Optimization: Principles and Algorithms

Michel Bierlaire

Companion website: **optimizationprinciplesalgorithms.com**

EPFL Press

Optimization: Principles and Algorithms

Michel Bierlaire

Every engineer and decision scientist must have a good mastery of optimization, an essential element in their toolkit. Thus, this articulate introductory textbook will certainly be welcomed by students and practicing professionals alike. Drawing from his vast teaching experience, the author skillfully leads the reader through a rich choice of topics in a coherent, fluid and tasteful blend of models and methods anchored on the underlying mathematical notions (only prerequisites: first year calculus and linear algebra). Topics range from the classics to some of the most recent developments in smooth unconstrained and constrained optimization, like descent methods, conjugate gradients, Newton and quasi-Newton methods, linear programming and the simplex method, trust region and interior point methods. Furthermore elements of discrete and combinatorial optimization like network optimization, integer programming and heuristic local search methods are also presented.

This book presents optimization as a modeling tool that beyond supporting problem formulation plus design and implementation of efficient algorithms, also is a language suited for interdisciplinary human interaction. Readers further become aware that while the roots of mathematical optimization go back to the work of giants like Newton, Lagrange, Cauchy, Euler or Gauss, it did not become a discipline on its own until World War Two. Also that its present momentum really resulted from its symbiosis with modern computers, which made it possible to routinely solve problems with millions of variables and constraints.

With his witty, entertaining, yet precise style, Michel Bierlaire captivates his readers and awakens their desire to try out the presented material in a creative mode. One of the outstanding assets of this book is the unified, clear and concise rendering of the various algorithms, which makes them easily readable and translatable into any high level programming language. This is an addictive book that I am very pleased to recommend.

Prof. Thomas M. Liebling

Michel Bierlaire holds a PhD in mathematics from the University of Namur, Belgium. He is full professor at the School of Architecture, Civil and Environmental Engineering at the Ecole Polytechnique Fédérale de Lausanne, Switzerland. He has been teaching optimization and operations research at the EPFL since 1998.

ISBN 978-2-88915-279-7

9 782889 152797 >

EPFL Press

Optimization:
Principles and Algorithms

Cover Illustrations: *Abstract cubes background*, © ProMotion and *Architectural detail of modern roof structure*, © Lucian Milasan – Fotolia.com

Optimization: Principles and Algorithms

Michel Bierlaire

EPFL Press

This book is published under the editorial direction of
Professor Robert Dalang (EPFL).

The publisher and author express their thanks to the Ecole Polytechnique Fédérale de Lausanne (EPFL) for its generous support towards the publication of this book.

EPFL Press

The EPFL Press is the English-language imprint of the Foundation of the Presses polytechniques et universitaires romandes (PPUR). The PPUR publishes mainly works of teaching and research of the Ecole polytechnique fédérale de Lausanne (EPFL), of universities and other institutions of higher education.

Presses polytechniques et universitaires romandes, EPFL – Rolex Learning Center,
Post office box 119, CH-1015 Lausanne, Switzerland
E-mail : ppur@epfl.ch
Phone : 021 / 693 21 30
Fax : 021 / 693 40 27

www.epflpress.org

ISBN 978-2-88915-279-7

ISBN 978-2-940222-78-0 (EPFL Press)
ISBN 978-1-4822-0345-5 (CRC Press)

Printed in Italy

To Patricia

Preface

Optimization algorithms are important tools for engineers, but difficult to use. In fact, none of them is universal, and a good understanding of the different methods is necessary in order to identify the most appropriate one in the context of specific applications. Designed to teach undergraduate engineering students about optimization, this book also provides professionals employing optimization methods with significant elements to identify the methods that are appropriate for their applications, and to understand the possible failures of certain methods on their problem. The content is meant to be formal, in the sense that the results presented are proven in detail, and all described algorithms have been implemented and tested by the author. In addition, the many numeric and graphic illustrations constitute a significant base for understanding the methods.

The book features eight parts. The first part focuses on the formulation and the analysis of the optimization problem. It describes the modeling process that leads to an optimization problem, as well as the transformations of the problem into an equivalent formulation. The properties of the problem and corresponding hypotheses are discussed independently of the algorithms. Subsequently, the optimality conditions, the theoretical foundations that are essential for properly mastering the algorithms, are analyzed in detail in Part II. Before explaining the methods for unconstrained continuous optimization in Part IV, algorithms for solving systems of non linear equations, based on Newton's method, are described in Part III. The algorithms for constrained continuous optimization constitute the fifth part. Part VI addresses optimization problems based on network structures, elaborating more specifically on the shortest path problem and the maximum flow problem. Discrete optimization problems, where the variables are constrained to take integer values, are introduced in Part VII, where both exact methods and heuristics are presented. The last part is an appendix containing the definitions and theoretical results used in the book.

Several chapters include exercises. Chapters related to algorithms also propose projects involving an implementation. It is advisable to use a mathematical programming language, such as Octave (Eaton, 1997) or Matlab (Moled, 2004). If a language such as C, C++, or Fortran is preferred, a library managing the linear algebra, such as LAPACK (Anderson et al., 1999), can be useful. When time limits do not allow a full implementation of the algorithms by the students, the teaching assistant may prepare the general structure of the program, including the implementation of optimization problems (objective function, constraints, and derivatives) in order for the

students to focus on the key points of the algorithms. The examples described in detail in this book enable the implementations to be verified.

Optimization is an active research field, that is, permanently stimulated by the needs of modern applications. Many aspects are absent from this book. "Every choice entails the rejection of what might have been better," said Andre Gide. Among the important topics not covered in this book, we can mention

- the numerical aspects related to the implementation, particularly important and tricky in this area (see, e.g., Dennis and Schnabel, 1996);
- the convergence analysis of the algorithms (see, e.g., Ortega and Rheinboldt, 1970, Dennis and Schnabel, 1996, Conn et al., 2000, and many others);
- automatic differentiation, allowing the automatic generation of analytical derivatives of a function (Griewank, 1989, Griewank, 2000);
- techniques to deal with problems of large size, such as updates with limited memory (Byrd et al., 1994) or partially separable functions (Griewank and Toint, 1982);
- homotopy methods (Forster, 1995);
- semidefinite optimization (Gärtner and Matousek, 2012);
- the vast field of convex optimization (Ben-Tal and Nemirovski, 2001, Boyd and Vandenberghe, 2004, Calafiore and El Ghaoui, 2014);
- stochastic programming (Birge and Louveaux, 1997, Shapiro et al., 2014);
- robust optimization (Ben-Tal et al., 2009), where uncertainty of the data is explicitly accounted for.

This book is the fruit of fifteen years of teaching optimization to undergraduate students in engineering at the Ecole Polytechnique Fédérale of Lausanne. Except for the parts of the book related to networks and discrete optimization that are new, the material in the book has been translated from Bierlaire (2006), a textbook in French. The main sources of inspiration are the following books:

- Bertsimas and Weismantel (2005)
- de Werra et al. (2003)
- Bonnans et al. (2003)
- Conn et al. (2000)
- Nocedal and Wright (1999)
- Bertsekas (1999)
- Wolsey (1998)
- Bertsekas (1998)
- Bertsimas and Tsitsiklis (1997)
- Wright (1997)
- Dennis and Schnabel (1996)
- Ahuja et al. (1993).

There are many books on optimization. Within the vast literature, we may cite the following books in English: Beck (2014), Calafiore and El Ghaoui (2014), Ben-Tal et al. (2009), Boyd and Vandenberghe (2004), Ben-Tal and Nemirovski (2001), Conn et al. (2000), Kelley (1999), Kelley (1995), Birge and Louveaux (1997), Dennis and Schnabel (1996), Axelsson (1994), Polyak (1987), Scales (1985), Coleman (1984), McCormick (1983), Gill et al. (1981), Fletcher (1980), Fletcher (1981), Ortega and Rheinboldt (1970), and Fiacco and McCormick (1968). Several books are also available in French. Among them, we can cite Korte et al. (2010), Dodge (2006), Cherruault (1999), Breton and Haurie (1999), Hiriart-Urruty (1998), Bonnans et al. (1997), and Gauvin (1992).

The bibliographic source for the biographies of Jacobi, Hesse, Lagrange, Fermat, Newton, Al Khwarizmi, Cauchy, and Lipschitz is Gillispie (1990). The information about Tucker (Gass, 2004), Dantzig (Gass, 2003), Little (Larson, 2004), Fulkerson (Bland and Orlin, 2005), and Gomory (Johnson, 2005) come from the series *IFORS' Operational Research Hall of Fame*. The source of information for Euler is his biography by Finkel (1897). Finally, the information on Davidon was taken from his web page

`www.haverford.edu/math/wdavidon.html`

and from Nocedal and Wright (1999). The selection of persons described in this work is purely arbitrary. Many other mathematicians have contributed significantly to the field of optimization and would deserve a place herein. I encourage the reader to read, in particular, the articles of the series *IFORS' Operational Research Hall of Fame* published in `International Transactions in Operational Research`, expressly dealing with Morse, Bellman, Kantorovich, Erlang, and Kuhn.

Online material

The book has a companion website:

`www.optimizationprinciplesalgorithms.com`

The algorithms presented in the book are coded in GNU Octave, a high-level interpreted language (www.gnu.org/software/octave), primarily intended for numerical computations. The code for the algorithms, as well as examples of optimization problems, are provided. All the examples have been run on GNU Octave, version 3.8.1. on a MacBook Pro running OS X Yosemite 10.10.2. If you use these codes, Michel Bierlaire, the author, grants you a nonexclusive license to run, display, reproduce, distribute and prepare derivative works of this code. The code has not been thoroughly tested under all conditions. The author, therefore, does not guarantee or imply its reliability, serviceability, or function. The author provides no program services for the code.

Acknowledgments

I would like to thank EPFL Press for their support in the translation of the French version and the publication of this book. The financial support of the School of Architecture, Civil and Environmental Engineering at EPFL is highly appreciated.

I am grateful to my PhD advisor, Philippe Toint, who passed on his passion for optimization to me. Among the many things he taught me, the use of geometric interpretations of certain concepts, especially algorithmic ones, proved particularly useful for my research and my teaching. I hope that they will now benefit the readers of this book.

I also thank Thomas Liebling who put his trust in me by welcoming me into his group at EPFL back in 1998. Among other things, he asked me to take care of the optimization and operations research courses, which year after year have enabled me to build the material that is gathered in this book. But I am especially grateful for his friendship, and all the things that he has imparted to me.

I would like to thank my doctoral students, teaching assistants, and postdocs, not only for the pleasure of working with them, but also for their valuable help in the optimization course over the years, and their comments on various versions of this book.

Earlier versions of the manuscript were carefully read by the members of the Transport and Mobility Laboratory at the EPFL, and in particular Stefan Binder, Anna Fernandez Antolin, Flurin Hänseler, Yousef Maknoon, Iliya Markov, Marija Nikolic, Tomas Robenek, Riccardo Scarinci, and Shadi Sharif. Thomas Liebling provided a great deal of comments, with excellent suggestions to improve the style and the content of the book. He caught several errors and imprecisions. It greatly improved the quality of the text. If there are still mistakes (and there are always some that escape scrutinity), I clearly take full responsibility. Errata will be published on the website as mistakes are caught.

I am so proud of my children, Aria and François, who have recently started the difficult challenge of obtaining a university degree. Finally, I dedicate this book to Patricia. I am really lucky to know such a wonderful person. Her love is a tremendous source of joy and energy for me.

Contents

Part I

Formulation and analysis of the problem

No one trusts a model except the man who wrote it; everyone trusts an observation, except the man who made it.

Harlow Shapley

Modeling is a necessity before any optimization process. How do we translate a specific problem statement into a mathematical formulation that allows its analysis and its resolution? In this first part, we propose modeling in the field of optimization. Then, we identify the properties of the optimization problem that are useful in the development of the theory and algorithms.

Chapter 1

Formulation

Contents

1.1 Modeling

The need to optimize is a direct result of the need to organize. Optimizing consists in identifying an optimal configuration or an optimum in a system in the broadest sense of the term. We use here Definition 1.1, given by Oxford University Press (2013).

Definition 1.1 (Optimum)**.** (In Latin *optimum*, the best). The most favorable or advantageous condition, value, or amount, especially under a particular set of circumstances.

As part of a scientific approach, this definition requires some details. How can we judge that the condition is favorable, and how can we formally describe the set of circumstances?

The answer to these questions is an essential step in any optimization: mathematical modeling (Definition 1.2 by Oxford University Press, 2013). The modeling process consists of three steps:

1. The identification of the decision variables. They are the components of the system that describe its state, and that the analyst wants to determine. Or, they represent configurations of the system that are possible to modify in order to improve its performance. In general, if these variables are n in number, they are represented by a (column)[1] vector of $\mathbb{R}^n$, often denoted by $x = (x_1 \dots x_n)^\top$, i.e.,

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

   In practice, this step is probably the most complicated and most important. The most complicated because only experience in modeling and a good knowledge of the specific problem can guide the selection. The most important because the rest of the process depends on it. An inappropriate selection of decision variables can generate an optimization problem that is too complicated and impossible to solve.
2. The description of the method to assess the state of the system in question, given a set of decision variables. In this book, we assume that the person performing the modeling is able to identify a formula, a function, providing a measure of the state of the system, a value that she wants to make the smallest or largest possible. This function, called *objective function*, is denoted by f and the aforementioned measure obtained for the decision variables x is a real number denoted by $f(x)$.
3. The mathematical description of the circumstances or constraints, specifying the values that the decision variables can take.

Definition 1.2 (Mathematical model)**.** Mathematical representation of a physical, economic, human phenomenon, etc., conducted in order to better study it.

The modeling process is both exciting and challenging. Indeed, there is no universal recipe, and the number of possible models for a given problem is only limited by the imagination of the modeler. However, it is essential to master optimization tools and to understand the underlying assumptions in order to develop the adequate model for the analysis in question. In this chapter, we provide some simple examples of modeling exercises. In each case, we present a possible modeling.

1.1.1 Projectile

We start with a simple problem. A projectile is launched vertically at a rate of 50 meters per second, in the absence of wind. After how long and at which altitude does

[1] See Appendix A about the mathematical notations used throughout the book.

it start to fall? Note that, in this case, the decision variables represent a state of the system that the analyst wants to calculate.

The modeling process consists of three steps.

