

**Mathematical Programming and
Operations Research**
Modeling, Algorithms, and Complexity
Examples in Excel and Python
(Work in progress)

Edited by: Robert Hildebrand

Contributors: Robert Hildebrand, Laurent Poirrier, Douglas Bish, Diego Moran

Version Compilation date: October 19, 2023

Preface

This entire book is a working manuscript. The first draft of the book is yet to be completed.

This book is being written and compiled using a number of open source materials. We will strive to properly cite all resources used and give references on where to find these resources. Although the material used in this book comes from a variety of licences, everything used here will be CC-BY-SA 4.0 compatible, and hence, the entire book will fall under a CC-BY-SA 4.0 license.

MAJOR ACKNOWLEDGEMENTS

I would like to acknowledge that substantial parts of this book were borrowed under a CC-BY-SA license. These substantial pieces include:

- "A First Course in Linear Algebra" by Lyryx Learning (based on original text by Ken Kuttler). A majority of their formatting was used along with selected sections that make up the appendix sections on linear algebra. We are extremely grateful to Lyryx for sharing their files with us. They do an amazing job compiling their books and the templates and formatting that we have borrowed here clearly took a lot of work to set up. Thank you for sharing all of this material to make structuring and forming this book much easier! See subsequent page for list of contributors.
- "Foundations of Applied Mathematics" with many contributors. See <https://github.com/Foundations-of-Applied-Mathematics>. Several sections from these notes were used along with some formatting. Some of this content has been edited or rearranged to suit the needs of this book. This content comes with some great references to code and nice formatting to present code within the book. See subsequent page with list of contributors.
- "Linear Inequalities and Linear Programming" by Kevin Cheung. See <https://github.com/dataopt/lineqlpbook>. These notes are posted on GitHub in a ".Rmd" format for nice reading online. This content was converted to \LaTeX using Pandoc. These notes make up a substantial section of the Linear Programming part of this book.
- Linear Programming notes by Douglas Bish. These notes also make up a substantial section of the Linear Programming part of this book.

I would also like to acknowledge Laurent Porrier and Diego Moran for contributing various notes on linear and integer programming.

I would also like to thank Jamie Fravel for helping to edit this book and for contributing chapters, examples, and code.

Contents

Contents	5
1 Resources and Notation	5
2 Mathematical Programming	9
2.1 Linear Programming (LP)	10
2.2 Mixed-Integer Linear Programming (MILP)	11
2.3 Non-Linear Programming (NLP)	13
2.3.1 Convex Programming	13
2.3.2 Non-Convex Non-linear Programming	14
2.3.3 Machine Learning	14
2.4 Mixed-Integer Non-Linear Programming (MINLP)	15
2.4.1 Convex Mixed-Integer Non-Linear Programming	15
2.4.2 Non-Convex Mixed-Integer Non-Linear Programming	15
I Linear Programming	17
II Nonlinear Programming	19
3 Nonlinear Programming (NLP)	21
3.1 Definitions and Theorems	21
3.1.1 Calculus: Derivatives	23
3.1.2 Calculus: Second derivatives	24
3.1.3 Multivariate Calculus Examples	25
3.1.4 Taylor's Theorem	27
3.1.5 Constrained Minimization	29
3.2 Convexity	29
3.2.1 Convex Sets	30
3.3 Convex Functions	32
3.3.1 Proving Convexity - Characterizations	35
3.3.2 Proving Convexity - Composition Tricks	35
3.4 Convex Optimization Examples	36
3.4.1 Unconstrained Optimization: Linear Regression	36
3.5 Machine Learning - SVM	40
3.5.0.1 Distance between hyperplanes	42
3.5.0.2 Approximate SVM	44

3.5.1	SVM with non-linear separators	44
3.5.2	Support Vector Machines	45
3.6	Markowitz Portfolio Optimization	47
3.6.1	Formulation	47
3.6.2	Interpretation	48
3.6.3	Alternative Formulation I: Bounding Risk	48
3.6.4	Alternative Formulation II: Bounding Expected Profits	48
3.6.5	Properties of the Covariance Matrix	48
3.7	Box Design: Minimizing Box Surface Area with a Volume Constraint	50
3.8	Modeling	52
3.8.1	Minimum distance to circles	53
3.9	Machine Learning	56
3.10	Machine learning - Supervised Learning - Classification	56
3.11	Resources	57
4	NLP Algorithms	59
4.1	Algorithms Introduction	59
4.2	1-Dimensional Algorithms	59
4.2.1	Golden Search Method - Derivative Free Algorithm	60
4.2.1.1	Example:	61
4.2.2	Bisection Method - 1st Order Method (using Derivative)	61
4.2.2.1	Minimization Interpretation	61
4.2.2.2	Root finding Interpretation	62
4.2.3	Gradient Descent - 1st Order Method (using Derivative)	62
4.2.4	Newton's Method - 2nd Order Method (using Derivative and Hessian)	63
4.3	Multi-Variate Unconstrained Optimizaition	63
4.3.1	Descent Methods - Unconstrained Optimization - Gradient, Newton	63
4.3.1.1	Choice of α_t	64
4.3.1.2	Choice of d_t using $\nabla f(x)$	64
4.3.2	Stochastic Gradient Descent - The mother of all algorithms.	64
4.3.3	Neural Networks	66
4.3.4	Choice of Δ_k using the hessian $\nabla^2 f(x)$	66
4.4	Constrained Convex Nonlinear Programming	66
4.4.1	Barrier Method	67
5	Computational Issues with NLP	71
5.1	Irrational Solutions	71
5.2	Discrete Solutions	71
5.3	Convex NLP Harder than LP	71
5.4	NLP is harder than IP	72
5.5	Karush-Huhn-Tucker (KKT) Conditions	73
5.6	Gradient Free Algorithms	75
5.6.1	Needler-Mead	75

6	Material to add...	77
6.0.1	Bisection Method and Newton's Method	77
6.1	Gradient Descent	77
7	Fairness in Algorithms	79
A	Linear Algebra	81
B	Results to put somewhere	83
C	Contributors	85
C.0.1	Graph Theory	3

Introduction

Letter to instructors

This is an introductory book for students to learn optimization theory, tools, and applications. The two main goals are (1) students are able to apply the of optimization in their future work or research (2) students understand the concepts underlying optimization solver and use this knowledge to use solvers effectively.

This book was based on a sequence of course at Virginia Tech in the Industrial and Systems Engineering Department. The courses are *Deterministic Operations Research I* and *Deterministic Operations Reserach II*. The first course focuses on linear programming, while the second course covers integer programming an nonlinear programming. As such, the content in this book is meant to cover 2 or more full courses in optimization.

The book is designed to be read in a linear fashion. That said, many of the chapters are mostly independent and could be rearranged, removed, or shortened as desited. This is an open source textbook, so use it as you like. You are encouraged to share adaptations and improvements so that this work can evolve over time.

Letter to students

This book is designed to be a resource for students interested in learning what optimizaiton is, how it works, and how you can apply it in your future career. The main focus is being able to apply the techniques of optimization to problems using computer technology while understanding (at least at a high level) what the computer is doing and what you can claim about the output from the computer.

Quite importantly, keep in mind that when someone claims to have *optimized* a problem, we want to know what kind of gaurantees they have about how good their solution is. Far too often, the solution that is provided is suboptimal by 10%, 20%, or even more. This can mean spending excess amounts of money, time, or energy that could have been saved. And when problems are at a large scale, this can easily result in millions of dollars in savings.

For this reason, we will learn the perspective of *mathematical programming* (a.k.a. mathematical optimization). The key to this study is that we provide gaurantees on how good a solution is to a given problem. We will also study how difficult a problem is to solve. This will help us know (a) how long it might take to solve it and (b) how good a of a solution we might expect to be able to find in reasonable amount of time.

We will later study *heuristic* methods - these methods typically do not come with gaurantees, but tend to help find quality solutions.

Note: Although there is some computer programming required in this work, this is not a course on programming. Thanks to the fantastic modelling packages available these days, we are able to solve complicated problems with little programming effort. The key skill we will need to learn is *mathematical modeling*: converting words and ideas into numbers and variables in order to communicate problems to a computer so that it can solve a problem for you.

As a main element of this book, we would like to make the process of using code and software as easy as possible. Attached to most examples in the book, there will be links to code that implements and solves the problem using several different tools from Excel and Python. These examples can should make it easy to solve a similar problem with different data, or more generally, can serve as a basis for solving related problems with similar structure.

How to use this book

Skim ahead. We recommend that before you come across a topic in lecture, that you skim the relevant sections ahead of time to get a broad overview of what is to come. This may take only a fraction of the time that it may take for you to read it.

Read the expected outcomes. At the beginning of each section, there will be a list of expected outcomes from that section. Review these outcomes before reading the section to help guide you through what is most relevant for you to take away from the material. This will also provide a brief look into what is to follow in that section.

Read the text. Read carefully the text to understand the problems and techniques. We will try to provide a plethora of examples, therefore, depending on your understanding of a topic, you many need to go carefully over all of the examples.

Explore the resources. Lastly, we recognize that there are many alternative methods of learning given the massive amounts of information and resources online. Thus, at the end of each section, there will be a number of superb resources that available on the internet in different formats. There are other free textbooks, informational websites, and also number of fantastic videos posted to youtube. We encourage to explore the resources to get another perspective on the material or to hear/read it taught from a differnt point of view or in presentation style.

Outline of this book

This book is divided in to 4 Parts:

Part I Linear Programming,

Part II Integer Programming,

Part III Discrete Algorithms,

Part IV Nonlinear Programming.

There are also a number of chapters of background material in the Appendix.

The content of this book is designed to encompass 2-3 full semester courses in an industrial engineering department.

Work in progress

This book is still a work in progress, so please feel free to send feedback, questions, comments, and edits to Robert Hildebrand at open.optimization@gmail.com.

1. Resources and Notation

Here are a list of resources that may be useful as alternative references or additional references.

FREE NOTES AND TEXTBOOKS

- Mathematical Programming with Julia by Richard Lusby & Thomas Stidsen
- Linear Programming by K.J. Mtetwa, David
- A first course in optimization by Jon Lee
- Introduction to Optimizaiton Notes by Komei Fukuda
- Convex Optimization by Bord and Vandenberghe
- LP notes of Michel Goemans from MIT
- Understanding and Using Linear Programming - Matousek and Gärtner [Downloadable from Springer with University account]
- Operations Research Problems Statements and Solutions - Raúl PolerJosefa Mula Manuel Díaz-Madroñero [Downloadable from Springer with University account]

NOTES, BOOKS, AND VIDEOS BY VARIOUS SOLVER GROUPS

- AIMMS Optimization Modeling
- Optimization Modeling with LINGO by Linus Schrage
- The AMPL Book
- Microsoft Excel 2019 Data Analysis and Business Modeling, Sixth Edition, by Wayne Winston - Available to read for free as an e-book through Virginia Tech library at Orielly.com.
- Lesson files for the Winston Book
- Video instructions for solver and an example workbook
- youtube-OR-course

GUROBI LINKS

- Go to <https://github.com/Gurobi> and download the example files.
- Essential ingredients
- Gurobi Linear Programming tutorial
- Gurobi tutorial MILP
- GUROBI - Python 1 - Modeling with GUROBI in Python
- GUROBI - Python II: Advanced Algebraic Modeling with Python and Gurobi
- GUROBI - Python III: Optimization and Heuristics
- Webinar Materials
- GUROBI Tutorials

HOW TO PROVE THINGS

- Hammack - Book of Proof

STATISTICS

- Open Stax - Introductory Statistics

LINEAR ALGEBRA

- Beezer - A first course in linear algebra
- Selinger - Linear Algebra
- Cherney, Denton, Thomas, Waldron - Linear Algebra

REAL ANALYSIS

- Mathematical Analysis I by Elias Zakon

DISCRETE MATHEMATICS, GRAPHS, ALGORITHMS, AND COMBINATORICS

- Levin - Discrete Mathematics - An Open Introduction, 3rd edition
- Github - Discrete Mathematics: an Open Introduction CC BY SA
- Keller, Trotter - Applied Combinatorics (CC-BY-SA 4.0)
- Keller - Github - Applied Combinatorics

PROGRAMMING WITH PYTHON

- A Byte of Python
- Github - Byte of Python (CC-BY-SA)

Also, go to <https://github.com/open-optimization/open-optimization-or-examples> to look at more examples.

Notation

- $\mathbf{1}$ - a vector of all ones (the size of the vector depends on context)
- \forall - for all
- \exists - there exists
- \in - in
- \therefore - therefore
- \Rightarrow - implies
- s.t. - such that (or sometimes "subject to".... from context?)
- $\{0, 1\}$ - the set of numbers 0 and 1
- \mathbb{Z} - the set of integers (e.g. $1, 2, 3, -1, -2, -3, \dots$)
- \mathbb{Q} - the set of rational numbers (numbers that can be written as p/q for $p, q \in \mathbb{Z}$ (e.g. $1, 1/6, 27/2$)
- \mathbb{R} - the set of all real numbers (e.g. $1, 1.5, \pi, e, -11/5$)
- \setminus - setminus, (e.g. $\{0, 1, 2, 3\} \setminus \{0, 3\} = \{1, 2\}$)
- \cup - union (e.g. $\{1, 2\} \cup \{3, 5\} = \{1, 2, 3, 5\}$)
- \cap - intersection (e.g. $\{1, 2, 3, 4\} \cap \{3, 4, 5, 6\} = \{3, 4\}$)
- $\{0, 1\}^4$ - the set of 4 dimensional vectors taking values 0 or 1, (e.g. $[0, 0, 1, 0]$ or $[1, 1, 1, 1]$)
- \mathbb{Z}^4 - the set of 4 dimensional vectors taking integer values (e.g., $[1, -5, 17, 3]$ or $[6, 2, -3, -11]$)

- \mathbb{Q}^4 - the set of 4 dimensional vectors taking rational values (e.g. $[1.5, 3.4, -2.4, 2]$)
- \mathbb{R}^4 - the set of 4 dimensional vectors taking real values (e.g. $[3, \pi, -e, \sqrt{2}]$)
- $\sum_{i=1}^4 i = 1 + 2 + 3 + 4$
- $\sum_{i=1}^4 i^2 = 1^2 + 2^2 + 3^2 + 4^2$
- $\sum_{i=1}^4 x_i = x_1 + x_2 + x_3 + x_4$
- \square - this is a typical Q.E.D. symbol that you put at the end of a proof meaning "I proved it."
- For $x, y \in \mathbb{R}^3$, the following are equivalent (note, in other contexts, these notations can mean different things)
 - $x^\top y$ *matrix multiplication*
 - $x \cdot y$ *dot product*
 - $\langle x, y \rangle$ *inner product*

and evaluate to $\sum_{i=1}^3 x_i y_i = x_1 y_1 + x_2 y_2 + x_3 y_3$.

A sample sentence:

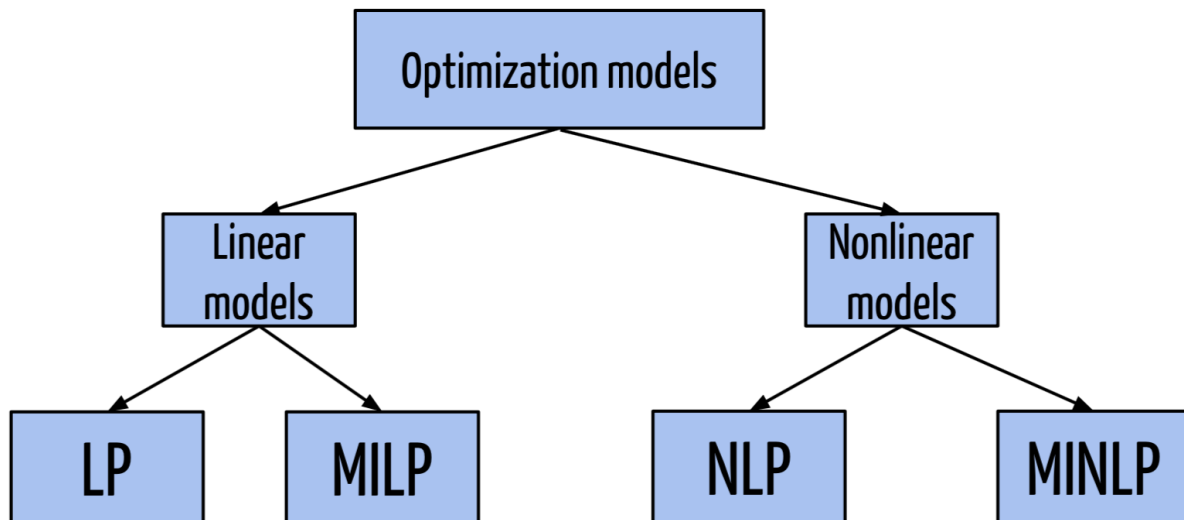
$$\forall x \in \mathbb{Q}^n \exists y \in \mathbb{Z}^n \setminus \{0\}^n \text{ s.t. } x^\top y \in \{0, 1\}$$

"For all non-zero rational vectors x in n -dimensions, there exists a non-zero n -dimensional integer vector y such that the dot product of x with y evaluates to either 0 or 1."

