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: November 12, 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 formating this book much easier! See subsequent page for list of contributors.
- "Foundations of Applied Mathmatics" 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 LaTeXusing 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

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 rearrranged, 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 compter 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 desigened 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

- $\mathbb{1}$ - 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. $1,2,3,-1,-2,-3,...$)
- $\mathbb{Q}$ - 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, π, 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. $[3, \pi, -e, \sqrt{2}]$)
- $\sum_{i=1}^4 i = 1+2+3+4$
- $\sum_{i=1}^4 i^2 = 1^2+2^2+3^2+4^4$
- $\sum_{i=1}^4 x_i = x_1+x_2+x_3+x_4$
- $\square$ - 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 $\sum_{i=1}^3 x_i y_i = x_1y_1 + x_2y_2 + x_3y_3$.

A sample sentence:

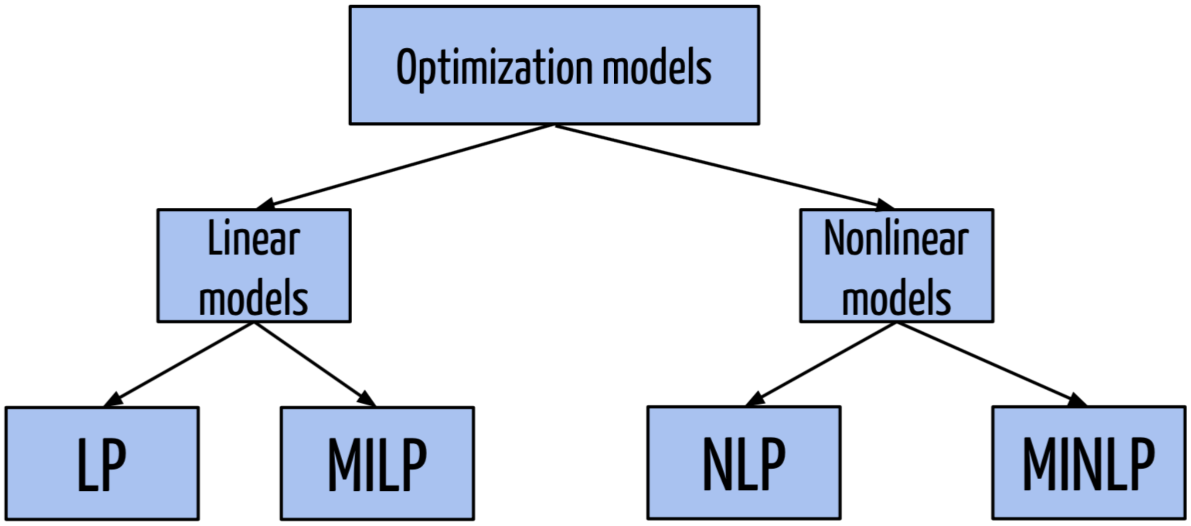
$$\forall x \in \mathbb{Q}^n \; \exists y \in \mathbb{Z}^n \setminus \{0\}^n s.t. x^\top y \in \{0,1\}$$

"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).



© Diego Moran CC0.[1]

Figure 2.1: ***Tree of optimization problems.***

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.

[1] *Tree of optimization problems.* from . Diego Moran CC0., 2017.

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

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

Linear programming can come in several forms, whether we are maximizing or minimizing, or if the constraints are $\le, =$ or $\ge$. 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

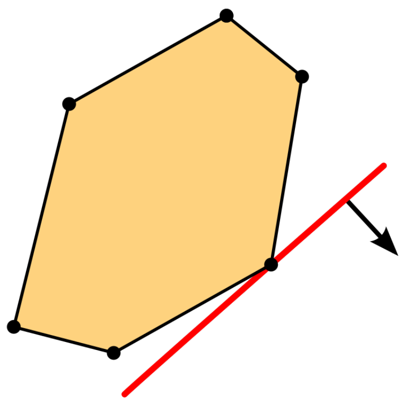
$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \ge 0 \end{aligned} \tag{2.2}$$

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.

[2] *A pictorial representation of a simple linear program with two variables and six inequalities. The set of feasible solutions is depicted in yellow and forms a polygon, a 2-dimensional polytope. The linear cost function is represented by the red line and the arrow: The red line is a level set of the cost function, and the arrow indicates the direction in which we are optimizing.* from `https://en.wikipedia.org/wiki/Linear_programming#/media/File:Linear_optimization_in_a_2-dimensional_polytope.svg`. Ylloh CC0., 2012.