Decision variables A single decision variable is used. Denoted by x, it represents the number of seconds from the launch of the projectile. Note that in this case, the decision variables represent a state of the system that the analyst wants to calculate.

Objective function We seek to identify the maximum altitude reached by the object. We must thus express the altitude as a function of the decision variable. Since we are dealing with the uniformly accelerating movement of an object subjected to gravity, we have

$$f(x) = -\frac{g}{2}x^2 + v_0 x + x_0 = -\frac{9.81}{2}x^2 + 50x\,,$$

where $g = 9.81$ is the acceleration experienced by the projectile, $v_0 = 50$ is its initial velocity, and $x_0 = 0$ is its initial altitude.

Constraints Time only goes forward. Therefore, we impose $x \geq 0$.

We obtain the optimization problem

$$\max_{x \in \mathbb{R}} -\frac{9.81}{2}x^2 + 50x\,, \tag{1.1}$$

subject to (s.t.)

$$x \geq 0. \tag{1.2}$$

1.1.2 Swisscom

The company Swisscom would like to install an antenna to connect four important new customers to its network. This antenna must be as close as possible to each client, giving priority to the best customers. However, to avoid the proliferation of telecommunication antennas, the company is not allowed to install the new antenna at a distance closer than 10 km from the other two antennas, respectively located at coordinates $(-5, 10)$ and $(5, 0)$ and represented by the symbol ✪ in Figure 1.1. The coordinates are expressed in kilometers from Swisscom's headquarters. Swisscom knows the geographic situation of each customer as well as the number of hours of communication that the customer is supposed to consume per month. This data is listed in Table 1.1. At which location should Swisscom install the new antenna?

The modeling process consists of three steps.

Decision variables Swisscom must identify the ideal location for the antenna, i.e., the coordinates of that location. We define two decision variables x_1 and x_2 representing these coordinates in a given reference system.

Objective function The distance $d_i(x_1, x_2)$ between a customer i located at the coordinates (a_i, b_i) and the antenna is given by

$$d_i(x_1, x_2) = \sqrt{(x_1 - a_i)^2 + (x_2 - b_i)^2}\,. \tag{1.3}$$

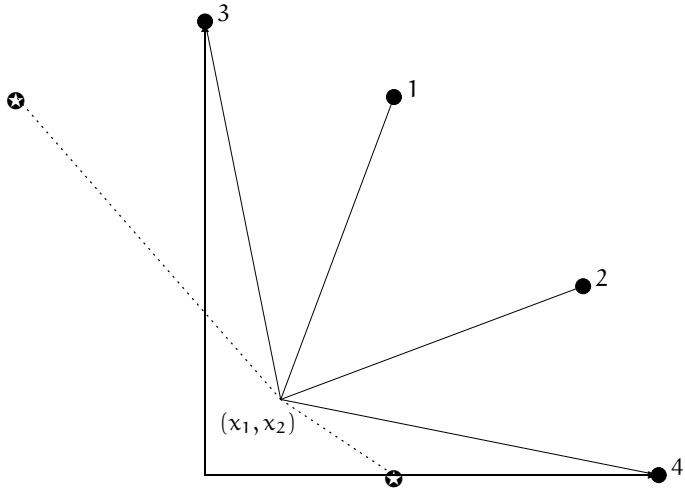


Figure 1.1: Swisscom problem

Table 1.1: Data for Swisscom customers

Customer	Coord.	Hours
1	$(5, 10)$	200
2	$(10, 5)$	150
3	$(0, 12)$	200
4	$(12, 0)$	300

To take into account the communication time, we measure the sum of the distances weighted by the number of consumed hours:

$$\begin{aligned} f(x_1, x_2) &= 200\, d_1(x_1, x_2) + 150\, d_2(x_1, x_2) \\ &\quad + 200\, d_3(x_1, x_2) + 300\, d_4(x_1, x_2) \\ &= 200\sqrt{(x_1 - 5)^2 + (x_2 - 10)^2} \\ &\quad + 150\sqrt{(x_1 - 10)^2 + (x_2 - 5)^2} \\ &\quad + 200\sqrt{x_1^2 + (x_2 - 12)^2} \\ &\quad + 300\sqrt{(x_1 - 12)^2 + x_2^2}\,. \end{aligned} \tag{1.4}$$

Constraints The constraints on the distances between the antennas can be expressed as

$$\sqrt{(x_1 + 5)^2 + (x_2 - 10)^2} \geq 10 \tag{1.5}$$

and

$$\sqrt{(x_1 - 5)^2 + x_2^2} \geq 10\,. \tag{1.6}$$

We can combine the various stages of modeling to obtain the following optimization problem:

$$\begin{aligned}\min_{x\in\mathbb{R}^2} f(x_1,x_2) = {} & 200\sqrt{(x_1-5)^2+(x_2-10)^2} \\ & + 150\sqrt{(x_1-10)^2+(x_2-5)^2} \\ & + 200\sqrt{x_1^2+(x_2-12)^2} \\ & + 300\sqrt{(x_1-12)^2+x_2^2}\end{aligned} \tag{1.7}$$

subject to (s.t.)

$$\begin{aligned}&\sqrt{(x_1+5)^2+(x_2-10)^2} \geq 10 \\ &\sqrt{(x_1-5)^2+(x_2-10)^2} \geq 10\,.\end{aligned} \tag{1.8}$$

1.1.3 Château Laupt-Himum

The Château Laupt-Himum produces rosé wine and red wine by buying grapes from local producers. This year they can buy up to one ton of Pinot (a red grape) from a winegrower, paying €3 per kilo. They can then vinify the grapes in two ways: either as a white wine to obtain a rosé wine or as a red wine to get Pinot Noir, a full-bodied red wine. The vinification of the rosé wine costs €2 per kilo of grapes, while that of the Pinot Noir costs €3.50 per kilo of grapes.

In order to take into account economies of scale, the Château wants to adjust the price of its wine to the quantity produced. The price for one liter of the rosé is €15 minus a rebate of €2 per hundred liters produced. Thus, if they produce 100 liters of rosé, they sell it for €13 per liter. If they produce 200, they sell it for €11 per liter. Similarly, they sell the Pinot Noir at a price of €23 per liter, minus a rebate of €1 per hundred liters produced.

How should the Château Laupt-Himum be organized in order to maximize its profit, when a kilo of grapes produces 1 liter of wine?

The modeling process consists of three steps.

Decision variables The strategy of the Château Laupt-Himum is to decide how many liters of rosé wine and Pinot Noir to produce each year, and the number of kilos of grapes to buy from the winegrower. Therefore, we define three decision variables:

- x_1 is the number of liters of rosé wine to produce each year,
- x_2 is the number of liters of Pinot Noir to produce,
- x_3 is the number of kilos of grapes to buy.

Objective function The objective of the Château Laupt-Himum is to maximize its profit. This gain is the income from the wine sales minus the costs.

Each liter of rosé wine that is sold gives (in €)

$$15 - \frac{2}{100}\,x_1$$

taking into account the reduction. Similarly, each liter of Pinot Noir gives (in €)

$$23 - \frac{1}{100}\,x_2\,.$$

The revenues corresponding to the production of x_1 liters of rosé wine and x_2 liters of Pinot Noir are equal to

$$x_1\left(15 - \frac{2}{100}\,x_1\right) + x_2\left(23 - \frac{1}{100}\,x_2\right).$$

It costs $3x_3$ to purchase the grapes. To produce a liter of wine, they need one kilo of vinified grapes, which costs €2 for the rosé and €3.50 for the Pinot Noir. The total costs are

$$2x_1 + 3.5\,x_2 + 3x_3\,.$$

The objective function that the Château Laupt-Himum should maximize is

$$x_1\left(15 - \frac{2}{100}\,x_1\right) + x_2\left(23 - \frac{1}{100}\,x_2\right) - (2x_1 + 3.5\,x_2 + 3x_3)\,.$$

Constraints The Château cannot buy more than 1 ton of grapes from the winegrower, i.e.,

$$x_3 \leq 1,000\,.$$

Moreover, they cannot produce more wine than is possible with the amount of grapes purchased. As one kilo of grapes produces one liter of wine, we have

$$x_1 + x_2 \leq x_3\,.$$

It is necessary to add constraints which are, although apparently trivial at the application level, essential to the validity of the mathematical model. These constraints specify the nature of the decision variables. In the case of Château Laupt-Himum, negative values of these variables would have no valid interpretation. It is necessary to impose

$$x_1 \geq 0\,, \qquad x_2 \geq 0\,, \qquad x_3 \geq 0\,. \tag{1.9}$$

We combine the modeling steps to obtain the following optimization problem:

$$\max_{x\in\mathbb{R}^3} f(x) = x_1\left(15 - \frac{2}{100}\,x_1\right) + x_2\left(23 - \frac{1}{100}\,x_2\right) - (2x_1 + 3.5\,x_2 + 3x_3) \tag{1.10}$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq x_3 \\ x_3 &\leq 1\,000 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_3 &\geq 0\,. \end{aligned} \tag{1.11}$$

1.1.4 Euclid

In about 300 BC, the Greek mathematician Euclid was interested in the following geometry problem: what is the rectangle with the greatest area among the rectangles with given perimeter L? This is considered as one of the first known optimization problems in history. We write it in three steps.

Decision variables The decision variables are the length x_1 and the height x_2 of the rectangle, expressed in any arbitrary unit.

Objective function We are looking for the rectangle with maximum area. Therefore, the objective function is simply equal to $x_1 x_2$.

Constraints The total length of the edges of the rectangle must be equal to L, that is

$$2x_1 + 2x_2 = L. \tag{1.12}$$

Moreover, the dimensions x_1 and x_2 must be non negative:

$$x_1 \geq 0 \text{ and } x_2 \geq 0. \tag{1.13}$$

Combining everything, we obtain the following optimization problem:

$$\max_{x \in \mathbb{R}^2} x_1 x_2 \tag{1.14}$$

subject to

$$\begin{aligned} 2x_1 + 2x_2 &= L \\ x_1 &\geq 0 \\ x_2 &\geq 0. \end{aligned} \tag{1.15}$$

1.1.5 Agent 007

James Bond, secret agent 007, has a mission to defuse a nuclear bomb on a yacht moored 100 meters from shore. Currently, James Bond is 80 meters from the nearest point to the yacht on the beach. He is capable of running on the beach at 20 km/h and swimming at 6 km/h. Given that he needs 30 seconds to defuse the bomb, and that the bomb is programmed to explode in 102 seconds, will James Bond have the time to save the free world? This crucial issue, illustrated in Figure 1.2, can be solved by an optimization problem.

The modeling process consists of three steps.

Decision variables The decision that James Bond must make in order to arrive as fast as possible on the yacht is to choose when to stop running on the beach and start swimming towards the yacht. We define a decision variable x representing the distance in meters to run on the beach before jumping into the water.

Objective function Since the objective is to minimize the time to get to the yacht, the objective function associates a decision x with the corresponding time in seconds. Since James Bond runs at 20 km/h, the x meters on the beach are covered in $3.6\,x/20$ seconds, i.e., $18x/100$ seconds. From there, he swims a distance of

$$\sqrt{100^2 + (80 - x)^2} \tag{1.16}$$

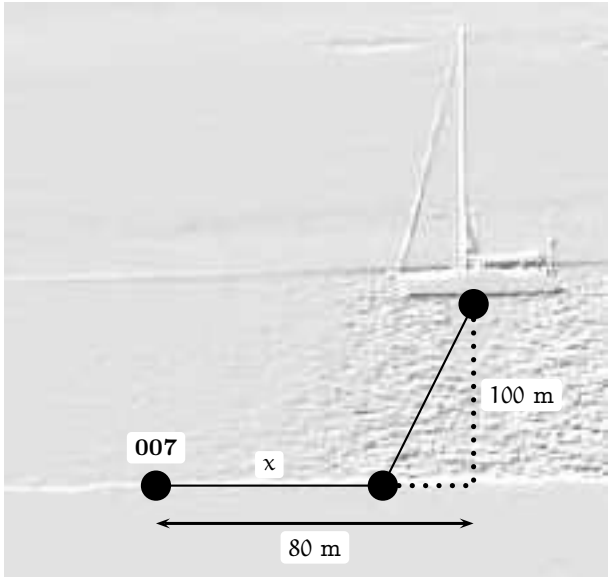


Figure 1.2: The setting for James Bond

at a speed of 6 km/h. This takes him

$$\frac{3.6}{6}\sqrt{100^2+(80-x)^2} = 0.6\sqrt{100^2+(80-x)^2} \tag{1.17}$$

seconds. The objective function is

$$f(x) = \frac{18}{100}x + 0.6\sqrt{100^2+(80-x)^2}\,. \tag{1.18}$$

Note that the trivial decisions such as $x = 0$ and $x = 80$ have disastrous consequences for the future of the planet. An optimization approach is essential.

Constraints A secret agent like James Bond suffers no constraint! However, it makes sense to require that he does not run backward or beyond the yacht, that is,

$$0 \le x \le 80\,.$$

We can combine the different steps of the modeling to obtain the following optimization problem:

$$\min_{x\in\mathbb{R}} f(x) = \frac{18}{100}x + 0.6\sqrt{100^2+(80-x)^2} \tag{1.19}$$

subject to

$$\begin{aligned} x &\ge 0 \\ x &\le 80\,. \end{aligned} \tag{1.20}$$

This example is inspired by Walker (1999).

1.1.6 Indiana Jones

During his quest to find the cross of Coronado, the famous archaeologist Indiana Jones gets stuck facing a huge room filled with *Pseudechis porphyriacus*, venomous snakes. This room is 10 meters long and 5 high. Given the aversion the adventurer has for these reptiles, it is impossible for him to wade through them, and he considers passing over them. However, the roof is not strong enough, so he cannot walk on it. Ever ingenious, he places the end of a ladder on the ground, blocked by a boulder, leans it on the wall, and uses it to reach the other end of the room (Figure 1.3). Once there, he uses his whip to get down to the floor on the other side of the snake room. Where on the floor must he place the end of the ladder, so that the length used is as small as possible, and the ladder thus less likely to break under his weight? We write the optimization problem that would help our hero.