2. Mathematical Programming

Outcomes

We will state main general problem classes to be associated with in these notes. These are Linear Programming (LP), Mixed-Integer Linear Programming (MILP), Non-Linear Programming (NLP), and Mixed-Integer Non-Linear Programming (MINLP).



© problem-class-diagram¹

Figure 2.1: problem-class-diagram

Along with each problem class, we will associate a complexity class for the general version of the problem. See ?? for a discussion of complexity classes. Although we will often state that input data for a problem comes from \mathbb{R} , when we discuss complexity of such a problem, we actually mean that the data is rational, i.e., from \mathbb{Q} , and is given in binary encoding.

¹problem-class-diagram, from problem-class-diagram. problem-class-diagram, problem-class-diagram.

2.1 Linear Programming (LP)

Some linear programming background, theory, and examples will be provided in ??.

Linear Programming (LP):

Polynomial time (P)

Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{2.1}$$

Linear programming can come in several forms, whether we are maximizing or minimizing, or if the constraints are \leq , $=$ or \geq . One form commonly used is *Standard Form* given as

Linear Programming (LP) Standard Form:

Polynomial time (P)

Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *linear programming* problem in *standard form* is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.2}$$

Figure 2.2

Exercise 2.1:

Start with a problem in form given as (2.1) and convert it to standard form (2.2) by adding at most m many new variables and by enlarging the constraint matrix A by at most m new columns.

²wiki/File/linear-programming.png, from wiki/File/linear-programming.png. wiki/File/linear-programming.png, wiki/File/linear-programming.png.

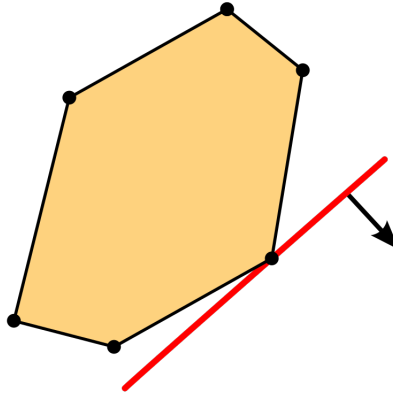
© wiki/File/linear-programming.png²

Figure 2.2: Linear programming constraints and objective.

2.2 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections ??, ??, ??, and ??. Recall that the notation \mathbb{Z} means the set of integers and the set \mathbb{R} means the set of real numbers. The first problem of interest here is a *binary integer program* (BIP) where all n variables are binary (either 0 or 1).

Binary Integer programming (BIP):

NP-Complete

Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned} \tag{2.1}$$

A slightly more general class is the class of *Integer Linear Programs* (ILP). Often this is referred to as *Integer Program* (IP), although this term could leave open the possibility of non-linear parts.

Figure 2.3

Integer Linear Programming (ILP):

NP-Complete

³wiki/File/integer-programming.png, from wiki/File/integer-programming.png. wiki/File/integer-programming.png, wiki/File/integer-programming.png.

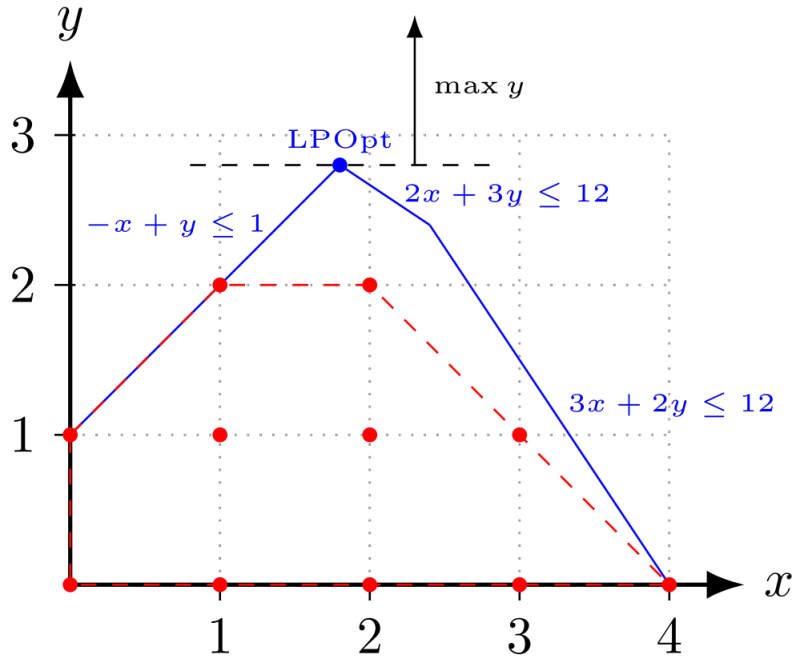
© wiki/File/integer-programming.png³

Figure 2.3: Comparing the LP relaxation to the IP solutions.

Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *integer linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned} \tag{2.2}$$

An even more general class is *Mixed-Integer Linear Programming (MILP)*. This is where we have n integer variables $x_1, \dots, x_n \in \mathbb{Z}$ and d continuous variables $x_{n+1}, \dots, x_{n+d} \in \mathbb{R}$. Succinctly, we can write this as $x \in \mathbb{Z}^n \times \mathbb{R}^d$, where \times stands for the *cross-product* between two spaces.

Below, the matrix A now has $n + d$ columns, that is, $A \in \mathbb{R}^{m \times (n+d)}$. Also note that we have not explicitly enforced non-negativity on the variables. If there are non-negativity restrictions, this can be assumed to be a part of the inequality description $Ax \leq b$.

Mixed-Integer Linear Programming (MILP):

NP-Complete

Given a matrix $A \in \mathbb{R}^{m \times (n+d)}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^{n+d}$, the *mixed-integer linear program*-

minimization problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \times \mathbb{R}^d \end{aligned} \tag{2.3}$$

2.3 Non-Linear Programming (NLP)

NLP:

NP-Hard

Given a function $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$ and other functions $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, the *nonlinear programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{2.1}$$

Nonlinear programming can be separated into convex programming and non-convex programming. These two are very different beasts and it is important to distinguish between the two.

2.3.1. Convex Programming

Here the functions are all **convex**!

Convex Programming:

Polynomial time (P) (typically)

Given a convex function $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$ and convex functions $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{2.2}$$

Observe that convex programming is a generalization of linear programming. This can be seen by letting $f(x) = c^\top x$ and $f_i(x) = A_i x - b_i$.

2.3.2. Non-Convex Non-linear Programming

When the function f or functions f_i are non-convex, this becomes a non-convex nonlinear programming problem. There are a few complexity issues with this.

IP AS NLP As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1-x) = 0$$

has exactly two solutions: $x = 0, x = 1$. Thus, quadratic constraints can be used to model binary constraints.

Binary Integer programming (BIP) as a NLP:

NP-Hard

Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & \cancel{x \in \{0,1\}^n} \\ & x_i(1-x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{2.3}$$

2.3.3. Machine Learning

Machine learning problems are often cast as continuous optimization problems, which involve adjusting parameters to minimize or maximize a particular objective. Frequently they are convex optimization problems, but many turn out to be nonconvex. Here are two examples of how these problems arise at a glance. We will see examples in greater detail later in the book.

Loss Function Minimization

In supervised learning, this objective is typically a loss function L that quantifies the discrepancy between the predictions of a model and the true data labels. The aim is to adjust the parameters θ of the model to minimize this loss. Mathematically, this can be represented as:

$$\min_{\theta} L(\theta) = \min_{\theta} \frac{1}{N} \sum_{i=1}^N l(y_i, f(x_i; \theta)) \quad (2.4)$$

where N is the number of data points, l is a per-data-point loss (e.g., squared error for regression or cross-entropy for classification), y_i is the true label for the i -th data point, and $f(x_i; \theta)$ is the model's prediction for the i -th data point with parameters θ .

Clustering Formulation

Clustering, on the other hand, seeks to group or partition data points such that data points in the same group are more similar to each other than those in other groups. One popular method is the k-means clustering algorithm. The objective of k-means is to partition the data into k clusters by minimizing the within-cluster sum of squares (WCSS). The mathematical formulation can be given as:

$$\min_{\mathbf{c}_1, \dots, \mathbf{c}_k} \sum_{j=1}^k \sum_{x_i \in C_j} \|x_i - \mathbf{c}_j\|^2 \quad (2.5)$$

where C_j represents the j -th cluster and \mathbf{c}_j is the centroid of that cluster.

This encapsulation presents a glimpse into how ML problems are framed mathematically. In practice, numerous algorithms, constraints, and regularizations add complexity to these basic formulations.

2.4 Mixed-Integer Non-Linear Programming (MINLP)

2.4.1. Convex Mixed-Integer Non-Linear Programming

2.4.2. Non-Convex Mixed-Integer Non-Linear Programming

Part I

Linear Programming

Part II

Nonlinear Programming

3. Nonlinear Programming (NLP)

In our last chapter, we covered integer programming, focusing on problems with discrete variables. This chapter will explore *nonlinear (continuous) optimization*, where we deal with continuous variables and now the objective and/or constraints might be nonlinear.

The foundation of continuous optimization is multi-variable calculus. By understanding its basic concepts, we can analyze whether problems are convex or non-convex. This understanding helps in creating efficient algorithms to tackle them.

Our approach in this chapter will be structured. We'll start with key definitions and then move to applications, especially in machine learning. It's worth noting that the problems in this area can vary widely in scale. Some might have millions of variables, while others only a few dozen. This means the choice of algorithm and approach can vary significantly based on the problem.

Our goals for this chapter are:

- Seeing practical applications of continuous optimization.
- Understanding the structure of problems.
- Recognizing the strengths and limits of different algorithms.
- Implementing solutions using Python tools like `scipy.optimize` and `scikitlearn`.

In the next chapter, we'll combine the continuous optimization techniques from this chapter with the integer programming methods from the last chapter. This combination will help us tackle more advanced problems and techniques.

We will provide a number of proofs throughout this chapter for completeness, although, memorizing these proofs may not be essential to fulfilling the outcomes of this chapter.

3.1 Definitions and Theorems

We consider a general mathematical formulation for the optimization problem. We will begin with the context of unconstrained optimization and then discuss later how some of the theorems can be altered to manage constrained versions of the problem.

Definition 3.1: Unconstrained Minimization

Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The problem is given by:

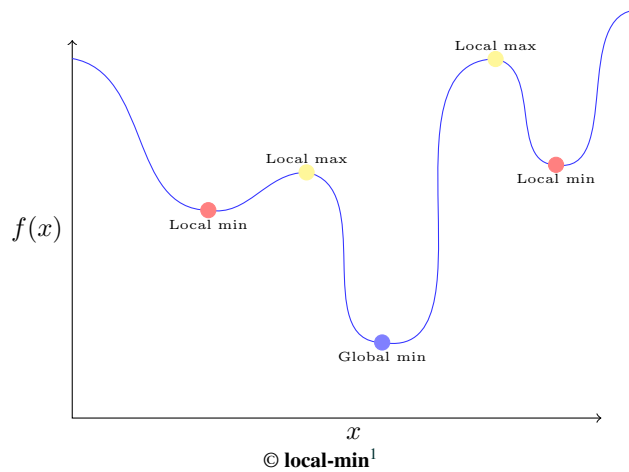
$$\min_{x \in \mathbb{R}^n} f(x)$$

Note that we may change any maximization problem to a minimization problem by simply minimizing the function $g(x) := -f(x)$.

Definition 3.2: Global and local optima

For Problem 3.1, a vector x^* is a

- global minimizer if $f(x^*) \leq f(x)$ for all $x \in \mathbb{R}^n$.
- local minimizer if $f(x^*) \leq f(x)$ for all x satisfying $\|x - x^*\| \leq \varepsilon$ for some $\varepsilon > 0$.
- strict local minimizer if $f(x^*) < f(x)$ for all $x \neq x^*$ satisfying $\|x - x^*\| \leq \varepsilon$ for some $\varepsilon > 0$.

**Theorem 3.3: Attaining a minimum**

Let S be a nonempty set that is closed and bounded. Suppose that $f : S \rightarrow \mathbb{R}$ is continuous. Then the problem $\min\{f(x) : x \in S\}$ attains its minimum.

Proof. By the Extreme Value Theorem, if a function f is continuous on a closed and bounded interval, then f attains both its maximum and minimum values, i.e., there exist points $c, d \in S$ such that:

$$f(c) \leq f(x) \leq f(d) \quad \text{for all } x \in S$$


Given that S is nonempty, closed, and bounded, and f is continuous on S , it follows from the theorem that f attains its minimum on S .

¹local-min, from local-min. local-min, local-min.

Thus, there exists an $x^* \in S$ such that:

$$f(x^*) \leq f(x) \quad \text{for all } x \in S$$

This means that the problem $\min\{f(x) : x \in S\}$ attains its minimum at x^* .

This completes the proof. 

3.1.1. Calculus: Derivatives

We generalize the notion a derivative from single variable calculus to multi-variable calculus.

Definition 3.4: Partial Derivative

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function defined on an open set containing a point $\mathbf{a} = (a_1, a_2, \dots, a_n)$. The partial derivative of f with respect to its i -th variable, x_i , at the point \mathbf{a} is defined as:

$$\frac{\partial f}{\partial x_i}(\mathbf{a}) = \lim_{h \rightarrow 0} \frac{f(a_1, \dots, a_{i-1}, a_i + h, a_{i+1}, \dots, a_n) - f(a_1, \dots, a_n)}{h}$$

provided the limit exists. It represents the rate of change of the function with respect to the variable x_i while keeping all other variables constant.

We can then combine these into a vector called the *Gradient*.

Definition 3.5: Gradient

Given a scalar-valued differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the gradient of f at a point $x \in \mathbb{R}^n$ is a vector of its first order partial derivatives. It is denoted by $\nabla f(x)$ and is given by:

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

The gradient points in the direction of the steepest ascent of the function at the point x .

Definition 3.6: Critical Point

A critical point is a point \bar{x} where $\nabla f(\bar{x}) = 0$.

Theorem 3.7

Suppose that $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable. If $\min\{f(x) : x \in \mathbb{R}^n\}$ has an optimizer x^* , then x^* is a critical point of f (i.e., $\nabla f(x^*) = 0$).

Proof. Suppose x^* is an optimizer of f but $\nabla f(x^*) \neq 0$. Without loss of generality, let's assume that the first component of $\nabla f(x^*)$ is positive (the argument for a negative component is similar).

Then, the directional derivative of f at x^* in the direction of the standard basis vector e_1 (which is the vector with a 1 in the first position and 0 elsewhere) is positive. Formally, this is:

$$D_{e_1}f(x^*) = \nabla f(x^*) \cdot e_1 > 0$$

This means that for sufficiently small $t > 0$, we have:

$$f(x^* + te_1) < f(x^*)$$

This contradicts the assumption that x^* is an optimizer of f , since we've found a point $x^* + te_1$ where f takes a smaller value than at x^* .

Therefore, our initial assumption that $\nabla f(x^*) \neq 0$ must be incorrect, and we conclude that $\nabla f(x^*) = 0$.