© Ylloh CC0.[2]
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

[3] *IP polytope with LP relaxation*, from `https://en.wikipedia.org/wiki/Integer_programming#/media/File:IP_polytope_with_LP_relaxation.svg`. Fanosta CC BY-SA 4.0., 2019.

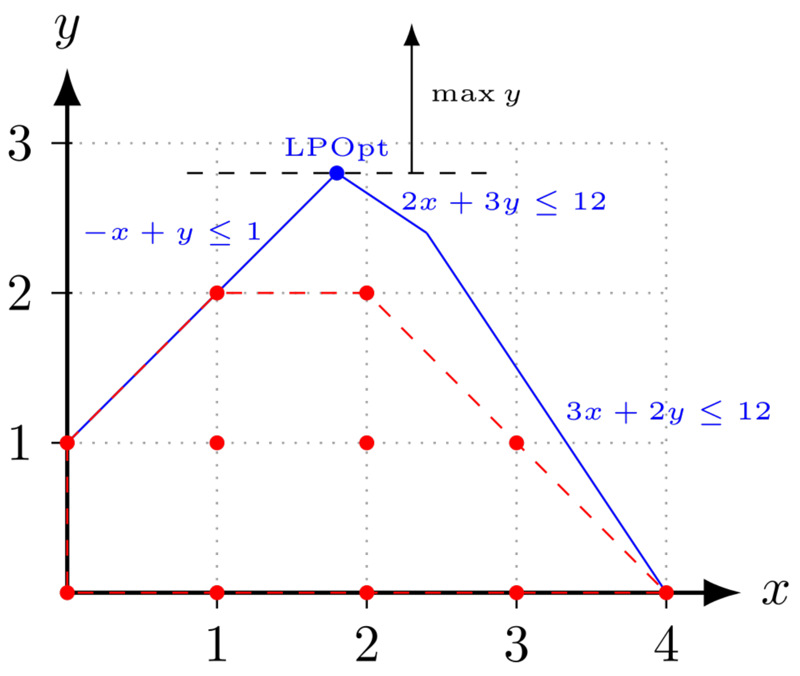


Figure 2.3: Comparing the LP relaxation to the IP solutions.

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

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

An even more general class is *Mixed-Integer Linear Programming (MILP)*. This is where we have n integer variables $x_1,\ldots,x_n \in \mathbb{Z}$ and d continuous variables $x_{n+1},\ldots,x_{n+d} \in \mathbb{R}$. 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 $Ax \le b$.

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-*

ming problem is

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

2.3 Non-Linear Programming (NLP)

NLP:

NP-Hard

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

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

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 $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *convex programming* problem is

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

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:

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

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:

$$\min_{\mathbf{c}_1,\ldots,\mathbf{c}_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \left\| x_i - \mathbf{c}_j \right\|^2 \tag{2.5}$$

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. Optimization Under Uncertainty

3.1 Robust Optimization

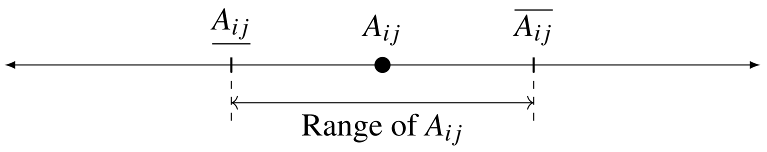
Robust optimization deals with problems of decision-making under uncertainty, particularly focusing on creating solutions that remain feasible under all variations within specified uncertainty sets. This section introduces two prevalent forms of uncertainty: interval uncertainty for individual coefficients and ellipsoidal uncertainty for rows of a coefficient matrix.

3.1.1. Interval Uncertainty

In interval uncertainty, each coefficient in the data matrix or vector is assumed to have an upper and lower bound, representing the range of its possible values. For a linear programming problem, this can be written as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b, \quad \text{for all } A \in [\underline{A}, \overline{A}], b \in [\underline{b}, \overline{b}] \end{aligned} \tag{3.1}$$

where x is the vector of decision variables, c is the cost vector, A and b are the technology matrix and resource vector with their respective ranges defined by the lower $\underline{A}, \underline{b}$ and upper $\overline{A}, \overline{b}$ bounds.



Example 3.1: Interval Uncertainty in LP

Consider the LP problem

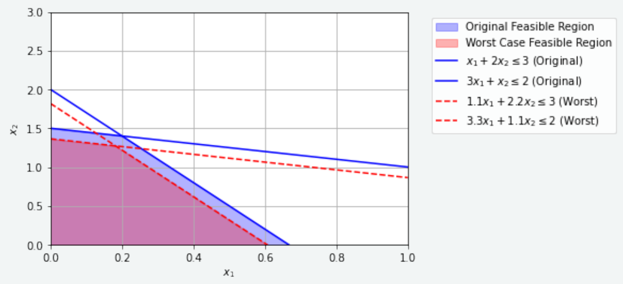
$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 2x_1 - 3x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 3, \\ & 3x_1 + x_2 \leq 2, \\ & x_1, x_2 \geq 0, \end{aligned} \tag{3.2}$$

but now we assume that the coefficients in the LP might be uncertain up to 10%.

