Operations Research

Lecture Notes

Master M1 — 2025–2026

Yaë Ulrich Gaba

“To optimize is to simplify without losing anything essential.”
— *George Dantzig*

March 25, 2026

Contents

Preface

Operations Research (OR) is a scientific discipline that relies on mathematical, statistical, and algorithmic methods to support decision-making. Born during World War II within Allied military headquarters, it has since become an indispensable tool in industry, logistics, finance, telecommunications, and many other fields.

Course objectives

This course is designed for second- and third-year undergraduate students (L2–L3) in mathematics, computer science, or engineering. Upon completion, students will be able to:

- Formulate real-world problems as mathematical programs (linear, integer, nonlinear).
- Solve these programs using fundamental algorithms (simplex, branch-and-bound, shortest path, maximum flow, etc.).
- Interpret results through duality and sensitivity analysis.
- Use software tools (Python, PuLP, SciPy, NetworkX) to solve realistically sized problems.
- Understand the principles of discrete-event simulation and queueing theory.

Organization of the book

The book is organized into eleven chapters, designed to be studied sequentially:

Chapter 1 — Introduction to OR. History, definitions, and general modelling methodology.

Chapter 2 — Linear Programming: Formulation and Geometry. Building linear models, geometric interpretation, feasible regions, and optimal vertices.

Chapter 3 — The Simplex Method. The simplex algorithm in tableau form, initialization via the Big-M method and the two-phase method.

Chapter 4 — LP Duality. Constructing the dual, duality theorems, economic interpretation.

Chapter 5 — Sensitivity Analysis. Coefficient ranging, parametric analysis, shadow price interpretation.

Chapter 6 — Transportation and Assignment Problems. Balanced and unbalanced transportation models, stepping-stone method, assignment problem and the Hungarian algorithm.

Chapter 7 — Applied Graph Theory. Shortest paths (Dijkstra, Bellman-Ford), minimum spanning trees (Kruskal, Prim).

Chapter 8 — Network Flows. Maximum flow (Ford-Fulkerson), minimum cut, minimum-cost flow.

Chapter 9 — Integer Programming. Branch-and-bound, cutting planes, linear relaxation.

Chapter 10 — Nonlinear Programming. Karush-Kuhn-Tucker conditions, convex programming, gradient methods.

Chapter 11 — Simulation and Queueing Theory. Discrete-event simulation, M/M/1 and M/M/c models, Little's law.

Prerequisites

The required background includes:

- **Linear algebra**: matrices, systems of linear equations, bases, and rank.
- **Calculus**: functions of several variables, partial derivatives, convexity.
- **Probability**: standard distributions, expectation, Poisson processes (for Chapter 11).
- **Algorithms**: basic notions of complexity and Python programming.

Conventions and notation

Throughout this book we use the following conventions:

Notation	Meaning
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^n$	Euclidean space of dimension n
$\mathbb{Z}$	Set of integers
$\mathbb{N}$	Set of natural numbers
$\mathbf{x}$	Column vector in $\mathbb{R}^n$
$A \in \mathbb{R}^{m\times n}$	Real $m \times n$ matrix
$\mathbf{c}^\top\mathbf{x}$	Inner (dot) product
$\|\mathbf{x}\|$	Euclidean norm
$\|S\|$	Cardinality of the set S
$\arg\min$, $\arg\max$	Argument of the minimum / maximum

Theorems, definitions, and propositions are numbered by chapter. Examples are highlighted in boxes, and exercises appear at the end of each chapter.

Remark 0.1. **Remark** boxes provide additional insights or caveats. **Best practice** boxes offer methodological advice for modelling and implementation.

Software used

Numerical examples are implemented in **Python 3** using the following libraries:

- `scipy.optimize.linprog` — solving linear programs;
- `PuLP` — modelling and solving LP/ILP;
- `NetworkX` — graphs, shortest paths, flows;
- `SimPy` — discrete-event simulation;
- `NumPy`, `Matplotlib` — numerical computing and visualization.

Installing dependencies

```python
# pip install numpy scipy matplotlib pulp networkx simpy
import numpy as np
from scipy.optimize import linprog
import pulp
import networkx as nx
```

How to use this course

Each chapter follows a recurring structure:

1. **Motivation** — a concrete problem that introduces the theoretical need.

2. **Theory** — definitions, theorems, and proofs.
3. **Algorithms** — formal description and worked example.
4. **Implementation** — commented Python code.
5. **Exercises** — problems of increasing difficulty ($\star$ to $\star\star\star$).

Active learning

Operations research is best learned by *doing*. We strongly recommend:

- Working through examples by hand before consulting the solution.
- Implementing algorithms in Python.
- Attempting exercises before looking at solutions.
- Exploring variants of the proposed problems.

Brief history

- **1937**: Kantorovich formulates the first linear programming problems for economic planning.
- **1939–1945**: British and American OR teams optimize military operations (convoys, radar, bombing campaigns).
- **1947**: George Dantzig publishes the simplex method.
- **1951**: Kuhn and Tucker state optimality conditions for nonlinear programming.
- **1956**: Ford and Fulkerson propose the maximum-flow algorithm.
- **1958**: Gomory introduces cutting planes for integer programming.
- **1960**: Land and Doig formalize branch-and-bound.
- **1979**: Khachiyan shows LP is solvable in polynomial time (ellipsoid method).
- **1984**: Karmarkar publishes the interior-point method.
- **1990s–2020s**: Development of commercial solvers (CPLEX, Gurobi) and open-source solvers (GLPK, CBC); widespread adoption in industry.

Main references

1. Hillier F.S., Lieberman G.J., *Introduction to Operations Research*, McGraw-Hill, 11th ed., 2021.
2. Bertsimas D., Tsitsiklis J.N., *Introduction to Linear Optimization*, Athena Scientific, 1997.

3. Wolsey L.A., *Integer Programming*, Wiley, 2nd ed., 2020.

4. Chvátal V., *Linear Programming*, W.H. Freeman, 1983.

5. Winston W.L., *Operations Research: Applications and Algorithms*, Cengage, 4th ed., 2003.

6. Bazaraa M.S., Jarvis J.J., Sherali H.D., *Linear Programming and Network Flows*, Wiley, 4th ed., 2010.

Acknowledgements

I thank the colleagues and students who contributed to improving this course through their comments and suggestions. Any remaining errors are my sole responsibility.

Happy reading and happy optimizing!

Chapter 1

Introduction to Operations Research

During World War II, the Royal Air Force faced a desperate logistics problem: how to deploy coastal radar stations to detect enemy bombers with a limited number of installations. A group of scientists — physicists, mathematicians, biologists — was assembled to analyze the problem *operationally*, using data, models, and quantitative methods. They called their discipline "operational research," and the results were spectacular: losses dropped, efficiency soared. After the war, these methods migrated to industry, transportation, healthcare, and finance, becoming what we now call *operations research*.

This chapter outlines the discipline: its origins, its methodology, and the major classes of problems it seeks to solve.

1.1 What is Operations Research?

Let us begin by defining the subject of our study precisely.

Definition 1.1 (Operations Research)**. Operations Research** (OR) is the application of scientific methods — mathematical, statistical, and computational — to decision-making in complex systems. Its goal is to determine the best allocation of limited resources to achieve a given objective.

OR is distinguished from other branches of applied mathematics by its focus on *modelling* real-world problems and its constant concern with providing *implementable* solutions.

Remark 1.2. The term "Operations Research" originated in the British military during World War II, where "operations" referred to military operations. The discipline is also known as *Management Science* or *Decision Science* in some contexts.

1.2 A brief history

1.2.1 Military origins (1937–1945)

The roots of OR trace back to **Leonid Kantorovich**'s 1937 work on optimizing production in Soviet industry. However, the discipline truly blossomed during World War II:

- In **1940**, physicist Patrick Blackett led *Blackett's Circus*, an interdisciplinary team tasked with optimizing radar deployment and convoy sizes.

CHAPTER 1. INTRODUCTION TO OPERATIONS RESEARCH

- In the United States, similar teams worked on bombing strategies and Pacific logistics.

1.2.2 The foundational era (1947–1960)

- **1947**: George **Dantzig** invents the **simplex method** for solving linear programs.
- **1951**: Harold **Kuhn** and Albert **Tucker** formalize optimality conditions for nonlinear programming (KKT conditions).
- **1956**: Lester **Ford** and Delbert **Fulkerson** publish the maximum-flow algorithm.
- **1958**: Ralph **Gomory** introduces cutting planes for integer programming.
- **1960**: Ailsa **Land** and Alison **Doig** formalize **branch-and-bound**.

1.2.3 Modern developments

- **1979**: Leonid **Khachiyan** proves that linear programming is solvable in polynomial time (ellipsoid method).
- **1984**: Narendra **Karmarkar** publishes the interior-point method, competitive with the simplex in practice.
- **2000s–2020s**: Modern solvers (CPLEX, Gurobi, CBC) solve problems with millions of variables routinely.

1.3 Application domains

OR is used across a wide range of sectors:

Sector	**Example problems**
Logistics	Vehicle routing, inventory management
Manufacturing	Scheduling, lot sizing
Finance	Portfolio optimization, risk management
Transportation	Network design, bus/train timetabling
Telecommunications	Routing, frequency assignment
Healthcare	Operating room scheduling, bed management
Energy	Unit commitment, power grid management
Aviation	Revenue management, gate assignment

1.4 The OR methodology

The general OR approach follows a six-step cycle:

1. **Problem definition**: identify the objective, the decision-makers, and operational constraints.

2. **Data collection**: model parameters, costs, capacities, demands.
3. **Model formulation**: translate the problem into mathematical terms (variables, objective, constraints).
4. **Solution**: apply an algorithm or solver.
5. **Validation**: verify the model's consistency and the solution's relevance.
6. **Implementation**: deploy the solution and monitor its performance.

Modelling means simplifying

A model is never an exact representation of reality. The art of modelling lies in finding the right balance between *accuracy* (faithful model) and *tractability* (solvable model).

1.5 Classification of optimization models

Definition 1.3 (Optimization problem)**.** An **optimization problem** is formulated generically as:

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \leq 0,\ i = 1,\dots,m, \qquad h_j(\mathbf{x}) = 0,\ j = 1,\dots,p,$$

where f is the **objective function**, $\mathcal{X}$ the domain, g_i the **inequality constraints**, and h_j the **equality constraints**.

Models are classified along several axes:

1.5.1 By the nature of objective and constraints

- **Linear programming (LP)**: f, g_i, h_j are all linear functions of $\mathbf{x}$.
- **Nonlinear programming (NLP)**: at least one function is nonlinear.
- **Quadratic programming**: quadratic objective with linear constraints.

1.5.2 By the nature of variables

- **Continuous variables**: $\mathbf{x} \in \mathbb{R}^n$.
- **Integer variables**: $\mathbf{x} \in \mathbb{Z}^n$ (integer programming, IP).
- **Mixed variables**: some continuous, some integer (mixed-integer programming, MIP).
- **Binary variables**: $x_i \in \{0, 1\}$ (0–1 programming).

CHAPTER 1. INTRODUCTION TO OPERATIONS RESEARCH

1.5.3 By information availability

- **Deterministic**: all parameters are known with certainty.
- **Stochastic**: some parameters are random variables.
- **Robust**: we seek a solution that performs well across all possible scenarios.

1.6 First modelling examples

Example 1.4 (Production problem)**.** A factory produces two products P_1 and P_2. Each unit of P_1 yields \$5 profit and requires 2 hours on machine A and 1 hour on machine B. Each unit of P_2 yields \$4 profit and requires 1 hour on machine A and 3 hours on machine B. Machine A is available 10 hours per day, machine B 12 hours.

Decision variables: x_1 = units of P_1, x_2 = units of P_2.

Model:

$$\begin{aligned} \max \quad & z = 5x_1 + 4x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \le 10 \quad \text{(machine A)} \\ & x_1 + 3x_2 \le 12 \quad \text{(machine B)} \\ & x_1, x_2 \ge 0 \end{aligned}$$

Example 1.5 (Blending problem)**.** A farmer wants to prepare an animal feed mix at minimum cost. Two ingredients are available:

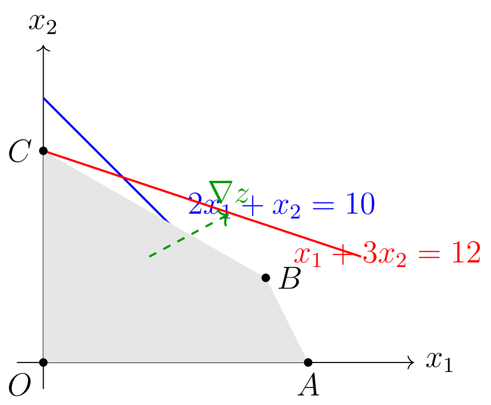
	Protein (%/kg)	Fat (%/kg)	Cost (\$/kg)
Ingredient 1	20	5	3
Ingredient 2	10	8	2
Minimum requirement	15	6	—

Let x_1, x_2 be the quantities (in kg) of the two ingredients. The mix must weigh 1 kg: $x_1 + x_2 = 1$.

$$\begin{aligned} \min \quad & z = 3x_1 + 2x_2 \\ \text{s.t.} \quad & 20x_1 + 10x_2 \ge 15 \quad \text{(protein)} \\ & 5x_1 + 8x_2 \ge 6 \qquad \text{(fat)} \\ & x_1 + x_2 = 1 \\ & x_1, x_2 \ge 0 \end{aligned}$$

1.7 Graphical solution (preview)

For problems with two variables, we can represent the feasible set in the plane and find the optimum graphically.



Vertex $B = (4.2, 1.6)$ maximizes $z = 5(4.2) + 4(1.6) = 27.4$. We will confirm this result formally in Chapter 3 using the simplex method.

1.8 Introduction to solvers

Solving with SciPy

```python
from scipy.optimize import linprog

# min c^T x  =>  negate for maximization
c = [-5, -4]  # max 5x1 + 4x2

# Constraints: A_ub @ x <= b_ub
A_ub = [[2, 1],    # 2x1 + x2 <= 10
        [1, 3]]    # x1 + 3x2 <= 12
b_ub = [10, 12]

# Bounds: x1, x2 >= 0
bounds = [(0, None), (0, None)]

result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds,
                 method='highs')
print(f"x* = {result.x}")
print(f"z* = {-result.fun:.2f}")
```

Output

```
x* = [4.2 1.6] z* = 27.40
```

Solving with PuLP

```python
import pulp

prob = pulp.LpProblem("Production", pulp.LpMaximize)

x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
```

CHAPTER 1. INTRODUCTION TO OPERATIONS RESEARCH

```python
# Objective
prob += 5*x1 + 4*x2, "Profit"

# Constraints
prob += 2*x1 + x2 <= 10, "Machine_A"
prob += x1 + 3*x2 <= 12, "Machine_B"

prob.solve(pulp.PULP_CBC_CMD(msg=0))

print(f"Status: {pulp.LpStatus[prob.status]}")
print(f"x1 = {x1.varValue}, x2 = {x2.varValue}")
print(f"Profit = {pulp.value(prob.objective):.2f}")
```

Output

```
Status: Optimal x1 = 4.2, x2 = 1.6 Profit = 27.40
```

1.9 Algorithmic complexity: review

Definition 1.6 (Big-$\mathcal{O}$ notation)**.** An algorithm has complexity $\mathcal{O}(g(n))$ if there exist constants $C > 0$ and n_0 such that the number of elementary operations is bounded above by $C \cdot g(n)$ for all $n \geq n_0$.