This completes the proof. 

3.1.2. Calculus: Second derivatives

We can also generalize multi-variate valuva

Definition 3.8: Hessian

For a scalar-valued function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ that has continuous second order partial derivatives, the Hessian matrix of f at a point $x \in \mathbb{R}^n$ is a square matrix of its second order partial derivatives. It is denoted by $\nabla^2 f(x)$ or $H_f(x)$ and is defined as:

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

The Hessian provides information about the local curvature of the function at the point x .

Definition 3.9: Positive Semidefinite

A square matrix A is said to be positive semidefinite if, for all non-zero vectors $x \in \mathbb{R}^n$, the quadratic form $x^\top Ax$ is non-negative, i.e.,

$$x^\top Ax \geq 0$$

for all $x \in \mathbb{R}^n$. Equivalently, all the eigenvalues of A are non-negative.

3.1.3. Multivariate Calculus Examples**Example 3.10: 2D Function: Gradient and Hessian**

Consider the function:

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

1. Gradient of f :

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

2. Hessian of f :

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

The eigenvalues of this Hessian are 4 and 0, both non-negative. Therefore, the Hessian is positive semidefinite.

Example 3.11: Non-Quadratic 2D Function: Gradient and Hessian

Consider the function:

$$h(x,y) = x^3y + xy^2$$

1. Gradient of h :

$$\nabla h(x,y) = \begin{bmatrix} \frac{\partial h}{\partial x} \\ \frac{\partial h}{\partial y} \end{bmatrix} = \begin{bmatrix} 3x^2y + y^2 \\ x^3 + 2xy \end{bmatrix}$$

2. Hessian of h :

$$H_h(x,y) = \begin{bmatrix} \frac{\partial^2 h}{\partial x^2} & \frac{\partial^2 h}{\partial x \partial y} \\ \frac{\partial^2 h}{\partial y \partial x} & \frac{\partial^2 h}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 6xy + 2y & 3x^2 + 2x \\ 3x^2 + 2x & 2x \end{bmatrix}$$

Note that the entries of the Hessian are not constant and vary depending on the point (x,y) . The positive semidefiniteness of the Hessian will vary depending on the values of x and y , and cannot be determined globally for this function without further analysis.

Example 3.12: 3D Function: Gradient and Hessian

Consider the function:

$$g(x, y, z) = x^2 + y^2 + z^2 + 2xz$$

1. Gradient of g :

$$\nabla g(x, y, z) = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \\ \frac{\partial g}{\partial z} \end{bmatrix} = \begin{bmatrix} 2x + 2z \\ 2y \\ 2z + 2x \end{bmatrix}$$

2. Hessian of g :

$$H_g(x, y, z) = \begin{bmatrix} \frac{\partial^2 g}{\partial x^2} & \frac{\partial^2 g}{\partial x \partial y} & \frac{\partial^2 g}{\partial x \partial z} \\ \frac{\partial^2 g}{\partial y \partial x} & \frac{\partial^2 g}{\partial y^2} & \frac{\partial^2 g}{\partial y \partial z} \\ \frac{\partial^2 g}{\partial z \partial x} & \frac{\partial^2 g}{\partial z \partial y} & \frac{\partial^2 g}{\partial z^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 2 \end{bmatrix}$$

The eigenvalues of this Hessian are approximately 4.73, 2, and -0.73 . Since one eigenvalue is negative, the Hessian is not positive semidefinite.

Lemma 3.13: Gradient of Quadratic in Matrix Notation

Given a multivariate quadratic function in the form:

$$f(x) = x^\top Qx$$

where $x = [x_1, x_2, \dots, x_n]^\top$ and Q is an $n \times n$ symmetric matrix. The gradient is given by

$$\nabla f(x) = 2Qx$$

Proof. We can expand the function in terms of individual elements as:

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n Q_{ij} x_i x_j$$

Now, let's differentiate with respect to x_k , for some $k \in \{1, 2, \dots, n\}$:

$$\frac{\partial f}{\partial x_k} = \frac{\partial}{\partial x_k} \left(\sum_{i=1}^n \sum_{j=1}^n Q_{ij} x_i x_j \right)$$

For terms where $i = k$:

$$\frac{\partial}{\partial x_k} (Q_{kj} x_k x_j) = Q_{kj} x_j$$

And for terms where $j = k$:

$$\frac{\partial}{\partial x_k} (Q_{ik} x_i x_k) = Q_{ik} x_i$$

Given that Q is symmetric, we have $Q_{kj} = Q_{jk}$. So, the sum of the two derivatives above for any given k is:

$$Q_{kj}x_j + Q_{jk}x_i = 2Q_{kj}x_j$$

Thus, the k -th component of the gradient vector is:

$$\frac{\partial f}{\partial x_k} = 2 \sum_{j=1}^n Q_{kj}x_j$$

In matrix notation, the gradient is:

$$\nabla f(x) = 2Qx$$



3.1.4. Taylor's Theorem

Theorem 3.14: Taylor's Theorem: Univariate Case

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a function such that f and its first n derivatives are continuous on an interval containing c and x , and its $(n+1)$ -th derivative exists on this interval. Then, for each x in the interval, there exists a point z between c and x such that:

$$f(x) = f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \cdots + \frac{f^{(n)}(c)}{n!}(x-c)^n + R_n(x)$$

where the remainder term is given by:

$$R_n(x) = \frac{f^{(n+1)}(z)}{(n+1)!}(x-c)^{n+1}$$

This can be generalized the the multivariate case. We present here just the version up to quadratic terms since this tends to be the most used version of the result.

Theorem 3.15: Taylor's Theorem: Multivariate Case (Quadratic Terms)

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar-valued function with continuous partial derivatives up to order 3 in a neighborhood of a point \mathbf{a} in \mathbb{R}^n . Then, for a vector \mathbf{h} in this neighborhood, the function can be expressed as:

$$f(\mathbf{a} + \mathbf{h}) = f(\mathbf{a}) + \nabla f(\mathbf{a})^\top \mathbf{h} + \frac{1}{2} \mathbf{h}^\top \nabla^2 f(\mathbf{a}) \mathbf{h} + R(\mathbf{h})$$

where the remainder term is:

$$R(\mathbf{h}) = O(\|\mathbf{h}\|^3)$$

and $R(\mathbf{h})$ depends on the third order derivatives of f , which are evaluated at some point between \mathbf{a} and $\mathbf{a} + \mathbf{h}$.

Lemma 3.16: 2nd Derivative and Critical Points

If $f''(a) > 0$ at a critical point a of a function f , then a is a local minimum of f .

Proof. A point a is a critical point of f if and only if $f'(a) = 0$. Using Taylor's theorem, we can expand f about the point $x = a$:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(c)}{2!}(x-a)^2$$

for some c between a and x .


Given that $f'(a) = 0$, the expansion reduces to:

$$f(x) = f(a) + \frac{f''(c)}{2!}(x-a)^2$$

Now, if $f''(a) > 0$, then by the continuity of the second derivative, there exists an open interval $(a - \delta, a + \delta)$ around a such that for all x in this interval (except possibly at $x = a$), $f''(x) > 0$.

For x in this interval, the term $(x-a)^2$ is always non-negative. Given that $f''(c) > 0$ for c between a and x , we deduce that:

$$f(x) - f(a) = \frac{f''(c)}{2}(x-a)^2 > 0$$

This implies that $f(x) > f(a)$ for x in $(a - \delta, a + \delta)$, except possibly at $x = a$. Therefore, a is a local minimum of f . 

Now let's look at the multivariate version of this.

Lemma 3.17: Hessian and Local Minima

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be a twice continuously differentiable function. If $\nabla f(\mathbf{a}) = 0$ (i.e., \mathbf{a} is a critical point) and the Hessian matrix $\nabla^2 f(\mathbf{a})$ is positive semidefinite at \mathbf{a} , then \mathbf{a} is a local minimum of f .

Proof. At the point \mathbf{a} , since $\nabla f(\mathbf{a}) = 0$, the first order Taylor expansion of f about \mathbf{a} is simply $f(\mathbf{a})$.

The second order Taylor expansion of f about \mathbf{a} is:

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \frac{1}{2}(\mathbf{x} - \mathbf{a})^\top \nabla^2 f(\mathbf{a})(\mathbf{x} - \mathbf{a})$$

Given that the Hessian $\nabla^2 f(\mathbf{a})$ is positive semidefinite, for any vector \mathbf{v} , we have:

$$\mathbf{v}^\top \nabla^2 f(\mathbf{a}) \mathbf{v} \geq 0$$

Choosing $\mathbf{v} = \mathbf{x} - \mathbf{a}$, we get:

$$(\mathbf{x} - \mathbf{a})^\top \nabla^2 f(\mathbf{a}) (\mathbf{x} - \mathbf{a}) \geq 0$$

Thus, for \mathbf{x} near \mathbf{a} :

$$f(\mathbf{x}) \geq f(\mathbf{a})$$

This shows that $f(\mathbf{a})$ is less than or equal to the values of f near \mathbf{a} , and hence \mathbf{a} is a local minimum of f .



3.1.5. Constrained Minimization

Definition 3.18: Constrained Minimization

Given functions $f: \mathbb{R}^n \rightarrow \mathbb{R}$ and $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$ for $i = 1, \dots, m$. The problem is formulated as:

$$\begin{array}{lll} \min_{x \in \mathbb{R}^n} & f(x) & \\ \text{subject to} & f_i(x) \leq 0 & \text{for } i = 1, \dots, m \end{array}$$

3.2 Convexity

A key property that will enable efficient algorithms is *convexity*. This comes in the form of *convex sets* and *convex functions*. Then the constraints to an optimization problem form a convex set and the objective function is a convex function, then we say that it is a convex optimization problem.

We explain these definitions in the next two subsections.

3.2.1. Convex Sets

Definition 3.19: Convex Combination

Given two points x, y , a convex combination is any point z that lies on the line between x and y . Algebraically, a convex combination is any point z that can be represented as $z = \lambda x + (1 - \lambda)y$ for some multiplier $\lambda \in [0, 1]$.

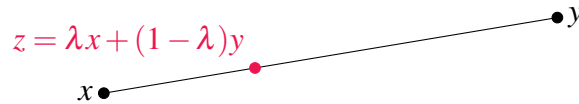
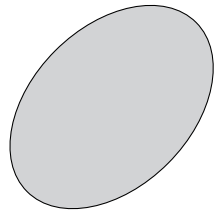


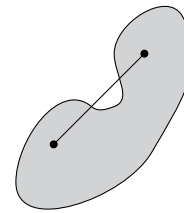
Figure 3.1: A convex combination of the points x and y is given by $z = \lambda x + (1 - \lambda)y$ with any $\lambda \in [0, 1]$. Here we demonstrate this using $\lambda = 2/3$.

Definition 3.20: Convex Set

A set C is convex if it contains all convex combinations of points in C . That is, for any $x, y \in C$, it holds that $\lambda x + (1 - \lambda)y \in C$ for all $\lambda \in [0, 1]$.



(a) Convex Set



(b) Non-convex Set

Examples Convex Sets

1. Hyperplane $H = \{x \in \mathbb{R}^n : a^\top x = b\}$
2. Halfspace $H = \{x \in \mathbb{R}^n : a^\top x \leq b\}$
3. Polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$
4. Ball $S = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i^2 \leq 1\}$
5. Second Order Cone $S = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} : \sum_{i=1}^n x_i^2 \leq t^2\}$

Example 3.21: Proof that a Polyhedron is Convex

Let P be a polyhedron defined by:

$$P = \{x \in \mathbb{R}^n : Ax \leq b\}$$

where A is an $m \times n$ matrix and b is an $m \times 1$ vector.

To prove the convexity of P , we'll use the definition of convexity: For any $x, y \in P$ and any $\lambda \in [0, 1]$, the point $z = \lambda x + (1 - \lambda)y$ must also belong to P .

Let $x, y \in P$. Then, by the definition of P , we have:

$$Ax \leq b \quad \text{and} \quad Ay \leq b$$

We want to demonstrate the inequality $Az \leq b$. We do this in just a few lines:

$$\begin{aligned} Az &= A(\lambda x + (1 - \lambda)y) && \text{Substituting the expression for } z \\ &= \lambda Ax + (1 - \lambda)Ay && \text{Expanding the matrix-vector product} \\ &\leq \lambda b + (1 - \lambda)b && \text{Explanation below} \\ &= b \end{aligned}$$

The inequality above uses the fact that $Ax \leq b$ and $Ay \leq b$, and since λ is in the interval $[0, 1]$, it follows that:

$$\lambda Ax \leq \lambda b \quad \text{and} \quad (1 - \lambda)Ay \leq (1 - \lambda)b$$

The computation shows that $z = \lambda x + (1 - \lambda)y$ satisfies $Az \leq b$ and therefore $z \in P$.

This shows that the polyhedron P is convex.

Lemma 3.22: Intersection of Convex Sets is Convex

Let C_1 and C_2 be convex sets. Then the intersection $C_1 \cap C_2$ is convex. In particular,

$$C_1 \cap C_2 := \{x : x \in C_1 \text{ and } x \in C_2\}.$$

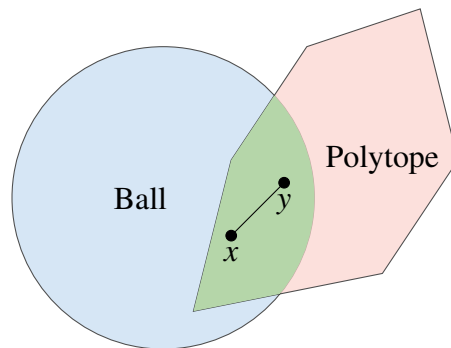


Figure 3.3: The green intersection of the convex sets that are the ball and the polytope is also convex. This can be seen by considering any points $x, y \in \text{Ball} \cap \text{Polytope}$. Since **Ball** is convex, the line segment between x and y is completely contained in **Ball**. And similarly, the line segment is completely contained in **Polytope**. Hence, the line segment is also contained in the intersection. This is how we can reason that the intersection is also convex.

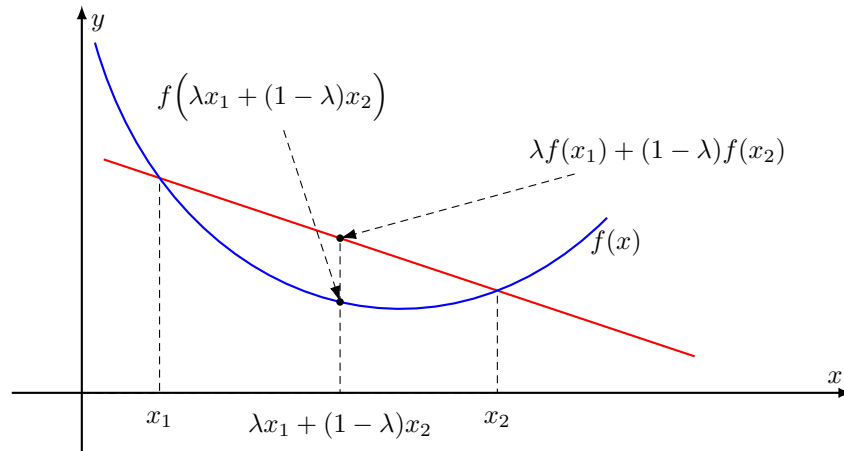
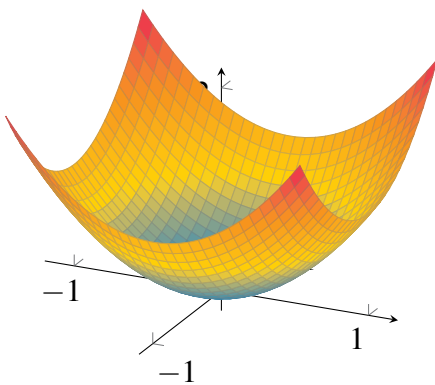
© tikz/convexity-definition.pdf²

Figure 3.5: Illustration explaining the definition of a convex function.