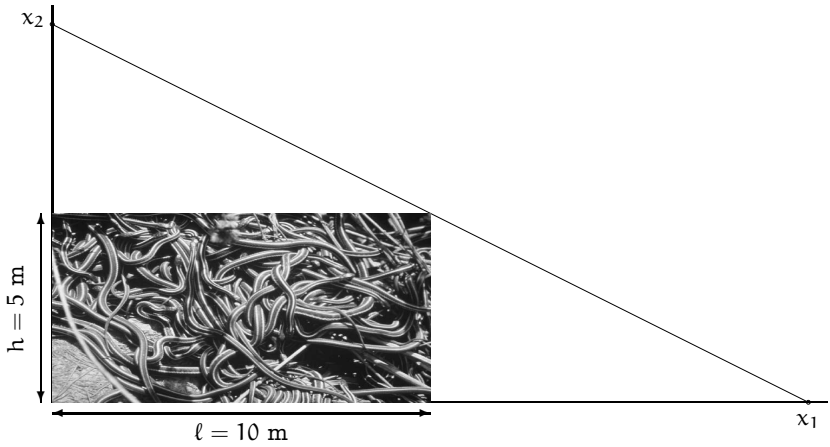


Figure 1.3: The setting for Indiana Jones

The modeling process consists of three steps.

Decision variables The decision variable is the position on the floor of the end of the ladder. To facilitate our modeling work, let us also use a decision variable for the other end of the ladder. Thus

- x_1 is the position on the floor of the ladder's end,
- x_2 is the height of the other end of the ladder at the other end of the room.

Objective function Since the objective is to find the smallest possible ladder, we minimize its length, i.e.,

$$f(x) = \sqrt{x_1^2 + x_2^2}\,.$$

Constraints The positions x_1 and x_2 should be such that the ladder is leaned exactly on the edge of the wall of the room. By using similar triangles, this constraint can be expressed as

$$\frac{x_2}{x_1} = \frac{h}{x_1 - \ell} = \frac{x_2 - h}{\ell}$$

or

$$x_1 x_2 - h x_1 - \ell x_2 = 0\,.$$

Finally, the ends of the ladder must be outside the room, and

$$x_1 \geq \ell \qquad \text{and} \qquad x_2 \geq h\,.$$

We can combine the modeling steps to obtain the following optimization problem:

$$\min_{x \in \mathbb{R}^2} \sqrt{x_1^2 + x_2^2} \tag{1.21}$$

subject to

$$\begin{aligned} x_1 x_2 - h x_1 - \ell x_2 &= 0 \\ x_1 &\geq \ell \\ x_2 &\geq h\,. \end{aligned} \tag{1.22}$$

1.1.7 Geppetto

The company Geppetto Inc. produces wooden toys. It specializes in the manufacture of soldiers and trains. Each soldier is sold for €27 and costs €24 in raw material. To produce a soldier, 1 hour of carpentry labor is required, as well as 2 hours of finishing. Each train is sold for €21 and costs €19 in raw material. To produce a train, it takes 1 hour of carpentry labor and 1 hour of finishing. Geppetto has two carpenters and two finishing specialists each working 40 hours per week. He himself puts in 20 hours per week on finishing work. The trains are popular, and he knows that he can always sell his entire production. However, he is not able to sell more than 40 soldiers per week. What should Geppetto do to optimize his income?

The modeling process is organized in three steps.

Decision variables Geppetto's strategy is to determine the number of soldiers and trains to produce per week. Thus, we define two decision variables:

- x_1 is the number of soldiers to produce per week,
- x_2 is the number of trains to produce per week.

Objective function The objective of Geppetto is to make as much money as possible. The quantity to maximize is the profit. Geppetto's gains consist of the income from the sales of the toys, minus the cost of the raw material. Geppetto's income is the sum of the income from the sales of the soldiers and the sales of the trains. The income (in €) from the sales of the soldiers is the number of soldiers sold multiplied by the selling price of one soldier, i.e., $27\,x_1$. The income from the sales of the trains is $21\,x_2$. The total income for Geppetto is $27\,x_1 + 21\,x_2$. Similarly, we evaluate the material costs to $24\,x_1 + 19 x_2$. The gain is

$$(27\,x_1 + 21\,x_2) - (24\,x_1 + 19 x_2), \tag{1.23}$$

or

$$f(x) = 3x_1 + 2x_2\,. \tag{1.24}$$

Constraints The production is subject to three main constraints. First, the available labor does not allow more than 100 finishing hours (performed by two workers and Geppetto) and 80 hours of carpentry (performed by two carpenters). Furthermore, to avoid unsold objects, they should not produce more than 40 soldiers per week. The number of hours per week of finishing is the number of hours of finishing for the soldiers, multiplied by the number of soldiers produced, plus the hours of finishing for the trains, multiplied by the number of trains produced. This number may not exceed 100, and we can express the following constraint:

$$2x_1 + x_2 \leq 100\,. \tag{1.25}$$

A similar analysis of the carpentry resources leads to the following constraint:

$$x_1 + x_2 \leq 80\,. \tag{1.26}$$

Finally, the constraint to avoid unsold products can simply be written as:

$$x_1 \leq 40\,. \tag{1.27}$$

At this stage, it seems that all the constraints of the problem have been described mathematically. However, it is necessary to add constraints that are, although apparently trivial at the application level, essential to the validity of the mathematical model. These constraints specify the nature of the decision variables. In the case of Geppetto, it is not conceivable to produce parts of trains or soldiers. The decision variables must absolutely take integer values, so that

$$x_1 \in \mathbb{N}, \qquad x_2 \in \mathbb{N}\,. \tag{1.28}$$

We combine the different stages of the modeling to obtain the following optimization problem:

$$\max_{x \in \mathbb{N}^2} f(x) = 3x_1 + 2x_2\,, \tag{1.29}$$

subject to

$$\begin{array}{rcrcl} 2x_1 & + & x_2 & \leq & 100 \\ x_1 & + & x_2 & \leq & 80 \\ x_1 & & & \leq & 40. \end{array} \tag{1.30}$$

Interestingly, the constraints (1.28), albeit of a trivial aspect, significantly complicate the optimization methods. Most of the book is devoted to problems where the integrality of the variables is not imposed. An introduction to discrete optimization is provided in Part VII. This example is inspired by Winston (1994).

1.2 Problem transformations

Even though the modeling step is completed, we are not yet out of the woods. Indeed, there are many ways to write a given problem mathematically. Algorithms and software often require a special formulation based on which they solve the problem. In this chapter, we study techniques that help us comply with these requirements.

Obtaining the mathematical formulation of a problem does not necessarily end the modeling process. In fact, the obtained formulation may not be adequate. In particular, the software capable of solving optimization problems often require the problems to be formulated in a specific way, not necessarily corresponding to the result of the approach described in Section 1.1. We review some rules for transforming an optimization problem into another equivalent problem.

Definition 1.3 (Equivalence)**.** Two optimization problems P_1 and P_2 are said to be equivalent if we can create a feasible point (i.e., satisfying the constraints) in P_2 from a feasible point in P_1 (and vice versa) with the same value of the objective function. In particular, the two problems have the same optimal cost, and we can obtain a solution of P_2 from a solution of P_1 (and vice versa).

1.2.1 Simple transformations

Here are some simple transformations that are often used in modeling.

1. Consider the optimization problem

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

where X is a subset of $\mathbb{R}^n$. Consider a function $g : \mathbb{R} \to \mathbb{R}$ that is strictly increasing on $\mathrm{Im}(f) = \{z \mid \exists x \in X \text{ such that } z = f(x)\}$, i.e., for any $z_1, z_2 \in \mathrm{Im}(f)$, $g(z_1) > g(z_2)$ if and only if $z_1 > z_2$. Thus,[2]

$$\operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} f(x) = \operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} h(x)\,, \tag{1.31}$$

where $h(x) = g\big(f(x)\big)$, and

$$\min_{x \in X \subseteq \mathbb{R}^n} g\big(f(x)\big) = g\left(\min_{x \in X \subseteq \mathbb{R}^n} f(x)\right). \tag{1.32}$$

In particular, adding or subtracting a constant to the objective function of an optimization problem does not change its solution

$$\forall c \in \mathbb{R}\,, \quad \operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} f(x) = \operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} \big(f(x) + c\big)\,. \tag{1.33}$$

[2] The operator argmin identifies the values of the decision variables that reach the minimum, while the operator min identifies the value corresponding to the objective function. See Appendix A.

Similarly, if the function f generates only positive values, taking the logarithm of the objective function or taking its square does not change its solution:

$$\operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} f(x) = \operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} \log\big(f(x)\big)\,, \tag{1.34}$$

and

$$\operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} f(x) = \operatorname{argmin}_{x \in X \subseteq \mathbb{R}^n} \big(f(x)\big)^2\,, \tag{1.35}$$

as $g(x) = x^2$ is strictly increasing for $x \geq 0$. Note that the log transformation is typically used in the context of maximum likelihood estimation of unknown parameters in statistics, where the objective function is a probability and, therefore, is positive. The square transform is relevant namely when $f(x)$ is expressed as a square root (see the example on Indiana Jones in Section 1.1.6). In this case, the square root can be omitted.

2. A maximization problem whose objective function is $f(x)$ is equivalent to a minimization problem whose objective function is $-f(x)$:

$$\operatorname{argmax}_x f(x) = \operatorname{argmin}_x -f(x)\,, \tag{1.36}$$

and

$$\max_x f(x) = -\min_x -f(x)\,. \tag{1.37}$$

Similarly, we have

$$\operatorname{argmin}_x f(x) = \operatorname{argmax}_x -f(x)\,, \tag{1.38}$$

and

$$\min_x f(x) = -\max_x -f(x)\,. \tag{1.39}$$

3. A constraint defined by a lower inequality can be multiplied by -1 to get an upper inequality

$$g(x) \leq 0 \iff -g(x) \geq 0\,. \tag{1.40}$$

4. A constraint defined by an equality can be replaced by two constraints defined by inequalities

$$g(x) = 0 \iff \begin{cases} g(x) \leq 0 \\ g(x) \geq 0\,. \end{cases} \tag{1.41}$$

Note that this transformation is primarily used when constraints are linear. When $g(x)$ is non linear, this transformation is generally not recommended.

5. Some software require that all decision variables be non negative. However, for problems such as the Swisscom example described in Section 1.1.2, such restrictions are not relevant. If a variable x can take any real value, it is then replaced by two artificial variables denoted by x^+ and x^-, such that

$$x = x^+ - x^-\,. \tag{1.42}$$

In this case, we can meet the requirements of the software and impose $x^+ \geq 0$ and $x^- \geq 0$, without loss of generality.

6. In the presence of a constraint $x \geq a$, with $a \in \mathbb{R}$, a simple change of variable

$$x = \widetilde{x} + a \tag{1.43}$$

transforms the constraint into

$$\widetilde{x} \geq 0\,. \tag{1.44}$$

To illustrate these transformations, let us consider the following optimization problem:

$$\max_{x,y} -x^2 + \sin y \tag{1.45}$$

subject to

$$6x - y^2 \geq 1 \tag{1.46}$$
$$x^2 + y^2 = 3 \tag{1.47}$$
$$x \geq 2 \tag{1.48}$$
$$y \in \mathbb{R}\,, \tag{1.49}$$

and transform it in such a way as to obtain a minimization problem, in which all the decision variables are non negative, and all constraints are defined by lower inequalities. We get the following problem:

$$-\min_{\widetilde{x},\, y^+,\, y^-} (\widetilde{x}+2)^2 - \sin(y^+ - y^-) \tag{1.50}$$

subject to

$$-6(\widetilde{x}+2) + \left(y^+ - y^-\right)^2 + 1 \leq 0 \tag{1.51}$$
$$(\widetilde{x}+2)^2 + \left(y^+ - y^-\right)^2 - 3 \leq 0 \tag{1.52}$$
$$-(\widetilde{x}+2)^2 - \left(y^+ - y^-\right)^2 + 3 \leq 0 \tag{1.53}$$
$$\widetilde{x} \geq 0 \tag{1.54}$$
$$y^+ \geq 0 \tag{1.55}$$
$$y^- \geq 0\,, \tag{1.56}$$

where

(1.50)	is obtained by applying	(1.37) to (1.45),
(1.51)	is obtained by applying	(1.40) to (1.46),
(1.52) and (1.53)	are obtained by applying	(1.41) to (1.47),
(1.54)	is obtained by applying	(1.43) to (1.48),
(1.55) and (1.56)	are obtained by applying	(1.42) to (1.49).

Note that the transformed problem has more decision variables (3 instead of 2 for the original problem) and more constraints (6 instead of 3 for the original problem).

When the solution $(\widetilde{x}^*, y^{+*}, y^{-*})$ of (1.50)–(1.56) is available, it is easy to obtain the solution to the original problem (1.45)–(1.49) by applying the inverse transformations, i.e.,

$$\begin{aligned} x^* &= \widetilde{x}^* + 2 \\ y^* &= y^{+*} - y^{-*}\,. \end{aligned} \tag{1.57}$$

1.2.2 Slack variables

A slack variable is introduced to replace an inequality constraint by an equality constraint. Such a variable should be non negative. There are several ways to define a slack variable.

- The slack variable y is introduced directly in the specification, and its value is explicitly restricted to be non negative:

$$g(x) \leq 0 \iff \begin{cases} g(x) + y = 0 \\ y \geq 0\,. \end{cases} \tag{1.58}$$

  The above specification does not completely eliminate inequality constraints. However, it simplifies considerably the nature of these constraints.
- The slack variable is introduced indirectly using a specification enforcing its non negativity. For example, the slack variable can be defined as $y = z^2$, and

$$g(x) \leq 0 \iff g(x) + z^2 = 0. \tag{1.59}$$

- The slack variable can also be introduced indirectly using an exponential, that is, $y = \exp(z)$, and

$$g(x) \leq 0 \iff g(x) + e^z = 0. \tag{1.60}$$

  The limitation of this approach is that there is no value of z such that $g(x)+e^z = 0$ when the constraint is active, that is when $g(x) = 0$. Strictly speaking, the two specifications are not equivalent. However, the slack variable $\exp(z)$ can be made as close to zero as desired by decreasing the value of z, as

$$\lim_{z \to -\infty} e^z = 0. \tag{1.61}$$

  So, loosely speaking, we can say that the two specifications are "asymptotically equivalent."

The slack variable (z^2, e^z or y) thus introduced measures the distance between the constraint and the point x. The special status of such variables can be effectively exploited for solving optimization problems.

Definition 1.4 (Slack variable)**.** A slack variable is a decision variable introduced in an optimization problem to transform an inequality constraint into an equality constraint, possibly with a non negativity constraint.