Class	Complexity	Example
$\mathcal{P}$	Polynomial	Simplex (in practice), Dijkstra
$\mathcal{NP}$	Verifiable in polynomial time	Knapsack problem
$\mathcal{NP}$-complete	As hard as any $\mathcal{NP}$ problem	TSP (decision)
$\mathcal{NP}$-hard	At least as hard as $\mathcal{NP}$-complete	TSP (optimization)

Simplex complexity

Although the simplex method has exponential worst-case complexity (Klee-Minty examples), it is extremely efficient in practice. Interior-point methods offer guaranteed polynomial complexity: $\mathcal{O}(n^{3.5}L)$ where L is the input size.

1.10 Fundamental vocabulary

Definition 1.7 (Feasible solution)**.** A solution $\mathbf{x}$ is called **feasible** (or **admissible**) if it satisfies all the constraints of the problem. The set of feasible solutions is denoted $\mathcal{F}$ (the *feasible set*).

Definition 1.8 (Optimal solution)**.** A feasible solution $\mathbf{x}^*$ is **optimal** if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{F}$ (in the case of minimization).

Definition 1.9 (Unbounded problem)**.** A problem is said to be **unbounded** if the objective function can take arbitrarily small values (minimization) or large values (maximization) over $\mathcal{F}$.

Definition 1.10 (Infeasible problem)**.** A problem is said to be **infeasible** if $\mathcal{F} = \emptyset$, i.e., no solution satisfies all constraints simultaneously.

1.11 Exercises

Exercise 1.1 ($\star$ — Modelling)**.** A company manufactures three products A, B, and C with the following data:

	Machine 1 (h)	Machine 2 (h)	Machine 3 (h)	Profit ($)
A	1	2	1	10
B	2	1	3	15
C	1	1	2	12
Availability	40	50	60	

Formulate the linear program that maximizes total profit.

Exercise 1.2 ($\star$ — Graphical solution)**.** Solve graphically:

$$\begin{aligned} \max \quad & z = 3x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + x_2 \le 4 \\ & x_1 + 3x_2 \le 6 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Exercise 1.3 ($\star\star$ — Diet problem)**.** A dietitian must compose a meal at minimum cost containing at least 2000 kcal, 55 g of protein, and 800 mg of calcium. Four foods are available:

	kcal/100g	Protein (g/100g)	Calcium (mg/100g)	Cost ($/100g)
Bread	265	9	15	0.30
Milk	65	3.5	120	0.15
Cheese	350	25	700	1.20
Apple	52	0.3	6	0.40

Formulate the corresponding LP.

Exercise 1.4 ($\star\star$ — Python code)**.** Implement and solve the problem from Exercise 1 using PuLP. Verify the result with `scipy.optimize.linprog`.

Exercise 1.5 ($\star\star\star$ — Modelling with binary variables)**.** An investor has $100,000 and must choose among 5 investment projects. Each project i requires an investment of c_i and returns r_i. The investor can only invest in a project entirely (no fractions). Moreover, if the investor chooses project 3, then project 1 must also be selected. Formulate this problem as an integer linear program (ILP).

CHAPTER 1. INTRODUCTION TO OPERATIONS RESEARCH

Chapter 2

Linear Programming — Formulation and Geometry

In the summer of 1947, at the Pentagon, a young mathematician named George Dantzig was working on planning problems for the United States Air Force. Resources were scarce, constraints were everywhere, and the objective was clear: do the most with the least. Dantzig formalised this challenge as a *linear program* and invented the simplex algorithm to solve it. Within months, the method was tackling logistics, diet planning, and industrial scheduling problems that had seemed intractable.

What makes linear programming so remarkable is the marriage of algebra and geometry. A linear program defines a convex polyhedron in high-dimensional space, and the optimal solution—if it exists—sits at a vertex of this polyhedron. The simplex algorithm simply walks from vertex to vertex along the edges, improving the objective at each step. This geometric insight, combined with the power of duality theory developed shortly after by John von Neumann, turned linear programming into one of the most widely used tools in all of applied mathematics.

2.1 General form of a linear program

Definition 2.1 (Linear program)**.** A **linear program** (LP) is an optimization problem of the form:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \ge \mathbf{0} \end{aligned}$$

where $\mathbf{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{x} \in \mathbb{R}^n$ is the vector of decision variables.

Remark 2.2. Maximizing $\mathbf{c}^\top\mathbf{x}$ is equivalent to minimizing $(-\mathbf{c})^\top\mathbf{x}$. Hence we can always convert to a minimization problem.

2.2 Equivalent forms

2.2.1 Standard form

Definition 2.3 (Standard form)**.** An LP is in **standard form** if:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

All constraints are equalities and all variables are nonnegative.

2.2.2 Converting to standard form

Conversion rules

1. $\leq$ **inequality**: add a slack variable $s_i \geq 0$:

$$a_{i1}x_1 + \cdots + a_{in}x_n + s_i = b_i$$

2. $\geq$ **inequality**: subtract a surplus variable $s_i \geq 0$:

$$a_{i1}x_1 + \cdots + a_{in}x_n - s_i = b_i$$

3. **Free variable** ($x_j \in \mathbb{R}$): set $x_j = x_j^+ - x_j^-$ with $x_j^+, x_j^- \geq 0$.
4. **Maximization**: $\max \mathbf{c}^\top \mathbf{x} = -\min(-\mathbf{c}^\top \mathbf{x})$.

Example 2.4 (Converting to standard form)**.** Consider:

$$\begin{aligned} \max \quad & 3x_1 + 5x_2 \\ \text{s.t.} \quad & x_1 + 2x_2 \leq 12 \\ & 2x_1 + x_2 \geq 4 \\ & x_1 \geq 0,\ x_2 \text{ free} \end{aligned}$$

Set $x_2 = x_2^+ - x_2^-$ with $x_2^+, x_2^- \geq 0$, and add slack/surplus variables:

$$\begin{aligned} \min \quad & -3x_1 - 5x_2^+ + 5x_2^- \\ \text{s.t.} \quad & x_1 + 2x_2^+ - 2x_2^- + s_1 = 12 \\ & 2x_1 + x_2^+ - x_2^- - s_2 = 4 \\ & x_1, x_2^+, x_2^-, s_1, s_2 \geq 0 \end{aligned}$$

2.3 Canonical form

Definition 2.5 (Canonical form)**.** An LP is in **canonical form** if:

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

(maximization, $\leq$ constraints, nonnegative variables).

2.4 Geometry of linear programming

2.4.1 Hyperplanes and half-spaces

Definition 2.6 (Hyperplane)**.** A **hyperplane** in $\mathbb{R}^n$ is a set of the form:

$$H = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^\top \mathbf{x} = \beta\}$$

where $\mathbf{a} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and $\beta \in \mathbb{R}$.

Definition 2.7 (Half-space)**.** A **half-space** is one of the two sets determined by a hyperplane:

$$H^+ = \{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq \beta\} \qquad \text{or} \qquad H^- = \{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \geq \beta\}$$

2.4.2 Convex sets and polyhedra

Definition 2.8 (Convex set)**.** A set $C \subseteq \mathbb{R}^n$ is **convex** if for all $\mathbf{x}, \mathbf{y} \in C$ and all $\lambda \in [0, 1]$:

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C$$

Definition 2.9 (Polyhedron)**.** A **polyhedron** is the intersection of finitely many half-spaces:

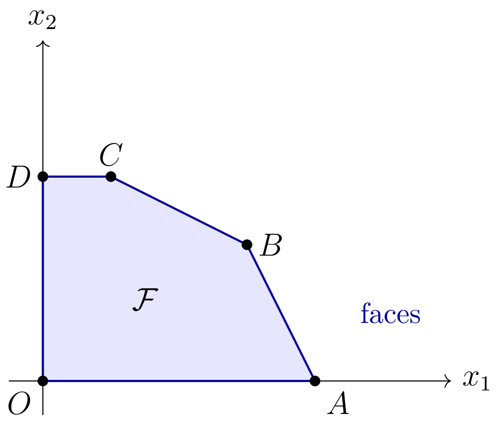
$$P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$$

A bounded polyhedron is called a **polytope**.

Theorem 2.10 (The feasible set is a polyhedron)**.** *The feasible set of a linear program:*

$$\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$$

is a convex polyhedron (possibly empty or unbounded).



2.4.3 Vertices and basic feasible solutions

Definition 2.11 (Vertex)**.** A point $\mathbf{v} \in P$ is a **vertex** (or **extreme point**) of polyhedron P if there do not exist two distinct points $\mathbf{x}, \mathbf{y} \in P$ such that $\mathbf{v} = \frac{1}{2}(\mathbf{x} + \mathbf{y})$.

Definition 2.12 (Basic feasible solution (BFS))**.** Consider the system $A\mathbf{x} = \mathbf{b}$ with $A \in \mathbb{R}^{m\times n}$, $m \leq n$, $\text{rank}(A) = m$. Partition the columns into B (basis, m linearly independent columns) and N (non-basic):

$$\mathbf{x}_B = B^{-1}\mathbf{b}, \qquad \mathbf{x}_N = \mathbf{0}$$

If $\mathbf{x}_B \geq \mathbf{0}$, then $\mathbf{x}$ is a **basic feasible solution**.

Theorem 2.13 (Vertex–BFS equivalence)**.** *Let $P = \{\mathbf{x} : A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0\}$. The following are equivalent:*

1. *$\mathbf{v}$ is a vertex of P.*
2. *$\mathbf{v}$ is a basic feasible solution.*
3. *The columns of A corresponding to the strictly positive components of $\mathbf{v}$ are linearly independent.*

2.5 Fundamental theorem of linear programming

Theorem 2.14 (Fundamental theorem of LP)**.** *Consider the LP $\min\{\mathbf{c}^\top\mathbf{x} : A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0\}$.*

1. *If the feasible set is nonempty, it has at least one vertex.*
2. *If the problem has a finite optimal solution, then there exists an optimal solution that is a vertex of the feasible set.*
3. *The number of vertices is finite (at most $\binom{n}{m}$).*

Why is the optimum at a vertex?

The level sets of $\mathbf{c}^\top\mathbf{x}$ are parallel hyperplanes. We slide these hyperplanes in the direction $-\mathbf{c}$ (for minimization) until they touch the polyhedron. The last contact point is necessarily a vertex (or a face containing vertices).

2.6 Formulation examples

Example 2.15 (Production planning)**.** A company produces n items on m machines over T periods. Let x_{it} be the quantity of item i produced in period t, and s_{it} the inventory at the end of the period.

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{t=1}^{T}(c_{it}x_{it} + h_{it}s_{it}) && \\
\text{s.t.} \quad & s_{i,t-1} + x_{it} - s_{it} = d_{it} && \forall\, i, t \quad \text{(flow balance)} \\
& \sum_{i=1}^{n} a_{ij}x_{it} \leq C_{jt} && \forall\, j, t \quad \text{(machine } j \text{ capacity)} \\
& x_{it}, s_{it} \geq 0 && \forall\, i, t
\end{aligned}$$

where c_{it} is unit production cost, h_{it} holding cost, d_{it} demand, a_{ij} unit machine time.

Example 2.16 (Diet problem (revisited))**.** With the data from the previous chapter:

$$\begin{aligned}
\min \quad & 0.30x_1 + 0.15x_2 + 1.20x_3 + 0.40x_4 \\
\text{s.t.} \quad & 265x_1 + 65x_2 + 350x_3 + 52x_4 \geq 2000 \\
& 9x_1 + 3.5x_2 + 25x_3 + 0.3x_4 \geq 55 \\
& 15x_1 + 120x_2 + 700x_3 + 6x_4 \geq 800 \\
& x_1, x_2, x_3, x_4 \geq 0
\end{aligned}$$

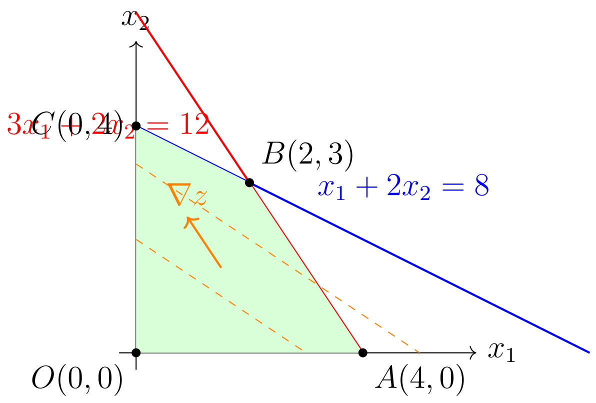
(each x_i is in hectograms, i.e., 100 g).

2.7 Detailed graphical solution

Example 2.17 (Complete graphical solution)**.** Solve:

$$\begin{aligned} \max \quad & z = 2x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + 2x_2 \le 8 \\ & 3x_1 + 2x_2 \le 12 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Step 1: Plot the constraint lines.



Step 2: Evaluate z at each vertex.

Vertex	$z = 2x_1 + 3x_2$
$O(0,0)$	0
$A(4,0)$	8
$B(2,3)$	**13**
$C(0,4)$	12

The optimum is $z^* = 13$ attained at $B = (2,3)$.

2.8 Special cases

Definition 2.18 (Infeasible problem)**.** An LP is **infeasible** if $\mathcal{F} = \emptyset$.

Example 2.19 (Infeasibility)**.**

$$x_1 + x_2 \le 2, \qquad x_1 + x_2 \ge 5, \qquad x_1, x_2 \ge 0$$

The first two constraints are contradictory.

Definition 2.20 (Unbounded problem)**.** An LP is **unbounded** if $z \to +\infty$ (max) or $z \to -\infty$ (min) over $\mathcal{F}$.

Definition 2.21 (Multiple optimal solutions)**.** There are **multiple optimal solutions** when an entire edge of the polyhedron is optimal (the gradient is parallel to a face).

Definition 2.22 (Degeneracy)**.** A BFS is **degenerate** if at least one basic variable equals zero. This occurs when more than n constraints are active at a vertex.