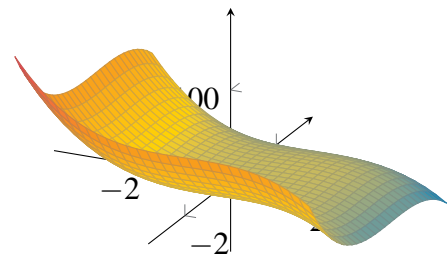
3.3 Convex Functions

Convex functions are "nice" functions that "open up". They represent an extremely important class of functions in optimization and typically can be optimized over efficiently.



(a) Convex Function

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



(b) Non-Convex Function

$$f(x, y) = x^2 + y^2 - 2(x - 0.3)^3 - 2(y - 0.4)^3.$$

Figure 3.4: Comparison of Convex and Non-Convex Functions.

Informally, a function is convex if whenever you draw a line between two points on the function, that line must be above the function.

Formally, we can make this definition using the idea of convex combinations.

Definition 3.23: Convex Functions

A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if for all $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$ we have

$$\lambda f(x) + (1 - \lambda)f(y) \geq f(\lambda x + (1 - \lambda)y). \quad (3.1)$$

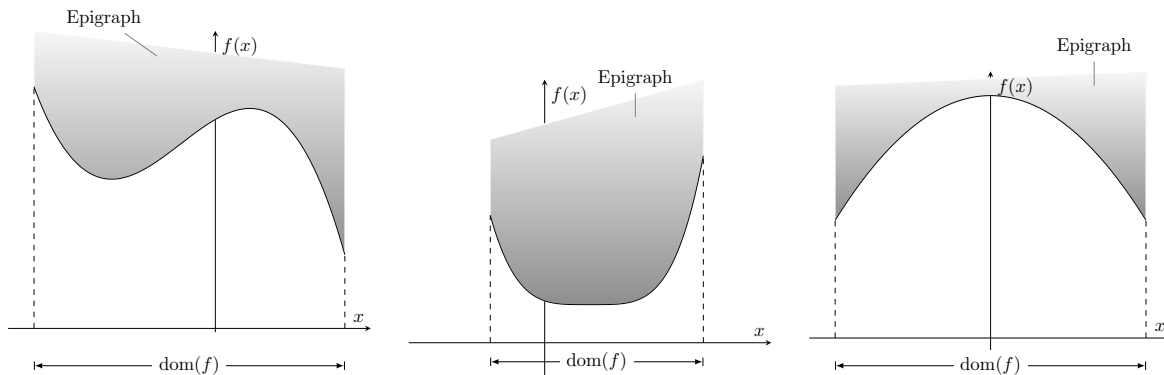
An equivalent definition of convex function are through the epigraph.

Definition 3.24: Epigraph

The epigraph of f is the set $\{(x, y) : y \geq f(x)\}$. This is the set of all points "above" the function.

Theorem 3.25

$f(x)$ is a convex function if and only if the epigraph of f is a convex set.



© epigraph.pdf³

4

Example 3.26: Examples of Convex functions

Some examples are

- $f(x) = ax + b$

²[tikz/convexity-definition.pdf](#), from [tikz/convexity-definition.pdf](#). [tikz/convexity-definition.pdf](#), [tikz/convexity-definition.pdf](#), [tikz/convexity-definition.pdf](#).

²<https://tex.stackexchange.com/questions/394923/how-one-can-draw-a-convex-function>

³[epigraph.pdf](#), from [epigraph.pdf](#). [epigraph.pdf](#), [epigraph.pdf](#).

⁴<https://tex.stackexchange.com/questions/261501/function-epigraph-possibly-using-fillbetween>

- $f(x) = x^2$
- $f(x) = x^4$
- $f(x) = |x|$
- $f(x) = e^x$
- $f(x) = -\sqrt{x}$ on the domain $[0, \infty)$.
- $f(x) = x^3$ on the domain $[0, \infty)$.
- $f(x, y) = \sqrt{x^2 + y^2}$
- $f(x, y) = x^2 + y^2 + x$
- $f(x, y) = e^{x+y}$
- $f(x, y) = e^x + e^y + x^2 + (3x + 4y)^6$

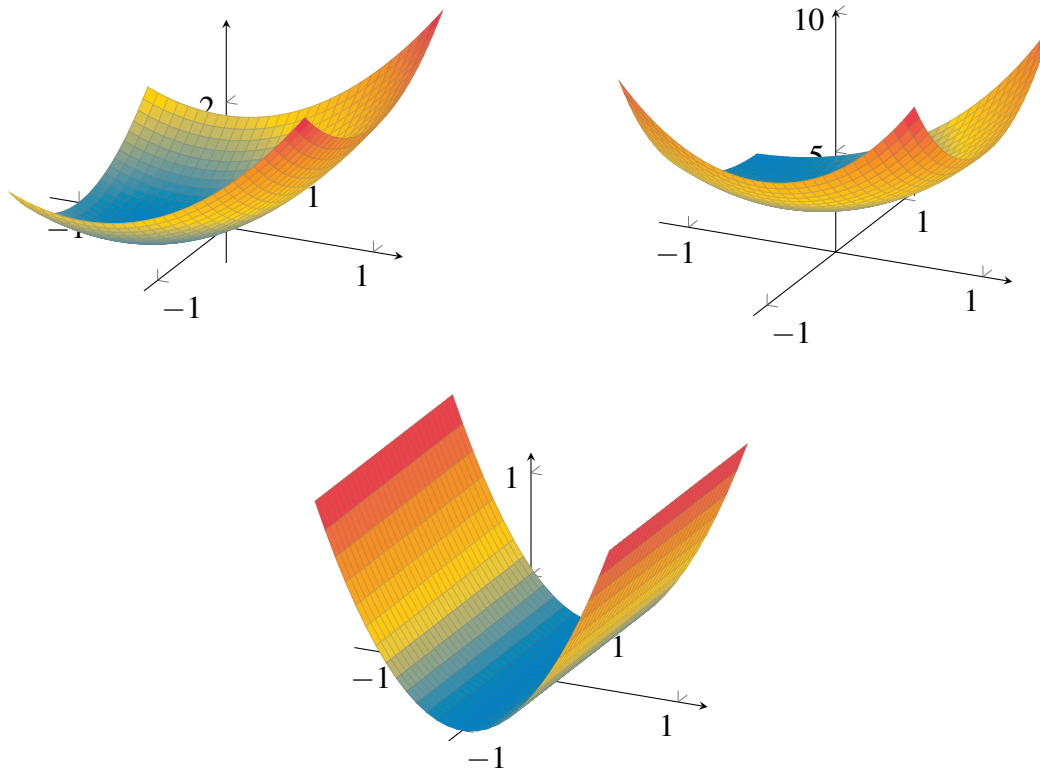


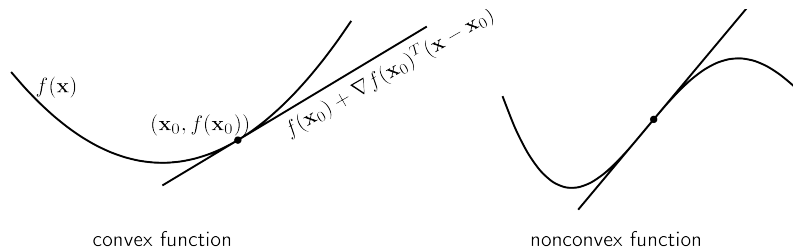
Figure 3.6: Convex Functions $f(x, y) = x^2 + y^2 + x$, $f(x, y) = e^{x+y} + e^{x-y} + e^{-x-y}$, and $f(x, y) = x^2$.

3.3.1. Proving Convexity - Characterizations

Theorem 3.27: Convexity: First order characterization - linear underestimates

Suppose that $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable. Then f is convex if and only if for all $\bar{x} \in \mathbb{R}^n$, then linear tangent is an underestimator to the function, that is,

$$f(\bar{x}) + (x - \bar{x})^\top \nabla f(\bar{x}) \leq f(x).$$



© first-order-convexity⁵
6

Theorem 3.28: Convexity: Second order characterization - positive curvature

We give statements for uni-variate functions and multi-variate functions.

- Suppose $f: \mathbb{R} \rightarrow \mathbb{R}$ is twice differentiable. Then f is convex if and only if $f''(x) \geq 0$ for all $x \in \mathbb{R}$.
- Suppose $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is twice differentiable. Then f is convex if and only if $\nabla^2 f(x) \succcurlyeq 0$ for all $x \in \mathbb{R}^n$.

3.3.2. Proving Convexity - Composition Tricks

Positive Scaling of Convex Function is Convex:

If f is convex and $\alpha > 0$, then αf is convex.

Example: $f(x) = e^x$ is convex. Therefore, $25e^x$ is also convex.

Sum of Convex Functions is Convex:

If f and g are both convex, then $f + g$ is also convex.

⁵<https://machinelearningcoban.com/2017/03/12/convexity/>

Example: $f(x) = e^x, g(x) = x^4$ are convex. Therefore, $e^x + x^4$ is also convex.

Composition with affine function:

If $f(x)$ is convex, then $f(a^\top x + b)$ is also convex.

Example: $f(x) = x^4$ are convex. Therefore, $(3x + 5y + 10z)^4$ is also convex.

Pointwise maximum:

If f_i are convex for $i = 1, \dots, t$, then $f(x) = \max_{i=1, \dots, t} f_i(x)$ is convex.

Example: $f_1(x) = e^{-x}, f_2(x) = e^x$ are convex. Therefore, $f(x) = \max(e^x, e^{-x})$ is also convex.

Other compositions:

Suppose

$$f(x) = h(g(x)).$$

1. If g is convex, h is convex and **non-decreasing**, then f is convex.
2. If g is concave, h is convex and **non-increasing**, then f is convex.

Example 1: $g(x) = x^4$ is convex, $h(x) = e^x$ is convex and non-decreasing. Therefore, $f(x) = e^{x^4}$ is also convex.

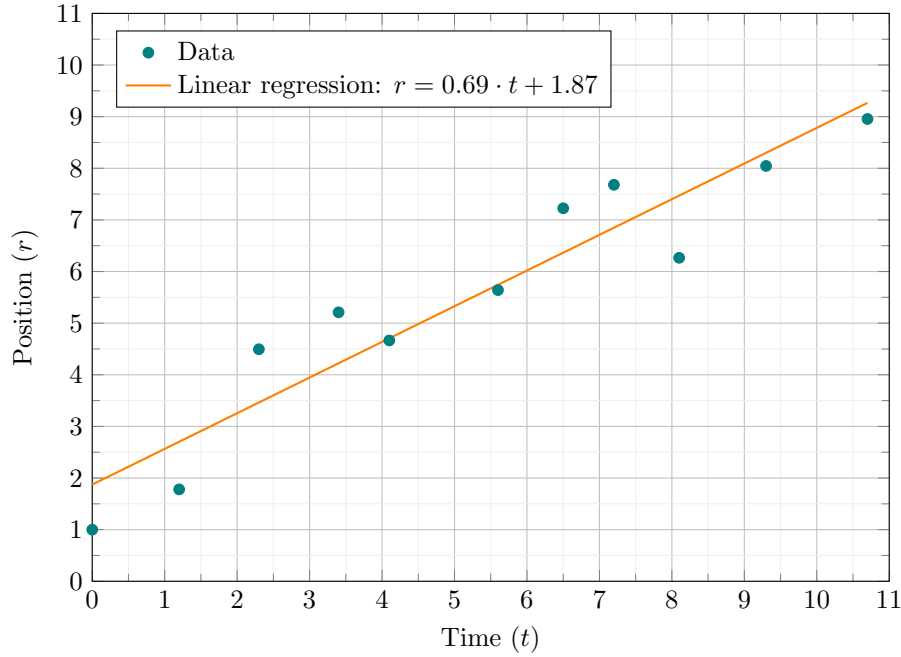
Example 2: $g(x) = \sqrt{x}$ is concave (on $[0, \infty)$), $h(x) = e^{-x}$ is convex and non-increasing. Therefore, $f(x) = e^{-\sqrt{x}}$ is convex on $x \in [0, \infty)$.

3.4 Convex Optimization Examples

3.4.1. Unconstrained Optimization: Linear Regression

Given data points $x^1, \dots, x^N \in \mathbb{R}^d$ and out values $y^i \in \mathbb{R}$, we want to find a linear function $y = \beta \cdot x$ that best approximates $x^i \cdot \beta \approx y^i$. For example, the data could $x = (\text{time})$ and the output could be $y = \text{position}$.

⁷ [tikz/linear-regression.pdf](#), from [tikz/linear-regression.pdf](#). [tikz/linear-regression.pdf](#), [tikz/linear-regression.pdf](#).

© tikz/linear-regression.pdf⁷**Figure 3.7: Line derived through linear regression.**

As is standard, we choose the error (or "loss") from each data point as the squared error. Hence, we can model this as the optimization problem:

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^N (x^i \cdot \beta - y^i)^2. \quad (3.1)$$

Equivalently, we could write this as

$$\min_{\beta} \|X\beta - \mathbf{y}\|_2,$$

where

$$X = \begin{bmatrix} \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^N \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_d^1 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \dots & x_d^N \end{bmatrix}$$

This problem has a nice closed form solution. We will derive this solution, and then in a later section discuss why using this solution might be too slow to compute on large data sets. In particular, the solution comes as a system of linear equations. But when N is really large, we may not have time to solve this system, so an alternative is to use decent methods, discussed later in this chapter.

Theorem 3.29: Linear Regression Solution

The solution to (3.1) is

$$\beta^* = (X^\top X)^{-1} X^\top \mathbf{y}. \quad (3.2)$$

Proof. Given the objective function:

$$L(\beta) = \sum_{i=1}^N (x^i \cdot \beta - y^i)^2$$

This can be written in matrix form as:

$$L(\beta) = (X\beta - \mathbf{y})^\top (X\beta - \mathbf{y})$$

Expanding the above expression:

$$L(\beta) = \beta^\top X^\top X\beta - 2\mathbf{y}^\top X\beta + \mathbf{y}^\top \mathbf{y}$$

To find the value of β that minimizes $L(\beta)$, differentiate with respect to β and set the result to zero:

$$\frac{\partial L(\beta)}{\partial \beta} = 2X^\top X\beta - 2X^\top \mathbf{y} = 0$$


From the above equation:

$$X^\top X\beta = X^\top \mathbf{y}$$

To solve for β :

$$\beta = (X^\top X)^{-1} X^\top \mathbf{y}$$

This completes the proof.

Note: The solution assumes that $X^\top X$ is invertible. If it's not, then the least squares solution may not be unique. 

Logistic Regression

Logistic regression is another important technique in statistical analysis and machine learning that builds off of the concepts of linear regression. As in linear regression, there is a set of predictor variables $\{x_{i,1}, x_{i,2}, \dots, x_{i,n}\}_{i=1}^m$ with corresponding outcome variables $\{y_i\}_{i=1}^m$. In logistic regression, the outcome variables y_i are binary and can be modeled by a *sigmoidal* relationship. The value of the predicted y_i can be thought of as the probability that $y_i = 1$. In mathematical terms,

$$\mathbb{P}(y_i = 1 | x_{i,1}, \dots, x_{i,n}) = p_i,$$

where

$$p_i = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i,1} + \dots + \beta_n x_{i,n}))}.$$

The parameters of the model are the real numbers $\beta_0, \beta_1, \dots, \beta_n$. Note that $p_i \in (0, 1)$ regardless of the values of the predictor variables and parameters.