For example, we consider the following optimization problem:

$$\min_{x_1, x_2} x_1^2 - x_2^2 \tag{1.62}$$

subject to

$$\sin x_1 \leq \frac{\pi}{2} \tag{1.63}$$

$$\ln\left(e^{x_1} + e^{x_2}\right) \geq \sqrt{e} \tag{1.64}$$

$$x_1 - x_2 \leq 100\,. \tag{1.65}$$

We introduce the slack variable z_1 for the constraint (1.63), the slack variable z_2 for the constraint (1.64), and the slack variable y_3 for the constraint (1.65). The obtained optimization problem is

$$\min_{x_1, x_2, z_1, z_2, y_3} x_1^2 - x_2^2, \tag{1.66}$$

subject to

$$\sin x_1 + z_1^2 = \frac{\pi}{2} \tag{1.67}$$

$$\ln\left(e^{x_1} + e^{x_2}\right) - e^{z_2} = \sqrt{e} \tag{1.68}$$

$$x_1 - x_2 + y_3 = 100 \tag{1.69}$$

$$y_3 \geq 0. \tag{1.70}$$

Note that the objective function is not affected by the introduction of slack variables.

1.3 Hypotheses

The methods and algorithms presented in this book are not universal. Each approach is subject to assumptions about the structure of the underlying problem. Specific assumptions are discussed for each method. However, there are three important assumptions that concern (almost) the whole book: the continuity hypothesis, the differentiability hypothesis, and the determinism hypothesis.

The continuity hypothesis consists in only considering problems for which the objective to optimize and the constraints are modeled by continuous functions of decision variables. This hypothesis excludes problems with integer variables (see discussion in Section 1.1.7). Such problems are treated by discrete optimization or integer programming.[3] An introduction to discrete optimization in provided in Part VII of this book. We also refer the reader to Wolsey (1998) for an introduction to combinatorial optimization, as well as to Schrijver (2003), Bertsimas and Weismantel (2005), and Korte and Vygen (2007).

The differentiability hypothesis (which obviously implies the continuity hypothesis) also requires that the functions involved in the model are differentiable. Non differentiable optimization is the subject of books such as Boyd and Vandenberghe (2004), Bonnans et al. (2006), and Dem'Yanov et al. (2012).

The determinism hypothesis consists in ignoring possible errors in the data for the problem. Measurement errors, as well as modeling errors, can have a non negligible impact on the outcome. Stochastic optimization (Birge and Louveaux, 1997) enables the use of models in which some pieces of data are represented by random variables. Robust optimization (see Ben-Tal and Nemirovski, 2001 and Ben-Tal et al., 2009) produces solutions that are barely modified by slight disturbances in the data of the problem.

[3] The term "programming", used in the sense of optimization, was introduced during the Second World War.

It is crucial to be aware of these hypotheses in the modeling stage. The use of inappropriate techniques can lead to erroneous results. For instance, it is shown in Section 25.4 that solving a discrete optimization problem by solving a continuous version (called the *relaxation*) and then rounding the solutions to the closest integer is inappropriate.

1.4 Problem definition

We now outline the main concepts that define the optimization problems, and analyze the desired properties.

We consider the following optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.71}$$

subject to

$$h(x) = 0 \tag{1.72}$$
$$g(x) \leq 0 \tag{1.73}$$

and

$$x \in X, \tag{1.74}$$

where f is a function of $\mathbb{R}^n$ in $\mathbb{R}$, h is a function of $\mathbb{R}^n$ in $\mathbb{R}^m$, g is a function of $\mathbb{R}^n$ in $\mathbb{R}^p$ and $X \subseteq \mathbb{R}^n$ is a convex set (Definition B.2). We say that this is an optimization problem with n decision variables, m equality constraints, and p inequality constraints. We assume that $n > 0$, i.e., that the problem involves at least one decision variable. However, we consider problems where m and p are zero, as well as problems where $X = \mathbb{R}^n$. By employing the transformations described in Section 1.2, it is possible to express any optimization problem satisfying the hypotheses described in Section 1.3 in the form (1.71)–(1.74).

We consider two types of solutions to this problem: local minima, where none of their neighbors also satisfying the constraints are better, and global minima, where no other point satisfying the constraints is better.

Definition 1.5 (Local minimum)**.** Let $Y = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0 \text{ and } x \in X\}$ be the feasible set, that is the set of vectors satisfying all constraints. The vector $x^* \in Y$ is called a local minimum of the problem (1.71)–(1.74) if there exists $\varepsilon > 0$ such that

$$f(x^*) \leq f(x), \quad \forall x \in Y \text{ such that } \|x - x^*\| < \varepsilon. \tag{1.75}$$

The notation $\|\cdot\|$ denotes a vector norm on $\mathbb{R}^n$ (Definition B.1).

Definition 1.6 (Strict local minimum). Let $Y = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0 \text{ and } x \in X\}$ be the feasible set, that is the set of vectors satisfying all the constraints. The vector $x^* \in Y$ is called a strict local minimum of the problem (1.71)–(1.74) if there exists $\varepsilon > 0$ such that

$$f(x^*) < f(x)\,, \quad \forall x \in Y,\ x \neq x^* \text{ such that } \|x - x^*\| < \varepsilon\,. \tag{1.76}$$

Definition 1.7 (Global minimum). Let $Y = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0 \text{ and } x \in X\}$ be the feasible set, that is the set of vectors satisfying all the constraints. The vector $x^* \in Y$ is called a global minimum of the problem (1.71)–(1.74) if

$$f(x^*) \leq f(x)\,, \qquad \forall x \in Y\,. \tag{1.77}$$

Definition 1.8 (Strict global minimum). Let $Y = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0 \text{ and } x \in X\}$ be the feasible set, that is the set of vectors satisfying all the constraints. The vector $x^* \in Y$ is called a strict global minimum of the problem (1.71)-(1.74) if

$$f(x^*) < f(x)\,, \qquad \forall x \in Y,\ x \neq x^*\,. \tag{1.78}$$

The notions of local maximum, strict local maximum, global maximum and strict global maximum are defined in a similar manner.

Example 1.9 (Local minimum and maximum). Figure 1.4 illustrates these definitions for the function

$$f(x) = -\frac{5}{6}x^3 + \frac{7}{2}x^2 - \frac{11}{3}x + 3\,. \tag{1.79}$$

The point $x^* \simeq 0.6972$ is a local minimum of f. Indeed, there is an interval $\left[x^* - \frac{1}{2}, x^* + \frac{1}{2}\right]$, represented by the dotted lines, such that $f(x^*) \leq f(x)$, for any x in the interval. Similarly, the point $\widetilde{x}^* \simeq 2.1024$ is a local maximum. Indeed, there exists an interval $\left[\widetilde{x}^* - \frac{1}{2}, \widetilde{x}^* + \frac{1}{2}\right]$, represented by the dotted lines, such that $f(\widetilde{x}^*) \geq f(x)$, for any x in the interval.

Example 1.10 (Binary optimization). Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable, and let us take the optimization problem

$$\min f(x)$$

with constraints $x \in \{0,1\}^n$, i.e., such that each variable can take only the value 0 or 1. However, we can express the constraints in the following manner:

$$h_i(x) = x_i(1 - x_i) = 0\,, \qquad i = 1, \ldots, n\,.$$

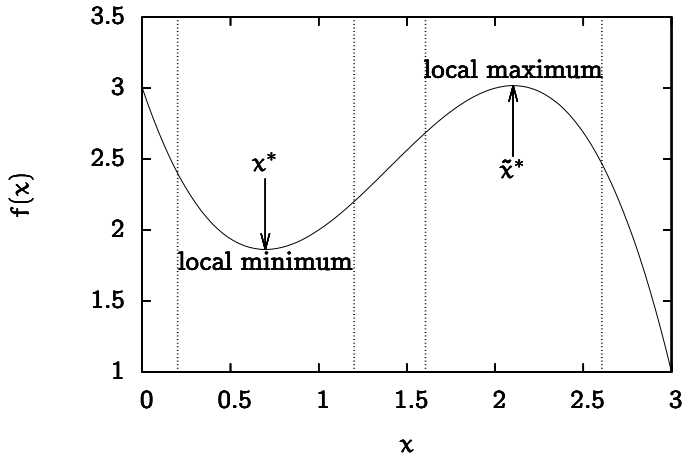


Figure 1.4: Local minimum and maximum for Example 1.9

We thus get n differentiable constraints and the hypotheses are satisfied. However, since each feasible point is isolated, each of them is a local minimum of the problem. Therefore, algorithms for continuous optimization designed to identify local minima are useless for this type of problem.

In general, it is desirable to exploit the particular structure of the problem, because an excess of generality penalizes optimization algorithms. We analyze special cases of the problem (1.71)–(1.74).

It is important to note that the existence of a solution is not always guaranteed. For example, the problem $\min_{x\in\mathbb{R}} f(x) = x$ has neither a minimum nor a maximum, whether local or global. In fact, this function is not bounded in the sense that it can take on values that are arbitrarily large and arbitrarily small.

Definition 1.11 (Function bounded from below)**.** The function $f : \mathbb{R}^n \to \mathbb{R}$ is bounded from below on $Y \subseteq \mathbb{R}^n$ if there exists a real M such that

$$f(x) \geq M\,, \qquad \forall x \in Y\,. \tag{1.80}$$

The function $f(x) = x$ is unbounded on $\mathbb{R}$. However, it is bounded on compact subsets of $\mathbb{R}$. Among all the bounds M of Definition 1.11, the largest is called the infimum of f.

Definition 1.12 (Infimum)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function bounded from below on the set $Y \subseteq \mathbb{R}^n$. The infimum of f on Y is denoted by

$$\inf_{y\in Y} f(y) \tag{1.81}$$

and is such that

$$\inf_{y \in Y} f(y) \leq f(x), \qquad \forall x \in Y \tag{1.82}$$

and

$$\forall M > \inf_{y \in Y} f(y), \quad \exists x \in Y \text{ such that } f(x) < M. \tag{1.83}$$

Example 1.13 (Infimum)**.** Consider $f(x) = e^x$ and $Y = \mathbb{R}$. We have

$$\inf_{y \in Y} f(y) = 0.$$

We verify Definition 1.12. We have

$$0 = \inf_{y \in Y} f(y) \leq f(x) = e^x, \qquad \forall x \in \mathbb{R},$$

and (1.82) is satisfied. Consider an arbitrary $M > 0$, and consider $x = \ln M/2$. In this case,

$$f(x) = \frac{M}{2} < M,$$

and the definition is satisfied.

This example shows that an optimization problem is not always well defined, and that no solution may exist. Theorem 1.14 identifies cases where the problem is well-defined, and where the infimum of a function on a set is reached by at least one point of the set. In this case, the point is a global minimum of the corresponding optimization problem.

Theorem 1.14 (Weierstrass theorem)**.** *Let $Y \subset \mathbb{R}^n$ be a closed and non empty subset of $\mathbb{R}^n$, and let $f : Y \to \mathbb{R}$ be a lower semi-continuous function (Definition B.21) on Y. If Y is compact (Definition B.23) or if f is coercive (that is, if it goes to $+\infty$ when x goes to $+\infty$ or $-\infty$, see Definition B.22), there exists $x^* \in Y$ such that*

$$f(x^*) = \inf_{y \in Y} f(y).$$

Proof. Consider a sequence $(x_k)_k$ of elements of Y such that

$$\lim_{k \to \infty} f(x_k) = \inf_{y \in Y} f(y).$$

If Y is compact, it is bounded and the sequence has at least one limit point x^*. If f is coercive (Definition B.22), the sequence $(x_k)_k$ is bounded and has at least one limit point x^*. In both cases, due to the lower semi-continuity of f in x^*, we have

$$f(x^*) \leq \lim_{k \to \infty} f(x_k) = \inf_{y \in Y} f(y),$$

and

$$f(x^*) = \inf_{y \in Y} f(y).$$

□

Note that some functions may include local minima and have no global minimum, as for Example 1.9, shown in Figure 1.4, where the function (1.79) is unbounded from below, as

$$\lim_{x \to +\infty} -\frac{5}{6}x^3 + \frac{7}{2}x^2 - \frac{11}{3}x + 3 = -\infty.$$

In some cases, an optimization problem with constraints can be simplified and the constraints ignored. Before detailing such simplifications in Section 3.1, we state now a general theoretical result when the optimum is an interior point of the set of constraints.

Definition 1.15 (Interior point)**.** Consider $Y \subset \mathbb{R}^n$ and $y \in Y$. We say that y is in the interior of Y if there exists a neighborhood of y in Y. Formally, y is in the interior of Y if there exists $\varepsilon > 0$ such that all points in a neighborhood of size ε of y belong to Y, that is such that

$$\|z - y\| \leq \varepsilon \implies z \in Y. \tag{1.84}$$

Theorem 1.16 (Solution in the interior of constraints)**.** *Let x^* be a local minimum of the optimization problem (1.71)–(1.74). Let $Y = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0 \text{ and } x \in X\}$ be the feasible set, and let $Y = Y_1 \cap Y_2$ such that x^* is an interior point of Y_1. Then x^* is a local minimum of the problem*

$$\min_{x \in Y_2} f(x).$$

In particular, if x^* is an interior point of Y, then the theorem applies with $Y_2 = \mathbb{R}^n$ and x^* is a solution of the unconstrained optimization problem.

Proof. According to Definition 1.15, there exists $\varepsilon_1 > 0$ such that

$$y \in Y_1\,, \quad \forall y \text{ such that } \|y - x^*\| \leq \varepsilon_1\,. \tag{1.85}$$

Since x^* is a local minimum (Definition 1.5), there exists $\varepsilon_2 > 0$ such that

$$f(x^*) \leq f(y)\,, \quad \forall y \in Y \text{ such that } \|y - x^*\| \leq \varepsilon_2\,. \tag{1.86}$$

Then, if $\varepsilon = \min(\varepsilon_1, \varepsilon_2)$, any point $y \in Y_2$ such that $\|y - x^*\| \leq \varepsilon$ belongs also to Y_1 and, therefore, is feasible. Consequently, x^* is better, in the sense of the objective function, than any of these y. Formally, we get

$$f(x^*) \leq f(y)\,, \quad \forall y \in Y_2 \text{ such that } \|y - x^*\| \leq \varepsilon\,, \tag{1.87}$$

which is exactly (1.75) where Y has been replaced by Y_2, and x^* is a local minimum of the problem with only the Y_2 constraints. □

This result is used particularly in the development of optimality conditions for problems with constraints (Chapter 6), as well as in the development of algorithms called interior point methods. Note that a problem containing equality constraints has no interior points.