2.9 Python implementation

Visualizing the feasible region

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog

# Solve
c = [-2, -3]
A_ub = [[1, 2], [3, 2]]
b_ub = [8, 12]
res = linprog(c, A_ub=A_ub, b_ub=b_ub,
              bounds=[(0, None), (0, None)], method='highs')
print(f"Optimum: z = {-res.fun:.1f} at x = {res.x}")

# Plot
fig, ax = plt.subplots(figsize=(6, 5))
x1 = np.linspace(0, 5, 300)

ax.plot(x1, (8 - x1)/2, 'b-', label=r'$x_1+2x_2=8$')
ax.plot(x1, (12 - 3*x1)/2, 'r-', label=r'$3x_1+2x_2=12$')
ax.fill([0, 4, 2, 0], [0, 0, 3, 4], alpha=0.2, color='green')

ax.set_xlim(-0.5, 5.5)
ax.set_ylim(-0.5, 5.5)
ax.set_xlabel(r'$x_1$'); ax.set_ylabel(r'$x_2$')
ax.legend(); ax.set_title('Feasible region')
plt.tight_layout()
plt.savefig('feasible_region.pdf')
```

Output

```
Optimum: z = 13.0 at x = [2. 3.]
```

2.10 Exercises

Exercise 2.1 (⋆ — Standard form conversion)**.** Convert the following LP to standard form:

$$\begin{aligned} \max \quad & 4x_1 - 2x_2 + 7x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 \le 20 \\ & 2x_1 - x_2 + 3x_3 \ge 10 \\ & x_1 \ge 0,\ x_2 \ge 0,\ x_3 \text{ free} \end{aligned}$$

Exercise 2.2 ($\star$ — Graphical solution)**.** Solve graphically and identify all vertices:

$$\begin{aligned} \max \quad & z = x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 5 \\ & 2x_1 + x_2 \leq 8 \\ & x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Exercise 2.3 ($\star\star$ — Empty, unbounded, or finite?)**.** For each LP below, determine whether it is infeasible, unbounded, or has a finite optimal solution:

1. $\min x_1 - x_2$ s.t. $x_1 + x_2 \geq 1$, $x_1, x_2 \geq 0$.
2. $\max 2x_1 + x_2$ s.t. $x_1 - x_2 \leq 1$, $x_1, x_2 \geq 0$.
3. $\min x_1$ s.t. $x_1 + x_2 \leq 4$, $x_1 - x_2 \leq 2$, $-x_1 + x_2 \leq 2$, $x_1, x_2 \geq 0$.

Exercise 2.4 ($\star\star$ — Complete formulation)**.** A refinery processes crude oil into three products: gasoline, diesel, and kerosene. Two types of crude (C_1 and C_2) are available. The following table gives yields (%) and costs:

	Gasoline	Diesel	Kerosene	Cost ($/barrel)
C_1	40%	35%	25%	50
C_2	30%	45%	25%	60
Min. demand (barrels)	3000	2000	1500	

Formulate the LP minimizing the total cost of crude oil purchase.

Exercise 2.5 ($\star\star\star$ — Proof)**.** Prove that the intersection of two convex sets is convex. Deduce that every polyhedron is convex.

Chapter 3

The Simplex Method

The geometric idea is luminous: since the optimum of a linear program sits at a vertex of the constraint polyhedron, one need only walk from vertex to vertex, improving the objective at each step. This is exactly what the simplex method does, invented by George Dantzig in 1947. Despite a theoretical worst-case exponential complexity (demonstrated by Klee and Minty in 1972), the simplex remains, in practice, one of the most efficient algorithms ever devised—a fascinating theoretical anomaly that still defies our understanding.

3.1 Principle of the algorithm

The fundamental theorem of LP (Theorem 2.14) states that if an LP has a finite optimum, it is attained at a vertex of the feasible polyhedron. The **simplex method**, due to George Dantzig (1947), traverses adjacent vertices while improving the objective at each iteration.

Simplex overview

1. Start from an initial vertex (BFS).
2. Test optimality (reduced costs).
3. If not optimal, pivot to an improving adjacent vertex.
4. Repeat until optimality or unboundedness is detected.

3.2 Notation and setup

Consider the LP in standard form:

$$\begin{aligned} \min \quad & z = \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0} \end{aligned}$$

with $A \in \mathbb{R}^{m\times n}$, $\operatorname{rank}(A) = m$, $\mathbf{b} \geq \mathbf{0}$.

Partition $A = [B \mid N]$, $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)^\top$, $\mathbf{c} = (\mathbf{c}_B, \mathbf{c}_N)^\top$:

$$B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b} \quad \Rightarrow \quad \mathbf{x}_B = B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N$$

Definition 3.1 (Reduced costs)**.** The **reduced costs** of the non-basic variables are:

$$\bar{c}_j = c_j - \mathbf{c}_B^\top B^{-1}\mathbf{a}_j$$

where $\mathbf{a}_j$ is the j-th column of A.

Theorem 3.2 (Optimality criterion)**.** *A BFS is optimal if and only if all reduced costs are nonnegative: $\bar{c}_j \geq 0$ for all non-basic j.*

3.3 The simplex tableau

The simplex tableau organizes all the necessary information:

$$\begin{array}{c|cccc|c}
 & x_1 & x_2 & \cdots & x_n & \\
\hline
z & \bar{c}_1 & \bar{c}_2 & \cdots & \bar{c}_n & -z \\
\hline
x_{B_1} & \bar{a}_{11} & \bar{a}_{12} & \cdots & \bar{a}_{1n} & \bar{b}_1 \\
x_{B_2} & \bar{a}_{21} & \bar{a}_{22} & \cdots & \bar{a}_{2n} & \bar{b}_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
x_{B_m} & \bar{a}_{m1} & \bar{a}_{m2} & \cdots & \bar{a}_{mn} & \bar{b}_m
\end{array}$$

where $\bar{A} = B^{-1}A$, $\bar{\mathbf{b}} = B^{-1}\mathbf{b}$.

3.4 A simplex iteration

One simplex iteration (minimization)

1. **Optimality test**: if $\bar{c}_j \geq 0$ for all j, STOP (optimal solution found).
2. **Entering variable**: choose $k = \arg\min_j \bar{c}_j$ (most negative reduced cost).
3. **Unboundedness test**: if $\bar{a}_{ik} \leq 0$ for all i, STOP (problem is unbounded).
4. **Leaving variable**: apply the minimum ratio rule:

$$r = \arg\min_{i:\bar{a}_{ik}>0} \frac{\bar{b}_i}{\bar{a}_{ik}}$$

5. **Pivot**: the pivot element is $\bar{a}_{rk}$. Perform Gauss-Jordan elimination to update the tableau.

Ratio rule

Only divide by **strictly positive** coefficients $\bar{a}_{ik} > 0$. If all are ≤ 0, the problem is unbounded: x_k can be increased indefinitely.

3.5 Complete example

Example 3.3 (Simplex step by step)**.** Solve:

$$\begin{aligned} \max \quad & z = 5x_1 + 4x_2 \\ \text{s.t.} \quad & 6x_1 + 4x_2 \leq 24 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Standard form: add slack variables $s_1, s_2 \geq 0$ and convert to minimization ($\min -5x_1 - 4x_2$).

Initial tableau:

	x_1	x_2	s_1	s_2	RHS
z	-5	-4	0	0	0
s_1	6	4	1	0	24
s_2	1	2	0	1	6

Iteration 1: Entering variable x_1 ($\bar{c}_1 = -5$). Ratios: $24/6 = 4$, $6/1 = 6$. Pivot: row s_1, column x_1 (pivot $= 6$).

Divide row 1 by 6, then eliminate x_1 from all other rows:

	x_1	x_2	s_1	s_2	RHS
z	0	$-\frac{2}{3}$	$\frac{5}{6}$	0	20
x_1	1	$\frac{2}{3}$	$\frac{1}{6}$	0	4
s_2	0	$\frac{4}{3}$	$-\frac{1}{6}$	1	2

Iteration 2: Entering variable x_2 ($\bar{c}_2 = -2/3$). Ratios: $4/(2/3) = 6$, $2/(4/3) = 3/2$. Pivot: row s_2, column x_2 (pivot $= 4/3$).

	x_1	x_2	s_1	s_2	RHS
z	0	0	$\frac{3}{4}$	$\frac{1}{2}$	21
x_1	1	0	$\frac{1}{4}$	$-\frac{1}{2}$	3
x_2	0	1	$-\frac{1}{8}$	$\frac{3}{4}$	$\frac{3}{2}$

All reduced costs are ≥ 0: optimal!
$x_1^* = 3$, $x_2^* = 3/2$, $z^* = 5(3) + 4(3/2) = 21$.

3.6 Initialization: Big-M method

When the initial tableau does not directly provide a feasible basis ($\geq$ or $=$ constraints), we use **artificial variables**.

Definition 3.4 (Big-M method)**.** For each equality or $\geq$ constraint (after adding a surplus variable), add an artificial variable $a_i \geq 0$ to the objective function with a penalty cost $+M$ (minimization) where M is a large positive number.

Example 3.5 (Big-M)**.**

$$\begin{aligned} \min \quad & z = 2x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 = 4 \\ & x_1 + 2x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

After adding surplus s_1 and artificials a_1, a_2:

$$\begin{aligned} \min \quad & z = 2x_1 + 3x_2 + Ma_1 + Ma_2 \\ \text{s.t.} \quad & x_1 + x_2 + a_1 = 4 \\ & x_1 + 2x_2 - s_1 + a_2 = 6 \\ & x_1, x_2, s_1, a_1, a_2 \geq 0 \end{aligned}$$

If at optimality $a_1 = a_2 = 0$, the solution is feasible for the original LP. Otherwise, the original LP is infeasible.

3.7 Initialization: two-phase method

Two-phase method

Phase I: Solve the auxiliary problem:

$$\min \sum_i a_i \quad \text{s.t.} \quad A\mathbf{x} + I\mathbf{a} = \mathbf{b}, \quad \mathbf{x}, \mathbf{a} \geq 0$$

- If the optimal value is > 0: the original LP is **infeasible**.
- If the optimal value is $= 0$: we obtain a BFS for the original LP.

Phase II: Use the BFS obtained as a starting point to solve the original LP with the original objective function.

Big-M vs. two-phase

- The Big-M method is simpler to implement but can cause numerical issues if M is poorly chosen.
- The two-phase method is numerically more stable and is preferred in professional implementations.

3.8 Degeneracy and cycling

Definition 3.6 (Degeneracy)**.** An iteration is **degenerate** if the minimum ratio is zero: the leaving variable is already zero and z does not change.

Theorem 3.7 (Cycling risk)**.** *In the presence of degeneracy, the simplex method can theoretically* ***cycle*** *(loop indefinitely among the same bases).*

Definition 3.8 (Bland's rule). **Bland's rule** (smallest-index rule) prevents cycling: choose as the entering variable the non-basic variable with the smallest index having a negative reduced cost, and as the leaving variable the basic variable with the smallest index achieving the minimum ratio.

Theorem 3.9 (Bland's anti-cycling guarantee). *Under Bland's rule, the simplex algorithm terminates in a finite number of iterations.*

3.9 Simplex complexity

Proposition 3.10 (Number of vertices). A polyhedron defined by m constraints and n variables has at most $\binom{n}{m}$ vertices.

Remark 3.11. The simplex has $\mathcal{O}(2^n)$ worst-case complexity (Klee-Minty examples, 1972), but in practice it performs roughly $2m$ to $3m$ iterations for a problem with m constraints.

3.10 Revised simplex

The **revised simplex** avoids maintaining the full tableau. It computes only the information needed at each iteration:

Revised simplex — one iteration

1. Compute $\boldsymbol{\pi}^\top = \mathbf{c}_B^\top B^{-1}$ (simplex multipliers).
2. For each $j \notin$ basis, compute $\bar{c}_j = c_j - \boldsymbol{\pi}^\top \mathbf{a}_j$.
3. If $\bar{c}_j \geq 0$ for all j: STOP.
4. Choose k with $\bar{c}_k < 0$ (entering variable).
5. Compute $\mathbf{d} = B^{-1}\mathbf{a}_k$ (direction).
6. Minimum ratio: $r = \arg\min_{i:d_i>0} \bar{b}_i / d_i$.
7. Update B^{-1} (product of elementary matrices).

3.11 Python implementation

Simplex tableau in Python

```python
import numpy as np

def simplex_tableau(c, A, b):
    """Solve max c^T x  s.t.  Ax <= b, x >= 0."""
    m, n = A.shape
    # Add slack variables
    tableau = np.zeros((m + 1, n + m + 1))
    tableau[0, :n] = -c          # objective row (min -c^T x)
    tableau[1:, :n] = A
    tableau[1:, n:n+m] = np.eye(m)
    tableau[1:, -1] = b

    basis = list(range(n, n + m))

    while True:
        # Entering variable: most negative reduced cost
        pivot_col = np.argmin(tableau[0, :-1])
        if tableau[0, pivot_col] >= -1e-10:
            break  # optimal

        # Ratios
        col = tableau[1:, pivot_col]
        rhs = tableau[1:, -1]
        ratios = np.full(m, np.inf)
        for i in range(m):
            if col[i] > 1e-10:
                ratios[i] = rhs[i] / col[i]

        pivot_row = np.argmin(ratios) + 1
        if ratios[pivot_row - 1] == np.inf:
            raise ValueError("Problem is unbounded")

        # Gauss-Jordan pivot
        pivot_val = tableau[pivot_row, pivot_col]
        tableau[pivot_row] /= pivot_val
        for i in range(m + 1):
            if i != pivot_row:
                tableau[i] -= (tableau[i, pivot_col]
                               * tableau[pivot_row])
        basis[pivot_row - 1] = pivot_col

    # Extract solution
    x = np.zeros(n)
    for i, var in enumerate(basis):
        if var < n:
            x[var] = tableau[i + 1, -1]

    return x, tableau[0, -1]  # (solution, -z_max)

# Test
c = np.array([5., 4.])
A = np.array([[6., 4.], [1., 2.]])
b = np.array([24., 6.])

x_opt, neg_z = simplex_tableau(c, A, b)
print(f"x* = {x_opt}")
print(f"z* = {-neg_z:.2f}")
```

Output

```
x* = [3. 1.5] z* = 21.00
```

Verification with SciPy

```python
from scipy.optimize import linprog

c_min = [-5, -4]
A_ub = [[6, 4], [1, 2]]
b_ub = [24, 6]
res = linprog(c_min, A_ub=A_ub, b_ub=b_ub,
              bounds=[(0, None), (0, None)], method='highs')
print(f"x* = {res.x}, z* = {-res.fun:.2f}")
```

Output

