) restriction, such that the optimal solution stands on the edge of feasibility and infeasibility. Knowing this, a penalty is given to possible solutions, often known as "interior solutions," while the constraint is not active. This method is simple for a single constraint (albeit it hasn't been encountered in the evolutionary computation literature), but the construction of internal penalty functions for multiple constraints is substantially more difficult [27].

Defintion (penalty)

A function $\mathcal{P} : \mathfrak{R}^n \to \mathfrak{R}$ is called a penalty function if $\mathcal{P}$ satisfies [30]:

i. $\mathcal{P}(x)$ is continuous on $\mathfrak{R}^n$.

ii. $\mathcal{P}(x) = 0$ if and only if $x \in S$

iii. $\mathcal{P}(x) > 0$ for all $x \in \mathfrak{R}^n \mid S$

An often-used class of penalty functions for optimization problems with only inequality constraints is: $P(x) = \sum_{i=1}^{m} (\text{Max}\{0, g(x)\})^p$, where p is a positive integer We refer to the function $f(x) + \mu P(x)$ as an auxiliary function The purpose of a penalty is to convert a constrained function into an unconstrained function Consider the problem below [53]:

$$\begin{cases} \text{minmize} & f(x) \\ \text{subject to} & h(x) = 0 \\ & g(x) \leq 0 \\ & x \in R^n \end{cases}$$

Assume this problem is substituted by unconstrained problem, where $\mu > 0$ and $\rho > 0$ are a large number [71].

$$\min f(x) + \mu p(x)$$

$\min f(x) + \mu p(x)$	New objective function
μ	Penalty Weight
$p(x)$	Penalty function

We keep the objective function the same and add a penalty function to it that consists of m (Penalty Weight) I keep changing and trying to find its value to make the penalty function infeasible area very high and in feasible area remains herself until minimum and $p(x)$ is Penalty function define first [1]

$$p(x) = \begin{cases} + & \text{is not satisfy the constraint} \\ 0 & \text{is satisfy the constraint} \end{cases}$$