The probability of observing the outcome variables y_i under this model, assuming they are independent, is given by the *likelihood function* $\mathcal{L} : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$

$$\mathcal{L}(\beta_0, \dots, \beta_n) = \prod_{i=1}^m p_i^{y_i} (1 - p_i)^{1-y_i}.$$

The goal of logistic regression is to find the parameters β_0, \dots, β_k that maximize this likelihood function. Thus, the problem can be written as:

$$\max_{(\beta_0, \dots, \beta_n)} \mathcal{L}(\beta_0, \dots, \beta_n).$$

Maximizing this function is often a numerically unstable calculation. Thus, to make the objective function more suitable, the logarithm of the objective function may be maximized because the logarithmic function is strictly monotone increasing. Taking the log and turning the problem into a minimization problem, the final problem is formulated as:

$$\min_{(\beta_0, \dots, \beta_n)} -\log \mathcal{L}(\beta_0, \dots, \beta_n).$$

A few lines of calculation reveal that this objective function can also be rewritten as

$$\begin{aligned} -\log \mathcal{L}(\beta_0, \dots, \beta_n) &= \sum_{i=1}^m \log(1 + \exp(-(\beta_0 + \beta_1 x_{i,1} + \dots + \beta_n x_{i,n}))) + \\ &\quad \sum_{i=1}^m (1 - y_i)(\beta_0 + \beta_1 x_{i,1} + \dots + \beta_n x_{i,n}). \end{aligned}$$

The values for the parameters $\{\beta_i\}_{i=1}^n$ that we obtain are known as the *maximum likelihood estimate* (MLE). To find the MLE, conjugate gradient can be used to minimize the objective function.

For a one-dimensional binary logistic regression problem, we have predictor data $\{x_i\}_{i=1}^m$ with labels $\{y_i\}_{i=1}^m$ where each $y_i \in \{0, 1\}$. The negative log likelihood then becomes the following.

$$-\log \mathcal{L}(\beta_0, \beta_1) = \sum_{i=1}^m \log(1 + e^{-(\beta_0 + \beta_1 x_i)}) + (1 - y_i)(\beta_0 + \beta_1 x_i) \quad (3.3)$$

Example 3.30: Challenger damage prediction

On January 28, 1986, less than two minutes into the Challenger space shuttle's 10th mission, there was a large explosion that originated from the spacecraft, killing all seven crew members and destroying the shuttle. The investigation that followed concluded that the malfunction was caused by damage to O-rings that are used as seals for parts of the rocket engines. There were 24 space shuttle missions before this disaster, some of which had noted some O-ring damage. Given the data, could this disaster have been predicted?

The file *challenger-data.csv* contains data for 23 missions (during one of the 24 missions, the engine was lost at sea). The first column (\mathbf{x}) contains the ambient temperature, in Fahrenheit, of the shuttle

launch. The second column (\mathbf{y}) contains a binary indicator of the presence of O-ring damage (1 if O-ring damage was present, 0 otherwise).

The logistic regression is implemented in *logistic-regression-challenger.ipynb* using the python package *sklearn*.

On the day on launch, the temperature was 31°F . The predicted probability (according to this model) of O-ring damage on the day the shuttle was launched, is 0.9996.

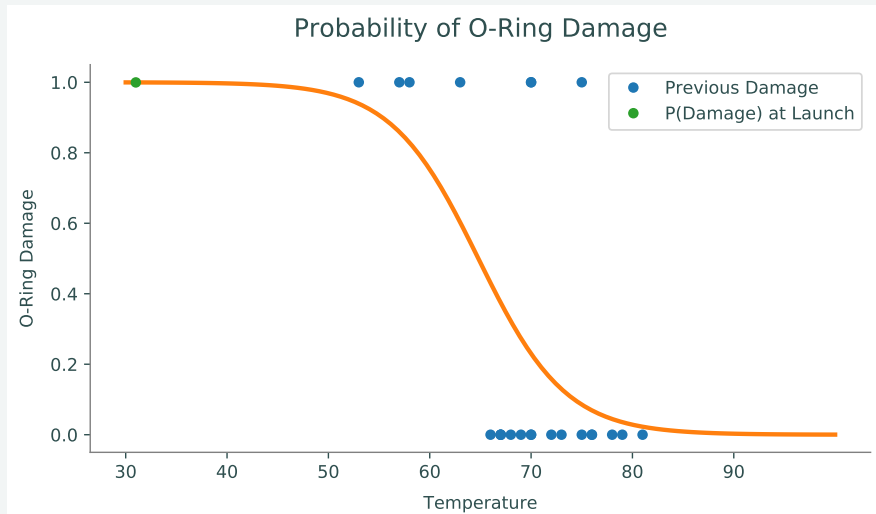


Figure 3.8: The logistic curve models the probability of O-ring damage on the Challenger shuttle. According to this model, given that the temperature was 31° on the day of launch, the shuttle had close to 100% likelihood of O-ring damage. This model had an initial guess of 1. and -1 for β_0 and β_1 respectively.

3.5 Machine Learning - SVM

Support Vector Machine (SVM) is a tool used in machine learning for classifying data points. For instance, if there are red and black data points, how can we find a good line that separates them? The input data that you are given is as follows:

INPUT:

- d -dimensional data points x^1, \dots, x^N
- 1-dimensional labels z^1, \dots, z^N (typically we will use z_i is either 1 or -1)

The output to the problem should be a hyperplane $w^\top x + b = 0$ that separates the two data types (either exact separation or approximate separation).

OUTPUT:

- A d -dimensional vector w
- A 1-dimensional value b

Given this output, we can construct a classification function $f(x)$ as

$$f(x) = \begin{cases} 1 & \text{if } w^\top x + b \geq 0, \\ -1 & \text{if } w^\top x + b < 0. \end{cases} \quad (3.1)$$

Support Vector Machine - Exact Classification:

Given labeled data (x^i, y_i) for $i = 1, \dots, N$, where $x^i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, find a vector $w \in \mathbb{R}^d$ and a number $b \in \mathbb{R}$ such that

$$x^i \cdot w + b > 0 \quad \text{if } y^i = 1 \quad (3.2)$$

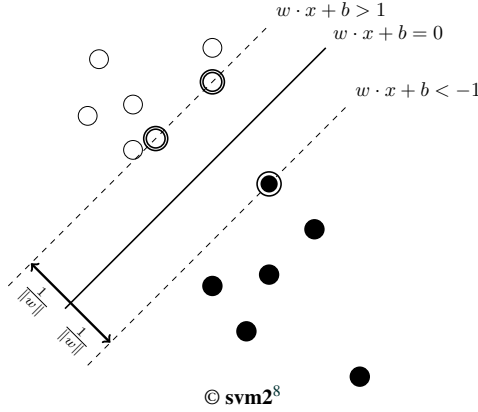
$$x^i \cdot w + b < 0 \quad \text{if } y^i = -1 \quad (3.3)$$

There are three versions to consider:

FEASIBLE SEPARATION If we only want to a line that separates the data points, we can use the following optimization model.

$$\begin{aligned} \min \quad & 0 \\ \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 \quad \text{for all } i = 1, \dots, N \\ & w \in \mathbb{R}^d \\ & b \in \mathbb{R} \end{aligned}$$

There may exist many solutions to this problem. Thus, we are interested in the "best" solution. Such a solution will maximize the separation between the two sets of points. To consider an equal margin on either side, we set the right hand sides to 1 and -1 and then compute the margin from the hyperplane. Notice that it is sufficient to use 1 and -1 on the right hand sides since any scaling can happen in w and b .



We will show that the margin under this model can be computed as $\frac{2}{\|w\|}$ where $\|w\| = \sqrt{w_1^2 + \dots + w_d^2}$. Hence, maximizing the margin is equivalent to minimizing $w_1^2 + \dots + w_d^2$. We arrive at the model

$$\min \sum_{i=1}^d w_i^2 \tag{3.4}$$

$$x^i \cdot w + b \geq 1 \quad \text{if } y^i = 1 \tag{3.5}$$

$$x^i \cdot w + b \leq -1 \quad \text{if } y^i = -1 \tag{3.6}$$

Or even more compactly written as

$$\min \sum_{i=1}^d w_i^2 \tag{3.7}$$

$$y^i(x^i \cdot w + b) \geq 1 \quad \text{for } i = 1, \dots, N \tag{3.8}$$

3.5.0.1. Distance between hyperplanes

We want to know a formula for the distance between two parallel hyperplanes. In particular, we will look at the hyperplane $H_0 = \{x : h^\top x = 0\}$ and $H_1 = \{x : h^\top x = 1\}$.

We choose a point $x^0 = 0 \in H_0$, and then find the point in H_1 that minimizes the distance between these two points.

To find the distance from the origin to the hyperplane H_1 defined as $H_1 = \{x : h^\top x = 1\}$, where h is the normal vector to the hyperplane:

1. Consider a point x on the hyperplane H_1 . This point can be expressed as $x = \lambda h$, where λ is a scalar.
2. We want to find the value of λ such that x satisfies the equation $h^\top x = 1$. Substituting $x = \lambda h$ into this equation:

⁸svm2, from svm2. svm2, svm2.

$$h^\top (\lambda h) = 1$$

3. Simplify the expression:

$$\lambda \|h\|^2 = 1$$

4. To solve for λ , divide both sides by $\|h\|^2$:

$$\lambda = \frac{1}{\|h\|^2}$$

Now, λ represents the scaling factor required to reach the point x on the hyperplane H_1 from the origin.

So, the distance d from the origin to the hyperplane H_1 is given by:

$$d = \lambda = \frac{1}{\|h\|^2}$$

Since we want the distance from the origin to H_1 in terms of the magnitude of the normal vector h , we can write:

$$d = \frac{1}{\|h\|^2}$$

Now, recall that $\|h\|^2 = \langle h, h \rangle$, where $\langle h, h \rangle$ represents the dot product of vector h with itself. If we take the square root of both sides:

$$d = \frac{1}{\sqrt{\langle h, h \rangle}}$$

This is the correct expression for the distance from the origin to the hyperplane H_1 , and it is indeed equal to $\frac{1}{\|h\|}$.

3.5.0.2. Approximate SVM

We can modify the objective function and the constraints to allow for approximate separation. This would be the case when you want to ignore outliers that don't fit well with the data set, or when exact SVM is not possible. This is done by changing the constraints to be

$$z^i(w^\top x^i + b) \geq 1 - \delta_i$$

where $\delta_i \geq 0$ is the error in the constraint for datapoint i . In order to reduce these errors, we add a penalty term in the objective function that encourages these errors to be small. For this, we can pick some number C and write the objective as

$$\min \|w\|_2^2 + C \sum_{i=1}^N \delta_i.$$

This creates the following optimization problem

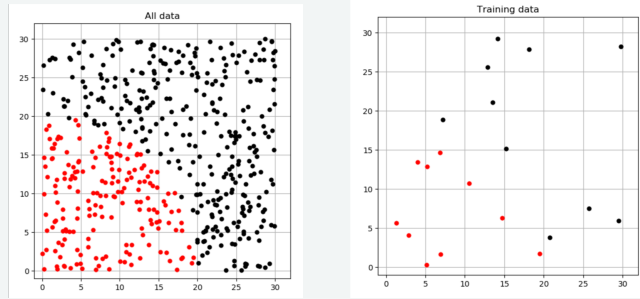
$$\begin{aligned} \min \quad & \|w\|_2^2 + C \sum_{i=1}^N \delta_i \\ \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 - \delta_i && \text{for all } i = 1, \dots, N \\ & w \in \mathbb{R}^d \\ & b \in \mathbb{R} \\ & \delta_i \geq 0 \text{ for all } i = 1, \dots, N \end{aligned}$$

See information about the scikit-learn module for svm here: <https://scikit-learn.org/stable/modules/svm.html>.

3.5.1. SVM with non-linear separators

Suppose for instance you are given data $x^1, \dots, x^N \in \mathbb{R}^2$ (2-dimensional data) and given labels are dependent on the distance from the origin, that is, all data points x with $x_1^2 + x_2^2 > r$ are given a label $+1$ and all data points with $x_1^2 + x_2^2 \leq r$ are given a label -1 . That is, we want to learn the function

$$f(x) = \begin{cases} 1 & \text{if } x_1^2 + x_2^2 > r, \\ -1 & \text{if } x_1^2 + x_2^2 \leq r. \end{cases} \quad (3.9)$$

Example 3.31

Here we have a classification problem where the data cannot be separated by a hyperplane. On the left, we have all of the data given to use. On the right, we have a subset of the data that we could try using for training and then test our learned function on the remaining data. As we saw in class, this amount of data was not sufficient to properly classify most of the data.

We cannot learn this classifier from the data directly using the hyperplane separation with SVM in the last section. But, if we modify the data set, then we can do this.

For each data point x , we transform it into a data point X by adding a third coordinate equal to $x_1^2 + x_2^2$. That is

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow X = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{pmatrix}. \quad (3.10)$$

In this way, we convert the data x^1, \dots, x^N into data X^1, \dots, X^N that lives in a higher-dimensional space. But with this new dataset, we can apply the hyperplane separation technique in the last section to properly classify the data.

This can be done with other nonlinear classification functions.

3.5.2. Support Vector Machines

One-Class SVM for Anomaly Detection: An Example in Credit Card Fraud Detection

The idea behind One-Class SVM is to find the best hyperplane that separates all the data points from the origin (in feature space) and maximizes the distance from this hyperplane to the origin. If a data point falls far from this hyperplane, it can be considered an anomaly.

PROBLEM STATEMENT You are given transactional data from a credit card company. The majority of transactions are legitimate, but a small fraction of them are fraudulent. The goal is to detect the fraudulent transactions.

DATA For each transaction, you have various features:

- Amount of the transaction
- Date and time
- Location
- Previous transaction history of the card
- Merchant information

Given the number of legitimate transactions is much higher than the number of fraudulent ones, this is a highly imbalanced dataset.

Steps for Using One-Class SVM

1. Data Preparation:

- Normalize the data since SVMs are sensitive to the scale of the input.
- If the data dimension is high, consider dimensionality reduction techniques (e.g., PCA) to make computation more feasible.

2. **Training the Model on Normal Data:** Train the One-Class SVM only on the legitimate transactions. The idea is to understand the "shape" of normal data. If a transaction does not fit this shape, it's likely an anomaly.

3. Hyperparameter Tuning:

- **Kernel:** Common choices include linear, polynomial, and radial basis function (RBF). The choice of kernel affects the decision boundary.
- **Nu:** This parameter, typically between 0 and 1, sets an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors.
- **Gamma:** If using the RBF kernel, gamma defines how much influence a single training example has. Low values mean 'far' and high values mean 'close'.

4. **Detection:** For each transaction, use the trained One-Class SVM to predict if it's an anomaly. If the SVM classifies it as not belonging to the trained class, it's flagged as an anomaly.

5. **Evaluation:** Since the dataset is imbalanced, accuracy might not be the best metric. Instead, focus on metrics like precision, recall, the F1 score, or area under the ROC curve.

ADVANTAGES

- Effective in high-dimensional spaces.
- Requires only normal data for training, making it suitable for datasets where anomalies are rare and not well-defined.

DISADVANTAGES

- Can be sensitive to the choice of hyperparameters.
- Might be computationally intensive on large datasets.

One-Class SVM provides a way to detect anomalies by learning the distribution of only the normal class and flagging observations that deviate from this. It can be particularly useful in scenarios where anomalies are rare and hard to define.

3.6 Markowitz Portfolio Optimization

Harry Markowitz introduced Modern Portfolio Theory in 1952, which provides a mathematical framework for assembling a portfolio of assets such that the expected return is maximized for a given level of risk. The fundamental insight is that by combining assets with different expected returns and volatilities, one can achieve diversification, leading to reduced portfolio risk.

3.6.1. Formulation

Given a set of n assets with expected returns $\mathbf{r} = [r_1, r_2, \dots, r_n]^\top$, and a covariance matrix Σ representing the covariances between the returns of the assets, the problem can be formulated as:

$$\begin{aligned} & \underset{\mathbf{w}}{\text{maximize}} && \mathbf{r}^\top \mathbf{w} - \frac{\gamma}{2} \mathbf{w}^\top \Sigma \mathbf{w} \\ & \text{subject to} && \mathbf{1}^\top \mathbf{w} = 1, \end{aligned} \tag{3.1}$$

where:

- $\mathbf{w} = [w_1, w_2, \dots, w_n]^\top$ is the vector of portfolio weights.
- γ is the risk aversion coefficient. A larger γ indicates greater aversion to risk.
- $\mathbf{1}$ is a vector of ones.

3.6.2. Interpretation

The objective function is comprised of two components: the expected portfolio return ($\mathbf{r}^\top \mathbf{w}$) and a penalty for portfolio variance ($\mathbf{w}^\top \Sigma \mathbf{w}$). By adjusting γ , an investor can trade off between risk and return.