```
x* = [3. 1.5], z* = 21.00
```

3.12 Exercises

Exercise 3.1 (⋆ — Simplex tableau)**.** Solve using the simplex tableau:

$$\begin{aligned} \max \quad & z = 3x_1 + 5x_2 \\ \text{s.t.} \quad & x_1 \le 4 \\ & 2x_2 \le 12 \\ & 3x_1 + 5x_2 \le 25 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Exercise 3.2 (⋆⋆ — Big-M)**.** Solve using the Big-M method:

$$\begin{aligned} \min \quad & z = 4x_1 + x_2 \\ \text{s.t.} \quad & 3x_1 + x_2 = 3 \\ & 4x_1 + 3x_2 \ge 6 \\ & x_1 + 2x_2 \le 4 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Exercise 3.3 (⋆⋆ — Two-phase method)**.** Re-do the previous exercise using the two-phase method.

Exercise 3.4 (⋆⋆ — Degeneracy)**.** Verify that degeneracy occurs in:

$$\begin{aligned} \max \quad & z = 2x_1 + x_2 \\ \text{s.t.} \quad & x_1 + x_2 \le 4 \\ & x_1 \le 3 \\ & x_2 \le 1 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Exercise 3.5 ($\star\star\star$ — Implementation)**.** Implement the two-phase method in Python and test it on an LP of your choice that has $\geq$ and $=$ constraints.

Chapter 4

LP Duality

Every optimisation problem hides a mirror: its *dual*. This idea, which traces back to John von Neumann and his work on game theory in the 1940s, is one of the deepest in all of optimisation. In linear programming, duality takes a particularly elegant form: to every minimisation problem corresponds a maximisation problem whose optimal value coincides with that of the primal (the strong duality theorem). The dual provides not only certified bounds, but also an economic interpretation: dual variables are "shadow prices" measuring the marginal value of each constraint.

4.1 Motivation

Consider a minimization problem. We seek to bound the optimal value:

- Every feasible solution provides an **upper bound**.
- Can we systematically obtain **lower bounds**?

Duality theory answers affirmatively by associating to each LP (the **primal**) another LP (the **dual**).

4.2 Constructing the dual

Definition 4.1 (Primal and dual)**.** Let the primal (P) be in canonical form:

$$(\mathrm{P})\qquad \begin{array}{rl} \max & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & A\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \ge \mathbf{0} \end{array}$$

The **dual** (D) is:

$$(\mathrm{D})\qquad \begin{array}{rl} \min & \mathbf{b}^\top \mathbf{y} \\ \text{s.t.} & A^\top \mathbf{y} \ge \mathbf{c} \\ & \mathbf{y} \ge \mathbf{0} \end{array}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the vector of **dual variables**.

Dual construction rules

Primal (max)		**Dual (min)**
Constraint $\leq$	$\leftrightarrow$	Variable $y_i \geq 0$
Constraint $\geq$	$\leftrightarrow$	Variable $y_i \leq 0$
Constraint $=$	$\leftrightarrow$	Variable y_i free
Variable $x_j \geq 0$	$\leftrightarrow$	Constraint $\geq$
Variable $x_j \leq 0$	$\leftrightarrow$	Constraint $\leq$
Variable x_j free	$\leftrightarrow$	Constraint $=$

Remark 4.2. The dual of the dual is the primal: $(D(D)) = (P)$.

4.3 Duality theorems

Theorem 4.3 (Weak duality)**.** *If* $\mathbf{x}$ *is feasible for (P) and* $\mathbf{y}$ *is feasible for (D), then:*

$$\mathbf{c}^\top \mathbf{x} \leq \mathbf{b}^\top \mathbf{y}$$

Proof. We have $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{y} \geq \mathbf{0}$, so $\mathbf{y}^\top A\mathbf{x} \leq \mathbf{y}^\top \mathbf{b}$. Similarly, $A^\top \mathbf{y} \geq \mathbf{c}$ and $\mathbf{x} \geq \mathbf{0}$ give $\mathbf{x}^\top A^\top \mathbf{y} \geq \mathbf{c}^\top \mathbf{x}$. Hence $\mathbf{c}^\top \mathbf{x} \leq \mathbf{y}^\top A\mathbf{x} \leq \mathbf{b}^\top \mathbf{y}$. $\square$

Corollary 4.4 (Bounds)**.**

1. *If (P) is unbounded* ($\max = +\infty$)*, then (D) is infeasible.*
2. *If (D) is unbounded* ($\min = -\infty$)*, then (P) is infeasible.*

Theorem 4.5 (Strong duality)**.** *If (P) has a finite optimal solution* $\mathbf{x}^*$*, then (D) also has a finite optimal solution* $\mathbf{y}^*$ *and:*

$$\mathbf{c}^\top \mathbf{x}^* = \mathbf{b}^\top \mathbf{y}^*$$

Interpretation

Strong duality states that the best lower bound (from the dual) coincides with the best upper bound (from the primal) at optimality. There is "no gap" between the two.

4.4 Complementary slackness

Theorem 4.6 (Complementary slackness)**.** *Let* $\mathbf{x}^*$ *and* $\mathbf{y}^*$ *be feasible solutions for the primal and dual respectively. They are both optimal if and only if:*

$$y_i^* \left(b_i - \sum_j a_{ij} x_j^* \right) = 0, \quad i = 1, \dots, m$$

$$x_j^* \left(\sum_i a_{ij} y_i^* - c_j \right) = 0, \quad j = 1, \dots, n$$

In other words:

- If a primal constraint is **not tight**, the corresponding dual variable is zero.
- If a primal variable is **strictly positive**, the corresponding dual constraint is tight.

4.5 Economic interpretation

Definition 4.7 (Shadow price)**.** The dual variable y_i^* associated with the i-th primal constraint is called the **shadow price** of resource i. It represents the marginal change in the optimal value when the right-hand side b_i increases by one unit:

$$y_i^* = \frac{\partial z^*}{\partial b_i}$$

Example 4.8 (Shadow price interpretation)**.** Consider the production problem from the previous chapter:

$$\begin{aligned} \max \quad & z = 5x_1 + 4x_2 \\ \text{s.t.} \quad & 6x_1 + 4x_2 \le 24 \quad (y_1) \\ & x_1 + 2x_2 \le 6 \quad (y_2) \\ & x_1, x_2 \ge 0 \end{aligned}$$

The dual is:

$$\begin{aligned} \min \quad & w = 24y_1 + 6y_2 \\ \text{s.t.} \quad & 6y_1 + y_2 \ge 5 \\ & 4y_1 + 2y_2 \ge 4 \\ & y_1, y_2 \ge 0 \end{aligned}$$

From the final simplex tableau (Chapter 3): $y_1^* = 3/4$, $y_2^* = 1/2$. Check: $w^* = 24(3/4) + 6(1/2) = 21 = z^*$.

Interpretation: increasing the capacity of constraint 1 from 24 to 25 increases z^* by approximately $3/4 = 0.75$.

4.6 Reading the dual from the simplex tableau

Proposition 4.9 (Dual variables from the final tableau)**.** In the final simplex tableau (primal in standard form with slack variables), the optimal dual variable values can be read from the **objective row** at the columns of the slack variables:

$$y_i^* = \bar{c}_{s_i} \quad \text{(reduced cost of slack variable } s_i\text{)}$$

4.7 The dual simplex method

The **dual simplex** method solves the dual directly on the primal tableau. It is useful when the basis is **dual feasible** (all reduced costs ≥ 0) but not primal feasible ($\bar{b}_i < 0$ for some i).

Dual simplex — one iteration

1. **Primal optimality test**: if $\bar{b}_i \geq 0$ for all i, STOP (optimal solution).
2. **Leaving variable**: choose $r = \arg\min_i \bar{b}_i$ (most negative row).
3. **Infeasibility test**: if $\bar{a}_{rj} \geq 0$ for all j, STOP (primal is infeasible).
4. **Entering variable**: apply the dual ratio:

$$k = \arg\min_{j:\bar{a}_{rj}<0} \frac{\bar{c}_j}{|\bar{a}_{rj}|}$$

5. **Pivot**: pivot on $\bar{a}_{rk}$.

4.8 Numerical example

Example 4.10 (Constructing and solving the dual)**.**

$$(\text{P}) \quad \begin{aligned} \max \quad & z = 4x_1 + 3x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 10 \\ & x_1 + 3x_2 \leq 12 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The dual:

$$(\text{D}) \quad \begin{aligned} \min \quad & w = 10y_1 + 12y_2 \\ \text{s.t.} \quad & 2y_1 + y_2 \geq 4 \\ & y_1 + 3y_2 \geq 3 \\ & y_1, y_2 \geq 0 \end{aligned}$$

Verification with PuLP

```python
import pulp

# Primal
P = pulp.LpProblem("Primal", pulp.LpMaximize)
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
P += 4*x1 + 3*x2
P += 2*x1 + x2 <= 10, "c1"
P += x1 + 3*x2 <= 12, "c2"
P.solve(pulp.PULP_CBC_CMD(msg=0))
print(f"Primal: z* = {pulp.value(P.objective):.2f}")
print(f"  x* = ({x1.varValue}, {x2.varValue})")

# Dual
D = pulp.LpProblem("Dual", pulp.LpMinimize)
y1 = pulp.LpVariable("y1", lowBound=0)
y2 = pulp.LpVariable("y2", lowBound=0)
D += 10*y1 + 12*y2
```

```
D += 2*y1 + y2 >= 4
D += y1 + 3*y2 >= 3
D.solve(pulp.PULP_CBC_CMD(msg=0))
print(f"Dual: w* = {pulp.value(D.objective):.2f}")
print(f"  y* = ({y1.varValue}, {y2.varValue})")
```

Output

```
Primal: z* = 22.80 x* = (5.4, 2.2) Dual: w* = 22.80 y* = (1.8, 0.4)
```

4.9 Summary of primal-dual relationships

	(P) feasible	**(P) infeasible**
(D) feasible	Equal finite optima	(D) unbounded
(D) infeasible	(P) unbounded	Both infeasible

Remark 4.11. It is possible for both primal and dual to be infeasible. Example: max $x_1 - x_2$ s.t. $-x_1 + x_2 \leq -1$, $x_1 - x_2 \leq -1$, $x_1, x_2 \geq 0$.

4.10 Exercises

Exercise 4.1 ($\star$ — Write the dual)**.** Write the dual of the following LPs:

1. max $3x_1 + 2x_2$ s.t. $x_1 + x_2 \leq 5$, $2x_1 + x_2 \leq 8$, $x_1, x_2 \geq 0$.
2. min $5x_1 + 3x_2$ s.t. $x_1 + x_2 = 4$, $2x_1 + x_2 \geq 6$, $x_1, x_2 \geq 0$.

Exercise 4.2 ($\star\star$ — Strong duality verification)**.** Solve both the primal and dual from Exercise 1(a) and verify that the optimal values are equal.

Exercise 4.3 ($\star\star$ — Complementary slackness)**.** Use complementary slackness to find the optimal dual solution given the primal solution $x_1^* = 3$, $x_2^* = 2$ for the LP: max $4x_1 + 3x_2$ s.t. $2x_1 + x_2 \leq 8$, $x_1 + 2x_2 \leq 7$, $x_1, x_2 \geq 0$.

Exercise 4.4 ($\star\star$ — Economic interpretation)**.** A factory has three resources R_1, R_2, R_3 with capacities 100, 80, and 120. The optimal shadow prices are $y_1^* = 2$, $y_2^* = 0$, $y_3^* = 3$. Interpret these values. A supplier offers 10 additional units of R_2 for \$5. Should the offer be accepted?

Exercise 4.5 ($\star\star\star$ — Proof)**.** Prove the weak duality theorem for the general case (standard form primal).

Exercise 4.6 ($\star\star\star$ — Dual simplex)**.** Apply the dual simplex to the problem:

$$\begin{aligned} \min \quad & z = 2x_1 + 3x_2 + x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 \geq 6 \\ & 2x_1 + x_3 \geq 4 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Chapter 5

Sensitivity Analysis

An optimisation model is never perfect: costs, capacities, and demands are estimated, and these estimates carry margins of error. The natural question is: how far can the optimal solution remain valid if the parameters change slightly? This is the question of *sensitivity analysis*, an indispensable tool for decision-makers. Thanks to the structure of the simplex and duality, one can answer this question without solving a new problem: it suffices to examine the ranges of parameter variation for which the optimal basis remains unchanged.

5.1 Introduction

In practice, the parameters of an LP (costs c_j, resources b_i, coefficients a_{ij}) are never known with absolute precision. **Sensitivity analysis** studies how the optimal solution and optimal value change when these parameters vary.

Definition 5.1 (Sensitivity analysis)**. Sensitivity analysis** (or **post-optimality analysis**) determines the ranges of parameter variation for which the **optimal basis** remains unchanged.

5.2 Variation of right-hand sides b_i

5.2.1 Effect on the solution

Let $\mathbf{b}' = \mathbf{b} + \Delta\mathbf{b}$. The basic solution becomes:

$$\mathbf{x}'_B = B^{-1}\mathbf{b}' = B^{-1}\mathbf{b} + B^{-1}\Delta\mathbf{b} = \bar{\mathbf{b}} + B^{-1}\Delta\mathbf{b}$$

The basis remains feasible as long as $\mathbf{x}'_B \geq \mathbf{0}$:

$$\bar{\mathbf{b}} + B^{-1}\Delta\mathbf{b} \geq \mathbf{0}$$

5.2.2 Variation of a single b_i

If only b_i changes by an amount δ:

$$\bar{b}_k + (B^{-1})_{ki} \cdot \delta \geq 0, \quad k = 1, \ldots, m$$

Validity range for b_i

The basis remains optimal for:

$$b_i \in \left[b_i - \min_{k:\,(B^{-1})_{ki}>0} \frac{\bar{b}_k}{(B^{-1})_{ki}},\ b_i + \min_{k:\,(B^{-1})_{ki}<0} \frac{-\bar{b}_k}{(B^{-1})_{ki}} \right]$$

Theorem 5.2 (Variation of z^*). *When b_i varies by δ within the validity range:*

$$z^*(\delta) = z^* + y_i^* \cdot \delta$$

where y_i^* is the shadow price (dual variable) of constraint i.

Example 5.3 (Variation of b_1). Consider the LP from Chapter 3:

$$\max\ z = 5x_1 + 4x_2 \quad \text{s.t.} \quad 6x_1 + 4x_2 \le 24,\ x_1 + 2x_2 \le 6$$

The final tableau gives $B^{-1} = \begin{pmatrix} 1/4 & -1/2 \\ -1/8 & 3/4 \end{pmatrix}$, $\bar{\mathbf{b}} = (3, 3/2)^\top$, $y_1^* = 3/4$, $y_2^* = 1/2$.

For variation of $b_1 = 24$:

- $(B^{-1})_{11} = 1/4 > 0$: $\delta \ge -\bar{b}_1/(1/4) = -12$
- $(B^{-1})_{21} = -1/8 < 0$: $\delta \le -\bar{b}_2/(-1/8) = 12$

So $b_1 \in [24 - 12, 24 + 12] = [12, 36]$.

If $b_1 = 30$: $z^*(30) = 21 + (3/4)(30 - 24) = 21 + 4.5 = 25.5$.

5.3 Variation of objective coefficients c_j

5.3.1 Basic variable

If x_j is a basic variable, changing c_j affects the reduced costs of non-basic variables. The basis remains optimal as long as all reduced costs remain ≥ 0 (minimization) or ≤ 0 (maximization tableau).

Validity range for c_j (basic variable)

The reduced costs of non-basic variables k become:

$$\bar{c}_k' = \bar{c}_k - \delta \cdot \bar{a}_{rk}$$

where r is the row of x_j in the basis and $\delta = c_j' - c_j$. The basis remains optimal for $\bar{c}_k' \ge 0$ for all non-basic k.

5.3.2 Non-basic variable

If x_j is non-basic with $\bar{c}_j > 0$ (maximization), the basis remains optimal as long as $\bar{c}_j + \delta \ge 0$, i.e., $c_j \ge c_j - \bar{c}_j$.

Example 5.4 (Variation of c_1). In our example, x_1 is basic. The reduced costs of non-basic variables (s_1, s_2) in the final tableau are: $\bar{c}_{s_1} = 3/4$, $\bar{c}_{s_2} = 1/2$.

Row of x_1 gives $\bar{a}_{1,s_1} = 1/4$, $\bar{a}_{1,s_2} = -1/2$.

- $\bar{c}'_{s_1} = 3/4 - \delta(1/4) \geq 0 \Rightarrow \delta \leq 3$
- $\bar{c}'_{s_2} = 1/2 - \delta(-1/2) = 1/2 + \delta/2 \geq 0 \Rightarrow \delta \geq -1$

So $c_1 \in [5-1, 5+3] = [4, 8]$.

5.4 Adding a new variable

If we add a variable x_{n+1} with cost c_{n+1} and technological column $\mathbf{a}_{n+1}$, compute:

$$\bar{c}_{n+1} = c_{n+1} - \mathbf{c}_B^\top B^{-1}\mathbf{a}_{n+1}$$

- If $\bar{c}_{n+1} \geq 0$ (minimization): the solution remains optimal without using x_{n+1}.
- If $\bar{c}_{n+1} < 0$: it is beneficial to bring x_{n+1} into the basis.

5.5 Adding a new constraint

If we add a constraint $\mathbf{a}_{m+1}^\top \mathbf{x} \leq b_{m+1}$:

1. Check whether the current optimal solution $\mathbf{x}^*$ satisfies the new constraint: $\mathbf{a}_{m+1}^\top \mathbf{x}^* \leq b_{m+1}$.
2. If yes: the solution remains optimal.
3. If no: apply the **dual simplex** by adding the constraint (with a slack variable) to the final tableau.

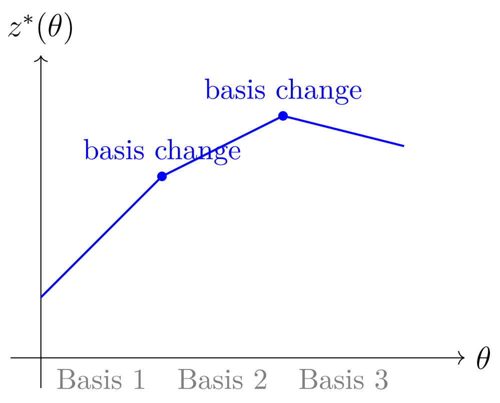
5.6 Parametric analysis

Definition 5.5 (Parametric analysis). **Parametric analysis** studies the continuous variation of z^* as a function of a parameter θ:

$$\mathbf{b}(\theta) = \mathbf{b} + \theta\mathbf{d} \qquad \text{or} \qquad \mathbf{c}(\theta) = \mathbf{c} + \theta\mathbf{e}$$

where $\mathbf{d}$ or $\mathbf{e}$ are fixed directions.

Theorem 5.6 (Shape of $z^*(\theta)$). *The optimal value $z^*(\theta)$ is a **piecewise linear** and **concave** function (when varying $\mathbf{b}$ in a maximization problem) of θ. Each segment corresponds to a different optimal basis.*



5.7 Python implementation

Sensitivity analysis with SciPy