Equality constraint

$$\begin{cases} \min f(x) \\ x = x^c \end{cases}$$

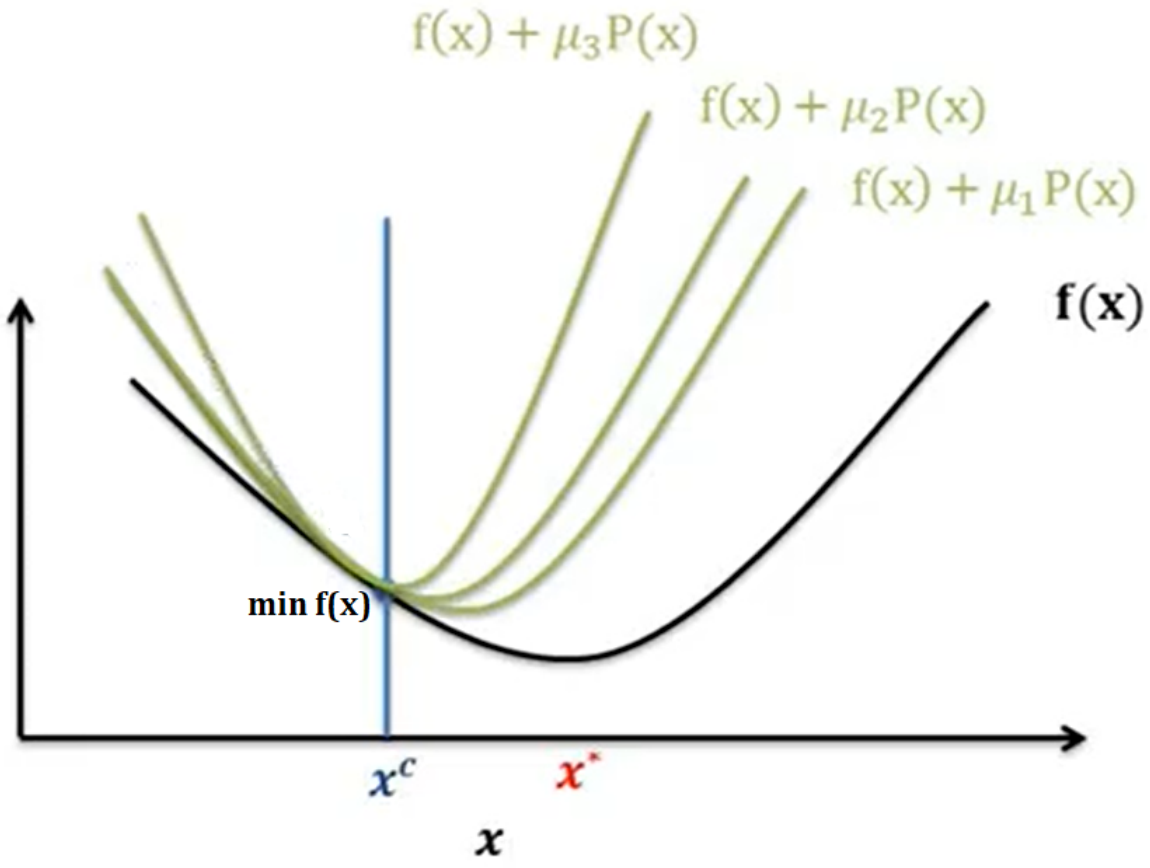


Figure 3.11: The black line represents the true function after adding μ_1 We notice that the ascending function appears to be rising above its value. change by μ. I keep going up in μ_3 as shown until we get a minimum function of it, which is equal to the constraint

Inequality constraint

$$\begin{cases} \min f(x) \\ x < x^c \end{cases}$$

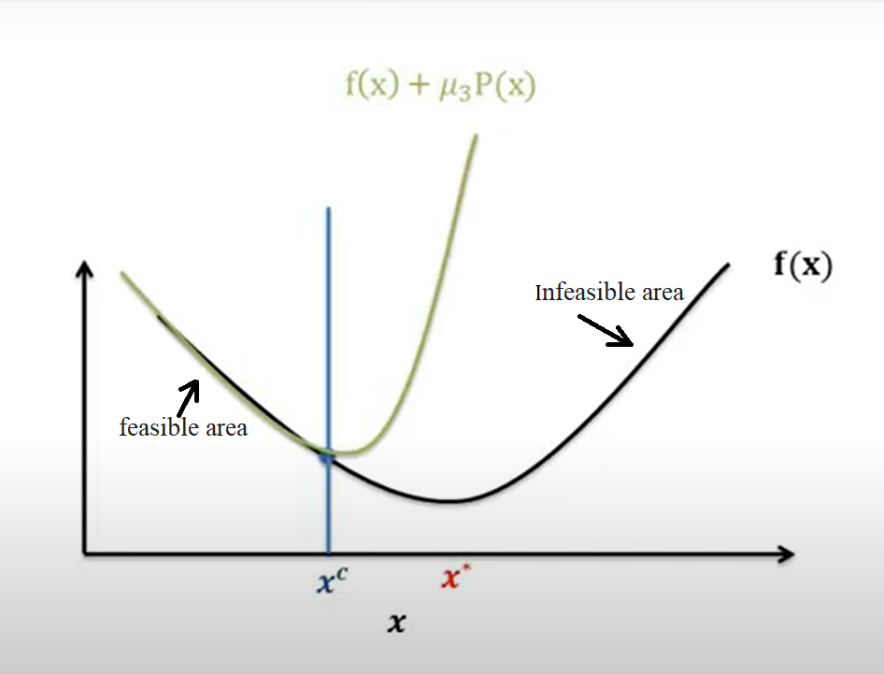


Figure 3.12: The black line represents the real function and $x < x^c$ the function will have the same value until we connect after c it appears to increase in value

- The penalty function:

· Quadratic penalty function, in which the penalty terms are the squares of the constraint violations.

$$\text{minf f}^{\text{new}}(X, C) = f(X) + \frac{\mu_1}{2}\sum_{i=1}^{m} h_i(X)^2 + \frac{\mu_2}{2}\sum_{j=1}^{n}\left(\max\{0, g_j(X)\}\right)^2$$

Example 3.3.4. : Consider the following problem:

$$\begin{cases} \underset{x_1,x_2}{\text{minimize}} & (x_1-3)^2 + 2x_2^2 \\ \text{subject to} & x_1 + x_2 = 4 \\ & (x_1 - x_2)^2 \leq 9 \\ & x_1, x_2 \geq 0 \end{cases}$$

After solving the prior problem with classical methods, we obtain $\overline{\mathbf{x}} = \left(\frac{7}{2}, \frac{1}{2}\right)$ with

objective function value $\frac{3}{4}$ Consider the following penalty function as a starting point for building theory around penalty functions:

$$\alpha\left(x_1, x_2\right)=\left(x_1+x_2-4\right)^2+\mathrm{m}\left\{0,\left(x_1-x_2\right)^2-9\right\}^2$$

constraint $_1=\left|x_1+x_2-4\right|$

constraint $_2=\max\left\{0,\left(x_1-x_2\right)^2-9\right\}$

Therefore, our goal is to minimize:

$$F\left(x_1, x_2\right)=\left(x_1-3\right)^2+2 x_2^2+\mu\left(x_1+x_2-4\right)^2+\mu \mathrm{m}\left\{0,\left(x_1-x_2\right)^2-9\right\}^2 .$$

We determine the minimum of each iteration based on the current value. Every best solution serves as a jumping off point for the following iteration. When the violation is less than , the loop is terminated. The scipy library is used to do all computations in Python. As can be seen in the table, we're getting close to the viable zone.