The constraint ensures that the sum of the portfolio weights is 1, meaning the portfolio is fully invested.

By solving this optimization problem, one can obtain the weights that maximize the expected return for a given level of risk, forming the efficient frontier.

3.6.3. Alternative Formulation I: Bounding Risk

Given a maximum acceptable risk level ρ , the optimization problem can be formulated as:

$$\begin{aligned} & \underset{\mathbf{w}}{\text{maximize}} && \mathbf{r}^\top \mathbf{w} \\ & \text{subject to} && \mathbf{w}^\top \Sigma \mathbf{w} \leq \rho, \\ & && \mathbf{1}^\top \mathbf{w} = 1. \end{aligned} \tag{3.2}$$

3.6.4. Alternative Formulation II: Bounding Expected Profits

Given a minimum acceptable return level ρ_r , the optimization problem can be formulated as:

$$\begin{aligned} & \underset{\mathbf{w}}{\text{minimize}} && \mathbf{w}^\top \Sigma \mathbf{w} \\ & \text{subject to} && \mathbf{r}^\top \mathbf{w} \geq \rho_r, \\ & && \mathbf{1}^\top \mathbf{w} = 1. \end{aligned} \tag{3.3}$$

3.6.5. Properties of the Covariance Matrix

Lemma 3.32: Covariance is Symmetric


The covariance matrix Σ is symmetric.

Proof. By definition, the covariance between two random variables X and Y is given by:

$$\text{cov}(X, Y) = E[(X - E[X])(Y - E[Y])] \tag{3.4}$$

From this definition, it's straightforward to deduce that:

$$\begin{aligned}\text{cov}(X, Y) &= E[(X - E[X])(Y - E[Y])] \\ &= E[(Y - E[Y])(X - E[X])] \\ &= \text{cov}(Y, X)\end{aligned}$$

Thus, the covariance matrix, which captures the covariances between all pairs of random variables, is symmetric. 

Lemma 3.33: Covariance is PSD

The covariance matrix Σ is positive semi-definite.

Proof.

Consider a covariance matrix Σ for a set of random variables $\mathbf{X} = [X_1, X_2, \dots, X_n]^T$. By definition, the i, j -th entry of Σ is:

$$\Sigma_{ij} = \text{cov}(X_i, X_j)$$

We wish to show that for any vector $\mathbf{w} \in \mathbb{R}^n$, the quadratic form $Q(\mathbf{w}) = \mathbf{w}^T \Sigma \mathbf{w} \geq 0$.

Starting with this quadratic form:

$$Q(\mathbf{w}) = \mathbf{w}^T \Sigma \mathbf{w} = \sum_{i=1}^n \sum_{j=1}^n w_i w_j \text{cov}(X_i, X_j) = \sum_{i=1}^n \sum_{j=1}^n w_i w_j E[(X_i - E[X_i])(X_j - E[X_j])]$$

Consider $Y = \sum_{i=1}^n w_i X_i$. Its expectation is:

$$E[Y] = E\left[\sum_{i=1}^n w_i X_i\right] = \sum_{i=1}^n w_i E[X_i].$$

The variance of Y is:

$$\begin{aligned}\text{var}(Y) &= E[(Y - E[Y])^2] = E\left[\left(\sum_{i=1}^n w_i X_i - \sum_{i=1}^n w_i E[X_i]\right)^2\right] \\ &= E\left[\left(\sum_{i=1}^n w_i (X_i - E[X_i])\right)\left(\sum_{j=1}^n w_j (X_j - E[X_j])\right)\right] = \sum_{i=1}^n \sum_{j=1}^n w_i w_j E[(X_i - E[X_i])(X_j - E[X_j])].\end{aligned}$$

From the above, we deduce:

$$Q(\mathbf{w}) = \text{var}(Y).$$

Since variance is non-negative, $Q(\mathbf{w}) \geq 0$ for all $\mathbf{w} \in \mathbb{R}^n$, proving that Σ is positive semidefinite.



In essence, the symmetry of the covariance matrix stems from the commutative property of the covariance calculation, while its positive semidefiniteness arises from the inherent non-negativity of variance for any linear combination of the concerned random variables.

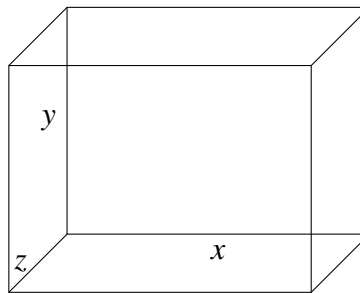
3.7 Box Design: Minimizing Box Surface Area with a Volume Constraint

Problem Description: We want to design a box with the smallest possible surface area such that its volume is at least 10 cubic units.

Mathematical Model:

Variables:

- x : length of the box
- y : width of the box
- z : height of the box



Objective Function: To minimize the surface area of the box, we aim to minimize the sum of the areas of all its sides. The surface area A of the box is:

$$A(x, y, z) = 2xy + 2xz + 2yz$$

Constraints:

1. The volume $V(x, y, z) = xyz$ of the box must be at least 10:

$$xyz \geq 10$$

2. All dimensions must be non-negative:

$$x \geq 0, \quad y \geq 0, \quad z \geq 0$$

Complete Mathematical Model:

Minimize: $A(x, y, z) = 2xy + 2xz + 2yz$

Subject to: $xyz \geq 10$

$$x \geq 0, \quad y \geq 0, \quad z \geq 0$$

Notice this is a non-convex quadratic objective function with a cubic constraint.

Notice that

$$\nabla A(x,y,z) = 2 \begin{bmatrix} y+z \\ x+z \\ x+y \end{bmatrix}, \quad \nabla^2 A(x,y,z) = 2 \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

This objective is non-convex as the hessian has both positive and negative eigenvalues.

But notice that at optimality, we expect the volume constraint to be tight!

Thus, we can assume at the optimal solution that

$$xyz = 10.$$

Thus, we have the relationships

$$xy = \frac{10}{z} \quad yz = \frac{10}{x} \quad xz = \frac{10}{y}.$$

Thus, we could recast the problem as

$$\begin{aligned} \text{Minimize: } & A(x,y,z) = 20 \left(\frac{1}{x} + \frac{1}{y} + \frac{1}{z} \right) \\ \text{Subject to: } & xyz = 10 \\ & x \geq 0, \quad y \geq 0, \quad z \geq 0 \end{aligned}$$

Now let's recast the problem one more time by setting $x' = \frac{1}{x}$, $y' = \frac{1}{y}$ and $z' = \frac{1}{z}$. Then the problem becomes

$$\begin{aligned} \text{Minimize: } & A(x',y',z') = 20 (x' + y' + z') \\ \text{Subject to: } & \frac{1}{x'y'z'} = 10 \\ & x' \geq 0, \quad y' \geq 0, \quad z' \geq 0 \end{aligned}$$

Resulting in

$$\begin{aligned} \text{Minimize: } & A(x',y',z') = 20 (x' + y' + z') \\ \text{Subject to: } & \frac{1}{10} = x'y'z' \\ & x' \geq 0, \quad y' \geq 0, \quad z' \geq 0 \end{aligned}$$

In fact, this can be relaxed to



$$\begin{aligned} \text{Minimize: } & A(x',y',z') = 20 (x' + y' + z') \\ \text{Subject to: } & \frac{1}{10} \leq x'y'z' \\ & x' \geq 0, \quad y' \geq 0, \quad z' \geq 0 \end{aligned}$$

And this problem actually has convex constraints and an linear (and hence convex) objective.

3.8 Modeling

We will discuss a few models and mention important changes to the models that will make them solvable.

IMPORTANT TIPS

1. **Find a convex formulation.** It may be that the most obvious model for your problem is actually non-convex. Try to reformulate your model into one that is convex and hence easier for solvers to handle.
2. **Intelligent formulation.** Understanding the problem structure may help reduce the complexity of the problem. Try to deduce something about the solution to the problem that might make the problem easier to solve. This may work for special cases of the problem.
3. **Identify problem type and select solver.** Based on your formulation, identify which type of problem it is and which solver is best to use for that type of problem. For instance,  Gurobi can handle some convex quadratic problems, but not all. Ipopt is a more general solver, but may be slower due to the types of algorithms that it uses.
4. **Add bounds on the variables.** Many solvers perform much better if they are provided bounds to the variables. This is because it reduces the search region where the variables live. Adding good bounds could be the difference in the solver finding an optimal solution and not finding any solution at all.
5. **Warm start.** If you know good possible solutions to the problem (or even just a feasible solution), you can help the solver by telling it this solution. This will reduce the amount of work the solver needs to do. In  JUMP this can be done by using the command `setvalue(x,[2 4 6])`, where here it sets the value of vector x to $[2\ 4\ 6]$. It may be necessary to specify values for all variables in the problem for it to start at.
6. **Rescaling variables.** It sometimes is useful to have all variables on the same rough scale. For instance, if minimizing $x^2 + 100^2y^2$, it may be useful to define a new variable $\bar{y} = 100y$ and instead minimize $x^2 + \bar{y}^2$.
7. **Provide derivatives.** Working out gradient and hessian information by hand can save the solver time. Particularly when these are sparse (many zeros). These can often be provided directly to the solver.

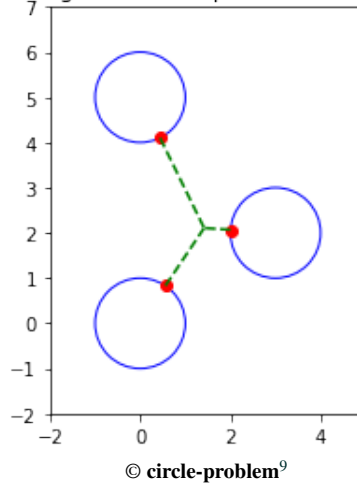
3.8.1. Minimum distance to circles

The problem we will consider here is: Given n circles, find a center point that minimizes the sum of the distances to all of the circles.

Minimize distance to circles:

Given circles described by center points (a_i, b_i) and radius r_i for $i = 1, \dots, n$, find a point $c = (c_x, c_y)$ that minimizes the sum of the distances to the circles.

Minimizing distance of points on three circles



Minimize distance to circles - Model attempt #1:

Non-convex

Let (x_i, y_i) be a point in circle i . Let w_i be the distance from (x_i, y_i) to c . Then we can model the problem as follows:

$$\begin{aligned}
 \min \quad & \sum_{i=1}^3 w_i && \text{Sum of distances} \\
 \text{s.t.} \quad & \sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} = r, \quad i = 1, \dots, n && (x_i, y_i) \text{ is in circle } i \\
 & \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} = w_i \quad i = 1, \dots, n && w_i \text{ is distance from } (x_i, y_i) \text{ to } c
 \end{aligned} \tag{3.1}$$

This model has several issues:

1. If the center c lies inside one of the circles, then the constraint $\sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} = r$ may not be valid. This is because the optimal choice for (x_i, y_i) in this case would be inside the circle, that is, satisfying $\sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} \leq r$.

⁹circle-problem, from circle-problem. circle-problem, circle-problem.

2. This model is **nonconvex**. In particular the equality constraints make the problem nonconvex.

Fortunately, we can relax the problem to make it convex and still model the correct solution. In particular, consider the constraint

$$\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} = w_i.$$

Since we are minimizing $\sum w_i$, it is equivalent to have the constraint

$$\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i.$$

This is equivalent because any optimal solution make w_i the smallest it can, and hence will meet that constraint at equality.

What is great about this change, it that it makes the constraint **convex!**. To see this we can write $f(z) = \|z\|_2^2$, $z = (x_i - c_x, y_i - c_y)$. Since $f(z)$ is convex and the transformation into variables x_i, c_x, y_i, c_y is linear, we have that $f(x_i - c_x, y_i - c_y)$ is convex. Then since $-w_i$ is linear, we have that

$$f(x_i - c_x, y_i - c_y) - w_i$$

is a convex function. Thus, the constraint

$$f(x_i - c_x, y_i - c_y) - w_i \leq 0$$

is a convex constraint.

This brings us to our second model.

Minimize distance to circles - Model attempt #2:

Convex, but has square roots

Let (x_i, y_i) be a point in circle i . Let w_i be the distance from (x_i, y_i) to c . Then we can model the problem as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^3 w_i & \text{Sum of distances} \\ \text{s.t.} & \sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} \leq r, \quad i = 1, \dots, n & (x_i, y_i) \text{ is in circle } i \\ & \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i \quad i = 1, \dots, n & w_i \text{ is distance from } (x_i, y_i) \text{ to } c \end{array} \quad (3.2)$$

Lastly, we would like to make this model better for a solver. For this we will

1. Add bounds on all the variables
2. Change format of non-linear inequalities

Minimize distance to circles - Model attempt #3:

Convex Model, easy to code

Let (x_i, y_i) be a point in circle i . Let w_i be the distance from (x_i, y_i) to c . Then we can model the problem as follows:

$$\begin{aligned}
 \min \quad & \sum_{i=1}^3 w_i && \text{Sum of distances} \\
 \text{s.t.} \quad & (x_i - a_i)^2 + (y_i - b_i)^2 \leq r^2, \quad i = 1, \dots, n && (x_i, y_i) \text{ is in circle } i \\
 & (x_i - c_x)^2 + (y_i - c_y)^2 \leq w_i^2 \quad i = 1, \dots, n && w_i \text{ is distance from } (x_i, y_i) \text{ to } c \\
 & 0 \leq w_i \leq u_i \\
 & a_i - r \leq x_i \leq a_i + r \\
 & b_i - r \leq y_i \leq b_i + r
 \end{aligned} \tag{3.3}$$

Example: Minimize distance to circles

Gurobipy

Here we minimize the distance of three circles of radius 1 centered at $(0,0)$, $(3,2)$, and $(0,5)$. Note: The bounds on the variables here are not chosen optimally.

$$\begin{aligned}
 \min \quad & w_1 + w_2 + w_3 \\
 \text{Subject to} \quad & (x_1 - 0)^2 + (y_1 - 0)^2 \leq 1 \\
 & (x_2 - 3)^2 + (y_2 - 2)^2 \leq 1 \\
 & (x_3 - 0)^2 + (y_3 - 5)^2 \leq 1 \\
 & (x_1 - c_x)^2 + (y_1 - c_y)^2 \leq w_1^2 \\
 & (x_2 - c_x)^2 + (y_2 - c_y)^2 \leq w_2^2 \\
 & (x_3 - c_x)^2 + (y_3 - c_y)^2 \leq w_3^2 \\
 & -1 \leq x_i \leq 10 \quad \forall i \in \{1, 2, 3\} \\
 & -1 \leq y_i \leq 10 \quad \forall i \in \{1, 2, 3\} \\
 & 0 \leq w_i \leq 40 \quad \forall i \in \{1, 2, 3\} \\
 & -1 \leq c_x \leq 10 \\
 & -1 \leq c_y \leq 10
 \end{aligned}$$

3.9 Machine Learning

We discuss briefly the field of machine learning and will go into this area deeper in later chapters.

There are two main fields of machine learning:

- Supervised Machine Learning,
- Unsupervised Machine Learning.

Supervised machine learning is composed of *Regression* and *Classification*. This area is thought of as being given labeled data that you are then trying to understand the trends of this labeled data.

Unsupervised machine learning is where you are given unlabeled data and then need to decide how to label this data. For instance, how can you optimally partition the people in a room into 5 groups that share the most commonalities?

3.10 Machine learning - Supervised Learning - Classification

The problem of data *classification* begins with *data* and *labels*. The goal is *classification* of future data based on sample data that you have by constructing a function to understand future data.

Goal: *Classification* - create a function $f(x)$ that takes in a data point x and outputs the correct label.

These functions can take many forms. In binary classification, the label set is $\{+1, -1\}$, and we want to correctly (as often as we can) determine the correct label for a future data point.

There are many ways to determine such a function $f(x)$. SVM is one way to go about this classification. SVM that determines the function by computing a hyperplane that separates the data labeled $+1$ from the data labeled -1 .

Later, we will learn about *neural networks* that describe much more complicated functions.