Before detailing algorithms that enable an optimization problem to be solved, it is essential to properly understand the nature of this problem. In Chapter 2, we analyze the objective function in detail. Constraints are discussed in Chapter 3. Finally, in Chapter 4, we analyze the ways in which it is possible to combine the objective function and constraints.

1.5 Exercises

Exercise 1.1 (Geometry)**.** We want to determine a parallelepiped with a volume of unity and minimal surface.

1. Formulate this problem as an optimization problem by determining
   (a) the decision variables,
   (b) the objective function,
   (c) the constraint(s).
2. Formulate this problem as a minimization problem with only lower inequality constraints.

Exercise 1.2 (Finance)**.** A bank wants to determine how to invest its assets for the year to come. Currently, the bank has a million euros that it can invest in bonds, real estate loans, leases or personal loans. The annual interest rate of different investment types are 6% for bonds, 10% for real estate loans, 8% for leases, and 13% for personal loans.

To minimize risk, the portfolio selected by the bank must satisfy the following restrictions:

- The amount allocated to personal loans must not exceed half of that invested in bonds.
- The amount allocated to real estate loans must not exceed that allocated to leases.
- At most 20% of the total invested amount can be allocated to personal loans.

1. Formulate this problem as an optimization problem by determining
   (a) the decision variables,
   (b) the objective function,
   (c) the constraint(s).
2. Formulate this problem as a minimization problem with only lower inequality constraints.
3. Formulate this problem as a maximization problem with equality constraints and with non negativity constraints on the decision variables.

Exercise 1.3 (Stock management)**.** The company Daubeliou sells oil and wants to optimize the management of its stock. The annual demand is estimated at 100,000 liters and is assumed to be homogeneous throughout the year. The cost of storage is €40 per thousand liters per year. When the company orders oil to replenish its stock, this costs €80. Assuming that the order arrives instantly, how many orders must the company Daubeliou place each year to satisfy demand and minimize costs?

Formulate this problem as an optimization problem by determining

1. the decision variables,
2. the objective function,
3. the constraint(s).

Advice:

- Set the amount of oil to order each time as a decision variable.
- Represent in a graph the evolution of the stock as a function of time.

Exercise 1.4 (Measurement errors)**.** Mr. O. Beese is obsessed with his weight. He owns 10 scales and weighs himself each morning on each of them. This morning he got the following measurement results:

100.8	99.4	101.3	97.6	102.5	102.4
104.6	102.6	95.1	96.6		

He wishes to determine an estimate of his weight while minimizing the sum of the squares of measurement errors from the 10 scales. Formulate the optimization problem that he needs to solve.

Exercise 1.5 (Congestion)**.** Every day, 10,000 people commute from Divonne to Geneva. By train, the journey takes 40 minutes. By road, the travel time depends on the level of congestion. It takes 20 minutes when the highway is completely deserted. When there is traffic, travel time increases by 5 minutes per thousand people using the highway (assuming that there is one person per car). If 4,000 people take the car and 6,000 take the train, the travel time by road is equal to $20 + 5 \times 4 = 40$ minutes, which is identical to the travel time by train. In this situation, it would be of no interest to change one's mode of transport. We say that the system is at equilibrium. However, from the point of view of the average travel time per person, is this situation optimal? Formulate an optimization problem to answer the question.

Chapter 2

Objective function

Before attempting to understand how to solve a problem, we first try to understand the problem itself. In this chapter, we identify the properties of the objective function which are useful in the development of theory and algorithms. A crucial part in optimization is to understand the geometry of the problem. Derivatives play a central role in this analysis. We also identify what constitutes a good or a bad geometry for a problem.

Contents

Several concepts can be used to characterize the objective function. It is important to identify the characteristics of the objective function because each optimization algorithm is based on specific hypotheses. When the hypotheses of an algorithm have not been verified for a given problem, there is no guarantee that the algorithm can be used to solve this problem.

2.1 Convexity and concavity

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex* if, for any vector x and y of $\mathbb{R}^n$, the graph of f between x and y is not above the line segment connecting $\big(x, f(x)\big)$ and $\big(y, f(y)\big)$ in $\mathbb{R}^{n+1}$. Definition 2.1 establishes this property formally.

Definition 2.1 (Convex function). A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be convex if, for any $x, y \in \mathbb{R}^n$ and for any $\lambda \in [0, 1]$, we have

$$f\big(\lambda x + (1-\lambda)y\big) \leq \lambda f(x) + (1-\lambda) f(y)\,. \tag{2.1}$$

Definition 2.1 is shown in Figure 2.1. The line segment connecting the points $\big(x, f(x)\big)$ and $\big(y, f(y)\big)$ is never below the graph of f. The point $z = \lambda x + (1-\lambda)y$ is somewhere between x and y when $0 \leq \lambda \leq 1$. The point with coordinates $\big(z, \lambda f(x) + (1-\lambda) f(y)\big)$ is on the line segment between the points $\big(x, f(x)\big)$ and $\big(y, f(y)\big)$. In order for the function to be convex, this point must never (i.e., for all x, y and $0 \leq \lambda \leq 1$) be below the graph of the function.

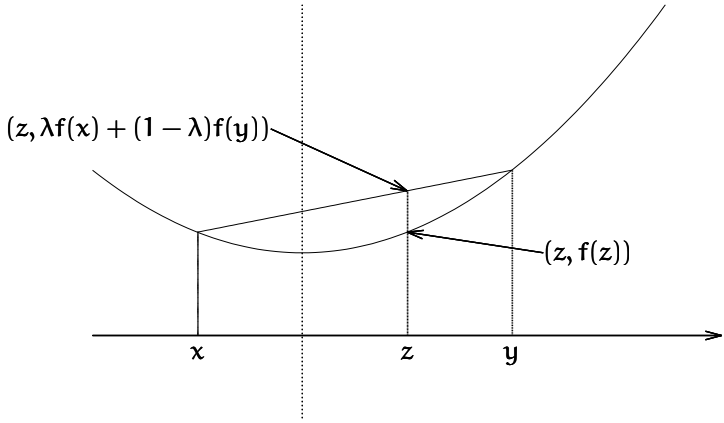


Figure 2.1: Illustration of Definition 2.1

Figure 2.2 shows a non convex function, in which there is an x and y such that the line connecting $\big(x, f(x)\big)$ and $\big(y, f(y)\big)$ is partially located below the graph of the function.

The notion of convexity of a function can be strengthened, giving strict convexity.

Definition 2.2 (Strictly convex function). A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be strictly convex if for all $x, y \in \mathbb{R}^n$, $x \neq y$, and for all $\lambda \in \,]0, 1[$, we have

$$f\big(\lambda x + (1-\lambda)y\big) < \lambda f(x) + (1-\lambda) f(y)\,. \tag{2.2}$$

The convexity of the function is an important feature in optimization. In general, when the function is not convex, it is particularly difficult to identify a global minimum of the problem (1.71)–(1.74).

Note that the importance of convexity is linked to minimization problems. When we study maximization problems, the concept of concavity should be used, as shown

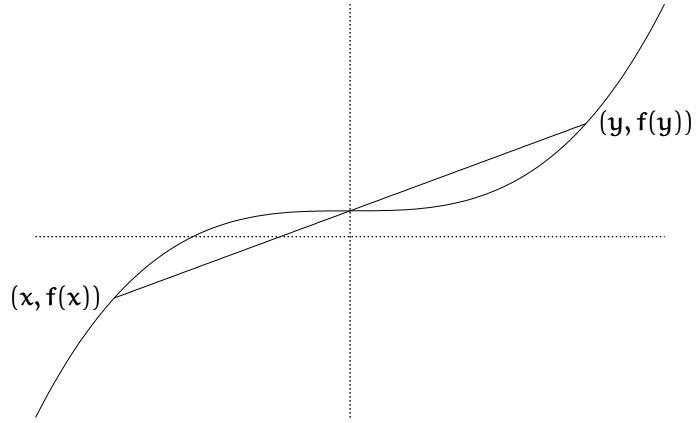


Figure 2.2: Illustration of a counterexample to Definition 2.1

by Definition 2.3. As discussed in Section 1.2.1, a maximization problem can always be easily transformed into a minimization problem using (1.37).

Definition 2.3 (Concave function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be concave if $-f$ is a convex function, i.e., if for all $x, y \in \mathbb{R}^n$ and for all $\lambda \in [0, 1]$, we have

$$f\big(\lambda x + (1-\lambda)y\big) \geq \lambda f(x) + (1-\lambda) f(y)\,. \tag{2.3}$$

Note that convexity and concavity are not complementary properties. A function may be neither convex nor concave. This is the case of the function represented in Figure 2.2.

2.2 Differentiability: the first order

An important assumption in this book is that the objective function f and the functions g and h describing the constraints are continuous (Definition B.5) and differentiable. We summarize here the main concepts related to differentiability.

Definition 2.4 (Partial derivative)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function. The function $\nabla_i f(x) : \mathbb{R}^n \to \mathbb{R}$, also written as $\partial f(x)/\partial x_i$ is called the i^{th} partial derivative of f and is defined as

$$\lim_{\alpha \to 0} \frac{f(x_1, \ldots, x_i + \alpha, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{\alpha}\,. \tag{2.4}$$

This limit may not exist.

If the partial derivatives $\partial f(x)/\partial x_i$ exist for all i, the *gradient* of f is defined as follows.

Definition 2.5 (Gradient). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. The function denoted by $\nabla f(x) : \mathbb{R}^n \to \mathbb{R}^n$ is called the gradient of f and is defined as

$$\nabla f(x) = \begin{pmatrix} \dfrac{\partial f(x)}{\partial x_1} \\ \vdots \\ \dfrac{\partial f(x)}{\partial x_n} \end{pmatrix}. \tag{2.5}$$

The gradient plays a key role in the development and analysis of optimization algorithms.

Example 2.6 (Gradient). Consider $f(x_1, x_2, x_3) = e^{x_1} + x_1^2 x_3 - x_1 x_2 x_3$. The gradient of f is given by

$$\nabla f(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1} + 2x_1 x_3 - x_2 x_3 \\ -x_1 x_3 \\ x_1^2 - x_1 x_2 \end{pmatrix}. \tag{2.6}$$

The analysis of the behavior of the function in certain directions is also important for optimization methods. We introduce the concept of a directional derivative.

Definition 2.7 (Directional derivative). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function. Consider $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^n$. The directional derivative of f in x in the direction d is given by

$$\lim_{\alpha \overset{>}{\longrightarrow} 0} \frac{f(x + \alpha d) - f(x)}{\alpha}, \tag{2.7}$$

if the limit exists. In addition, when the gradient exists, the directional derivative is the scalar product between the gradient of f and the direction d, i.e.,

$$\nabla f(x)^T d. \tag{2.8}$$

Definition 2.8 (Differentiable function). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function. If, for any $d \in \mathbb{R}^n$, the directional derivative of f in the direction d exists, the function f is said to be differentiable.

This concept is sometimes called "Gâteaux differentiability," in the sense that other types of differentiability can be defined (like the Fréchet differentiability). In

this book, we deal with continuous differentiable functions for which a distinction is unnecessary.

Example 2.9 (Directional derivative). Consider $f(x_1, x_2, x_3) = e^{x_1} + x_1^2 x_3 - x_1 x_2 x_3$, and

$$d = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}. \tag{2.9}$$

The directional derivative of f in the direction d is

$$\begin{pmatrix} d_1 & d_2 & d_3 \end{pmatrix} \nabla f(x_1, x_2, x_3) = \\ d_1(e^{x_1} + 2x_1x_3 - x_2x_3) - d_2x_1x_3 + d_3(x_1^2 - x_1x_2), \tag{2.10}$$

where $\nabla f(x_1, x_2, x_3)$ is given by (2.6).

A numerical illustration of the directional derivative is given in Example 2.14. Note that the partial derivatives are in fact directional derivatives in the direction of the coordinate axes and

$$\frac{\partial f(x)}{\partial x_i} = \nabla f(x)^T e_i, \tag{2.11}$$

where e_i is the column i of the identity matrix, i.e., a vector with all entries equal to 0, except the one at line i which is 1.

The directional derivative provides information about the slope of the function in the direction d, just as the derivative gives information of the slope of functions of one variable. In particular, the function increases in the direction d if the directional derivative is positive and decreases if it is negative. In the latter case, we say that it is a descent direction.

Definition 2.10 (Descent direction). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Consider x, $d \in \mathbb{R}^n$. The direction d is a descent direction in x if

$$d^T \nabla f(x) < 0. \tag{2.12}$$

The terminology "descent direction" is justified by Theorem 2.11. It shows the decrease of the function along the descent direction. The theorem also states that the decrease is proportional to the slope, that is, the directional derivative, in the neighborhood of x.

Theorem 2.11 (Descent direction). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Consider $x \in \mathbb{R}^n$ such that $\nabla f(x) \neq 0$ and $d \in \mathbb{R}^n$. If d is a descent direction, in the sense of Definition 2.10, there exists $\eta > 0$ such that*

$$f(x + \alpha d) < f(x), \qquad \forall 0 < \alpha \leq \eta. \tag{2.13}$$

Moreover, for any $\beta < 1$, *there exists* $\widehat{\eta} > 0$ *such that*

$$f(x + \alpha d) < f(x) + \alpha\beta \nabla f(x)^T d\,, \tag{2.14}$$

for all $0 < \alpha \leq \widehat{\eta}$.