Code 38:Penalty Method

```
  import math
import numpy as np
import pandas as pd
from scipy . optimize import minimize
def penalty(x):
    obj = (x[0]-3)**2+2*x[1]**2
    obj += mu*(x[0]+x[1]-4)**2
    obj += mu*max(0, (x[0]-x[1])**2-9)**2
    return obj
eps = 10**-5
mu = 10
delta = 10
```

```
x0 = np.zeros(2)
violence = np . zeros(2)
violence [0] = max(0, (x0[0]-x0[1])**2-9)
violence [1] = abs(x0[0]+x0[1]-4)
viol = max(violence)
i = 1
while ( viol >= eps ):
    x0 = minimize(penalty , x0).x
    violence [0] = max(0, (x0[0]-x0[1])**2-9)
    violence [1] = abs (x0[0]+x0[1]-4)
    viol = max(violence)
    objective = (x0[0]-3)**2+2*x0[1]**2
    vi = 2*mu*(x0[0]+x0[1]-4)
    ui = 2*mu*max(0, (x0[0]-x0[1])**2-9)
    print ("{} -> mu= {}, x = {},vi = {},ui = {} viol = {},
     obj-value = {}, penalty = {}".
    format (i, mu, x0, vi, ui, viol, objective , penalty(x0)))
    mu = mu*delta
    i +=1
```

Iteration	μ	x_k	objective value	violation	penalty
1	10	$(3.46544586, 0.4648014)$	0.64872	0.06975	0.697524529300386
2	100	$(3.49631232, 0.49624341)$	0.73884	0.00744	0.7443997738138434
3	1000	$(3.49962861, 0.49962167)$	0.74887	0.00075	0.7494361881650267
4	10000	$(3.49996271, 0.49996202)$	0.74988	7.527	0.7499435813936578
5	100000	$(3.49999615, 0.49999608)$	0.74999	7.77561	0.7499943652807513

Table 3.1: Penalty Function

In this table3.1 Five Iteration In each Iteration, μ is a variable and not fixed, x_k represents the value of x_1 and x_2 for each Iteration, v_i and u_i represents the values of the first and second constraints, the objective value represents the objective function, and Violation represents the greatest value after compensating the value of x_1, x_2 in the two constraints and finally the penalty value after applying the law

3.3.3 Interior Penalty Function Methods

Interior Penalty techniques are also known as Barrier methods or interior point approaches. Martinez and Santos (1995), Nash and Sofer (1993), Wright (1992), Luenberger and Ye (2008) [65]produced some theoretical findings. These techniques are useful for form-related problems:

$$\begin{cases} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & g_i(x) \leq 0 \quad i = 1, \ldots\ldots\ldots, n \\ & x \in R^n \end{cases}$$

Where $f, g_1, \ldots, g_n$ are continuous functions defined on R^n these strategies turn a restricted problem into an unconstrained one, but the barrier prevents the present solution from ever leaving the viable zone. This necessitates the existence of nonempty interiors in the feasible sets. As a consequence, they're most often utilized to solve problems with merely inequality restrictions. As a result, once an unconstrained minimization is started from any possible x_1, the successive points produced will always be inside the feasible area, since the constraint boundaries function as barriers throughout the minimizing process.

Defintion(interior penalty Method) A function B : $R^n \to R$ is called an interior penalty (Barrier) function if B(x) satisfies:

1. $B(x)$ is continuous on $\mathbb{R}^n$
2. $B(x) \geq 0$ each $x \in$ intS, where S denotes the feasible region for Problem
3. $B(x) \to +\infty$ as x approaches the border of S.

write the barrier problem as:

$$\begin{cases} \theta(x, \mu) = \text{ minimize } f(x) + \alpha B(x) \\ \text{subject to } g_i(x) \leq 0, i = 1, 2, \ldots, n \\ x \in \mathbb{R}^n. \end{cases}$$

where $\alpha > 0$, it is so-called barrier parameter and we pick α small ($\alpha \to 0$)

The most commonly used types of interior penalty functions are:

1. Inverse function: $B(x) = -\sum_{i=1}^{M} \frac{1}{g_i(x)}$, for $g_i(x) < 0$
2. Logarithm function: $B(x) = -\sum_{i=1}^{M} \log[-g_i(x)]$, for $g_i(x) < 0$

From this we observe that the barrier problem itself is a constrained problem, and actually, the constraint is some what more complex than in the true problem. The characteristic of this problem, that it can be solved by using an unconstrained search

technique

Example 3.3.5.