Another method is to create a *decision tree*. These are typically more interpretable functions (neural networks are often a bit mysterious) and thus sometimes preferred in settings where the classification should be easily understood, such as a medical diagnosis. We will not discuss this method here since it fits less well with the theme of nonlinear programming.

3.11 Resources

Resources

SVM

- *Python SGD implementation and video for support vector machines*
- https://www.youtube.com/watch?time_continue=6&v=N1v0golbjSc

Box design

- *Box design optimization using Scipy*

Classification

- https://www.youtube.com/watch?v=bwZ3QiuJ3i8&list=PL9ooVrP1hQOHUfd-g8GUpKI3hH0wM_9Dn&index=13
- <https://towardsdatascience.com/solving-a-simple-classification-problem-with-python>

Neural Networks:

- <https://www.youtube.com/watch?v=bVQUSndD1lU>
- <https://www.youtube.com/watch?v=8bNIkfrJZpo>
- <https://www.youtube.com/watch?v=Dws9Zveu9ug>

4. NLP Algorithms

4.1 Algorithms Introduction

We will begin with unconstrained optimization and consider several different algorithms based on what is known about the objective function. In particular, we will consider the cases where we use

- Only function evaluations (also known as *derivative free optimization*),
- Function and gradient evaluations,
- Function, gradient, and hessian evaluations.

We will first look at these algorithms and their convergence rates in the 1-dimensional setting and then extend these results to higher dimensions.

4.2 1-Dimensional Algorithms

We suppose that we solve the problem

$$\min f(x) \tag{4.1}$$

$$x \in [a, b]. \tag{4.2}$$

That is, we minimize the univariate function $f(x)$ on the interval $[l, u]$.

For example,

$$\min (x^2 - 2)^2 \tag{4.3}$$

$$0 \leq x \leq 10. \tag{4.4}$$

Note, the optimal solution lies at $x^* = \sqrt{2}$, which is an irrational number. Since we will consider algorithms using floating point precision, we will look to return a solution \bar{x} such that $\|x^* - \bar{x}\| < \varepsilon$ for some small $\varepsilon > 0$, for instance, $\varepsilon = 10^{-6}$.

4.2.1. Golden Search Method - Derivative Free Algorithm

Resources

- *Youtube! - Golden Section Search Method*

Suppose that $f(x)$ is unimodal on the interval $[a, b]$, that is, it is a continuous function that has a single minimizer on the interval.

Without any extra information, our best guess for the optimizer is $\bar{x} = \frac{a+b}{2}$ with a maximum error of $\varepsilon = \frac{b-a}{2}$. Our goal is to reduce the size of the interval where we know x^* to be, and hence improve our best guess and the maximum error of our guess.

Now we want to choose points in the interior of the interval to help us decide where the minimizer is. Let x_1, x_2 such that

$$a < x_2 < x_1 < b.$$

Next, we evaluate the function at these four points. Using this information, we would like to argue a smaller interval in which x^* is contained. In particular, since f is unimodal, it must hold that

1. $x^* \in [a, x_2]$ if $f(x_1) \leq f(x_2)$,
2. $x^* \in [x_1, b]$ if $f(x_2) < f(x_1)$,

After comparing these function values, we can reduce the size of the interval and hence reduce the region where we think x^* is.

We will now discuss how to choose x_1, x_2 in a way that we can

1. Reuse function evaluations,
2. Have a constant multiplicative reduction in the size of the interval.

We consider the picture:

To determine the best d , we want to decrease by a constant factor. Hence, we decrease by a factor γ , which we will see is the golden ration (GR). To see this, we assume that $(b - a) = 1$, and ask that $d = \gamma$. Thus, $x_1 - a = \gamma$ and $b - x_2 = \gamma$. If we are in case 1, then we cut off $b - x_1 = 1 - \gamma$. Now, if we iterate and do this again, we will have an initial length of γ and we want to cut off the interval $x_2 - x_1$ with this being a proportion of $(1 - \gamma)$ of the remaining length. Hence, the second time we will cut off $(1 - \gamma)\gamma$, which we set as the length between x_1 and x_2 .

Considering the geometry, we have

$$\text{length } a \text{ to } x_1 + \text{length } x_2 \text{ to } b = \text{total length} + \text{length } x_2 \text{ to } x_1$$

hence

$$\gamma + \gamma = 1 + (1 - \gamma)\gamma.$$

Simplifying, we have

$$\gamma^2 + \gamma - 1 = 0.$$

Applying the quadratic formula, we see

$$\gamma = \frac{-1 \pm \sqrt{5}}{2}.$$

Since we want $\gamma > 0$, we take

$$\gamma = \frac{-1 + \sqrt{5}}{2} \approx 0.618$$

This is exactly the Golden Ratio (or, depending on the definition, the golden ratio minus 1).

4.2.1.1. Example:

We can conclude that the optimal solution is in $[1.4, 3.8]$, so we would guess the midpoint $\bar{x} = 2.6$ as our approximate solution with a maximum error of $\varepsilon = 1.2$.

Convergence Analysis of Golden Search Method:

After t steps of the Golden Search Method, the interval in question will be of length

$$(b - a)(GR)^t \approx (b - a)(0.618)^t$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b - a)(0.618)^t.$$

4.2.2. Bisection Method - 1st Order Method (using Derivative)

4.2.2.1. Minimization Interpretation

Assumptions: f is convex, differentiable

We can look for a minimizer of the function $f(x)$ on the interval $[a, b]$.

4.2.2.2. Root finding Interpretation

Instead of minimizing, we can look for a root of $f'(x)$. That is, find x such that $f'(x) = 0$.

Assumptions: $f'(a) < 0 < f'(b)$, **OR**, $f'(b) < 0 < f'(a)$. f' is continuous

The goal is to find a root of the function $f'(x)$ on the interval $[a, b]$. If f is convex, then we know that this root is indeed a global minimizer.

Note that if f is convex, it only makes sense to have the assumption $f'(a) < 0 < f'(b)$.

Convergence Analysis of Bisection Method:

After t steps of the Bisection Method, the interval in question will be of length

$$(b - a) \left(\frac{1}{2}\right)^t.$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b - a) \left(\frac{1}{2}\right)^t.$$

4.2.3. Gradient Descent - 1st Order Method (using Derivative)

Input: $f(x)$, $\nabla f(x)$, initial guess x^0 , learning rate α , tolerance ε

Output: An approximate solution x

1. Set $t = 0$
2. While $\|f(x^t)\|_2 > \varepsilon$:
 - (a) Set $x^{t+1} \leftarrow x^t - \alpha \nabla f(x^t)$.
 - (b) Set $t \leftarrow t + 1$.
3. Return x^t .

4.2.4. Newton's Method - 2nd Order Method (using Derivative and Hessian)

Input: $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$, initial guess x^0 , learning rate α , tolerance ε

Output: An approximate solution x

1. Set $t = 0$
2. While $\|f(x^t)\|_2 > \varepsilon$:
 - (a) Set $x^{t+1} \leftarrow x^t - \alpha[\nabla^2 f(x^t)]^{-1} \nabla f(x^t)$.
 - (b) Set $t \leftarrow t + 1$.
3. Return x^t .

4.3 Multi-Variate Unconstrained Optimization

We will now use the techniques for 1-Dimensional optimization and extend them to multi-variate case. We will begin with unconstrained versions (or at least, constrained to a large box) and then show how we can apply these techniques to constrained optimization.

4.3.1. Descent Methods - Unconstrained Optimization - Gradient, Newton

Outline for Descent Method for Unconstrained Optimization:

Input:

- A function $f(x)$
- Initial solution x^0
- Method for computing step direction d_t
- Method for computing length t of step
- Number of iterations T

Output:

- A point x_T (hopefully an approximate minimizer)

Algorithm

1. For $t = 1, \dots, T$,

set $x_{t+1} = x_t + \alpha_t d_t$

4.3.1.1. Choice of α_t

There are many different ways to choose the step length α_t . Some choices have proofs that the algorithm will converge quickly. An easy choice is to have a constant step length $\alpha_t = \alpha$, but this may depend on the specific problem.

4.3.1.2. Choice of d_t using $\nabla f(x)$

Choice of descent methods using $\nabla f(x)$ are known as *first order methods*. Here are some choices:

1. **Gradient Descent:** $d_t = -\nabla f(x_t)$
2. **Nesterov Accelerated Descent:** $d_t = \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1}))$

Here, μ, γ are some numbers. The number μ is called the momentum.

4.3.2. Stochastic Gradient Descent - The mother of all algorithms.

A popular method is called *stochastic gradient descent* (SGD). This has been described as "The mother of all algorithms". This is a method to **approximate the gradient** typically used in machine learning or stochastic programming settings.

Stochastic Gradient Descent:

Suppose we want to solve

$$\min_{x \in \mathbb{R}^n} F(x) = \sum_{i=1}^N f_i(x). \quad (4.1)$$

We could use *gradient descent* and have to compute the gradient $\nabla F(x)$ at each iteration. But! We see that in the **cost to compute the gradient** is roughly $O(nN)$, that is, it is very dependent on the number of function N , and hence each iteration will take time dependent on N .

Instead! Let i be a uniformly random sample from $\{1, \dots, N\}$. Then we will use $\nabla f_i(x)$ as an approximation of $\nabla F(x)$. Although we lose a bit by using a guess of the gradient, this approximation only takes $O(n)$ time to compute. And in fact, in expectation, we are doing the same thing. That is,

$$N \cdot \mathbb{E}(\nabla f_i(x)) = N \sum_{i=1}^N \frac{1}{N} \nabla f_i(x) = \sum_{i=1}^N \nabla f_i(x) = \nabla \left(\sum_{i=1}^N f_i(x) \right) = \nabla F(x).$$

Hence, the SGD algorithm is:

1. Set $t = 0$
2. While ...(some stopping criterion)

- (a) Choose i uniformly at random in $\{1, \dots, N\}$.
- (b) Set $d_t = \nabla f_i(x_t)$
- (c) Set $x_{t+1} = x_t - \alpha d_t$

There can be many variations on how to decide which functions f_i to evaluate gradient information on. Above is just one example.

Linear regression is an excellent example of this.

Example 4.1: Linear Regression with SGD

Given data points $x^1, \dots, x^N \in \mathbb{R}^d$ and output $y^1, \dots, y^N \in \mathbb{R}$, find $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that $a^\top x^i + b \approx y^i$. This can be written as the optimization problem

$$\min_{a,b} \sum_{i=1}^N g_i(a,b) \quad (4.2)$$

where $g_i(a,b) = (a^\top x^i + b)^2$.

Notice that the objective function $G(a,b) = \sum_{i=1}^N g_i(a,b)$ is a convex quadratic function. The gradient of the objective function is

$$\nabla G(a,b) = \sum_{i=1}^N \nabla g_i(a,b) = \sum_{i=1}^N 2x^i(a^\top x^i + b)$$

Hence, if we want to use gradient descent, we must compute this large sum (think of $N \approx 10,000$). Instead, we can **approximate the gradient!**. Let $\tilde{\nabla}G(a,b)$ be our approximate gradient. We will compute this by randomly choosing a value $r \in \{1, \dots, N\}$ (with uniform probability). Then set

$$\tilde{\nabla}G(a,b) = \nabla g_r(a,b).$$

It holds that the expected value is the same as the gradient, that is,

$$\mathbb{E}(\tilde{\nabla}G(a,b)) = \nabla G(a,b).$$

Hence, we can make probabilistic arguments that these two will have the same (or similar) convergence properties (in expectation).

4.3.3. Neural Networks

Resources

- *ML Zero to Hero - Part 1: Intro to machine learning*
- *ML Zero to Hero - Part 2: Basic Computer Vision with ML*

4.3.4. Choice of Δ_k using the hessian $\nabla^2 f(x)$

These choices are called *second order methods*

1. **Newton's Method:** $\Delta_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
2. **BFGS (Quasi-Newton):** $\Delta_k = -(B_k)^{-1} \nabla f(x_k)$

Here

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \end{aligned}$$

and

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

This serves as an approximation of the hessian and can be efficiently computed. Furthermore, the inverse can be easily computed using certain updating rules. This makes for a fast way to approximate the hessian.

4.4 Constrained Convex Nonlinear Programming

Given a convex function $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$ and convex functions $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{4.1}$$

4.4.1. Barrier Method

Constrained Convex Programming via Barrier Method:

We convert 4.1 into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^m \log(-f_i(x)) \\ & x \in \mathbb{R}^d \end{aligned} \quad (4.2)$$

Here $\phi > 0$ is some number that we choose. As $\phi \rightarrow 0$, the optimal solution $x(\phi)$ to (4.2) tends to the optimal solution of (4.1). That is $x(\phi) \rightarrow x^*$ as $\phi \rightarrow 0$.

Constrained Convex Programming via Barrier Method - Initial solution:

Define a variable $s \in \mathbb{R}$ and add that to the right hand side of the inequalities and then minimize it in the objective function.

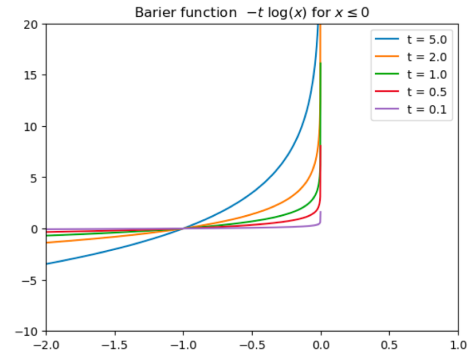
$$\begin{aligned} \min \quad & s \\ \text{s.t.} \quad & f_i(x) \leq s \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \quad (4.3)$$

Note that this problem is feasible for all x values since s can always be made larger. If there exists a solution with $s \leq 0$, then we can use the corresponding x solution as an initial feasible solution. Otherwise, the problem is infeasible.

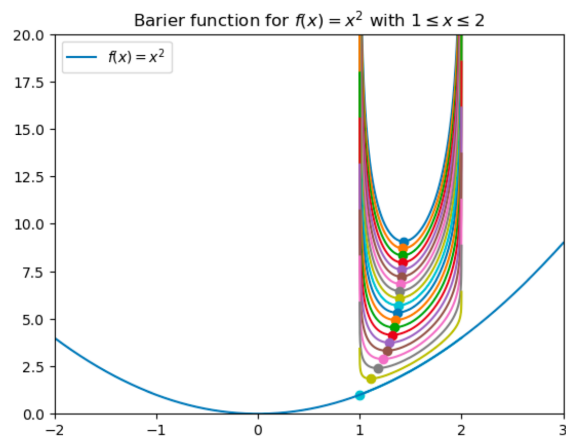
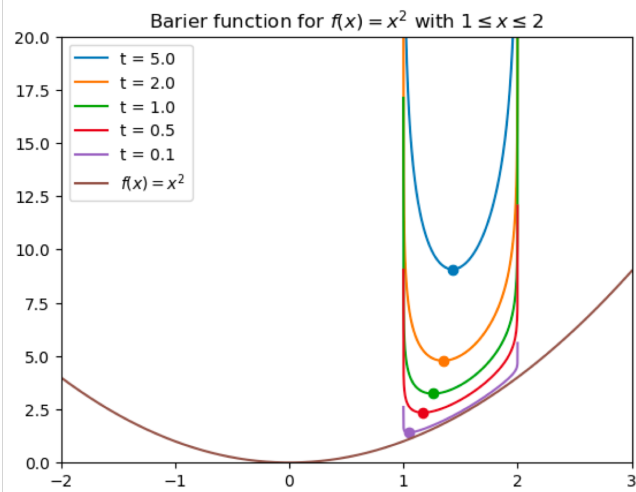
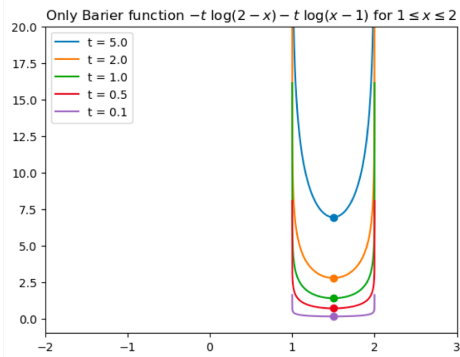
Now, convert this problem into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^m \log(-(f_i(x) - s)) \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \quad (4.4)$$

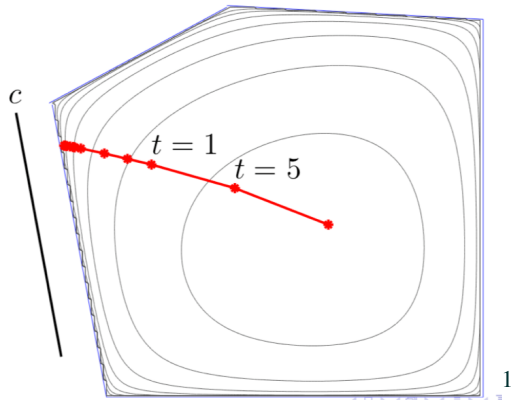
This problem has an easy time of finding an initial feasible solution. For instance, let $x = 0$, and then $s = \max_i f_i(x) + 1$.