Proof. We use (C.1) from Taylor's theorem (Theorem C.1) to evaluate the function in $x + \alpha d$ by employing the information in x. We have

$$\begin{aligned} f(x + \alpha d) &= f(x) + \alpha d^T \nabla f(x) + o\big(\alpha \|d\|\big) && \text{Taylor} \\ &= f(x) + \alpha d^T \nabla f(x) + o(\alpha) && \|d\| \text{ does not depend on } \alpha. \end{aligned}$$

The result follows from the fact that $d^T \nabla f(x) < 0$ by assumption, and that $o(\alpha)$ can be made as small as needed. More formally, according to the definition of the Landau notation $o(\,\cdot\,)$ (Definition B.17), for any $\varepsilon > 0$, there exists η such that

$$\frac{\big|o(\alpha)\big|}{\alpha} < \varepsilon\,, \qquad \forall 0 < \alpha \leq \eta\,.$$

We choose $\varepsilon = -d^T \nabla f(x)$, which is positive according to Definition 2.10. Then,

$$\frac{o(\alpha)}{\alpha} \leq \frac{\big|o(\alpha)\big|}{\alpha} < -d^T \nabla f(x)\,, \qquad \forall 0 < \alpha \leq \eta\,. \tag{2.15}$$

We now need only multiply (2.15) by α to obtain

$$\alpha d^T \nabla f(x) + o(\alpha) < 0\,, \qquad \forall 0 < \alpha \leq \eta\,,$$

and $f(x + \alpha d) < f(x)$, $\forall 0 < \alpha \leq \eta$. The result (2.14) is obtained in a similar manner, by choosing $\varepsilon = (\beta - 1)\nabla f(x)^T d$ in the definition of the Landau notation. We have $\varepsilon > 0$ because $\beta < 1$ and $\nabla f(x)^T d < 0$. □

Among all directions d from a point x, the one in which the slope is the steepest is the direction of the gradient $\nabla f(x)$. To show this, we consider all directions d that have the same norm (the same length) as the gradient and compare the directional derivative for each of them.

Theorem 2.12 (Steepest ascent). *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a differentiable function. Consider* $x \in \mathbb{R}^n$ *and* $d^* = \nabla f(x)$. *Then, for all* $d \in \mathbb{R}^n$ *such that* $\|d\| = \big\|\nabla f(x)\big\|$, *we have*

$$d^T \nabla f(x) \leq d^{*T} \nabla f(x) = \nabla f(x)^T \nabla f(x)\,. \tag{2.16}$$

Proof. Let d be any direction. We have

$$\begin{aligned} d^T \nabla f(x) &\leq \|d\| \left\|\nabla f(x)\right\| && \text{Cauchy-Schwartz (Theorem C.13)} \\ &= \left\|\nabla f(x)\right\|^2 && \text{assumption } \|d\| = \left\|\nabla f(x)\right\| \\ &= \nabla f(x)^T \nabla f(x) && \text{definition of a scalar product} \\ &= {d^*}^T \nabla f(x) && \text{definition of } d^* . \end{aligned}$$

Since ${d^*}^T \nabla f(x) = \left\|\nabla f(x)\right\|^2 \geq 0$, the function increases in the direction d^*, which corresponds to the steepest ascent. □

If the direction of the gradient corresponds to the steepest ascent of the function x, we need only consider the direction opposite the gradient $-\nabla f(x)$ to find the steepest descent.

Corollary 2.13 (Steepest descent). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Consider $x \in \mathbb{R}^n$ and $d^* = -\nabla f(x)$. Then, for any $d \in \mathbb{R}^n$ such that $\|d\| = \left\|\nabla f(x)\right\|$, we have*

$$-\nabla f(x)^T \nabla f(x) = {d^*}^T \nabla f(x) \leq d^T \nabla f(x) \tag{2.17}$$

and the direction opposite the gradient is that in which the function has its steepest descent.

Proof. Let d be any direction. We have

$$\begin{aligned} -d^T \nabla f(x) &\leq \nabla f(x)^T \nabla f(x) && \text{by applying Theorem 2.12 at } -d \\ &= -\left(d^*\right)^T \nabla f(x) && \text{according to the definition of } d^* \end{aligned}$$

and $-d^T \nabla f(x) \leq -\left(d^*\right)^T \nabla f(x)$. We get (2.17) by multiplying this last inequality by -1. Since ${d^*}^T \nabla f(x) \leq 0$, the function is decreasing in the direction d^*, which corresponds to the steepest descent. □

Example 2.14 (Steepest ascent). Consider $f(x) = \frac{1}{2}x_1^2 + 2x_2^2$, and $x = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$. We consider three directions

$$d_1 = \nabla f(x) = \begin{pmatrix} 1 \\ 4 \end{pmatrix}, \qquad d_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \text{and} \qquad d_3 = \begin{pmatrix} -1 \\ 3 \end{pmatrix}.$$

The directional derivative in f in each direction equals:

$$\begin{aligned} d_1^T \nabla f(x) &= 17 \\ d_2^T \nabla f(x) &= 5 \\ d_3^T \nabla f(x) &= 11 . \end{aligned}$$

The shape of the function in each of these directions is shown in Figure 2.3. For each direction d_i, the function $f(x + \alpha d_i)$ is represented for values of α between 0 and 1.

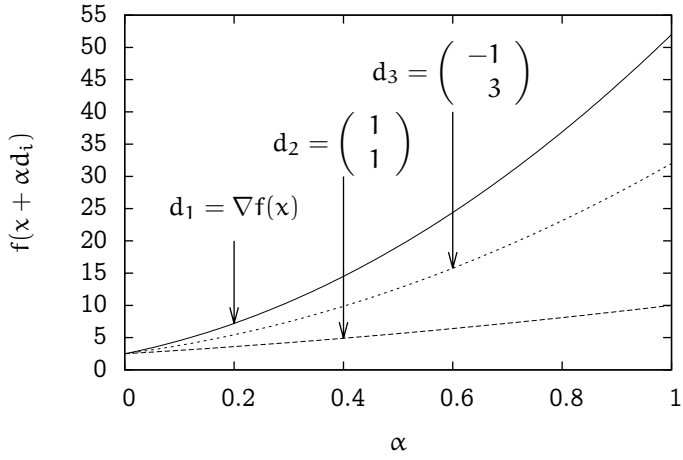


Figure 2.3: Shape of the function $\frac{1}{2}x_1^2 + 2x_2$ at point $(1,1)^T$ in several directions

Example 2.15 (Steepest descent). Consider $f(x) = \frac{1}{2}x_1^2 + 2x_2^2$, and $x = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$. We consider three directions

$$d_1 = -\nabla f(x) = \begin{pmatrix} -1 \\ -4 \end{pmatrix}, \qquad d_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \qquad \text{and} \qquad d_3 = \begin{pmatrix} 1 \\ -3 \end{pmatrix}.$$

The directional derivative in f in each direction equals:

$$\begin{aligned} d_1^T \nabla f(x) &= -17 \\ d_2^T \nabla f(x) &= -5 \\ d_3^T \nabla f(x) &= -11\,. \end{aligned}$$

The shape of the function in each of these directions is shown in Figure 2.4. For each direction d_i, the function $f(x + \alpha d_i)$ is represented for values of α between 0 and 1. It is important to note in this figure that the function does not constantly decrease along one descent direction. The descent feature is local, i.e., valid in a neighborhood of x. Even if the function increases later, the steepest descent is locally observed in the direction $-\nabla f(x)$.

In addition to providing information on the slope of the function, the gradient also enables to verify whether the function is convex or concave.

Theorem 2.16 (Convexity according to the gradient). *Let $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ be a differentiable function on an open convex set X. f is convex on X if and only if*

$$f(y) - f(x) \geq (y - x)^T \nabla f(x)\,, \qquad \forall x, y \in X\,. \tag{2.18}$$

f is strictly convex on X if and only if

$$f(y) - f(x) > (y - x)^T \nabla f(x)\,, \qquad \forall x, y \in X\,. \tag{2.19}$$

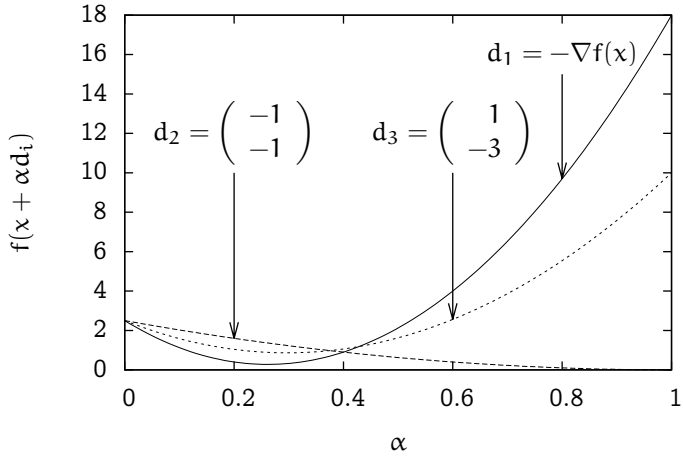


Figure 2.4: Shape of the function $\frac{1}{2}x_1^2 + 2x_2$ at point $(1,1)^T$ in several directions

Proof. Necessary condition. We first show that (2.18) is a necessary condition. Let $x, y \in X$ be arbitrary and let us consider $d = y - x$. We evaluate the directional derivative of f in the direction d and obtain

$$\begin{aligned}
(y-x)^T \nabla f(x) &= d^T \nabla f(x) && \text{definition of } d \\
&= \lim_{\alpha \xrightarrow{>} 0} \frac{f(x+\alpha d) - f(x)}{\alpha} && \text{Definition 2.7} \\
&= \lim_{\alpha \xrightarrow{>} 0} \frac{f\big(x+\alpha(y-x)\big) - f(x)}{\alpha} && \text{definition of } d.
\end{aligned}$$

Since the limit is reached for $\alpha \to 0$, we can assume without loss of generality that $\alpha < 1$. We obtain, by convexity of f, and by applying Definition 2.1 with $\lambda = 1 - \alpha$,

$$(y-x)^T \nabla f(x) \leq \lim_{\alpha \xrightarrow{>} 0} \frac{(1-\alpha)f(x) + \alpha f(y) - f(x)}{\alpha}. \tag{2.20}$$

We now need only simplify (2.20) to obtain (2.18).

Sufficient condition. We now assume that (2.18) is satisfied, and let us demonstrate the convexity of f. Let $x, y \in X$ be arbitrary, and let us take $z = \lambda x + (1-\lambda)y$. $z \in X$ because X is convex. We apply (2.18) first for z and x, and then for z and y:

$$\begin{aligned}
f(x) - f(z) &\geq (x-z)^T \nabla f(z) \\
f(y) - f(z) &\geq (y-z)^T \nabla f(z).
\end{aligned} \tag{2.21}$$

We multiply the first inequality by λ and the second by $(1-\lambda)$ and sum them up to obtain

$$\lambda f(x) + (1-\lambda) f(y) - f(z) \geq \big(\lambda x + (1-\lambda)y - z\big)^T \nabla f(z). \tag{2.22}$$

According to the definition of z, we obtain the characterization (2.2) of Definition 2.1, and f is convex.

The proof for the strictly convex case is identical. □

If we write (2.18) in a slightly different way:

$$f(y) \geq f(x) + (y - x)^T \nabla f(x), \qquad \forall x, y \in \mathbb{R}^n, \tag{2.23}$$

the right term is nothing else than the equation of the hyperplane that is tangent to the function f at point x. In this case, we see that a function is convex if and only if the graph is never below the hyperplane tangent. Figure 2.5 illustrates this idea in the case of a function with one variable.

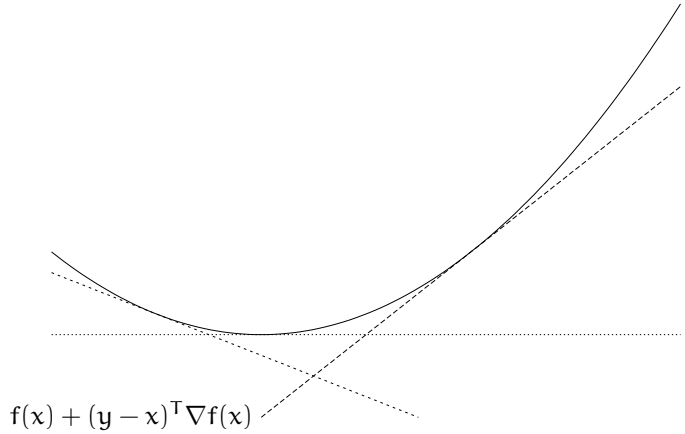


Figure 2.5: Hyperplane tangent to a convex function

We conclude this section by generalizing the notion of the gradient for the functions of $\mathbb{R}^n \to \mathbb{R}^m$. In this case, the various partial derivatives are arranged in a matrix called the *gradient matrix*. Each column of this matrix is the gradient of the corresponding component of f.

Definition 2.17 (Gradient matrix)**.** Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be such that $f_i : \mathbb{R}^n \to \mathbb{R}$ is differentiable, for $i = 1, \ldots, m$. In this case, f is differentiable, and the function $\nabla f(x) : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ is called a *gradient matrix* and is defined by

$$\begin{aligned} \nabla f(x) &= \begin{pmatrix} | & & | \\ \nabla f_1(x) & \cdots & \nabla f_m(x) \\ | & & | \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}. \end{aligned} \tag{2.24}$$

The gradient matrix is often used in its transposed form and is then called the Jacobian matrix of f.

Definition 2.18 (Jacobian matrix)**.** Consider $f : \mathbb{R}^n \to \mathbb{R}^m$. The function $J(x) : \mathbb{R}^n \to \mathbb{R}^{m \times n}$ is called a Jacobian matrix and is defined as

$$J(x) = \nabla f(x)^T = \begin{pmatrix} \text{——} & \nabla f_1(x)^T & \text{——} \\ & \vdots & \\ \text{——} & \nabla f_m(x)^T & \text{——} \end{pmatrix}. \tag{2.25}$$

Born in Potsdam (Germany) on December 10, 1804, and died in Berlin on February 18, 1851, Jacobi taught at Königsberg with Neumann and Bessel. He contributed significantly to the theory of elliptic functions, in competition with Abel. His work on first-order partial differential equations and determinants are of prime importance. Although introduced by Cauchy in 1815, the determinant function is called Jacobian thanks to a long thesis published by Jacobi in 1841. The determinant of the Jacobian matrix (Definition 2.18) is called the *Jacobian*.

Figure 2.6: Carl Gustav Jacob Jacobi

2.3 Differentiability: the second order

We can perform the same differentiability analysis as that on the function f in Section 2.2 for each of the functions $\nabla_i f(x)$ of Definition 2.4. The jth partial derivative of $\nabla_i f(x)$ is the second derivative of f with respect to the variables i and j, because

$$\frac{\partial \nabla_i f(x)}{\partial x_j} = \frac{\partial \left(\partial f(x) / \partial x_i \right)}{\partial x_j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}. \tag{2.26}$$

It is common to organize the second derivatives in an $n \times n$ matrix in which the element of line i and column j is $\partial^2 f(x) / \partial x_i \partial x_j$. This matrix is called Hessian and gets its name from the German mathematician Otto Hesse (Figure 2.7).