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

$$\text{subject to} \quad g(x) = 1 - x_1 - x_2 \leq 0$$

$$x \in \Re^2$$

. **Solution**: Define the barrier function

$$P(x) = -\log[-g(x)] = -\log[x_1 + x_2 - 1]$$

The unconstrained problem is

$$\text{Minimize } \phi_{\mu_k} = x_1^2 + 2x_2^2 - \mu_k \log[x_1 + x_2 - 1].$$

The necessary conditions for the optimal solution $\nabla f(x) = 0$ yield the following:

$$\frac{\partial \phi_{\mu_k}}{\partial x_1} = 2x_1 - \frac{\mu_k}{(x_1 + x_2 - 1)} = 0$$
$$\frac{\partial \phi_{\mu_k}}{\partial x_2} = 4x_2 - \frac{\mu_k}{(x_1 + x_2 - 1)} = 0$$

and we get,

$$x_1 = \frac{1 \pm \sqrt{1 + 3\mu_k}}{3}$$

and

$$x_2 = \frac{1 \pm \sqrt{1 + 3\mu_k}}{6}$$

Since the negative signs lead to infeasibility, we have

$$x_1 = \frac{1 + \sqrt{1 + 3\mu_k}}{3} \text{ and } x_2 = \frac{1 + \sqrt{1 + 3\mu_k}}{6}$$

Starting with $\mu_1 = 1, \gamma = 0.1$ and $x_1 = (1, 0.5)$ and using a tolerance of 0.005 , we have the following: Thus, this solution approach to the exact optimal solution $x^* = (2/3, 1/3)$.

Code 39:interior penalty Method

```
import math
m=int(input("k= "))
x1=(1+(math.sqrt(1+(3*m))))/3
x2=(1+(math.sqrt(1+(3*m))))/6
g=1-x1-x2
p=-(math.log(-g,10))
mp=m*p
fx=((x1**2)+(2*(x2**2)))
fii=((x1**2)+(2*(x2**2)))-(m*(math.log((x1+x2-1),10)))
print("x:(",x1,",",x2,")")
print("g(x): ",g)
print("P(x): ",p)
print("MP(x): ",mp)
print("f(x): ",fx)
print("fii(x): ",fii)
```

out put solution by using python code.From the table, we observe that every points at each iteration is in the interior of the feasible region, and the final solution itself remains in the interior

k	μ_k	x^k	$g\left(x^k\right)$	$P\left(x^k\right)$	$\mu_k P\left(x^k\right)$	$f\left(x^k\right)$	$\phi_{\mu_k}\left(x^k\right)$
1	1	$(1.000, 0.5000)$	-0.5000	0.30103	0.3010	1.5	1.80103
2	0.1000	$(0.714, 0.357)$	-0.071	1.1487	0.1149	0.765	0.8788
3	0.0100	$(0.672, 0.336)$	-0.008	2.0969	0.02097	0.677	0.6979
4	0.0010	$(0.6672, 0.3336)$	-0.0008	3.0969	0.003097	0.668	0.6708
5	0.0001	$(0.6666, 0.3333)$	-0.0001	4.6576	0.000466	0.6667	0.6672
6	0.00001	$(0.666671, 0.333335)$	-0.000007	5.6576	0.0000566	0.666667	0.6667

Table 3.2: **Interior Penalty Function Methods**

In this table3.2, k represents the number of iteration, μ is a variable in each iteration, x^k represents the value of x_1, x_2, and $g(x^k)$ represents the constraint value, $P(x^k)$ represents the barrier function and $f(x^k)$ represents the objective function and $\phi_{\mu_k}\left(x^k\right)$ represents interior penalty Method

3.3.4 Exterior Penalty Function Methods

A limited problem may be turned into a single unconstrained problem or a series of unconstrained problems using exterior penalty function techniques If we use a large penalty factor, the search will trend to local optimums, but if we use a small penalty factor, we will have to spend a long time searching the feasible area. As a result, finding an efficient penalty factor is a significant advancement in the limited optimization problem domain. We suppose that the optimization problem has the following

properties:

$$\begin{cases} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & h_i(x) = 0, \quad i = 1, 2, \ldots, k \\ & g_j(x) \leq 0, \quad j = 1, 2, \ldots, n \\ & x \in \mathbb{R}^n. \end{cases}$$

Where $f, h_1, \ldots, h_k, g_1, \ldots, g_n$ are continuous on $\mathbb{R}^n$, suppose this problem is substitute by the following unconstrained problem:

$$E(x, \mu) = f(x) + \mu\Psi(x)$$

Subject to $x \in \mathbb{R}^n$ Where the penalty function $\Psi(x)$ is defined by

$$\Psi(x) = \sum_{j=1}^{n} \| \text{ m}\{0, g_j(x)\}\|^q + \sum_{i=1}^{k} \|h_i(x)\|^q$$

Example 3.3.6.

$$\begin{cases} \underset{x}{\text{minimize}} & f(x) = 2\text{x} \\ \text{subject to} & g(x) = 3 - x \leq 0, \\ & x \in \text{R}. \end{cases}$$

Note that the minimizer is $x^* = 3$

Solution

Let $\alpha(x) = [\max\{g(x), 0\}]^2$

i.e., $\alpha(x) = 0$, for $x \geq 3$ or $\alpha(x) = (3 - x)^2$, for $x < 3$.