Images below: the value t is the value ϕ discussed above



Minimizing $c^T x$ subject to $Ax \leq b$.



¹Image taken from unknown source.

5. Computational Issues with NLP

We mention a few computational issues to consider with nonlinear programs.

5.1 Irrational Solutions

Consider nonlinear problem (this is even convex)

$$\begin{array}{ll} \min & -x \\ \text{s.t.} & x^2 \leq 2. \end{array} \quad (5.1)$$

The optimal solution is $x^* = \sqrt{2}$, which cannot be easily represented. Hence, we would settle for an **approximate solution** such as $\bar{x} = 1.41421$, which is feasible since $\bar{x}^2 \leq 2$, and it is close to optimal.

5.2 Discrete Solutions

Consider nonlinear problem (not convex)

$$\begin{array}{ll} \min & -x \\ \text{s.t.} & x^2 = 2. \end{array} \quad (5.1)$$

Just as before, the optimal solution is $x^* = \sqrt{2}$, which cannot be easily represented. Furthermore, the only two feasible solutions are $\sqrt{2}$ and $-\sqrt{2}$. Thus, there is no chance to write down a feasible rational approximation.

5.3 Convex NLP Harder than LP

Convex NLP is typically polynomially solvable. It is a generalization of linear programming.

Convex Programming:

Polynomial time (P) (typically)

Given a convex function $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$ and convex functions $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{5.1}$$

Example 5.1: C

Convex programming is a generalization of linear programming. This can be seen by letting $f(x) = c^\top x$ and $f_i(x) = A_i x - b_i$.

5.4 NLP is harder than IP

As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1-x) = 0$$

has exactly two solutions: $x = 0, x = 1$. Thus, quadratic constraints can be used to model binary constraints.

Binary Integer programming (BIP) as a NLP:

NP-Hard

Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and vector $c \in \mathbb{R}^n$, the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & \cancel{x \in \{0,1\}^n} \\ & x_i(1-x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{5.1}$$

5.5 Karush-Huhn-Tucker (KKT) Conditions

The KKT conditions use the augmented Lagrangian problem to describe sufficient conditions for optimality of a convex program.

KKT Conditions for Optimality:

Given a convex function $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$ and convex functions $g_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{5.1}$$

Given $(\bar{x}, \bar{\lambda})$ with $\bar{x} \in \mathbb{R}^d$ and $\bar{\lambda} \in \mathbb{R}^m$, if the KKT conditions hold, then \bar{x} is optimal for the convex programming problem.

The KKT conditions are

1. (Stationary).

$$-\nabla f(\bar{x}) = \sum_{i=1}^m \bar{\lambda}_i \nabla g_i(\bar{x}) \tag{5.2}$$

2. (Complimentary Slackness).

$$\bar{\lambda}_i g_i(\bar{x}) = 0 \text{ for } i = 1, \dots, m \tag{5.3}$$

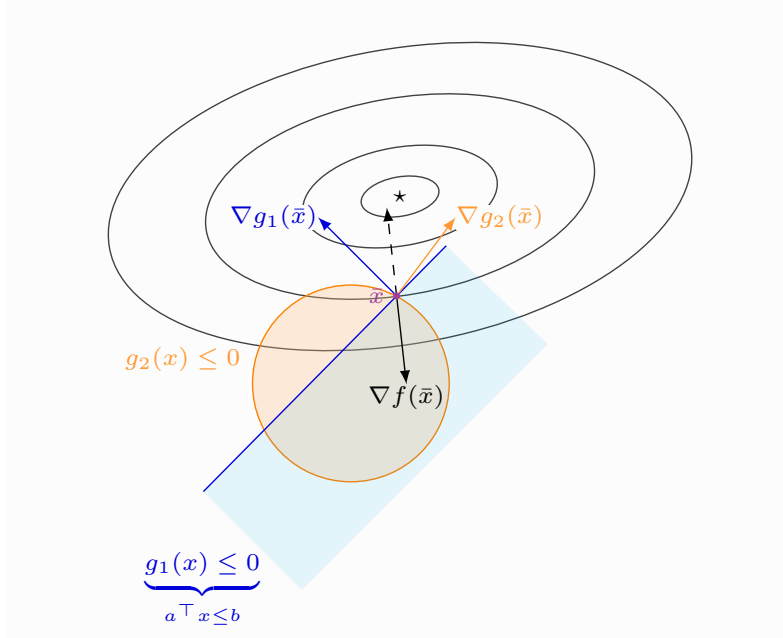
3. (Primal Feasibility).

$$g_i(\bar{x}) \leq 0 \text{ for } i = 1, \dots, m \tag{5.4}$$

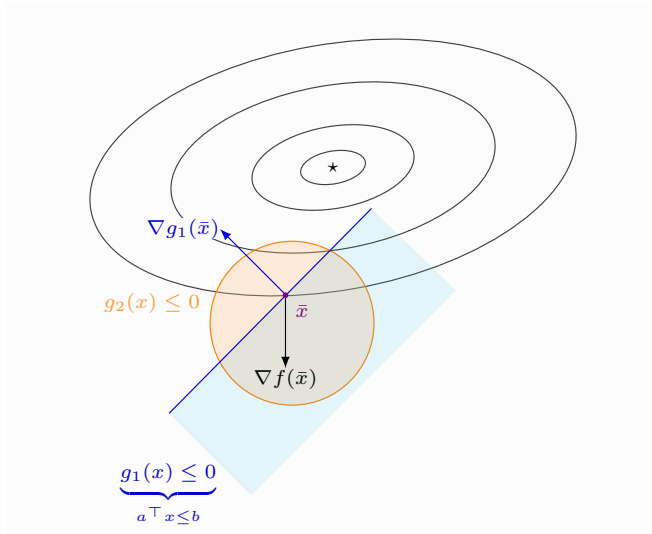
4. (Dual Feasibility).

$$\bar{\lambda}_i \geq 0 \text{ for } i = 1, \dots, m \tag{5.5}$$

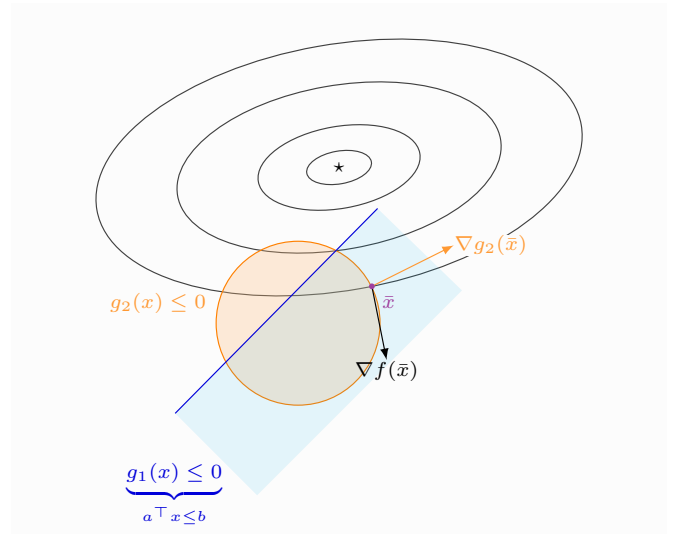
If certain properties are true of the convex program, then every optimizer has these properties. In particular, this holds for Linear Programming.



© tikz/kkt-optimal¹



© tikz/kkt-non-optimal1²



© tikz/kkt-non-optimal2³

tikz/kkt-non-optimal1, from **tikz/kkt-non-optimal1**. **tikz/kkt-non-optimal2**, from **tikz/kkt-non-optimal2**.
tikz/kkt-non-optimal1, **tikz/kkt-non-optimal1**. **tikz/kkt-non-optimal2**, **tikz/kkt-non-optimal2**.

¹**tikz/kkt-optimal**, from **tikz/kkt-optimal**. **tikz/kkt-optimal**, **tikz/kkt-optimal**.

5.6 Gradient Free Algorithms

5.6.1. Needler-Mead

Resources

- *Wikipedia*
- *Youtube*

6. Material to add...

6.0.1. Bisection Method and Newton's Method

Resources

- See section 4 of the following nodes: <http://www.seas.ucla.edu/~vandenbe/133A/133A-notes.pdf>

6.1 Gradient Descent

Resources

Recap Gradient and Directional Derivatives:

- <https://www.youtube.com/watch?v=tIpKfDc295M>
- <https://www.youtube.com/watch?v=-02ze7tf08>
- https://www.youtube.com/watch?v=N_ZRcLheNv0
- <https://www.youtube.com/watch?v=4RBkIJPG6Yo>

Idea of Gradient descent:

- <https://youtu.be/IHZwWFHwa-w?t=323>

Vectors:

- https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=2&t=0s

7. Fairness in Algorithms

Resources

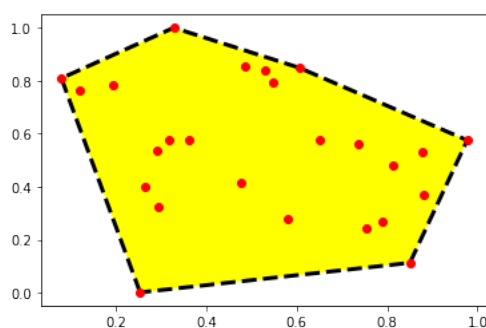
- *Simons Institute - Michael Kearns (University of Pennsylvania) - "The Ethical Algorithm"*

A. Linear Algebra

B. Results to put somewhere

Definition B.1: Convex Hull

Let $S \subseteq \mathbb{R}^n$. The convex hull $\text{conv}(S)$ is the smallest convex set containing S .



© convex-hull-random¹

Theorem B.2: Caratheodory's Theorem

Let $x \in \text{conv}(S)$ and $S \subseteq \mathbb{R}^n$. Then there exist $x^1, \dots, x^k \in S$ such that $x \in \text{conv}(\{x^1, \dots, x^k\})$ and $k \leq n + 1$.

C. Contributors



Champions of Access to Knowledge



OPEN TEXT

All digital forms of access to our high-quality open texts are entirely FREE! All content is reviewed for excellence and is wholly adaptable; custom editions are produced by Lyryx for those adopting Lyryx assessment. Access to the original source files is also open to anyone!



ONLINE ASSESSMENT

We have been developing superior online formative assessment for more than 15 years. Our questions are continuously adapted with the content and reviewed for quality and sound pedagogy. To enhance learning, students receive immediate personalized feedback. Student grade reports and performance statistics are also provided.



SUPPORT

Access to our in-house support team is available 7 days/week to provide prompt resolution to both student and instructor inquiries. In addition, we work one-on-one with instructors to provide a comprehensive system, customized for their course. This can include adapting the text, managing multiple sections, and more!



INSTRUCTOR SUPPLEMENTS

Additional instructor resources are also freely accessible. Product dependent, these supplements include: full sets of adaptable slides and lecture notes, solutions manuals, and multiple choice question banks with an exam building tool.

Contact Lyryx Today!

info@lyryx.com

¹This book was not produced by Lyryx, but this book has made substantial use of their open source material. We leave this page in here as a tribute to Lyryx for sharing their content.

lyryx
advancing learning
A First Course in Linear Algebra
an Open Text

BE A CHAMPION OF OER!

Contribute suggestions for improvements, new content, or errata:

A new topic

A new example

An interesting new question

A new or better proof to an existing theorem

Any other suggestions to improve the material

Contact Lyryx at info@lyryx.com with your ideas.

CONTRIBUTIONS

Ilijas Farah, York University

Ken Kuttler, Brigham Young University

Lyryx Learning Team

Foundations of Applied Mathematics

<https://github.com/Foundations-of-Applied-Mathematics>

CONTRIBUTIONS

List of Contributors

E. Evans

Brigham Young University

R. Evans

Brigham Young University

J. Grout

Drake University

J. Humpherys

Brigham Young University

T. Jarvis

Brigham Young University

J. Whitehead

Brigham Young University

J. Adams

Brigham Young University

J. Bejarano

Brigham Young University

Z. Boyd

Brigham Young University

M. Brown

Brigham Young University

A. Carr

Brigham Young University

C. Carter

Brigham Young University

T. Christensen

Brigham Young University

M. Cook

Brigham Young University

R. Dorff

Brigham Young University

B. Ehlert

Brigham Young University

M. Fabiano

Brigham Young University

K. Finlinson

Brigham Young University

J. Fisher

Brigham Young University

R. Flores

Brigham Young University

R. Fowers

Brigham Young University

A. Frandsen

Brigham Young University

R. Fuhrman

Brigham Young University

S. Giddens

Brigham Young University

C. Gigena

Brigham Young University

M. Graham

Brigham Young University

F. Glines

Brigham Young University

C. Glover

Brigham Young University

M. Goodwin

Brigham Young University

R. Grout

Brigham Young University

D. Grundvig

Brigham Young University

E. Hannesson

Brigham Young University

J. Hendricks

Brigham Young University

A. Henriksen

Brigham Young University

I. Henriksen

Brigham Young University

C. Hettinger

Brigham Young University

S. Horst
Brigham Young University
K. Jacobson
Brigham Young University
J. Leete
Brigham Young University
J. Lytle
Brigham Young University
R. McMurray
Brigham Young University
S. McQuarrie
Brigham Young University
D. Miller
Brigham Young University
J. Morrise
Brigham Young University
M. Morrise
Brigham Young University
A. Morrow
Brigham Young University
R. Murray
Brigham Young University
J. Nelson
Brigham Young University
E. Parkinson
Brigham Young University
M. Probst
Brigham Young University
M. Proudfoot
Brigham Young University
D. Reber
Brigham Young University

H. Ringer
Brigham Young University
C. Robertson
Brigham Young University
M. Russell
Brigham Young University
R. Sandberg
Brigham Young University
C. Sawyer
Brigham Young University
M. Stauffer
Brigham Young University
J. Stewart
Brigham Young University
S. Suggs
Brigham Young University
A. Tate
Brigham Young University
T. Thompson
Brigham Young University
M. Victors
Brigham Young University
J. Webb
Brigham Young University
R. Webb
Brigham Young University
J. West
Brigham Young University
A. Zaitzeff
Brigham Young University

This project is funded in part by the National Science Foundation, grant no. TUES Phase II DUE-1323785.

C.0.1. Graph Theory

Chapter on Graph Theory adapted from: CC-BY-SA 3.0 Math in Society A survey of mathematics for the liberal arts major Math in Society is a free, open textbook. This book is a survey of contemporary mathematical topics, most non-algebraic, appropriate for a college-level quantitative literacy topics course for liberal arts majors. The text is designed so that most chapters are independent, allowing the instructor to choose a selection of topics to be covered. Emphasis is placed on the applicability of the mathematics. Core material for each topic is covered in the main text, with additional depth available through exploration exercises appropriate for in-class, group, or individual investigation. This book is appropriate for Washington State Community Colleges' Math 107.

The current version is 2.5, released Dec 2017. <http://www.opentextbookstore.com/mathinsociety/2.5/GraphTheory.pdf>

Communicated by Tricia Muldoon Brown, Ph.D. Associate Professor of Mathematics Georgia Southern University Armstrong Campus Savannah, GA 31419 <http://math.armstrong.edu/faculty/brown/MATH1001.html>