For the objective function, coefficients of x_1 and x_2 are within intervals [1.8, 2.2] and [-3.3, -2.7], respectively.
The constraint coefficients are within the following intervals: for the first constraint, x_1 is in [0.9, 1.1] and x_2 is in [1.8, 2.2]; for the second constraint, x_1 is in [2.7, 3.3] and x_2 is in [0.9, 1.1]. The bounds on b are the constant terms on the right side of the constraints, which are assumed to be fixed for this example.
The LP formulation considering the worst-case scenario for the coefficient intervals is as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 2.2x_1 - 2.7x_2 \\ \text{subject to} \quad & 1.1x_1 + 2.2x_2 \leq 3, \\ & 3.3x_1 + 1.1x_2 \leq 2, \\ & x_1, x_2 \geq 0. \end{aligned} \tag{3.3}$$

This LP takes the upper bound of coefficients for the objective function and the lower bound for the constraints, thereby presenting the most conservative estimate of the solution space in face of the given uncertainties.

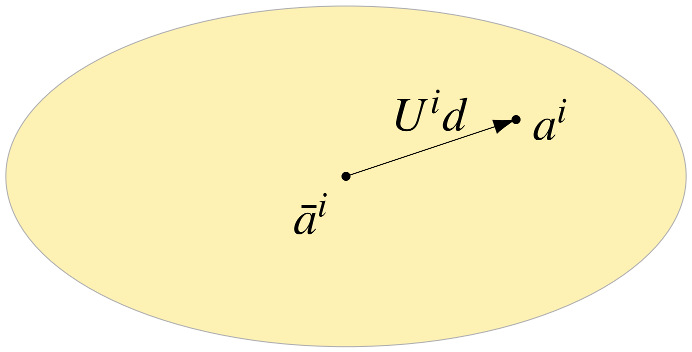


3.1.2. Ellipsoidal Uncertainty - Stolen From Boyd Convex Optimization

Ellipsoidal uncertainty models the uncertainty in a row a^i of the matrix A as lying within an ellipsoid or ball. The uncertain row a^i can be represented as:

$$a^i \in \mathscr{E}_i := \{\bar{a}^i + U^i d \ : \ \|d\|_2 \leq 1\}, \tag{3.4}$$

where $\bar{a}^i$ is the nominal row vector, U^i is a matrix specifying the directions of deviation for row i, d is a deviation vector within the unit ball.



Robust linear programming We consider a linear program in inequality form,

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad i = 1, \ldots, m \end{array}$$

in which there is some uncertainty or variation in the parameters c, a_i, b_i. To simplify the exposition we assume that c and b_i are fixed, and that a_i are known to lie in given ellipsoids:

$$a_i \in \mathcal{E}_i = \{\bar{a}_i + P_i u \mid \|u\|_2 \leq 1\}$$

where $P_i \in \mathbf{R}^{n \times n}$. (If P_i is singular we obtain 'flat' ellipsoids, of dimension rank P_i; $P_i = 0$ means that a_i is known perfectly.)

We will require that the constraints be satisfied for all possible values of the parameters a_i, which leads us to the robust linear program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i \text{ for all } a_i \in \mathcal{E}_i, \quad i = 1, \ldots, m. \end{array}$$

The robust linear constraint, $a_i^T x \leq b_i$ for all $a_i \in \mathcal{E}_i$, can be expressed as

$$\sup \left\{ a_i^T x \mid a_i \in \mathcal{E}_i \right\} \leq b_i$$

the lefthand side of which can be expressed as

$$\begin{aligned} \sup \left\{ a_i^T x \mid a_i \in \mathcal{E}_i \right\} &= \bar{a}_i^T x + \sup \left\{ u^T P_i^T x \mid \|u\|_2 \leq 1 \right\} \\ &= \bar{a}_i^T x + \left\| P_i^T x \right\|_2 \end{aligned}$$

Thus, the robust linear constraint can be expressed as

$$\bar{a}_i^T x + \left\| P_i^T x \right\|_2 \leq b_i,$$

which is evidently a second-order cone constraint. Hence the robust LP (4.37) can be expressed as the SOCP

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \bar{a}_i^T x + \left\| P_i^T x \right\|_2 \leq b_i, \quad i = 1, \ldots, m \end{array}$$

Note that the additional norm terms act as regularization terms; they prevent x from being large in directions with considerable uncertainty in the parameters a_i.