If $\alpha(x) = 0$, then the optimal solution to minimize $\text{F}(x) = f(x) + \mu\alpha(x)$ is at $x^* = -\infty$ and this is infeasible. So,

$$\text{F} = 2x + \mu(3 - x)^2.$$

Since this function is quadratic form, we can evaluate the minimizer by using first derivative,

$F'(x) = 2 - 2\mu(3 - x) = 0$. This implies that $x = 3 - \frac{1}{\mu}$, which converges to $x^* = 3$ as $\mu \to \infty$. Therefore, the minimizer of the original problem is $x^* = 3$

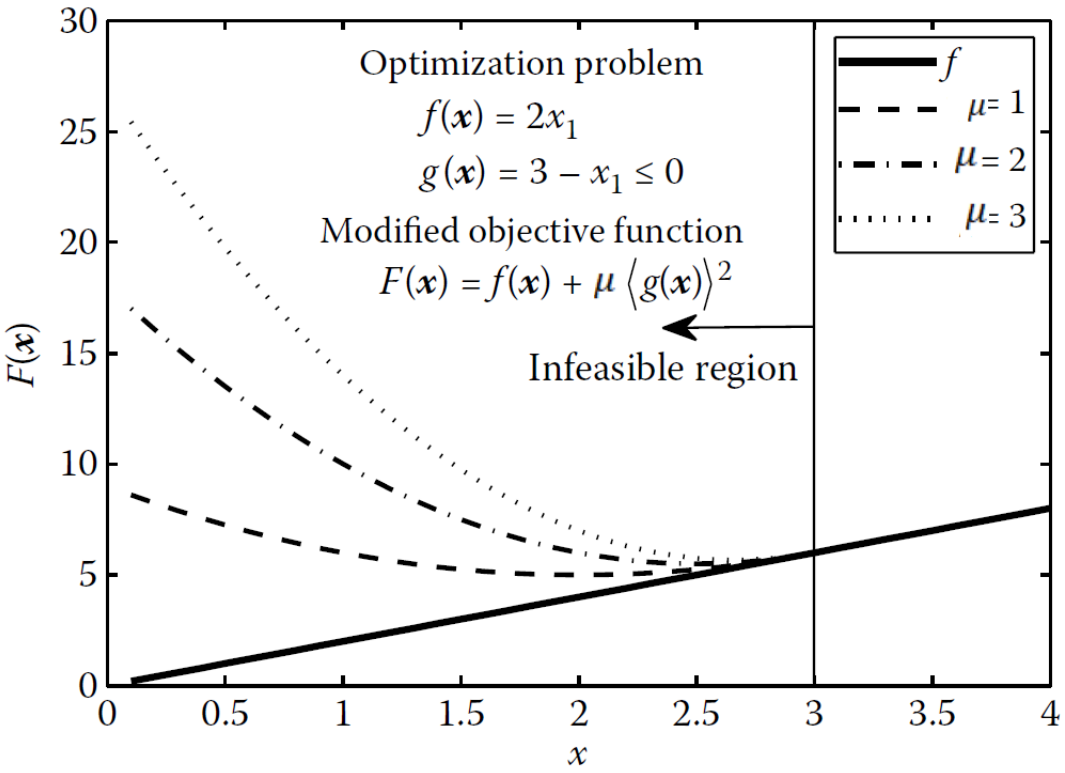


Figure 3.13: Exterioe penalty method

Code 40:Exterioe penalty method

```
  import math
import numpy as np
import pandas as pd
from scipy . optimize import minimize

def exterior(x):
    obj = (2*x[0])
    obj += mu*(3-x[0])**2
    return obj

eps = 10**-5
mu = 10
delta = 10
x0 = np.zeros(2)
violence = np . zeros(2)
violence [0] = abs(3-x0[0])
viol = max(violence)
i = 1
while ( viol >= eps ):
    x0 = minimize(exterior , x0).x
    violence [0] = abs (3-x0[0])
    viol = max(violence)
    objective = (2*x0[0])
    vi = 2*mu*(3-x0[0])
    print ("{} -> mu= {}, x = {},vi = {},viol = {}, obj-value = {}, exterior = {}")
    format (i, mu, x0, vi, viol, objective , exterior(x0)))
```

```
mu = mu*delta
i +=1
```

k	μ	x_k	objective value	vi	violation	exterior
1	10	(2.899)	5.799999917851817	2.00000082	0.100000041074	5.900000000000017
2	100	2.99	5.97999999017209	2.00000098	0.0100000049139	5.990000000000002
3	1000	2.99899	5.997999985	2.0000144	0.0010000072057	5.999000000000052
4	10000	2.99989	5.9997999851464	2.000148	0.0001000074267	5.999900000000552
5	100000	2.99998	5.99997998510464	2.001489	1.000744767676	5.999990000005547
6	1000000	2.99998	5.999997985099326	2.0149006	061.00745033693	5.999999000055