```
from scipy.optimize import linprog
import numpy as np

c = [-5, -4]  # max 5x1 + 4x2
A_ub = [[6, 4], [1, 2]]
b_ub = [24, 6]
bounds = [(0, None), (0, None)]

# Base solution
res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds,
              method='highs')
print(f"z* = {-res.fun:.2f}, x* = {res.x}")

# Sensitivity on b1
print("\nVariation of b1:")
for delta in [-12, -6, 0, 6, 12]:
    b_new = [24 + delta, 6]
    r = linprog(c, A_ub=A_ub, b_ub=b_new, bounds=bounds,
                method='highs')
    print(f"  b1={24+delta:3d} -> z*={-r.fun:.2f}, "
          f"x*=[{r.x[0]:.2f}, {r.x[1]:.2f}]")

# Sensitivity on c1
print("\nVariation of c1:")
for c1 in [3, 4, 5, 6, 8]:
    c_new = [-c1, -4]
    r = linprog(c_new, A_ub=A_ub, b_ub=b_ub, bounds=bounds,
                method='highs')
    print(f"  c1={c1} -> z*={-r.fun:.2f}, "
          f"x*=[{r.x[0]:.2f}, {r.x[1]:.2f}]")
```

Output

```
z* = 21.00, x* = [3. 1.5]
Variation of b1: b1= 12 -> z*= 12.00, x*=[0.00, 3.00] b1= 18 -> z*= 16.50,
x*=[1.50, 2.25] b1= 24 -> z*= 21.00, x*=[3.00, 1.50] b1= 30 -> z*= 25.50,
x*=[4.50, 0.75] b1= 36 -> z*= 30.00, x*=[6.00, 0.00]
Variation of c1: c1=3 -> z*= 15.00, x*=[3.00, 1.50] c1=4 -> z*= 18.00, x*=[3.00,
1.50] c1=5 -> z*= 21.00, x*=[3.00, 1.50] c1=6 -> z*= 24.00, x*=[3.00, 1.50] c1=8
-> z*= 30.00, x*=[3.00, 1.50]
```

5.8 Sensitivity report

Commercial solvers provide a **sensitivity report** containing for each constraint and variable:

Information	**Description**
Optimal value	Value of the variable/constraint at optimality
Shadow price	Marginal change in z^* per unit of b_i
Reduced cost	Marginal change in c_j needed for x_j to enter the basis
Allowable increase	Upper bound on variation
Allowable decrease	Lower bound on variation

5.9 Exercises

Exercise 5.1 ($\star$ — Validity ranges)**.** For the LP:

$$\max\ z = 3x_1 + 2x_2 \quad \text{s.t.} \quad x_1 + x_2 \leq 10,\ 2x_1 + x_2 \leq 14,\ x_1, x_2 \geq 0$$

Determine the validity ranges for b_1 and b_2.

Exercise 5.2 ($\star\star$ — Objective variation)**.** For the same LP, determine the range of c_1 for which the optimal basis does not change.

Exercise 5.3 ($\star\star$ — New variable)**.** Should we introduce a new product P_3 with profit $c_3 = 6$ and resource consumption $(a_{13}, a_{23}) = (3, 1)$?

Exercise 5.4 ($\star\star$ — New constraint)**.** Add the constraint $x_1 + x_2 \leq 5$. Is the optimal solution still feasible? If not, find the new optimal solution.

Exercise 5.5 ($\star\star\star$ — Complete parametric analysis)**.** Study $z^*(\theta)$ for $\mathbf{b}(\theta) = (10 + \theta, 14 - 2\theta)$ as θ varies from -5 to $+5$. Plot the curve and identify basis changes.

Exercise 5.6 ($\star\star\star$ — Implementation)**.** Write a Python script that:

1. Solves an LP with PuLP.
2. Automatically computes validity ranges for each b_i via numerical perturbation.
3. Plots $z^*(\theta)$ for a parametric variation of b_1.

CHAPTER 5. SENSITIVITY ANALYSIS

Chapter 6

Transportation and Assignment Problems

How does one ship goods from several factories to several warehouses at minimum cost? This problem, formulated independently by the Soviet mathematician Leonid Kantorovich (1939) and the American statistician Frank Hitchcock (1941), is one of the oldest and most elegant in operations research. It possesses a special LP structure—all variables appear with coefficient 1 in the constraints—which allows specialised algorithms far faster than the general simplex. Kantorovich received the Nobel Prize in Economics in 1975 for this work.

6.1 The transportation problem

Definition 6.1 (Transportation problem)**.** The **transportation problem** determines the quantities x_{ij} to ship from m sources S_i (supply a_i) to n destinations D_j (demand b_j) so as to minimize total transportation cost:

$$\begin{aligned} \min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\, x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1,\dots,m \quad \text{(supply)} \\ & \sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1,\dots,n \quad \text{(demand)} \\ & x_{ij} \ge 0 \end{aligned}$$

Definition 6.2 (Balanced problem)**.** The problem is **balanced** if $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$. Otherwise, add a **dummy** source or destination with zero costs.

Remark 6.3. The transportation problem is a special case of LP. Its special structure (totally unimodular constraint matrix) guarantees that every BFS is integer if the a_i and b_j are integers.

6.2 Initial feasible solutions

6.2.1 Northwest corner method

Northwest corner

1. Start at cell $(1, 1)$.
2. Allocate $x_{ij} = \min(a_i, b_j)$.
3. Update a_i and b_j.
4. Move to the next cell (right if $a_i = 0$, down if $b_j = 0$).
5. Repeat until complete allocation.

6.2.2 Minimum cost method

Minimum cost

1. Select the cell (i, j) with minimum cost c_{ij}.
2. Allocate $x_{ij} = \min(a_i, b_j)$.
3. Update capacities and eliminate the saturated row or column.
4. Repeat.

6.2.3 Vogel's approximation method

Vogel's Approximation Method (VAM)

1. For each row and column, compute the **penalty**: difference between the two smallest costs.
2. Select the row or column with the largest penalty.
3. Allocate the maximum possible to the minimum-cost cell in that row/column.
4. Repeat.

Initial solution quality

Northwest corner $<$ Minimum cost $<$ Vogel. Vogel's method generally gives a solution close to optimal.

6.3 Complete example

Example 6.4 (Transportation problem).

	D_1	D_2	D_3	Supply
S_1	4	8	1	30
S_2	7	3	5	40
S_3	2	6	9	20
Demand	25	35	30	90

The problem is balanced $(30 + 40 + 20 = 25 + 35 + 30 = 90)$.
Northwest corner solution:

	D_1	D_2	D_3
S_1	**25**	**5**	0
S_2	0	**30**	**10**
S_3	0	0	**20**

Cost: $25(4) + 5(8) + 30(3) + 10(5) + 20(9) = 100 + 40 + 90 + 50 + 180 = 460$.

6.4 Optimality test: MODI method

The **MODI** (Modified Distribution) method uses multipliers u_i and v_j such that for each basic cell:

$$u_i + v_j = c_{ij}$$

Set $u_1 = 0$ and solve the system.

Definition 6.5 (Reduced cost of a non-basic cell).

$$\bar{c}_{ij} = c_{ij} - u_i - v_j$$

Theorem 6.6 (Transportation optimality). *The solution is optimal if and only if $\bar{c}_{ij} \geq 0$ for all non-basic cells.*

6.5 Improvement: stepping-stone method

If $\bar{c}_{ij} < 0$ for a non-basic cell (i, j):

1. Identify the unique **cycle** connecting (i, j) to basic cells (alternating $+/-$).

2. Determine $\theta = \min$ of the values on $-$ cells of the cycle.

3. Add θ to $+$ cells and subtract θ from $-$ cells.

6.6 The assignment problem

Definition 6.7 (Assignment problem)**.** The **assignment problem** assigns n agents to n tasks (bijection) to minimize total cost:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\dots,n \\ & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,\dots,n \\ & x_{ij} \in \{0,1\} \end{aligned}$$

Remark 6.8. This is a special case of the transportation problem with $a_i = b_j = 1$. The integrality constraint is automatically satisfied (total unimodularity).

6.7 Hungarian algorithm

Hungarian algorithm (Kuhn-Munkres method)

1. **Row reduction**: subtract the row minimum from each row.
2. **Column reduction**: subtract the column minimum from each column.
3. **Cover zeros**: draw the minimum number of lines (horizontal/vertical) covering all zeros.
4. If this number $= n$: an optimal assignment exists among the zeros.
5. Otherwise: find the smallest uncovered element ε, subtract it from uncovered elements, add it to intersection elements. Return to step 3.

Example 6.9 (Hungarian algorithm)**.** Cost matrix:

$$C = \begin{pmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{pmatrix}$$

Step 1 — Row reduction (subtract 2, 3, 1, 4):

$$\begin{pmatrix} 7 & 0 & 5 & 6 \\ 3 & 1 & 0 & 4 \\ 4 & 7 & 0 & 7 \\ 3 & 2 & 5 & 0 \end{pmatrix}$$

Step 2 — Column reduction (subtract 3, 0, 0, 0):

$$\begin{pmatrix} 4 & 0 & 5 & 6 \\ 0 & 1 & 0 & 4 \\ 1 & 7 & 0 & 7 \\ 0 & 2 & 5 & 0 \end{pmatrix}$$

Optimal assignment: $(1, 2), (2, 1), (3, 3), (4, 4)$.
Optimal cost: $2 + 6 + 1 + 4 = 13$.

6.8 Python implementation

Transportation problem with PuLP

```python
import pulp
import numpy as np

# Data
cost = [[4, 8, 1], [7, 3, 5], [2, 6, 9]]
supply = [30, 40, 20]
demand = [25, 35, 30]
m, n = len(supply), len(demand)

prob = pulp.LpProblem("Transport", pulp.LpMinimize)
x = [[pulp.LpVariable(f"x_{i}_{j}", lowBound=0)
      for j in range(n)] for i in range(m)]

# Objective
prob += pulp.lpSum(cost[i][j] * x[i][j]
                   for i in range(m) for j in range(n))

# Supply constraints
for i in range(m):
    prob += pulp.lpSum(x[i][j] for j in range(n)) == supply[i]

# Demand constraints
for j in range(n):
    prob += pulp.lpSum(x[i][j] for i in range(m)) == demand[j]

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print(f"Optimal cost: {pulp.value(prob.objective):.0f}")
for i in range(m):
    row = [x[i][j].varValue for j in range(n)]
    print(f"  S{i+1} -> {row}")
```

Output

```
Optimal cost: 290 S1 -> [0.0, 0.0, 30.0] S2 -> [5.0, 35.0, 0.0] S3 -> [20.0, 0.0, 0.0]
```

Assignment problem with SciPy

```python
from scipy.optimize import linear_sum_assignment
import numpy as np

C = np.array([[9, 2, 7, 8],
              [6, 4, 3, 7],
                 [5, 8, 1, 8],
                 [7, 6, 9, 4]])

row_ind, col_ind = linear_sum_assignment(C)
print("Optimal assignment:")
for i, j in zip(row_ind, col_ind):
    print(f"  Agent {i+1} -> Task {j+1} (cost {C[i,j]})")
print(f"Total cost: {C[row_ind, col_ind].sum()}")
```

Output