3.2 Linear programming with random constraints - stolen from Boyd

The robust LP described above can also be considered in a statistical framework. Here we suppose that the parameters a_i are independent Gaussian random vectors, with mean $\bar{a}_i$ and covariance Σ_i. We require that each constraint $a_i^T x \leq b_i$ should hold with a probability (or confidence) exceeding η, where $\eta \geq 0.5$, i.e.,

$$\text{prob}\left(a_i^T x \leq b_i\right) \geq \eta$$

We will show that this probability constraint can be expressed as a second-order cone constraint.

Letting $u = a_i^T x$, with σ^2 denoting its variance, this constraint can be written as

$$\text{prob}\left(\frac{u-\bar{u}}{\sigma} \leq \frac{b_i-\bar{u}}{\sigma}\right) \geq \eta.$$

Since $(u-\bar{u})/\sigma$ is a zero mean unit variance Gaussian variable, the probability above is simply $\Phi\left((b_i-\bar{u})/\sigma\right)$, where

$$\Phi(z) = \frac{1}{\sqrt{2\pi}}\text{int}_{-\infty}^{z} e^{-t^2/2} dt$$

is the cumulative distribution function of a zero mean unit variance Gaussian random variable. Thus the probability constraint (4.38) can be expressed as

$$\frac{b_i-\bar{u}}{\sigma} \geq \Phi^{-1}(\eta)$$

or, equivalently,

$$\bar{u} + \Phi^{-1}(\eta)\sigma \leq b_i$$

From $\bar{u} = \bar{a}_i^T x$ and $\sigma = \left(x^T \Sigma_i x\right)^{1/2}$ we obtain

$$\bar{a}_i^T x + \Phi^{-1}(\eta)\left\|\Sigma_i^{1/2} x\right\|_2 \leq b_i.$$

By our assumption that $\eta \geq 1/2$, we have $\Phi^{-1}(\eta) \geq 0$, so this constraint is a second-order cone constraint. In summary, the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \text{prob}\left(a_i^T x \leq b_i\right) \geq \eta, \quad i = 1, \ldots, m \end{array}$$

can be expressed as the SOCP

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \bar{a}_i^T x + \Phi^{-1}(\eta)\left\|\Sigma_i^{1/2} x\right\|_2 \leq b_i, \quad i = 1, \ldots, m \end{array}$$

3.3 Forecasting Demand with SARIMA Models

Forecasting future demand is an essential task for businesses of all sizes, as it can help optimize production, inventory, and staffing levels. Time series forecasting is a popular method for demand forecasting, as it can capture the patterns and trends in the data over time.

SARIMA (Seasonal Autoregressive Integrated Moving Average) models are a class of time series models that can be used to capture the autocorrelation and seasonality in the data. SARIMA models have four main parameters: the autoregressive order (p), the differencing order (d), the moving average order (q), and the seasonal order (P, D, Q, s).

To fit a SARIMA model to a time series of demand data, we can use the ‘SARIMAX‘ function from the ‘statsmodels‘ package in Python. Once we have fit a SARIMA model to the data, we can use the ‘forecast‘ method to generate point forecasts for future periods.

To generate a distribution of demand for each future period, we can assume that the forecast errors are normally distributed and use the standard deviation of the forecast errors to generate a normal distribution around each point forecast. We can then sample from these distributions using the ‘np.random.normal‘ function to generate a 12-tuple of random variables representing the sampled demand for each month in the future.