Table 3.3: **Exterior Penalty Method**

In this table 3.3 Five Iteration In each Iteration, μ is a variable and not fixed, x_k represents the value of x_1 and x_2 for each Iteration, v_i and u_i represents the values of the first and second constraints, the objective value represents the objective function, and Violation represents the greatest value after compensating the value of x_1, x_2 in the two constraints and finally the penalty value after applying the law

Conclusions

The optimization models are a subset of numerical optimization. This study's objective is to find the greater design which leads us to the optimal solution of the linear and nonlinear optimization problems, this research provided efficient algorithms which imply good design codes, and new approaches to algorithms to solve different mathematical modeling applications for optimization. This study developed design codes based on different mathematical models which included, Tyler's rounding technique, linear programming, nonlinear programming, double linear, nonlinear, Lagrange multiplier, Karush-Kuhn-Tucker(KKT) condition, and internal penalty and external penalty methods. This study depends on open software which is Python language. The implementation of all design codes gave us less than allocation memory of the computer process. Moreover, this research focus on developing the theoretical converges properties.

REFERENCES

[1] Ahmed Al-Jilawi. *Solving the Semidefinite Programming Relaxation of Max-cut Using an Augmented Lagrangian Method*. Northern Illinois University, 2019.

[2] Ahmed Alridha, Fadhil Abdalhasan Wahbi, and Mazin Kareem Kadhim. Training analysis of optimization models in machine learning. *International Journal of Nonlinear Analysis and Applications*, 12(2):1453–1461, 2021.

[3] Ping An, Hongbin Chen, Haixia Ren, Juan Su, Mengyao Ji, Jian Kang, Xiaoda Jiang, Yifei Yang, Jiao Li, Xiaoguang Lv, et al. Gastrointestinal symptoms onset in covid-19 patients in wuhan, china. *Digestive diseases and sciences*, 66(10):3578–3587, 2021.

[4] Niclas Andréasson, Anton Evgrafov, and Michael Patriksson. *An introduction to continuous optimization: foundations and fundamental algorithms*. Courier Dover Publications, 2020.

[5] Neculai Andrei et al. *Nonlinear conjugate gradient methods for unconstrained optimization*. Springer, 2020.

[6] Rajesh Kumar Arora. *Optimization: algorithms and applications*. CRC Press, 2015.

[7] Ashwini Kumar Arya, Seong Jin Kim, Sungik Park, Dong-Hoon Kim, Rehab S Hassan, Kyeongjun Ko, and Sanghoek Kim. Shark-fin antenna for railway

communications in lte-r, lte, and lower 5g frequency bands. *Progress In Electromagnetics Research*, 167:83–94, 2020.

[8] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan M Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[9] LDR Beal, D Hill, and RA Martin. and hedengren. *JD" GEKKO Optimization Suite." Processes*, 6(8), 2018.

[10] Brian Beavis and Ian Dobbs. *Optimisation and stability theory for economic analysis*. Cambridge university press, 1990.

[11] David M Beazley. *Python essential reference*. Addison-Wesley Professional, 2009.

[12] Genrich Belitskii et al. *Matrix norms and their applications*, volume 36. Birkhäuser, 2013.

[13] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[14] Dimitri Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.

[15] Dimitri Bertsekas, Angelia Nedic, and Asuman Ozdaglar. *Convex analysis and optimization*, volume 1. Athena Scientific, 2003.

[16] Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[17] Dimitri P Bertsekas, W Hager, and O Mangasarian. Nonlinear programming. athena scientific belmont. *Massachusets, USA*, 1999.

[18] Rajendra Bhatia and Chandler Davis. A cauchy-schwarz inequality for operators with applications. *Linear algebra and its applications*, 223:119–129, 1995.

[19] Karl Heinz Borgwardt. *The Simplex Method: a probabilistic analysis*, volume 1. Springer Science & Business Media, 2012.

[20] Albrecht Böttcher and David Wenzel. The frobenius norm and the commutator. *Linear algebra and its applications*, 429(8-9):1864–1885, 2008.

[21] Nicolas Bourbaki. *Topological vector spaces: Chapters 1–5*. Springer Science & Business Media, 2013.

[22] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[23] Michael W Carter and Camille C Price. *Operations research: a practical introduction*. Crc Press, 2017.