```
Optimal assignment: Agent 1 -> Task 2 (cost 2) Agent 2 -> Task 1 (cost 6) Agent
3 -> Task 3 (cost 1) Agent 4 -> Task 4 (cost 4) Total cost: 13
```

6.9 Variants

- **Unbalanced transportation**: add a dummy source/destination.
- **Transshipment**: some nodes are both sources and destinations.
- **Capacitated arcs**: $x_{ij} \leq u_{ij}$.
- **Maximization assignment**: convert to minimization ($c'_{ij} = M - c_{ij}$).

6.10 Exercises

Exercise 6.1 ($\star$ — Northwest corner and minimum cost)**.** Solve using both methods:

	D_1	D_2	D_3	Supply
S_1	3	1	7	40
S_2	4	6	2	50
S_3	8	3	5	30
Demand	30	40	50	

Exercise 6.2 ($\star\star$ — MODI)**.** Verify the optimality of the Vogel solution and improve if necessary using the MODI method.

Exercise 6.3 ($\star\star$ — Unbalanced problem)**.** Solve the transportation problem with supply $(20, 30, 25)$ and demand $(15, 25, 20, 15)$.

Exercise 6.4 ($\star\star$ — Hungarian algorithm)**.** Solve by hand:

$$C = \begin{pmatrix} 10 & 5 & 13 & 15 \\ 3 & 9 & 18 & 13 \\ 10 & 7 & 2 & 8 \\ 5 & 11 & 9 & 6 \end{pmatrix}$$

Exercise 6.5 ($\star\star\star$ — Implementation)**.** Implement Vogel's approximation method in Python and test it on a 5×5 problem.

Chapter 7

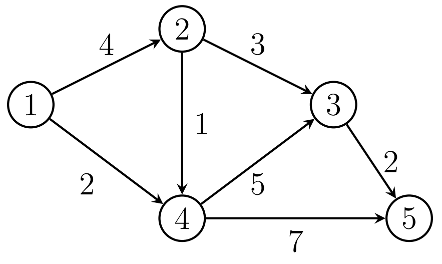
Applied Graph Theory

In 1736, Leonhard Euler turned his attention to a recreational problem posed by the inhabitants of Königsberg: is it possible to cross the city's seven bridges, passing over each one exactly once? By proving that it is impossible, Euler did not merely solve a puzzle about walks: he founded graph theory, a branch of mathematics that would become indispensable to operations research. Today, graphs model transportation networks, electronic circuits, social networks, and supply chains. Finding the shortest path, the minimum spanning tree, the maximum flow — these fundamental problems have algorithms (Dijkstra, Kruskal, Ford-Fulkerson) that are executed millions of times daily in computer systems around the world.

7.1 Fundamental definitions

Definition 7.1 (Graph)**.** A **graph** $G = (V, E)$ is a pair where V is a finite set of **vertices** (or nodes) and E is a set of **edges** (undirected graph) or **arcs** (directed graph, denoted $G = (V, A)$).

Definition 7.2 (Weighted graph)**.** A graph is **weighted** if each edge/arc (i, j) is endowed with a weight $w(i, j) \in \mathbb{R}$ (distance, cost, capacity, etc.).



Definition 7.3 (Path and cycle)**.** A **path** from u to v is a sequence of vertices $u = v_0, v_1, \ldots, v_k = v$ such that $(v_{i-1}, v_i) \in E$ for all i. Its **length** is the sum of the weights. A **cycle** is a closed path ($u = v$) of nonzero length.

Definition 7.4 (Connected graph)**.** An undirected graph is **connected** if there exists a path between every pair of vertices. A directed graph is **strongly connected** if there exists a directed path between every ordered pair of vertices.

Definition 7.5 (Tree)**.** A **tree** is a connected graph without cycles. It has exactly $|V| - 1$ edges.

7.2 Graph representations

Data structures

- **Adjacency matrix** $A \in \{0,1\}^{n\times n}$: $A_{ij} = 1$ if $(i,j) \in E$. Memory: $\mathcal{O}(n^2)$.
- **Adjacency lists**: for each vertex, the list of its neighbors. Memory: $\mathcal{O}(n+m)$.
- **Weight matrix** W: $W_{ij} = w(i,j)$ or $+\infty$ if $(i,j) \notin E$.

7.3 Single-source shortest paths

7.3.1 Dijkstra's algorithm

Theorem 7.6 (Dijkstra's correctness)**.** *Dijkstra's algorithm computes shortest paths from a source vertex s in a graph with* ***nonnegative*** *weights in time $\mathcal{O}((n+m)\log n)$ using a binary heap.*

Dijkstra's Algorithm

```
1. Initialize $d[s] = 0$, $d[v] = +\infty$ for $v \neq s$.
2. $Q \leftarrow V$ (priority queue).
3. While $Q \neq \emptyset$:
    (a) Extract $u = \arg\min_{v\in Q} d[v]$.
    (b) For each neighbor $v$ of $u$ with $(u,v) \in E$: if $d[u] + w(u,v) < d[v]$:
        $d[v] \leftarrow d[u] + w(u,v)$, $\pi[v] \leftarrow u$.
```

Example 7.7 (Dijkstra)**.** On the graph above with source $s = 1$:

Step	$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$
Init	0	∞	∞	∞	∞
$u=1$	0	4	∞	2	∞
$u=4$	0	3	7	2	9
$u=2$	0	3	6	2	9
$u=3$	0	3	6	2	8
$u=5$	0	3	6	2	8

Shortest path $1 \to 5$: $1 \to 4 \to 3 \to 5$ of length 8.

7.3.2 Bellman-Ford algorithm

Theorem 7.8 (Bellman-Ford)**.** *The Bellman-Ford algorithm computes shortest paths from s in the presence of* ***negative*** *weights (but no reachable negative cycle) in time $\mathcal{O}(nm)$.*

Bellman-Ford Algorithm

1. Initialize $d[s] = 0$, $d[v] = +\infty$ for $v \neq s$.
2. Repeat $|V| - 1$ times:
   (a) For each arc $(u, v) \in A$: if $d[u] + w(u, v) < d[v]$: $d[v] \leftarrow d[u] + w(u, v)$, $\pi[v] \leftarrow u$.
3. **Negative cycle detection**: for each arc (u, v): if $d[u] + w(u, v) < d[v]$, a negative cycle exists.

Negative cycles

In the presence of a negative cycle reachable from s, shortest paths are undefined (distances can go to $-\infty$). Bellman-Ford detects this situation.

7.4 All-pairs shortest paths

Floyd-Warshall Algorithm

Complexity: $\mathcal{O}(n^3)$.

1. Initialize $D^{(0)} = W$ (weight matrix).
2. For $k = 1, \ldots, n$:

$$D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$$

7.5 Minimum spanning trees

Definition 7.9 (Minimum spanning tree (MST))**.** A **minimum spanning tree** of a connected weighted graph $G = (V, E)$ is a spanning subgraph without cycles of minimum total weight.

7.5.1 Kruskal's algorithm

Kruskal's Algorithm

```
1. Sort edges by increasing weight.
2. For each edge $(u, v)$ in weight order:
    (a) If $u$ and $v$ are in different components: add $(u, v)$ to the MST and merge the components (Union-Find).
3. Stop after $|V| - 1$ edges.
```

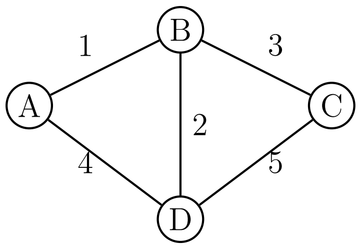
Complexity: $\mathcal{O}(m \log m)$.

7.5.2 Prim's algorithm

Prim's Algorithm

```
1. Choose an initial vertex $s$. $T \leftarrow \{s\}$.
2. While $|T| < |V|$:
    (a) Find the minimum-weight edge $(u, v)$ with $u \in T$, $v \notin T$.
    (b) Add $v$ to $T$ and $(u, v)$ to the MST.
```

Complexity: $\mathcal{O}((n + m) \log n)$ with a binary heap.



Example 7.10 (MST by Kruskal)**.**
Sorted edges: $(A, B) = 1$, $(B, D) = 2$, $(B, C) = 3$, $(A, D) = 4$, $(C, D) = 5$.
MST: $\{(A, B), (B, D), (B, C)\}$, total weight $= 1 + 2 + 3 = 6$.

7.6 Python implementation with NetworkX

Shortest paths and MST

```python
import networkx as nx

# Create directed graph
G = nx.DiGraph()
edges = [(1,2,4), (1,4,2), (2,3,3), (2,4,1),
         (3,5,2), (4,3,5), (4,5,7)]
G.add_weighted_edges_from(edges)

# Dijkstra
dist, path = nx.single_source_dijkstra(G, source=1)
```

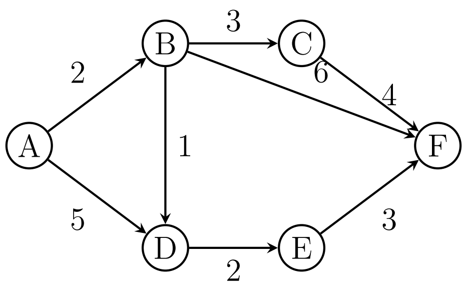
```
print("Distances from 1:", dist)
print("Path 1->5:", path[5])

# Bellman-Ford
dist_bf = nx.single_source_bellman_ford_path_length(G, 1)
print("Bellman-Ford:", dict(dist_bf))

# Undirected graph for MST
H = nx.Graph()
H.add_weighted_edges_from([
    ('A','B',1), ('A','D',4), ('B','C',3),
    ('B','D',2), ('C','D',5)
])

mst = nx.minimum_spanning_tree(H, algorithm='kruskal')
print("\nMST (Kruskal):")
for u, v, d in mst.edges(data=True):
    print(f"  {u}--{v} : {d['weight']}")
print(f"Total weight: {sum(d['weight']
       for _,_,d in mst.edges(data=True))}")
```

7.7 Applications

- **GPS and navigation**: shortest paths (Dijkstra, A^*).
- **Telecommunications**: MST for optimal cabling.
- **Project planning**: PERT/CPM (critical path = longest path in a DAG).
- **Scheduling**: topological sort in a precedence graph.

7.8 Exercises

Exercise 7.1 ($\star$ – Dijkstra)**.** Apply Dijkstra to the following graph from vertex A:

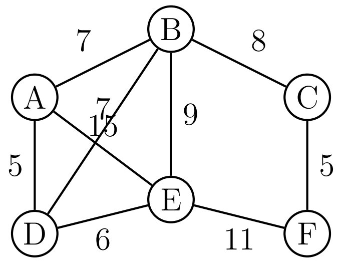


Exercise 7.2 ($\star\star$ – Bellman-Ford)**.** Apply Bellman-Ford to the same graph with an added arc (E, B) of weight -4. Determine if a negative cycle exists.

Exercise 7.3 ($\star\star$ – Floyd-Warshall)**.** Compute the all-pairs shortest distance matrix for a complete graph on 4 vertices of your choice.

CHAPTER 7. APPLIED GRAPH THEORY

Exercise 7.4 (⋆⋆ – Kruskal and Prim)**.** Find the MST of the following graph using both methods and verify that they yield the same result:



Exercise 7.5 (⋆ ⋆ ⋆ – PERT/CPM)**.** A project has the following tasks:

Task	Duration	Predecessors
A	3	–
B	5	A
C	2	A
D	4	B, C
E	6	C
F	2	D, E

Build the project graph, find the critical path and the minimum project duration.

Chapter 8

Network Flows

8.1 Introduction

In 1956, at the height of the Cold War, mathematicians Lester Ford and Delbert Fulkerson at the RAND Corporation worked on a strategic problem: what is the maximum capacity of the Soviet railway network between the USSR and Eastern Europe? To answer this, they developed the Ford–Fulkerson algorithm and proved the max-flow min-cut theorem, one of the most elegant results in combinatorial optimization: the maximum flow from a source to a sink equals exactly the minimum capacity of any cut separating them. This profound duality between flows and cuts now irrigates logistics, telecommunications, computer vision, and even bioinformatics.

Intuition

Imagine a network of pipes connecting a water source to a reservoir. Each pipe has a maximum capacity. The maximum flow problem asks for the greatest rate at which water can be shipped from source to reservoir while respecting every pipe's capacity.

8.2 Fundamental definitions

Definition 8.1 (Flow network)**.** A **flow network** is a directed graph $G = (V, A)$ equipped with a capacity function $c : A \to \mathbb{R}_+$, a **source** vertex $s \in V$, and a **sink** vertex $t \in V$ with $s \neq t$.

Definition 8.2 (Flow)**.** A **flow** in a network (G, c, s, t) is a function $f : A \to \mathbb{R}_+$ satisfying:

1. **Capacity constraint**: $0 \leq f(u, v) \leq c(u, v)$ for all $(u, v) \in A$.
2. **Flow conservation**: for every $v \in V \setminus \{s, t\}$,

$$\sum_{u:(u,v)\in A} f(u, v) = \sum_{w:(v,w)\in A} f(v, w).$$

The **value** of the flow is $|f| = \sum_{w:(s,w)\in A} f(s, w) - \sum_{u:(u,s)\in A} f(u, s)$.

Definition 8.3 (Residual graph)**.** Given a flow f, the **residual graph** $G_f = (V, A_f)$ is defined by:

- For each arc $(u, v) \in A$ with $f(u, v) < c(u, v)$: forward arc (u, v) with residual capacity $c_f(u, v) = c(u, v) - f(u, v)$.
- For each arc $(u, v) \in A$ with $f(u, v) > 0$: backward arc (v, u) with residual capacity $c_f(v, u) = f(u, v)$.

Definition 8.4 (Augmenting path)**.** An **augmenting path** is a path from s to t in the residual graph G_f. Its **bottleneck capacity** is the minimum residual capacity along the path.

Definition 8.5 (Cut)**.** A **cut** (S, T) is a partition of V into S and $T = V \setminus S$ with $s \in S$ and $t \in T$. Its **capacity** is:

$$c(S, T) = \sum_{\substack{u \in S,\, v \in T \\ (u,v) \in A}} c(u, v).$$

8.3 Max-flow min-cut theorem

Theorem 8.6 (Max-flow min-cut)**.** *In any flow network, the maximum value of a flow equals the minimum capacity of a cut:*

$$\max_f |f| = \min_{(S,T)} c(S, T).$$

Proof. **Step 1** (weak duality): For any flow f and any cut (S, T):

$$|f| = \sum_{u \in S,\, v \in T} f(u, v) - \sum_{u \in T,\, v \in S} f(u, v) \le \sum_{u \in S,\, v \in T} c(u, v) = c(S, T).$$

Step 2: If f has no augmenting path in G_f, define $S = \{v \in V : v \text{ is reachable from } s \text{ in } G_f\}$ and $T = V \setminus S$. Then $s \in S$ and $t \in T$.

Step 3: For every arc (u, v) with $u \in S, v \in T$, we have $f(u, v) = c(u, v)$ (otherwise $(u, v) \in A_f$ and v would be in S). For every arc (v, u) with $v \in T, u \in S$, $f(v, u) = 0$. Hence $|f| = c(S, T)$. □

Key flow properties

- $|f| \le c(S, T)$ for every cut (S, T).
- A flow is maximum $\iff$ no augmenting path exists.
- $\max |f| = \min c(S, T)$ (max-flow min-cut theorem).
- If capacities are integral, there exists an integral max flow.

8.4 Ford-Fulkerson algorithm

Ford-Fulkerson