However, this approach assumes that the demand for each future period is an independent random variable. If the demand for each period is dependent on the demand in previous periods or on external factors, we may need to use a multivariate time series model such as vector autoregression (VAR) or dynamic linear regression (DLR) to capture these dependencies.

```
Date,Sales
2018-01-01, 1500
2018-02-01, 2100
2018-03-01, 1800
2018-04-01, 2200
2018-05-01, 2400
2018-06-01, 2600
2018-07-01, 2800
2018-08-01, 3200
2018-09-01, 2800
2018-10-01, 3300
2018-11-01, 3900
2018-12-01, 4500
2019-01-01, 1700
2019-02-01, 2000
2019-03-01, 2200
2019-04-01, 2500
2019-05-01, 2700
2019-06-01, 2900
```

```
2019-07-01, 3200
2019-08-01, 3600
2019-09-01, 3300
2019-10-01, 3800
2019-11-01, 4200
2019-12-01, 4900
2020-01-01, 1800
2020-02-01, 2200
2020-03-01, 2400
2020-04-01, 2700
2020-05-01, 2900
2020-06-01, 3100
2020-07-01, 3600
2020-08-01, 3800
2020-09-01, 3500
2020-10-01, 4000
2020-11-01, 4400
2020-12-01, 5200
2021-01-01, 2000
2021-02-01, 2300
2021-03-01, 2600
2021-04-01, 2800
2021-05-01, 3100
2021-06-01, 3300
2021-07-01, 3800
2021-08-01, 4100
2021-09-01, 3800
2021-10-01, 4400
2021-11-01, 4800
2021-12-01, 5700
2022-01-01, 2200
2022-02-01, 2400
2022-03-01, 2800
2022-04-01, 3100
2022-05-01, 3500
2022-06-01, 3700
2022-07-01, 4100
2022-08-01, 4600
2022-09-01, 4200
2022-10-01, 4800
2022-11-01, 5100
2022-12-01, 6100
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Load sales data into a pandas dataframe
sales_data = pd.read_csv('sales_data.csv', index_col='Date', parse_dates=True)

# Check the data and make sure it's in the right format
print(sales_data.head())
print(sales_data.info())

# Fit a SARIMA model to the sales data
model = SARIMAX(sales_data, order=(1,1,1), seasonal_order=(1,1,1,12))
model_fit = model.fit()

# Get the residuals of the model on the training data
residuals = model_fit.resid

# Generate point forecasts for the next 12 months
forecast = model_fit.forecast(steps=12)

# Combine the residuals with the point forecasts to obtain the forecasted sales for the nex
forecasted_sales = np.concatenate((sales_data['Sales'].values[-12:], forecast))

# Calculate the standard deviation of the forecast errors
forecast_errors = forecasted_sales[-12:] - sales_data['Sales'].values[-12:]
stddev = np.sqrt(np.mean(forecast_errors ** 2))

# Sample demand for each month in 2023 from the forecast distribution 100 times
demands_2023_samples = []
for i in range(100):
    forecast_distributions = [np.random.normal(forecast[i], stddev) for i in range(12)]
    demands_2023_samples.append([np.random.normal(forecast_distributions[i], stddev) for i

# Plot all samples on the same graph
plt.figure(figsize=(10, 6))
plt.title('Sampled Demand for Each Month in 2023')
plt.xlabel('Month')
plt.ylabel('Demand')
for i in range(100):
    plt.plot(demands_2023_samples[i], alpha=0.2)
plt.show()
```

3.4 Forecasting Demand with SARIMA Models

Forecasting future demand is an essential task for businesses of all sizes, as it can help optimize production, inventory, and staffing levels. Time series forecasting is a popular method for demand forecasting, as it can capture the patterns and trends in the data over time.

SARIMA (Seasonal Autoregressive Integrated Moving Average) models are a class of time series models that can be used to capture the autocorrelation and seasonality in the data. SARIMA models have four main parameters: the autoregressive order (p), the differencing order (d), the moving average order (q), and the seasonal order (P, D, Q, s).

A SARIMA(p,d,q)(P,D,Q)s model is defined as:

$$\phi_p(B)\Phi_P(B^s)(1-B)^d(1-B^s)^D y_t = \theta_q(B)\Theta_Q(B^s)\varepsilon_t$$

where y_t is the time series, B is the backshift operator, ε_t is white noise with zero mean and variance σ^2, and $\phi_p(B)$, $\theta_q(B)$, $\Phi_P(B^s)$, and $\Theta_Q(B^s)$ are polynomials in B and B^s with the following general forms:

$$\begin{aligned}\phi_p(B) &= 1-\phi_1 B-\phi_2 B^2-\cdots-\phi_p B^p\\ \theta_q(B) &= 1+\theta_1 B+\theta_2 B^2+\cdots+\theta_q B^q\\ \Phi_P(B^s) &= 1-\Phi_1 B^s-\Phi_2 B^{2s}-\cdots-\Phi_P B^{Ps}\\ \Theta_Q(B^s) &= 1+\Theta_1 B^s+\Theta_2 B^{2s}+\cdots+\Theta_Q B^{Qs}\end{aligned}$$

The parameters p, d, and q are the orders of the autoregressive, differencing, and moving average components, respectively. The parameters P, D, Q, and s are the orders of the seasonal autoregressive, seasonal differencing, seasonal moving average, and the length of the seasonal cycle, respectively.

To fit a SARIMA model to a time series of demand data, we can use the ‘SARIMAX‘ function from the ‘statsmodels‘ package in Python. Once we have fit a SARIMA model to the data, we can use the ‘forecast‘ method to generate point forecasts for future periods.