[24] Zhongtao Cheng, Dong Liu, and Lei Zhang. Random two-frame phase-shifting interferometry via minimization of coefficient of variation. *Applied Physics Letters*, 115(12):121107, 2019.

[25] M Chiang. Ele539a: Optimization of communication systems lecture 2: Convex optimization and lagrange duality, 2007.

[26] Wei-Yu Chiu and Bor-Sen Chen. Mobile location estimation in urban areas using mixed manhattan/euclidean norm and convex optimization. *IEEE Transactions on Wireless Communications*, 8(1):414–423, 2009.

[27] David W Coit and Alice E Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on reliability*, 45(2):254–260, 1996.

[28] Riet De Smet and Kathleen Marchal. Advantages and limitations of current network inference methods. *Nature Reviews Microbiology*, 8(10):717–729, 2010.

[29] Kalyanmoy Deb and Himanshu Gupta. Searching for robust pareto-optimal solutions in multi-objective optimization. In *International conference on evolutionary multi-criterion optimization*, pages 150–164. Springer, 2005.

[30] Urmila M Diwekar. *Introduction to applied optimization*, volume 22. Springer Nature, 2020.

[31] Uday Shanker Dixit, Manjuri Hazarika, and J Paulo Davim. Emergence of production and industrial engineering. In *A Brief History of Mechanical Engineering*, pages 127–146. Springer, 2017.

[32] Mahmoud Mahmoud El-Alem. *A global convergence theory for a class of trust region algorithms for constrained optimization*. PhD thesis, Rice University, 1988.

[33] Isaac Elishakoff and Makoto Ohsaki. *Optimization and anti-optimization of structures under uncertainty*. World Scientific, 2010.

[34] Christodoulos A Floudas, Ioannis G Akrotirianakis, S Caratzoulas, Clifford A Meyer, and Josef Kallrath. Global optimization in the 21st century: Advances and challenges. *Computers & Chemical Engineering*, 29(6):1185–1202, 2005.

[35] Pascal Francq. Neutrality in internet regulation: three regulatory principles. Technical report, 2014.

[36] Lorenzo Garlappi and Georgios Skoulakis. Taylor series approximations to expected utility and optimal portfolio choice. *Mathematics and Financial Economics*, 5(2):121–156, 2011.

[37] Saul I Gass. *Linear programming: methods and applications*. Courier Corporation, 2003.

[38] Franco Giannessi, Antonino Maugeri, and Panos M Pardalos. *Equilibrium problems: nonsmooth optimization and variational inequality models*, volume 58. Springer Science & Business Media, 2006.

[39] Michel X Goemans and Martin Skutella. Cooperative facility location games. *Journal of Algorithms*, 50(2):194–214, 2004.

[40] Alain Goriely and Martine Ben Amar. On the definition and modeling of incremental, cumulative, and continuous growth laws in morphoelasticity. *Biomechanics and modeling in mechanobiology*, 6(5):289–296, 2007.

[41] William E Hart. Python optimization modeling objects (pyomo). In *Operations Research and Cyber-Infrastructure*, pages 3–19. Springer, 2009.

[42] Dorit S Hochbaum and J George Shanthikumar. Convex separable optimization is not much harder than linear optimization. *Journal of the ACM (JACM)*, 37(4):843–862, 1990.

[43] Laurence D Hoffmann, Gerald L Bradley, and Kenneth H Rosen. *Calculus for business, economics, and the social and life sciences*. McGraw-Hill, 1989.

[44] Christi M Jaeger. *Phylogeny of Tortricidae (Lepidoptera): A morphological approach with enhanced whole mount staining techniques*. Mississippi State University, 2017.

[45] K Jamuna and KS Swarup. Multi-objective biogeography based optimization for optimal pmu placement. *Applied Soft Computing*, 12(5):1503–1510, 2012.

[46] John P Jasa, John T Hwang, and Joaquim RRA Martins. Open-source coupled aerostructural optimization using python. *Structural and Multidisciplinary Optimization*, 57(4):1815–1827, 2018.

[47] Mazin Kareem Kadhim, Fadhil Abdalhasan Wahbi, and Ahmed Hasan Alridha. Mathematical optimization modeling for estimating the incidence of clinical diseases. *International Journal of Nonlinear Analysis and Applications*, 13(1):185–195, 2022.

[48] Howard Karloff. *Linear programming*. Springer Science & Business Media, 2008.

[49] Catherine Lewis. Linear programming: theory and applications. *Whitman College Mathematics Department*, 2008.

[50] Jun Liu, Shuiwang Ji, and Jieping Ye. Multi-task feature learning via efficient l2, 1-norm minimization. *arXiv preprint arXiv:1205.2631*, 2012.

[51] Jérôme Malick. The spherical constraint in boolean quadratic programs. *Journal of Global Optimization*, 39(4):609–622, 2007.