Definition 2.19 (Hessian matrix)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function. The function $\nabla^2 f(x) : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ is called the Hessian matrix or Hessian of f and

is defined by

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix}. \tag{2.27}$$

The Hessian matrix is always symmetric.

Note that the Hessian of f is the gradient matrix and the Jacobian matrix of ∇f.

Example 2.20 (Hessian). Consider $f(x_1, x_2, x_3) = e^{x_1} + x_1^2 x_3 - x_1 x_2 x_3$. The Hessian of f is given by

$$\nabla^2 f(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1} + 2x_3 & -x_3 & 2x_1 - x_2 \\ -x_3 & 0 & -x_1 \\ 2x_1 - x_2 & -x_1 & 0 \end{pmatrix}. \tag{2.28}$$

Just like the gradient, the Hessian gives us information about the convexity of the function.

Theorem 2.21 (Convexity by the Hessian). *Let $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function on an open convex set X. If $\nabla^2 f(x)$ is positive semidefinite (resp. positive definite) for all x in X, then f is convex (resp. strictly convex) on X.*

Proof. Consider x and y in X. We utilize (C.4) of Taylor's theorem (Theorem C.2) to evaluate the function in y by using the information in x. By writing $d = y - x$, we have

$$f(y) = f(x) + (y - x)^T \nabla f(x) + \frac{1}{2} d^T \nabla^2 f(x + \alpha d) d\,. \tag{2.29}$$

If $0 < \alpha \leq 1$, $x + \alpha d \in X$ by convexity (Definition B.2) of X, and the matrix $\nabla^2 f(x + \alpha d)$ is positive semidefinite (Definition B.8). Therefore,

$$d^T \nabla^2 f(x + \alpha d) d \geq 0\,. \tag{2.30}$$

Then, we have

$$f(y) \geq f(x) + (y - x)^T \nabla f(x)\,. \tag{2.31}$$

We now need only invoke Theorem 2.16 to prove the convexity of f. The strict convexity is proven in a similar manner. □

It is interesting to analyze the second order information along a given direction d. If we have a twice differentiable function f, a point x and a direction d, we can calculate the derivatives in this direction by considering the function of one variable

$$g : \mathbb{R}_+ \longrightarrow \mathbb{R} : \alpha \rightsquigarrow f(x + \alpha d)\,. \tag{2.32}$$

According to the chain differentiation rule, we have

$$g'(\alpha) = d^T \nabla f(x + \alpha d)\,. \tag{2.33}$$

Note that $g'(0)$ is the directional derivative (Definition 2.7) of f at x along d. We also have

$$g''(\alpha) = d^T \nabla^2 f(x + \alpha d) d\,. \tag{2.34}$$

Since the second derivative of a function of one variable gives us curvature information, (2.34) provides us with information about the curvature of the function f in the direction d. In particular, when $\alpha = 0$, this expression gives information on the curvature of f in x. To avoid the length of the direction influencing the notion of curvature, it is important to normalize. We obtain the following definition.

Definition 2.22 (Curvature)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function. Consider x, $d \in \mathbb{R}^n$. The quantity

$$\frac{d^T \nabla^2 f(x) d}{d^T d} \tag{2.35}$$

represents the curvature of the function f in x in the direction d.

In linear algebra, the quantity (2.35) is often called the Rayleigh quotient of $\nabla^2 f(x)$ in the direction d. One should immediately note that the curvature of the function x in the direction $-d$ is exactly the same as in the direction d.

Example 2.23 (Curvature)**.** Consider $f(x) = \frac{1}{2}x_1^2 + 2x_2^2$, and $x = (1, 1)^T$, as in Examples 2.14 and 2.15. The curvature of the function in different directions is given below:

d	$-d$	$d^T \nabla^2 f(x) d / d^T d$
$(\ 1 \quad 4\ )^T$	$(\ -1 \quad -4\ )^T$	3.8235
$(\ 1 \quad 1\ )^T$	$(\ -1 \quad -1\ )^T$	2.5
$(\ -1 \quad 3\ )^T$	$(\ 1 \quad -3\ )^T$	3.7

Otto Hesse was born in Königsberg (currently Kaliningrad, Russia) on April 22, 1811, and died in Munich (Germany) on August 4, 1874. He was a student of Jacobi and in 1845 was appointed Extraordinary Professor at Königsberg, where he taught for 16 years. Kirchkoff and Lipschitz attended his courses. Hesse was also affiliated with the University of Halle, in Heidelberg and with Munich Polytechnicum. He worked mainly on the theory of algebraic functions and the theory of invariants.

Figure 2.7: Ludwig Otto Hesse

2.4 Linearity and non linearity

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *linear* if its value is a linear combination of variables.

Definition 2.24 (Linear function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be linear if it is defined as

$$f(x) = c^T x = \sum_{i=1}^{n} c_i x_i \,, \tag{2.36}$$

where $c \in \mathbb{R}^n$ is a constant vector, i.e., independent of x. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is linear if each of its components $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$, is linear. In this case, it can be written as

$$f(x) = Ax \,, \tag{2.37}$$

where $A \in \mathbb{R}^{m \times n}$ is a matrix of constants.

When a constant term is added to a linear function, the result is said to be *affine*.

Definition 2.25 (Affine function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be affine if it is written as

$$f(x) = c^T x + d = \sum_{i=1}^{n} c_i x_i + d \,, \tag{2.38}$$

where $c \in \mathbb{R}^n$ is a vector of constants and $d \in \mathbb{R}$. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is affine if each of its components $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$, is affine. In this case, it can be written as

$$f(x) = Ax + b \,, \tag{2.39}$$

where $A \in \mathbb{R}^{m \times n}$ is a matrix and $b \in \mathbb{R}^m$ is a vector.

Note that minimizing (2.38) is equivalent to minimizing (2.36). Note that all linear functions are affine. By abuse of language, a non linear function is actually a function that is not affine.

Definition 2.26 (Non linear function)**.** Any function that is not affine is said to be *non linear*.

The set of non linear functions is vast, and one needs to be a little more precise in their characterization. Intuitively, the function shown in Figure 2.8 seems more non linear than the one in Figure 2.9. The slope of the former changes quickly with x, which is not the case for the latter. This is formally captured by the Lipschitz continuity (Definition B.16) of the gradient.

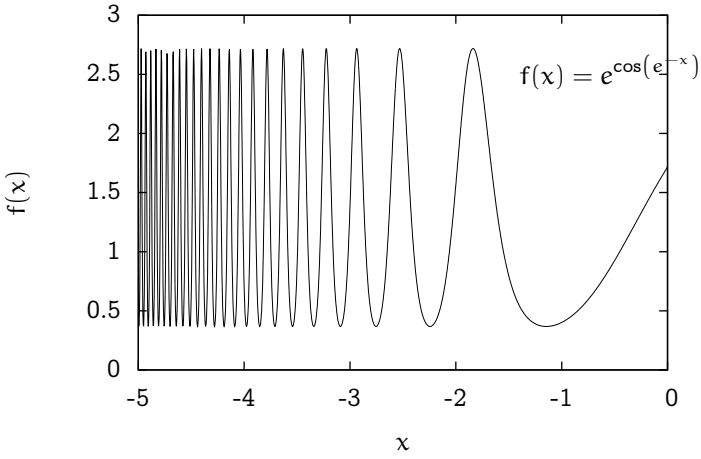


Figure 2.8: Example of a non linear function

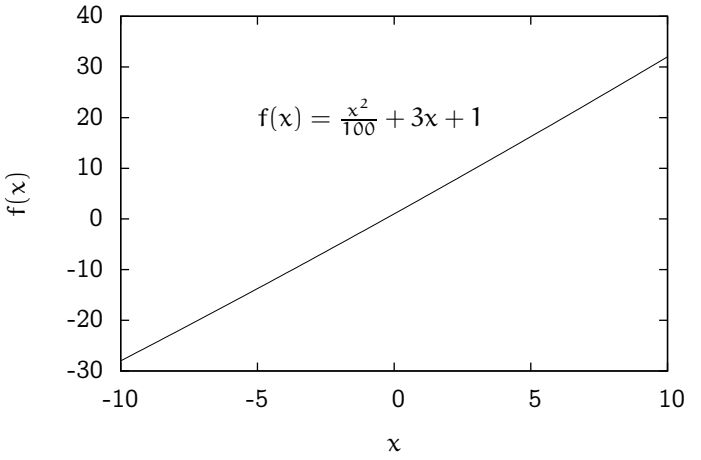


Figure 2.9: Example of another non linear function

Definition 2.27 (Lipschitz continuity of the gradient)**.** Consider $f : X \subseteq \mathbb{R}^n \to \mathbb{R}^m$. The gradient matrix of the function is Lipschitz continuous on X if there exists a constant $M > 0$ such that, for all $x, y \in X$, we have

$$\left\|\nabla f(x) - \nabla f(y)\right\|_{n\times m} \leq M\left\|x - y\right\|_n , \tag{2.40}$$

where $\|\cdot\|_{n\times m}$ is a norm on $\mathbb{R}^{n\times m}$ and $\|\cdot\|_n$ is a norm on $\mathbb{R}^n$. The constant M is called the Lipschitz constant.

Intuitively, the definition says that the slopes of the function at two close points are close as well. And the more so when M is small. Actually, when f is linear, the slope is the same at any point, and (2.40) is verified with $M = 0$. The value of M for the function represented in Figure 2.9 is low, while it is large for the function illustrated in Figure 2.8, where the slope varies dramatically with small modifications of x.

The constant M can be interpreted as an upper bound on the curvature of the function. The greater M is, the smaller the curvature is. If $M = 0$, the curvature is zero, and the function is linear. Note that this constant is essentially theoretical and that it is generally difficult to obtain a value for it.

Among the non linear functions, quadratic functions play an important role in optimization algorithms.

Definition 2.28 (Quadratic function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be quadratic if it can be written as

$$f(x) = \frac{1}{2}x^T Q x + g^T x + c = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij}x_i x_j + \sum_{i=1}^{n} g_i x_i + c , \tag{2.41}$$

where Q is a symmetric matrix $n \times n$, $g \in \mathbb{R}^n$ and $c \in \mathbb{R}$. We have

$$\nabla f(x) = Qx + g \qquad \text{and} \qquad \nabla^2 f(x) = Q . \tag{2.42}$$

The presence of the factor $\frac{1}{2}$ enables a simplification of the expression of $\nabla f(x)$. Note also that the fact that Q is symmetric is not restrictive. Indeed, if Q was not symmetric, we would have

$$x^T Q x = \sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij}x_i x_j = \sum_{i=1}^{n}\sum_{j=i}^{n} \frac{1}{2}(Q_{ij} + Q_{ji})x_i x_j .$$

We now define the symmetric matrix Q' such that $Q'_{ij} = Q'_{ji} = \frac{1}{2}(Q_{ij} + Q_{ji})$. We obtain $x^T Q x = x^T Q' x$, and the same function can be written using a symmetric matrix.

2.5 Conditioning and preconditioning

In linear algebra, the notion of conditioning is related to the analysis of numerical errors that can occur when solving a linear system (see Golub and Van Loan, 1996).

Definition 2.29 (Condition number)**.** Let $A \in \mathbb{R}^{n \times n}$ be a non singular symmetric matrix. The condition number for A is

$$\kappa(A) = \|A\| \left\|A^{-1}\right\| . \tag{2.43}$$

If the matrix norm used is the norm 2, we have

$$\kappa_2(A) = \|A\|_2 \left\|A^{-1}\right\|_2 = \frac{\sigma_1}{\sigma_n} , \tag{2.44}$$

where σ_1 is the largest singular value of A (Definition B.28) and σ_n is the smallest. By extension, the condition number of a singular matrix (i.e., such that $\sigma_n = 0$) is $+\infty$. If A is symmetric positive semidefinite, the singular values of A are its eigenvalues (Definition B.7).

We propose a geometric interpretation of the condition number. For this, we consider a non linear function $f : \mathbb{R}^n \to \mathbb{R}$ and a vector $x \in \mathbb{R}^n$. We assume that the matrix $\nabla^2 f(x)$ is positive definite[1], and let λ_1 be its largest eigenvalue and λ_n its smallest. Let d_1 be an eigenvector corresponding to λ_1. We have

$$A d_1 = \lambda_1 d_1 . \tag{2.45}$$

By premultiplying by d_1^T and normalizing, we obtain

$$\lambda_1 = \frac{d_1^T A d_1}{d_1^T d_1} . \tag{2.46}$$

According to Definition 2.22, the eigenvalue λ_1 corresponds to the curvature of the function in the direction of the eigenvector d_1. In addition, according to the Rayleigh-Ritz theorem (Theorem C.4), this is the greatest curvature among all possible directions. A similar reasoning for the smallest eigenvalue allows us to determine λ_n as the smallest curvature of the function in all possible directions. The condition number of the Hessian matrix at a point x is the ratio between the largest and the smallest curvature among the directions when starting from x.

Definition 2.30 (Conditioning)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function, and let us take a vector $x \in \mathbb{R}^n$. The conditioning of f at x is the condition number of $\nabla^2 f(x)$.

[1] $\nabla^2 f(x)$ is always symmetric.

By using this interpretation related to curvature, an ill-conditioned function is characterized by a large difference in curvature between two directions. In the case of a quadratic function with two dimensions (Figure 2.10(a)), this translates into level curves forming elongated ellipses (Figure 2.10(b)). A well conditioned function is characterized by a homogeneous curvature in the various directions. In the case of a quadratic function with two dimensions (Figure 2.11(a)), this translates into nearly circular level curves (Figure 2.11(b)).