To generate a distribution of demand for each future period, we can assume that the forecast errors are normally distributed and use the standard deviation of the forecast errors to generate a normal distribution around each point forecast. We can then sample from these distributions using the ‘np.random.normal‘ function to generate a 12-tuple of random variables representing the sampled demand for each month in the future.

However, this approach assumes that the demand for each future period is an independent random variable. If the demand for each period is dependent on the demand in previous periods or on external factors, we may need to use a multivariate time series model such as vector autoregression (VAR) or dynamic linear regression (DLR) to capture these dependencies.

3.5 Inventory Problem

Let d_t be the forecasted demand for month t, and let c_t be the cost of ordering inventory in month t. We can define the following decision variables:

x_t: an integer variable that represents the amount of inventory ordered in month t The objective is to minimize the total cost of inventory over the planning horizon, which is defined as:

$$\text{minimize} \quad \sum_{t=1}^{12} c_t x_t$$

The constraints are as follows:

Inventory balance constraint: the amount of inventory on hand at the end of month t is equal to the amount of inventory on hand at the end of month $t-1$, plus the amount of inventory ordered in month t, minus the demand in month t. This can be expressed as:

$$x_t - d_t + y_{t-1} = y_t, \quad \forall t$$

where y_0 is the starting inventory level and y_t is the inventory level at the end of month t.

Non-negativity constraint: the inventory ordered in a given month must be non-negative. This can be expressed as:

$$x_t \geq 0, \quad \forall t$$

Maximum inventory constraint: the amount of inventory on hand at the end of each month must be non-negative and cannot exceed a maximum inventory level M. This can be expressed as:

$$0 \leq y_t \leq M, \quad \forall t$$

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

# Define parameters
M = 1000  # maximum inventory level
T = 12  # number of months in planning horizon
c = np.random.rand(T)  # cost of ordering inventory in each month
d = np.array([100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650])  # forecasted de
y0 = 0  # initial inventory level

# Create model
m = gp.Model()
```

```
# Add decision variables
x = m.addVars(T, vtype=GRB.INTEGER, name="x")
y = m.addVars(T, lb=0, ub=M, name="y")

# Set objective function
m.setObjective(gp.quicksum(c[t] * x[t] for t in range(T)), GRB.MINIMIZE)

# Add constraints
for t in range(T):
    if t == 0:
        m.addConstr(y[t] == y0 + x[t] - d[t])
    else:
        m.addConstr(y[t] == y[t-1] + x[t] - d[t])
    m.addConstr(x[t] >= 0)