```
1. Initialize $f(u,v) = 0$ for all arcs.
2. While there exists an augmenting path $P$ from $s$ to $t$ in $G_f$:
    (a) Compute $\delta = \min_{(u,v)\in P} c_f(u,v)$.
    (b) For each arc $(u,v) \in P$: update flow along $P$ by $\delta$.
3. Return $f$.
```

Termination of Ford-Fulkerson

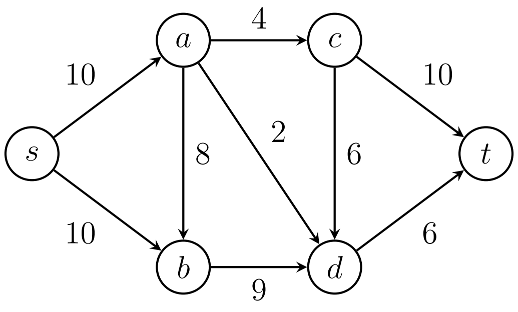
With irrational capacities, Ford-Fulkerson may not terminate. With integer capacities, the complexity is $\mathcal{O}(|A| \cdot |f^*|)$ where $|f^*|$ is the max flow value, which can be very large.

8.5 Edmonds-Karp algorithm

Theorem 8.7 (Edmonds-Karp). *The Edmonds-Karp algorithm is Ford-Fulkerson with BFS (shortest augmenting path in terms of number of arcs). Its complexity is* $\mathcal{O}(|V| \cdot |A|^2)$.

Proof. One shows that the BFS distance from s to any vertex v in G_f never decreases across augmentations. Since each augmentation saturates at least one *critical* arc, and each arc can become critical at most $|V|/2$ times, the total number of augmentations is $\mathcal{O}(|V| \cdot |A|)$. Each BFS costs $\mathcal{O}(|A|)$. □

Example 8.8 (Maximum flow). Consider the network with source s and sink t:



Applying Edmonds-Karp yields a maximum flow of value 16.

8.6 Minimum cost flows

Definition 8.9 (Minimum cost flow problem). Given a network $G = (V, A)$ with capacities c, unit costs $w : A \to \mathbb{R}$, and supply/demand $b : V \to \mathbb{R}$ with $\sum_v b(v) = 0$:

$$\min \sum_{(u,v)\in A} w(u,v) f(u,v) \quad \text{s.t.} \quad \begin{cases} 0 \le f(u,v) \le c(u,v) & \forall (u,v) \in A, \\ \displaystyle\sum_u f(u,v) - \sum_w f(v,w) = b(v) & \forall v \in V. \end{cases}$$

Theorem 8.10 (Optimality condition). *A feasible flow f has minimum cost if and only if there is no **negative-cost cycle** in the residual graph G_f.*

8.7 Applications

8.7.1 Bipartite matching

Proposition 8.11 (Matching via max flow)**.** The **maximum matching** in a bipartite graph $G = (U \cup W, E)$ reduces to max flow: add source s connected to all of U and sink t connected to all of W, with all capacities equal to 1. The max flow value equals the maximum matching size.

Theorem 8.12 (König's theorem)**.** *In a bipartite graph, the size of a maximum matching equals the size of a minimum vertex cover.*

Proposition 8.13 (Menger's theorem)**.** The maximum number of edge-disjoint paths from s to t equals the minimum number of edges whose removal disconnects s from t.

8.8 Python implementation

Edmonds-Karp implementation

```python
from collections import deque

def bfs(graph, s, t, parent):
    """BFS in the residual graph."""
    visited = {s}
    queue = deque([s])
    while queue:
        u = queue.popleft()
        for v in graph[u]:
            if v not in visited and graph[u][v] > 0:
                visited.add(v)
                parent[v] = u
                if v == t:
                    return True
                queue.append(v)
    return False

def edmonds_karp(graph, s, t):
    """Maximum flow via Edmonds-Karp."""
    max_flow = 0
    while True:
        parent = {}
        if not bfs(graph, s, t, parent):
            break
        delta = float('inf')
        v = t
        while v != s:
            u = parent[v]
            delta = min(delta, graph[u][v])
            v = u
        v = t
        while v != s:
            u = parent[v]
            graph[u][v] -= delta
            graph[v][u] += delta
            v = u
        max_flow += delta
    return max_flow
```

8.9 Exercises

Exercise 8.1 (⋆ – Maximum flow)**.** Find the maximum flow in the network with s, a, b, t and arcs $s \to a$ (5), $s \to b$ (3), $a \to b$ (2), $a \to t$ (4), $b \to t$ (6). Also find a minimum cut.

Exercise 8.2 (⋆⋆ – Edmonds-Karp trace)**.** Detail every iteration of Edmonds-Karp on the 6-vertex example network. Show the residual graph at each step.

Exercise 8.3 (⋆⋆ – Bipartite matching)**.** Four employees $\{E_1, E_2, E_3, E_4\}$ can fill positions $\{P_1, P_2, P_3, P_4\}$ with qualifications: E_1: P_1, P_2; E_2: P_2, P_3; E_3: P_1, P_3, P_4; E_4: P_3. Model as a flow problem and find a maximum matching.

Exercise 8.4 (⋆⋆⋆ – Minimum cost flow)**.** A company ships goods from 2 factories to 3 warehouses. Formulate as a minimum cost flow problem and solve.

Exercise 8.5 (⋆⋆⋆ – Menger's theorem)**.** Show that Menger's theorem follows from max-flow min-cut. Apply to a 6-node telecommunications reliability problem.

Chapter 9

Integer Programming

9.1 Introduction

In linear programming, optimal solutions live in a continuous world: one can assign 3.7 trucks to a route. But in the real world, a truck is an integer. You cannot cut an airplane in half to serve two routes. This integrality constraint, seemingly innocuous, radically transforms the nature of the problem: integer programming is NP-hard in general, and the solution methods are fundamentally different. The *branch and bound* method, developed by Ailsa Land and Alison Doig in 1960, intelligently explores a tree of subproblems; *Gomory cuts* (1958) add constraints to tighten the continuous relaxation; *branch and cut* combines both. These algorithms are at the heart of modern solvers (CPLEX, Gurobi) that solve industrial problems with millions of variables.

Intuition

Continuous linear programming may produce fractional solutions (e.g., build 3.7 tables). Naively rounding is generally neither optimal nor feasible. Integer programming provides systematic methods to find the best integer solution.

9.2 Formulation

Definition 9.1 (Integer linear program)**.** An **integer linear program** (ILP) is:

$$\min\ c^\top x \quad \text{s.t.} \quad Ax \le b, \quad x \in \mathbb{Z}_+^n.$$

If only some variables are integer, it is a **mixed-integer program** (MIP). If all variables are binary ($x \in \{0,1\}^n$), it is a **binary program**.

Definition 9.2 (LP relaxation)**.** The **LP relaxation** of an ILP is the linear program obtained by replacing $x \in \mathbb{Z}_+^n$ with $x \in \mathbb{R}_+^n$. Its optimal value provides a **lower bound** (for minimization) on the ILP optimum.

Proposition 9.3 (Relaxation bound)**.** Let z_{LP}^* and z_{IP}^* be the LP and IP optima. Then $z_{\text{LP}}^* \le z_{\text{IP}}^*$. If the LP optimum is integral, it is also optimal for the ILP.

CHAPTER 9. INTEGER PROGRAMMING

Common binary modeling tricks

- **Selection**: $x_j = 1$ if item j is selected.
- **Implication**: $x_i = 1 \Rightarrow x_j = 1$ modeled as $x_i \le x_j$.
- **Disjunction**: at least one of $x_1, \dots, x_k$: $x_1 + \dots + x_k \ge 1$.
- **Big-M**: $y \le Mx$ forces $y = 0$ when $x = 0$.

9.3 Branch and Bound

Definition 9.4 (Branch and Bound)**.** **Branch and Bound** (B&B) solves an ILP by implicit enumeration over a search tree. At each node, the LP relaxation is solved and pruning rules eliminate suboptimal branches.

Branch and Bound

1. Solve the LP relaxation of the original problem.
2. If the solution is integral, it is optimal. **Stop**.
3. Otherwise, choose a fractional variable x_j (value $\bar{x}_j$).
4. **Branching**: create two subproblems:
   - Left branch: add $x_j \le \lfloor \bar{x}_j \rfloor$.
   - Right branch: add $x_j \ge \lceil \bar{x}_j \rceil$.
5. **Bounding**: solve the LP relaxation of each subproblem.
6. **Pruning**: eliminate a node if:
   - the subproblem is infeasible,
   - its lower bound $\ge$ best known solution (incumbent),
   - its solution is integral (update incumbent).
7. Repeat until all nodes are pruned.

Example 9.5 (Branch and Bound)**.** Consider the ILP:

$$\max\ 5x_1 + 4x_2 \quad \text{s.t.} \quad 6x_1 + 4x_2 \le 24, \quad x_1 + 2x_2 \le 6, \quad x_1, x_2 \in \mathbb{Z}_+.$$

The LP relaxation gives $x_1 = 3.6$, $x_2 = 0.6$, $z = 20.4$. Branching on x_1:

- Branch $x_1 \le 3$: LP optimum $x_1 = 3, x_2 = 1.5$, $z = 21$.
- Branch $x_1 \ge 4$: LP optimum $x_1 = 4, x_2 = 0$, $z = 20$ (integral).

Continuing on the left branch by branching on x_2: $x_2 \le 1$ gives $z = 19$ (integral); $x_2 \ge 2$ eventually yields $z = 18$. The optimum is $x_1 = 4, x_2 = 0$ with $z^* = 20$.

9.4 Cutting plane methods

Definition 9.6 (Valid inequality)**.** A **valid inequality** (or cut) is a linear constraint satisfied by all feasible integer solutions but violated by the current fractional LP solution.

9.4.1 Gomory cuts

Theorem 9.7 (Gomory cut)**.** *Let $x_i = \bar{b}_i - \sum_j \bar{a}_{ij} x_j$ be a row of the final simplex tableau with $\bar{b}_i \notin \mathbb{Z}$. Then the inequality:*

$$\sum_j \{\bar{a}_{ij}\}\, x_j \geq \{\bar{b}_i\}$$

is a valid cut, where $\{y\} = y - \lfloor y \rfloor$ is the fractional part.

Proof. From the simplex row: $x_i + \sum_j \bar{a}_{ij} x_j = \bar{b}_i$. Decompose into integer and fractional parts:

$$\underbrace{x_i + \sum_j \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor \bar{b}_i \rfloor}_{\in \mathbb{Z}} = \{\bar{b}_i\} - \sum_j \{\bar{a}_{ij}\} x_j.$$

The left side is integer. Since $\{\bar{b}_i\} > 0$ and each $\{\bar{a}_{ij}\} x_j \geq 0$, we obtain $\sum_j \{\bar{a}_{ij}\} x_j \geq \{\bar{b}_i\}$. □

Remark 9.8. In practice, **Branch and Cut** methods combine B&B with cut generation at each node. Modern solvers (Gurobi, CPLEX) use many families of cuts beyond Gomory: mixed-integer rounding, lift-and-project, Chvátal cuts, etc.

9.5 Total unimodularity

Definition 9.9 (Totally unimodular matrix)**.** A matrix $A \in \mathbb{Z}^{m \times n}$ is **totally unimodular** (TU) if every square submatrix has determinant in $\{-1, 0, 1\}$.

Theorem 9.10 (Exact relaxation)**.** *If A is TU and $b \in \mathbb{Z}^m$, then every vertex of the polyhedron $\{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ is integral. Hence the LP relaxation automatically yields an integer solution.*

Proof. Every vertex solves a system $A'x = b'$ where A' is an invertible square submatrix. Since A is TU, $\det(A') \in \{-1, 1\}$, so by Cramer's rule, $x = (A')^{-1} b'$ is integral. □

Example 9.11 (Classical TU matrices)**.**

- The vertex-arc incidence matrix of a directed graph is TU.
- The constraint matrix of the transportation problem is TU.
- Consequence: flow and transportation problems are solved by the simplex and automatically yield integer solutions.

CHAPTER 9. INTEGER PROGRAMMING

9.6 Classical applications

9.6.1 Knapsack problem

Definition 9.12 (0-1 Knapsack)**.** Given n items with values $v_1, \dots, v_n$ and weights $w_1, \dots, w_n$, and a knapsack of capacity W:

$$\max \sum_{j=1}^{n} v_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{n} w_j x_j \le W, \quad x_j \in \{0, 1\}.$$

9.6.2 Traveling salesman problem

Definition 9.13 (TSP formulation)**.** The **Traveling Salesman Problem** (TSP) on n cities:

$$\min \sum_{i \ne j} d_{ij} x_{ij} \quad \text{s.t.} \quad \sum_{j \ne i} x_{ij} = 1 \; \forall i, \quad \sum_{i \ne j} x_{ij} = 1 \; \forall j, \quad x_{ij} \in \{0, 1\},$$

plus subtour elimination constraints (e.g., Miller-Tucker-Zemlin or exponentially many subtour cuts).

TSP complexity

TSP is NP-hard. State-of-the-art solvers (Concorde) use advanced Branch and Cut and can solve instances with several thousand cities to optimality.

9.7 Exercises

Exercise 9.1 ($\star$ – Branch and Bound)**.** Solve by B&B: max $3x_1 + 2x_2$ s.t. $2x_1 + x_2 \le 6$, $x_1 + 3x_2 \le 9$, $x_1, x_2 \in \mathbb{Z}_+$. Draw the search tree.

Exercise 9.2 ($\star\star$ – Gomory cuts)**.** Consider max x_1+x_2 s.t. $3x_1+2x_2 \le 6$, $x_1+4x_2 \le 4$, $x_1, x_2 \ge 0$ integer. Generate a Gomory cut from the final simplex tableau and solve.

Exercise 9.3 ($\star\star$ – Knapsack)**.** Solve the 0-1 knapsack with values $(10, 6, 12, 7)$, weights $(3, 2, 5, 3)$, capacity $W = 8$ by Branch and Bound.

Exercise 9.4 ($\star\star\star$ – TSP formulation)**.** Formulate TSP on 5 cities with MTZ constraints. Solve the LP relaxation and measure the integrality gap.

Exercise 9.5 ($\star\star\star$ – Total unimodularity)**.** Prove that the vertex-arc incidence matrix of a directed graph is TU. Deduce that the minimum cost flow problem always has an integer optimal solution when the data are integral.

Chapter 10

Nonlinear Programming

10.1 Introduction

When the objective function or constraints are nonlinear, we enter the realm of **nonlinear optimization**. This chapter covers optimality conditions, classical descent methods, and approaches for constrained problems.

Intuition

In linear programming, the optimum always lies at a vertex of the polyhedron. In nonlinear optimization, the optimum can be anywhere — inside the feasible region, on its boundary, or even at a saddle point. More sophisticated analytical tools are needed.

10.2 Unconstrained optimization

10.2.1 Optimality conditions

Theorem 10.1 (First-order necessary condition)**.** *If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and x^* is a local minimum, then:*

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

Theorem 10.2 (Second-order sufficient condition)**.** *If f is twice differentiable, $\nabla f(x^*) = 0$, and the Hessian $\nabla^2 f(x^*)$ is* ***positive definite****, then x^* is a* ***strict local minimum****.*

Local vs. global minima

For a nonconvex function, a point satisfying first- and second-order conditions is only a **local** minimum. Only convexity of f guarantees that every local minimum is global.

10.2.2 Gradient descent

Definition 10.3 (Gradient descent)**. Gradient descent** generates a sequence (x_k) by:

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

where $\alpha_k > 0$ is the **step size** (or learning rate).

Theorem 10.4 (Convergence for L-smooth convex functions)**.** *If f is convex and ∇f is L-Lipschitz, gradient descent with constant step $\alpha = 1/L$ satisfies:*

$$f(x_k) - f(x^*) \leq \frac{L \left\| x_0 - x^* \right\|^2}{2k}.$$

The convergence rate is $\mathcal{O}(1/k)$.

Step size selection

- **Constant step**: $\alpha_k = \alpha < 2/L$ (simple, convergent).
- **Exact line search**: $\alpha_k = \arg\min_{\alpha>0} f(x_k - \alpha\nabla f(x_k))$.
- **Backtracking (Armijo)**: reduce α by factor $\beta \in (0,1)$ until $f(x_k - \alpha\nabla f(x_k)) \leq f(x_k) - c\alpha \left\|\nabla f(x_k)\right\|^2$, $c \in (0,1)$.

10.2.3 Newton's method

Definition 10.5 (Newton's method)**. Newton's method** uses second-order information:

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

Theorem 10.6 (Quadratic convergence)**.** *If f is twice continuously differentiable, $\nabla^2 f$ is Lipschitz, and x_0 is sufficiently close to x^* (where $\nabla^2 f(x^*)$ is positive definite), the convergence is **quadratic**:*

$$\left\|x_{k+1} - x^*\right\| \leq C \left\|x_k - x^*\right\|^2.$$

Remark 10.7. Newton's method converges very fast near the solution but requires computing and inverting the Hessian ($\mathcal{O}(n^3)$ per iteration). **Quasi-Newton** methods (BFGS, L-BFGS) approximate the Hessian to reduce cost.

Example 10.8 (Gradient descent vs. Newton)**.** Consider $f(x_1, x_2) = 10x_1^2 + x_2^2$ (ill-conditioned quadratic, $L/\mu = 10$). From $x_0 = (10, 1)^\top$:

- Gradient ($\alpha = 1/L = 1/20$): slow convergence, zigzag trajectory.
- Newton: converges in **one iteration** since f is quadratic.

10.3 Constrained optimization: KKT conditions

Definition 10.9 (Constrained optimization problem)**.**

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad g_i(x) \leq 0,\ i = 1, \dots, m, \qquad h_j(x) = 0,\ j = 1, \dots, p.$$

Theorem 10.10 (Karush-Kuhn-Tucker conditions)**.** *If x^* is a local minimum and a constraint qualification holds (e.g., LICQ), then there exist multipliers $\lambda^* \in \mathbb{R}^m_+$ and $\nu^* \in \mathbb{R}^p$ such that:*

1. ***Stationarity**: $\nabla f(x^*) + \sum_i \lambda_i^* \nabla g_i(x^*) + \sum_j \nu_j^* \nabla h_j(x^*) = 0$.*

2. ***Primal feasibility***: $g_i(x^*) \le 0$, $h_j(x^*) = 0$.
3. ***Dual feasibility***: $\lambda_i^* \ge 0$.
4. ***Complementary slackness***: $\lambda_i^* g_i(x^*) = 0$.

Remark 10.11. For convex problems (convex objective, convex inequality constraints, affine equalities), the KKT conditions are also **sufficient** for global optimality.

10.4 Penalty and barrier methods

10.4.1 Exterior penalty method

Definition 10.12 (Quadratic penalty). Replace the constrained problem by the sequence of unconstrained problems:

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

with $\rho \to +\infty$.

Theorem 10.13 (Penalty convergence). *If $\rho_k \to +\infty$ and x_k is an (approximate) minimizer of the penalized problem with parameter ρ_k, then every accumulation point of (x_k) solves the constrained problem.*

10.4.2 Barrier method (interior point)

Definition 10.14 (Logarithmic barrier). For inequality constraints $g_i(x) \le 0$:

$$\min_x \; f(x) - \frac{1}{t}\sum_{i=1}^{m}\ln(-g_i(x)),$$

with $t \to +\infty$. The barrier prevents leaving the interior of the feasible region.

10.5 Sequential quadratic programming (SQP)

Definition 10.15 (SQP). **Sequential Quadratic Programming** solves a nonlinear constrained problem by solving at each iteration a **quadratic subproblem**:

$$\min_d \; \nabla f(x_k)^\top d + \frac{1}{2}d^\top H_k d \quad \text{s.t.} \quad g_i(x_k) + \nabla g_i(x_k)^\top d \le 0,$$

where H_k approximates the Hessian of the Lagrangian. Then $x_{k+1} = x_k + d_k$.

Remark 10.16. SQP generalizes Newton's method to the constrained case. It is one of the most efficient methods for medium-scale nonlinear optimization, with superlinear convergence near the solution.

10.6 Python implementation

Nonlinear optimization with SciPy