[52] Glenn Marion and Daniel Lawson. An introduction to mathematical modelling. *Edinburgh: Bioinformatics and Statistics Scotland, University of Bristol*, 2008.

[53] Lucija Mihalj. *Penalty methods in constrained optimization*. PhD thesis, University of Zagreb. Faculty of Science. Department of Mathematics, 2019.

[54] Andreas C Müller and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

[55] Masakatsu Muramatsu and Tsunehiro Suzuki. A new second-order cone programming relaxation for max-cut problems. *Journal of the Operations Research Society of Japan*, 46(2):164–177, 2003.

[56] Duy-Dinh Nguyen, Kazuto Yukita, Akinori Katou, and Shinji Yoshida. Design optimization of a three-phase dual-active-bridge converter for charging stations. In *2019 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–6. IEEE, 2019.

[57] Michael J Panik. *Linear programming and resource allocation modeling*. John Wiley & Sons, 2018.

[58] Nikolaos Ploskas, Nikolaos Samaras, et al. *Linear Programming Using MATLAB®*, volume 127. Springer, 2017.

[59] RA Poliquin and R Tyrrell Rockafellar. Tilt stability of a local minimum. *SIAM Journal on Optimization*, 8(2):287–299, 1998.

[60] Jean Rabault, Feng Ren, Wei Zhang, Hui Tang, and Hui Xu. Deep reinforcement learning in fluid mechanics: A promising method for both active flow control and shape optimization. *Journal of Hydrodynamics*, 32(2):234–246, 2020.

[61] Singiresu S Rao. *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[62] Michel F Sanner et al. Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61, 1999.

[63] Ankur Sinha, Tharo Soun, and Kalyanmoy Deb. Using karush-kuhn-tucker proximity measure for solving bilevel optimization problems. *Swarm and evolutionary computation*, 44:496–510, 2019.

[64] Wenyu Sun and Ya-Xiang Yuan. *Optimization theory and methods: nonlinear programming*, volume 1. Springer Science & Business Media, 2006.

[65] Porfirio Suñagua and Aurelio Ribeiro Leite Oliveira. A constructive global convergence of the mixed barrier-penalty method for mathematical optimization problems. *Pesquisa Operacional*, 40, 2020.

[66] Michael J Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001.

[67] Guido Van Rossum et al. Python programming language. In *USENIX annual technical conference*, volume 41, pages 1–36, 2007.

[68] Sorin Vlase, Marin Marin, and Andreas Öchsner. *Eigenvalue and Eigenvector Problems in Applied Mechanics*. Springer, 2019.

[69] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4):722–732, 2008.

[70] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[71] FATMA YILDIZ. The diplomatic relationships between the republic of turkey and ethiopia of african intellectual history and cul... 2018.

الملخص

تعد البرمجة الخطية وغير الخطية في الأساس مجالًا فرعيًا للتحسين العددي ، والتحسين الخطي وغير الخطي هو تقنية برمجة رياضية للعثور على القيمة المثلى للوظائف الموضوعية الخطية وغير الخطية. الهدف من هذه الدراسة هو ترميز التصميم وهو أداة مهمة لنمذجة مجالات التحسين العددي. بحثت هذه الدراسة واقترحت برنامج Python للنمذجة العامة للتحسين الخطي وغير الخطي. يوفر مفهومنا الأساس الأساسي لتعلم وتحسين قدرات حل المشكلات الرياضية باستخدام لغة Python. نحن نقدم أساليب وخوارزميات جديدة وأنشأنا تطبيقات مختلفة للنمذجة الرياضية من أجل التحسين. أعطت النتائج العددية والتطبيقية مجال جدوى جيد تم اشتقاق الحل الأمثل منه. طورت هذه الدراسة رموز التصميم بناءً على نماذج رياضية مختلفة والتي تضمنت أسلوب تايلر للتقريب والبرمجة الخطية والبرمجة غير الخطية ومضاعف لاغرانج والخطي المزدوج وغير الخطي ومضاعف لاغرانج وحالة كاروش كون تاكر والعقوبة (الجزاء الداخلي والخارجي) ، قدموا جميعًا خوارزميات فعالة وهي حزم مفتوحة المصدر تصنع برامج رياضية يتم وصفها بلغة برمجة Python

جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة بابل
كلية التربية للعلوم الصرفة
قسم الرياضيات

تصميم تحسين لترميز بايثون

رسالة

مقدمة الى مجلس كلية التربية للعلوم الصرفة / جامعة بابل وهي جزء من متطلبات نيل درجة الماجستير في التربية / الرياضيات

من قبل الطالبة
نادية علي عباس حسين

بإشراف
أ.م. د. احمد صباح احمد الجيلاوي

1444 هـ 2022 م