# Set Gurobi parameters
m.setParam("OutputFlag", 0)

# Optimize model
m.optimize()

# Print solution
print("Optimal objective value:", m.objVal)
for t in range(T):
    print("Order", int(x[t].x), "units in month", t)
    print("Ending inventory level in month", t, "is", int(y[t].x))
```

Stochastic integer program for inventory management

Suppose we want to manage inventory over a planning horizon of T months, where the demand for each month is uncertain and follows a probability distribution. Let d_t be the random demand for month t, and let c_t be the cost of ordering inventory in month t. We also assume that there is a cost c_h for holding inventory, which is incurred for each unit of inventory held in each month.

We can model this problem as a stochastic integer program, where we seek to minimize the expected total cost of inventory over the planning horizon. We can define the following decision variables:

- x_t: an integer variable that represents the amount of inventory ordered in month t - $y_{t,k}$: a non-negative variable that represents the amount of inventory on hand at the end of month t for demand scenario k

The objective is to minimize the total cost of inventory over the planning horizon, which is defined as:

$$\text{minimize} \quad \sum_{t=1}^{T} c_t x_t + \frac{1}{100} \sum_{t=1}^{T} \sum_{k=1}^{100} c_h y_{t,k}$$

The first term represents the cost of ordering inventory, and the second term represents the expected cost of holding inventory. The factor of $1/100$ in the second term accounts for the fact that we are averaging over 100 demand scenarios.

The constraints are as follows:

- Inventory balance constraint: for each demand scenario k, the amount of inventory on hand at the end of month t is equal to the amount of inventory on hand at the end of month $t-1$, plus the amount of inventory ordered in month t, minus the demand in month t. This can be expressed as:

$$y_{t,k} = \begin{cases} y_0 + x_t - d_t & \text{if } t = 1 \\ y_{t-1,k} + x_t - d_t & \text{if } t > 1 \end{cases} \quad \forall t, k$$

where y_0 is the starting inventory level and $y_{T,k}$ is the ending inventory level for demand scenario k.

- Non-negativity constraint: the inventory ordered in a given month must be non-negative. This can be expressed as:

$$x_t \geq 0, \quad \forall t$$

- Maximum inventory constraint: the amount of inventory on hand at the end of each month for each demand scenario must be non-negative and cannot exceed a maximum inventory level M. This can be expressed as:

$$0 \leq y_{t,k} \leq M, \quad \forall t, k$$

We can solve this stochastic integer program using Gurobi by sampling 100 demand scenarios from the forecast distribution and solving the optimization problem using scenario generation. In practice, more sophisticated methods such as dynamic programming or robust optimization may be needed to handle more complex inventory models.

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

# Define parameters
M = 1000  # maximum inventory level
T = 12  # number of months in planning horizon
c_order = np.random.rand(T)  # cost of ordering inventory in each month
c_hold = 1  # cost of holding inventory per unit per month
y0 = 0  # initial inventory level
```

```
# Sample demand from the forecast distribution 100 times
np.random.seed(0)  # for reproducibility
d_samples = [np.random.normal(mu, sigma, size=(T, 100)) for mu, sigma in forecast_distribut

# Create model
m = gp.Model()

# Add decision variables
x = m.addVars(T, vtype=GRB.INTEGER, name="x")
y = m.addVars(T, 100, lb=0, ub=M, name="y")
d = m.addVars(T, 100, lb=0, name="d")

# Set objective function
m.setObjective(gp.quicksum(c_order[t] * x[t] for t in range(T)) +
               gp.quicksum(gp.quicksum(c_hold * y[t, k] for k in range(100)) for t in range

# Add constraints for each demand scenario
for k in range(100):
    for t in range(T):
        if t == 0:
            m.addConstr(y[t, k] == y0 + x[t] - d[t, k])
        else:
            m.addConstr(y[t, k] == y[t-1, k] + x[t] - d[t, k])
        m.addConstr(d[t, k] >= 0)
    m.addConstrs((x[t] >= 0 for t in range(T)))

# Set Gurobi parameters
m.setParam("OutputFlag", 0)

# Optimize model
m.optimize()

# Print solution
print("Optimal objective value:", m.objVal)
for t in range(T):
    print("Order", int(x[t].x), "units in month", t)
    print("Average ending inventory level in month", t, "is", int(np.mean([y[t, k].x for k
```

Bender's decomposition for stochastic integer programming

Bender's decomposition is a classical algorithm for solving large-scale stochastic optimization problems. The algorithm decomposes the original problem into a master problem and subproblems, each of which is solved iteratively until convergence.

In the case of a stochastic integer programming problem, Bender's decomposition works by decomposing the problem into two levels. The first level, called the master problem, includes the deterministic decision variables and a relaxation of the stochastic variables. The second level, called the subproblem, includes the stochastic variables and their associated constraints. The algorithm iteratively solves the subproblem for each scenario, and generates cuts that are added to the master problem to ensure that the relaxation is tightened.

More specifically, Bender's decomposition works as follows:

1. Solve the master problem without considering the stochastic variables. 2. For each scenario, solve the subproblem using the current solution to the master problem. 3. If the current solution violates the stochastic constraints for a scenario, generate a cut that is added to the master problem to ensure that the relaxation is tightened. 4. Return to step 1 and repeat until the solution to the master problem is feasible for all scenarios.

Bender's decomposition can be particularly effective for large-scale stochastic optimization problems, as it reduces the size of the problem that needs to be solved at each iteration. In addition, the use of cuts can significantly reduce the number of iterations needed to reach convergence.

In practice, the performance of Bender's decomposition depends on the structure of the problem, the number of scenarios, and the quality of the cuts generated. Other solution methods, such as stochastic dual dynamic programming, may be more effective for some problems.

Bender's decomposition for the stochastic inventory problem

In the stochastic inventory problem, the objective is to determine the inventory levels and order quantities that minimize the expected cost of ordering and holding inventory over a finite time horizon. The problem includes a set of deterministic decision variables, such as the order quantities, as well as a set of stochastic variables, such as the demand in each time period.

To apply Bender's decomposition to this problem, we first decompose the problem into a master problem and subproblems. The master problem includes the deterministic decision variables and a relaxation of the stochastic variables, while the subproblem includes the stochastic variables and their associated constraints.

In this particular case, we can define the master problem as follows:

1. Define the decision variables x_t, which represent the order quantity in each time period t. 2. Define the decision variables $y_{t,k}$, which represent the ending inventory level at the end of time period t for each demand scenario k. 3. Define the objective function as the expected cost of ordering and holding inventory

over the time horizon, given the expected demand scenario. 4. Define constraints that enforce the inventory balance for each demand scenario, using the decision variables x_t and $y_{t,k}$.