```python
import numpy as np
from scipy.optimize import minimize

# Rosenbrock function
def rosenbrock(x):
    return (1 - x[0])**2 + 100*(x[1] - x[0]**2)**2

x0 = np.array([-1.0, 1.0])

# Quasi-Newton (L-BFGS-B)
res = minimize(rosenbrock, x0, method='L-BFGS-B')
print("L-BFGS-B:", res.x, "in", res.nit, "iterations")

# With constraints (SQP via SLSQP)
cons = ({'type': 'ineq', 'fun': lambda x: x[0] + x[1] - 1})
res_c = minimize(rosenbrock, x0, method='SLSQP',
                 constraints=cons)
print("SLSQP:", res_c.x)
```

10.7 Exercises

Exercise 10.1 (⋆ – KKT conditions)**.** Find the KKT points of: min $x_1^2+x_2^2$ s.t. $x_1+x_2 \geq 1$. Verify that the solution is the global minimum.

Exercise 10.2 (⋆⋆ – Gradient descent)**.** Implement gradient descent with backtracking for $f(x) = x_1^4 + x_2^4 - 2x_1x_2$ from $x_0 = (2,2)^\top$. Plot the trajectory and the decrease of f.

Exercise 10.3 (⋆⋆ – Newton's method)**.** Apply Newton's method to $f(x) = e^{x_1+x_2} + x_1^2 + x_2^2$ from $x_0 = (1,1)^\top$. Compute the first three iterations by hand.

Exercise 10.4 (⋆⋆⋆ – Penalty method)**.** Solve min $x_1^2 + x_2^2$ s.t. $x_1 + x_2 = 1$ by quadratic penalty for $\rho \in \{1, 10, 100, 1000\}$. Observe convergence to the exact solution.

Exercise 10.5 (⋆⋆⋆ – SQP)**.** Formulate the QP subproblem of SQP for: min $(x_1-2)^2 + (x_2-1)^2$ s.t. $x_1^2 + x_2^2 \leq 1$. Perform two iterations from $x_0 = (0,0)^\top$.

Chapter 11

Simulation and Queueing Theory

11.1 Introduction

Simulation is an essential tool in operations research when analytical models are too complex or intractable. This chapter covers discrete event simulation, random variate generation, Monte Carlo estimation, and variance reduction techniques.

Intuition

When a system is too complex for mathematical analysis (queues with priorities, logistics networks, etc.), we *simulate* it: generate random realizations of the system and observe the results statistically. It is like running thousands of virtual experiments.

11.2 Discrete event simulation

Definition 11.1 (Discrete event simulation (DES))**. Discrete event simulation** models a system as a sequence of instantaneous events (customer arrival, service completion, failure, etc.) that modify the system state at discrete time points.

Definition 11.2 (DES components)**.**

- **System state**: variables describing the system at time t (number of customers, server status, etc.).
- **Simulation clock**: current time t.
- **Future event list** (FEL): list of scheduled events, sorted by time.
- **Statistical counters**: accumulators for performance measures.

DES main loop

```
1. Initialize state, clock $t = 0$, and FEL.
2. While the stopping condition is not met:
    (a) Extract the next event $(t_e, \text{type})$ from the FEL.
    (b) Advance the clock: $t \leftarrow t_e$.
    (c) Process the event (update state, schedule new events).
```

CHAPTER 11. SIMULATION AND QUEUEING THEORY

   (d) Update statistical counters.

3. Compute and return performance indicators.

11.3 Random variate generation

Theorem 11.3 (Inverse transform method). *If $U \sim \mathcal{U}(0,1)$ and F is the CDF of a continuous random variable X, then:*

$$X = F^{-1}(U)$$

has the distribution of X.

Proof. For any $x \in \mathbb{R}$: $\mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x)$, since $U \sim \mathcal{U}(0,1)$ and F is nondecreasing. □

Example 11.4 (Exponential distribution). For $X \sim \text{Exp}(\lambda)$, $F(x) = 1 - e^{-\lambda x}$, so $F^{-1}(u) = -\frac{1}{\lambda}\ln(1-u)$. We generate:

$$X = -\frac{1}{\lambda}\ln(U), \quad U \sim \mathcal{U}(0,1).$$

(We use $\ln(U)$ instead of $\ln(1-U)$ since U and $1-U$ have the same distribution.)

Generating classical distributions

- **Exponential**(λ): $X = -\frac{1}{\lambda}\ln(U)$.
- **Bernoulli**(p): $X = \mathbb{1}_{U \leq p}$.
- **Poisson**(λ): count U_i until $\prod U_i < e^{-\lambda}$.
- **Normal** (Box-Muller): $X = \sqrt{-2\ln U_1}\cos(2\pi U_2)$.

Theorem 11.5 (Acceptance-rejection method). *Let f be the target density and g a proposal density with $f(x) \leq Mg(x)$ for all x. To generate $X \sim f$:*

1. *Generate $Y \sim g$ and $U \sim \mathcal{U}(0,1)$.*
2. *If $U \leq \frac{f(Y)}{Mg(Y)}$, accept $X = Y$; otherwise reject and repeat.*

The acceptance rate is $1/M$.

11.4 Monte Carlo estimation

Definition 11.6 (Monte Carlo estimator). To estimate $\theta = \mathbb{E}[h(X)]$, the Monte Carlo estimator is:

$$\hat{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} h(X_i),$$

where $X_1, \ldots, X_n$ are i.i.d. copies of X.

Theorem 11.7 (Properties of the MC estimator)**.** *1.* ***Unbiased:*** $\mathbb{E}[\hat{\theta}_n] = \theta$.

2. ***Consistency:*** $\hat{\theta}_n \xrightarrow{a.s.} \theta$ *(strong law of large numbers).*

3. ***CLT:*** $\sqrt{n}(\hat{\theta}_n - \theta) \xrightarrow{d} \mathcal{N}(0, \sigma^2)$ *where* $\sigma^2 = \mathrm{Var}(h(X))$.

4. ***95% confidence interval:*** $\hat{\theta}_n \pm 1.96 \cdot \hat{\sigma}/\sqrt{n}$.

Example 11.8 (Estimating π)**.** Generate points (X, Y) uniformly in $[0, 1]^2$ and compute the proportion $\hat{p}$ satisfying $X^2 + Y^2 \leq 1$. Then $\hat{\pi} = 4\hat{p}$.

11.5 Variance reduction techniques

Definition 11.9 (Antithetic variates)**.** Instead of n independent samples $U_1, \ldots, U_n$, use the pairs $(U_i, 1-U_i)$. If h is monotone, the negative covariance between $h(U)$ and $h(1-U)$ reduces the variance of the mean.

Proposition 11.10 (Antithetic variance reduction)**.** If h is monotone and $U \sim \mathcal{U}(0, 1)$:

$$\mathrm{Var}\left(\frac{h(U) + h(1-U)}{2}\right) \leq \frac{\mathrm{Var}(h(U))}{2}.$$

Definition 11.11 (Control variates)**.** Let Y be an auxiliary variable with known mean $\mathbb{E}[Y] = \mu_Y$. The control variate estimator is:

$$\hat{\theta}_c = \hat{\theta}_n - c(\bar{Y}_n - \mu_Y),$$

with optimal coefficient $c^* = \mathrm{Cov}(h(X), Y)/\mathrm{Var}(Y)$.

Definition 11.12 (Importance sampling)**.** To estimate $\theta = \mathbb{E}_f[h(X)]$, sample from a proposal distribution g and reweight:

$$\hat{\theta}_{\mathrm{IS}} = \frac{1}{n}\sum_{i=1}^{n} h(X_i)\frac{f(X_i)}{g(X_i)}, \quad X_i \sim g.$$

The optimal proposal is $g^*(x) \propto |h(x)|\, f(x)$.

11.6 Applications in operations research

- **Queueing systems**: simulation of M/M/1, M/G/k with priorities and breakdowns.
- **Inventory management**: (s, S) policy with random demand.
- **Logistics**: supply chain simulation.
- **Finance**: option pricing via Monte Carlo.
- **Reliability**: estimating failure probability of complex systems.

11.7 Python implementation

M/M/1 discrete event simulation

```python
import numpy as np
import heapq

def simulate_mm1(lam, mu, n_clients):
    """Simulate an M/M/1 queue."""
    t = 0.0
    fel = []  # future event list (time, type)
    n_in_system = 0
    arrivals = 0
    total_wait = 0.0
    heapq.heappush(fel,
        (np.random.exponential(1/lam), 'A'))
    while arrivals < n_clients:
        event_time, event_type = heapq.heappop(fel)
        t = event_time
        if event_type == 'A':  # Arrival
            arrivals += 1
            n_in_system += 1
            if n_in_system == 1:
                service = np.random.exponential(1/mu)
                heapq.heappush(fel, (t + service, 'D'))
            heapq.heappush(fel,
                (t + np.random.exponential(1/lam), 'A'))
        else:  # Departure
            n_in_system -= 1
            if n_in_system > 0:
                service = np.random.exponential(1/mu)
                heapq.heappush(fel, (t + service, 'D'))
    return t / arrivals
```

11.8 Exercises

Exercise 11.1 ($\star$ – Inverse transform)**.** Generate 1000 realizations of an exponential distribution with parameter $\lambda = 2$ using the inverse transform method. Compare the histogram to the theoretical density.

Exercise 11.2 ($\star\star$ – M/M/1 simulation)**.** Simulate an M/M/1 queue with $\lambda = 4$ and $\mu = 5$ for 10,000 customers. Estimate the mean waiting time and compare to the theoretical value $W = \frac{1}{\mu-\lambda}$.

Exercise 11.3 ($\star\star$ – Monte Carlo integration)**.** Estimate $I = \int_0^1 e^{-x^2}\,dx$ by Monte Carlo with $n = 10^4$. Construct a 95% confidence interval. Compare with antithetic variates.

Exercise 11.4 ($\star\star\star$ – Control variates)**.** Estimate $\mathbb{E}[e^U]$ where $U \sim \mathcal{U}(0,1)$. Use $Y = U$ as a control variate. Compute the optimal coefficient c^* and the theoretical variance reduction.

Exercise 11.5 ($\star\star\star$ – Acceptance-rejection)**.** Generate realizations of a Beta$(2, 5)$ distribution by acceptance-rejection with $g = \mathcal{U}(0, 1)$. Compute the constant M and the acceptance rate.

CHAPTER 11. SIMULATION AND QUEUEING THEORY

Bibliography

[1] Hillier, F.S. and Lieberman, G.J., *Introduction to Operations Research*, 11th ed., McGraw-Hill, 2021.

[2] Bertsimas, D. and Tsitsiklis, J.N., *Introduction to Linear Optimization*, Athena Scientific, 1997.

[3] Gondran, M. and Minoux, M., *Graphes et Algorithmes*, 4th ed., Lavoisier, 2009.