The subproblem, on the other hand, can be defined as follows:

1. Define the decision variables $d_{t,k}$, which represent the demand in each time period t for each demand scenario k. 2. Define the objective function as the negative ending inventory level in the subproblem. 3. Define constraints that enforce the inventory balance for each demand scenario, using the decision variables x_t, $y_{t,k}$, and $d_{t,k}$.

The algorithm then proceeds as follows:

1. Solve the master problem without considering the stochastic variables. 2. For each scenario, solve the subproblem using the current solution to the master problem. 3. If the current solution violates the stochastic constraints for a scenario, generate a cut that is added to the master problem to ensure that the relaxation is tightened. 4. Return to step 1 and repeat until the solution to the master problem is feasible for all scenarios.

In this particular problem, the cuts generated in step 3 can be expressed as constraints that enforce the total amount of inventory ordered to be at least 1 over all demand scenarios. These cuts ensure that the inventory level is not too low in any scenario, and therefore the solution to the master problem is feasible for all scenarios.

Bender's decomposition can be particularly effective for this problem, as it allows us to decompose the problem into a deterministic master problem and a stochastic subproblem. The use of cuts can significantly reduce the number of iterations needed to reach convergence, and can provide a more efficient solution method for large-scale problems with demand uncertainty.

In practice, the performance of Bender's decomposition for this problem depends on the structure of the problem, the number of scenarios, and the quality of the cuts generated. By solving the master problem and subproblems iteratively, we can obtain a more accurate and efficient solution to the stochastic inventory problem.

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

# Define parameters
M = 1000  # maximum inventory level
T = 12  # number of months in planning horizon
c_order = np.random.rand(T)  # cost of ordering inventory in each month
c_hold = 1  # cost of holding inventory per unit per month
y0 = 0  # initial inventory level

# Sample demand from the forecast distribution 100 times
np.random.seed(0)  # for reproducibility
d_samples = [np.random.normal(mu, sigma, size=(T, 100)) for mu, sigma in forecast_distribut
```

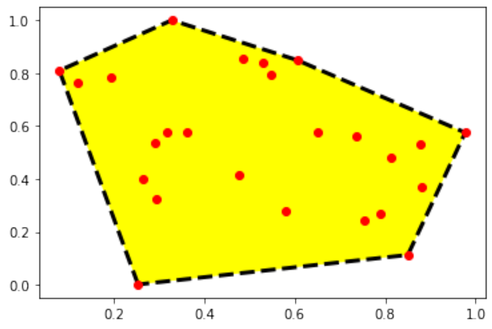
```
# Create master problem
m_master = gp.Model()
x = m_master.addVars(T, vtype=GRB.INTEGER, name="x")
y_avg = m_master.addVars(T, lb=0, ub=M, name="y_avg")
m_master.setObjective(gp.quicksum(c_order[t] * x[t] for t in range(T)) +
                      gp.quicksum(c_hold * y_avg[t] for t in range(T)), GRB.MINIMIZE)
for t in range(T):
    m_master.addConstr(y_avg[t] == (1/100) * gp.quicksum(y[t, k] for k in range(100)))

# Create subproblem
m_sub = gp.Model()
d = m_sub.addVars(T, lb=0, name="d")
m_sub.setObjective(gp.quicksum(-1 * y[t, k] for t in range(T) for k in range(100)), GRB.MAX
for t in range(T):
    m_sub.addConstr(y[t, k] == y0 + gp.quicksum(x[i] for i in range(t+1)) - gp.quicksum(d[i
m_sub.Params.OutputFlag = 0

# Define Bender's cuts
cuts = []

# Set Gurobi parameters
m_master.Params.LazyConstraints = 1
m_master.Params.OutputFlag = 0

# Define callback function for adding Bender's cuts
def callback(model, where):
    if where == GRB.Callback.MIPSOL:
        y_vals = model.cbGetSolution(y)
        for k in range(100):
            d_vals = [model.cbGetSolution(d[t, k]) for t in range(T)]
            m_sub.setObjective(gp.quicksum(-1 * y_vals[t, k] for t in range(T)), GRB.MAXIMI
            m_sub.optimize()
            if m_sub.objVal > -1e-6:
                cuts.append(master.addConstr(gp.quicksum(d_vals[t] for t in range(T)) >= 1)

# Optimize master problem with Bender's decomposition
m_master.optimize(callback)

# Print solution
print("Optimal objective value:", m_master.objVal)
for t in range(T):
    print("Order", int(x[t].x), "units in month", t)
    print("Average ending inventory level in month", t, "is", int(y_avg[t].x))
```

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-random[1]

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

lyryx
advancing learning [1]

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

[1] This 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. Fuhriman
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