Example 2.31 (Conditioning)**.** The quadratic function

$$f(x_1, x_2) = 2x_1^2 + 9x_2^2 \tag{2.47}$$

is such that its condition number is $9/2$ for all x, because

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} 4 & 0 \\ 0 & 18 \end{pmatrix}. \tag{2.48}$$

We now apply the change of variables

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 3\sqrt{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{2.49}$$

i.e.,

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & \sqrt{2}/6 \end{pmatrix} \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}. \tag{2.50}$$

We obtain

$$f(x_1', x_2') = \frac{1}{2} {x_1'}^2 + \frac{1}{2} {x_2'}^2, \tag{2.51}$$

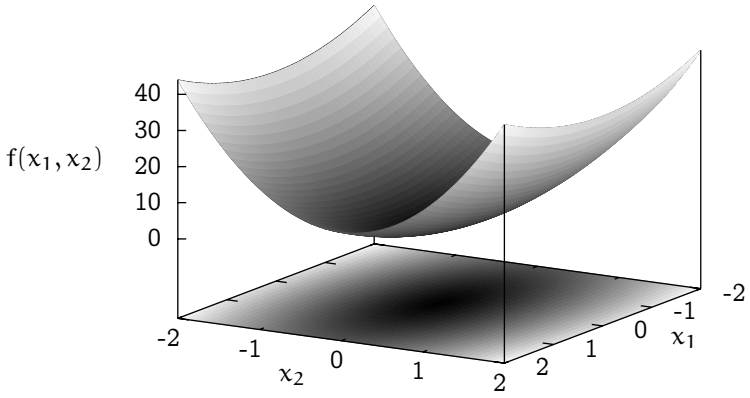
for which the Hessian is the identity matrix, and the condition number is 1, for all (x_1', x_2').

We see that it is possible to reduce the condition by a change of variables. In general, a change of variables is defined by an invertible matrix M.

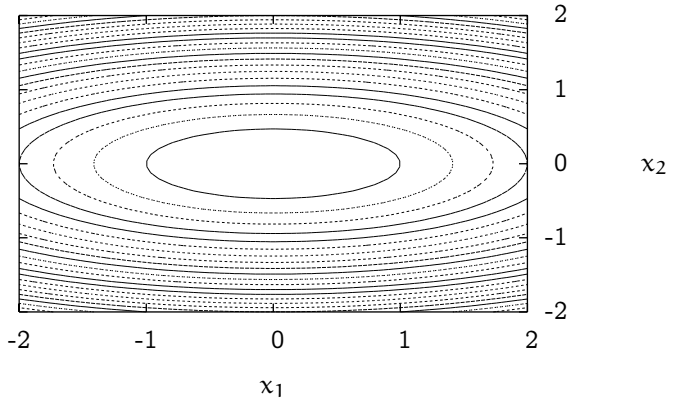
Definition 2.32 (Change of variables)**.** Consider $x \in \mathbb{R}^n$. Let $M \in \mathbb{R}^{n \times n}$ be an invertible matrix. The change of variables is the linear application defined by M that transforms x into $x' = Mx$.

Consider a function $f(x)$ and apply to it the change of variables $x' = Mx$ to obtain the function $\tilde{f}(x')$. We have, according to the chain differentiation rule (Theorem C.3),

$$\begin{aligned} \tilde{f}(x') &= f(M^{-1}x') \\ \nabla \tilde{f}(x') &= M^{-T} \nabla f(M^{-1}x') \\ \nabla^2 \tilde{f}(x') &= M^{-T} \nabla^2 f(M^{-1}x') M^{-1} \\ &= M^{-T} \nabla^2 f(x) M^{-1}. \end{aligned} \tag{2.52}$$



(a) Function



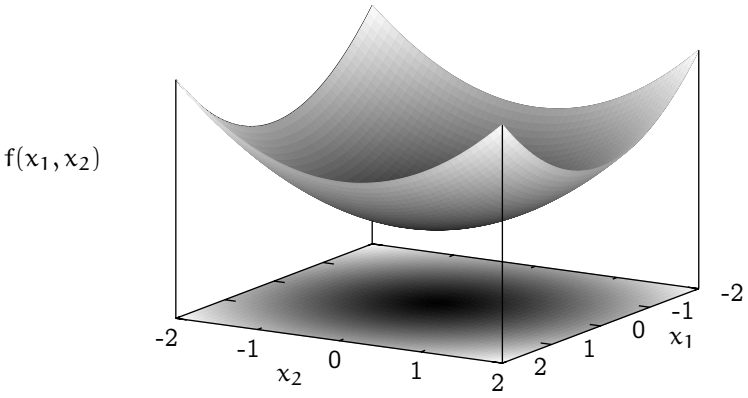
(b) Level curves

Figure 2.10: Function (2.47) of Example 2.31

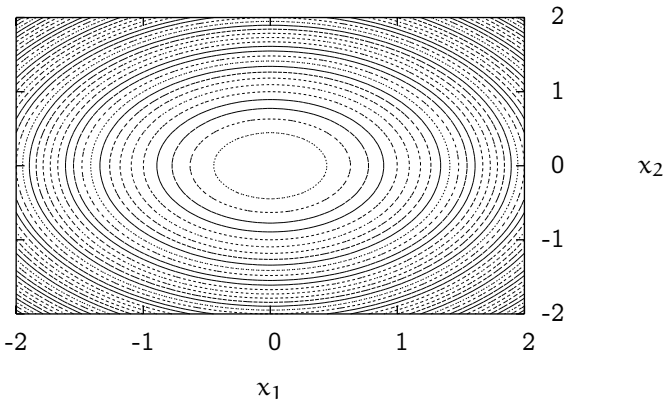
The condition of $\tilde{f}$ in x' is the condition number of the matrix $M^{-T}\nabla^2 f(x)M^{-1}$ (where M^{-T} is the inverse of the transpose of M). Choosing a change of variables such that the condition is as close as possible to 1 is called *preconditioning*.

Definition 2.33 (Preconditioning)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function, and let us take a vector $x \in \mathbb{R}^n$. The preconditioning of f in x involves defining a change of variables $x' = Mx$ and a function $\tilde{f}(x') = f(M^{-1}x')$, such that the conditioning of $\tilde{f}$ in Mx is better than the conditioning of f in x.

In the context of optimization, the matrix for the change of variables should be positive definite in order to preserve the nature of the problem.



(a) Function



(b) Level curves

Figure 2.11: Function (2.51) of Example 2.31

If $\nabla^2 f(x)$ is positive definite, we can calculate the Cholesky decomposition (Definition B.18)

$$\nabla^2 f(x) = LL^T, \tag{2.53}$$

where L is a lower triangular matrix. We now choose the change of variables

$$x' = L^T x \iff x = L^{-T} x'. \tag{2.54}$$

In this case

$$\begin{aligned} \nabla^2 \tilde{f}(x') &= L^{-1} \nabla^2 f(x) L^{-T} && \text{according to (2.52)} \\ &= L^{-1} L L^T L^{-T} && \text{according to (2.53)} \\ &= I. \end{aligned}$$

The conditioning of the function $\tilde{f}$ in x' is 1. According to Definition 2.29, $\kappa_2\left(\nabla^2\tilde{f}(x')\right) \geq 1$ and the obtained conditioning is the best possible. The best preconditioning of f in x consists in defining the change of variables based on the Cholesky factorization of $\nabla^2 f(x)$.

Example 2.34 (Preconditioning)**.** Consider

$$f(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{25}{2}x_2^2 + 3x_1x_2 - 12x_1 - \sqrt{\pi}x_2 - 6. \tag{2.55}$$

We have

$$\nabla f(x_1, x_2) = \begin{pmatrix} x_1 + 3x_2 - 12 \\ 3x_1 + 25x_2 - \sqrt{\pi} \end{pmatrix} \tag{2.56}$$

and

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} 1 & 3 \\ 3 & 25 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix}^T. \tag{2.57}$$

We define

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 0 & 4 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{2.58}$$

i.e.,

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & -3/4 \\ 0 & 1/4 \end{pmatrix}\begin{pmatrix} x_1' \\ x_2' \end{pmatrix}. \tag{2.59}$$

We obtain

$$\begin{aligned} \tilde{f}(x_1', x_2') = {} & \frac{1}{2}\left(x_1' - \frac{3}{4}x_2'\right)^2 + \frac{25}{2}\left(\frac{1}{4}x_2'\right)^2 + \frac{3}{4}\left(x_1' - \frac{3}{4}x_2'\right)x_2' \\ & - 12\left(x_1' - \frac{3}{4}x_2'\right) - \frac{\sqrt{\pi}}{4}x_2' - 6 \\ = {} & \frac{1}{2}x_1'^2 + \frac{1}{2}x_2'^2 - 12x_1' + \left(9 - \frac{\sqrt{\pi}}{4}\right)x_2' - 6. \end{aligned} \tag{2.60}$$

It is easy to verify that $\nabla^2\tilde{f}(x_1', x_2') = I$. Note that there are no longer any crossed terms in $x_1'\ x_2'$.

2.6 Exercises

Exercise 2.1. Among the following functions, which are convex and which are concave? Justify your answer.

1. $f(x) = 1 - x^2$.
2. $f(x) = x^2 - 1$.
3. $f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$.
4. $f(x) = x^3$.
5. $f(x_1, x_2, x_3) = \sin(a)x_1 + \cos(b)x_2 + e^{-c}x_3,\ a, b, c \in \mathbb{R}$.

Exercise 2.2. For each of the following functions:

- Calculate the gradient.
- Calculate the Hessian.
- Specify (and justify) whether the function is convex, concave, or neither.
- Calculate the curvature of the function in a direction d at the specified point $\bar{x}$.
- Make a change of variables to precondition the function, using the Hessian at the specified point $\bar{x}$. Please note that the matrix for a change of variables must be positive definite.

1. $f(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2,\ \bar{x} = (0, 0)^T$.
2. $f(x_1, x_2) = \frac{1}{3}x_1^3 + x_2^3 - x_1 - x_2,\ \bar{x} = (9, 1)^T$.
3. $f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2)^2 x_2^2 + (x_2 + 1)^2,\ \bar{x} = (2, -1)^T$.
4. $f(x_1, x_2) = x_1^2 + 2x_1x_2 + 2x_2^2,\ \bar{x} = (1, 1)^T$.
5. $f(x_1, x_2) = x_1^2 - x_1x_2 + 2x_2^2 - 2x_1 + e^{x_1 + x_2},\ \bar{x} = (0, 0)^T$.

Exercise 2.3. Consider $f : \mathbb{R}^n \to \mathbb{R}$, a point $x \in \mathbb{R}^n$ and a direction $d \in \mathbb{R}^n$, $d \neq 0$ such that $f(x + \alpha d) = f(x)$ for all $\alpha \in \mathbb{R}$. What is the curvature of f in x in the direction d? What is the conditioning of f in x?

Chapter 3

Constraints

Life would be easier without constraints. Or would it? In this chapter, we investigate ways to remove some of them, or even all of them. And when some remain, they need to be properly understood in order to be verified. As algorithms need to move along directions that are compatible with the constraints, such directions are characterized in various contexts. We put a special emphasis on linear constraints, which the analysis simplifies significantly.

Contents

In this chapter, we analyze the constraints of the optimization problem: $\min_{x \in \mathbb{R}^n} f(x)$ subject to $h(x) = 0, g(x) \leq 0, x \in X$. A vector satisfying all constraints is called a *feasible point*.

Definition 3.1 (Feasible point)**.** Consider the optimization problem (1.71)–(1.74). A point $x \in \mathbb{R}^n$ is said to be *feasible* if it satisfies all constraints (1.72)–(1.74). Note that, in the literature, this concept is sometimes called a *feasible solution* or *feasible vector*.

3.1 Active constraints

The concept of active constraints is relevant mainly for inequality constraints. We introduce the concept with an example.

Example 3.2 (Inequality constraints)**.** Consider the optimization problem

$$\min_{x\in\mathbb{R}} x^2 \tag{3.1}$$

subject to

$$\begin{array}{rcl} x & \leq & 4 \\ x & \geq & -10. \end{array} \tag{3.2}$$

It is illustrated in Figure 3.1, where the inequality constraints are represented by vertical lines, associated with an arrow pointed towards the feasible domain. The solution to this problem is $x^* = 0$. In fact, one could also choose to ignore the constraints and still obtain the same solution. We say that the constraints are inactive at the solution. When using the notation (1.73), the problem can be written as

$$\min_{x\in\mathbb{R}} x^2 \tag{3.3}$$

subject to

$$\begin{array}{rcrcl} g_1(x) & = & x-4 & \leq & 0 \\ g_2(x) & = & -x-10 & \leq & 0. \end{array} \tag{3.4}$$

We have $g_1(x^*) = -4$ and $g_2(x^*) = -10$, and $g_1(x^*) < 0$ and $g_2(x^*) < 0$. The fact that the inequality constraints are strictly verified characterizes inactive constraints.

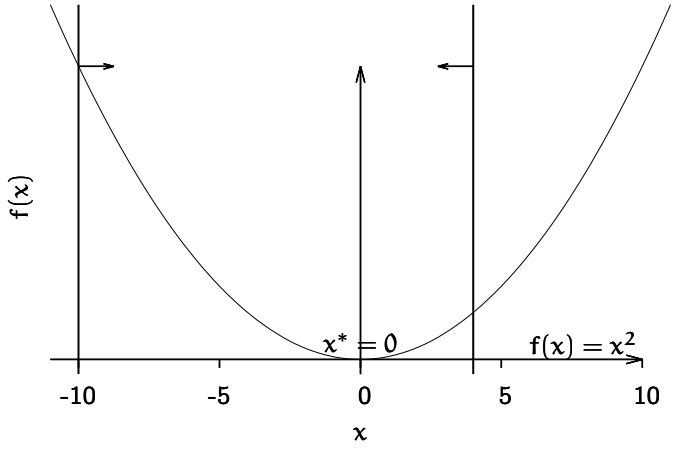


Figure 3.1: Illustration of Example 3.2

Example 3.3 (Inequality constraints II)**.** Consider the optimization problem

$$\min_{x \in \mathbb{R}} x^2 \tag{3.5}$$

subject to

$$\begin{array}{rcl} x & \leq & 4 \\ x & \geq & 1. \end{array} \tag{3.6}$$

It is illustrated in Figure 3.2, where the inequality constraints are represented by vertical lines, associated with an arrow pointed towards the feasible domain. The solution to this problem is $x^* = 1$. In this case, we can ignore the constraint $x \leq 4$. However, the constraint $x \geq 1$ cannot be ignored without modifying the solution. It is said to be *active*. Using the notation (1.73), the problem can be written as

$$\min_{x \in \mathbb{R}} x^2 \tag{3.7}$$

subject to

$$\begin{array}{rcccl} g_1(x) & = & x-4 & \leq & 0 \\ g_2(x) & = & 1-x & \leq & 0. \end{array} \tag{3.8}$$

We have $g_1(x^*) = -3$ and $g_2(x^*) = 0$, and $g_1(x^*) < 0$ and $g_2(x^*) = 0$. The first constraint is verified with strict inequality and is inactive. The second constraint is verified with equality and is active.

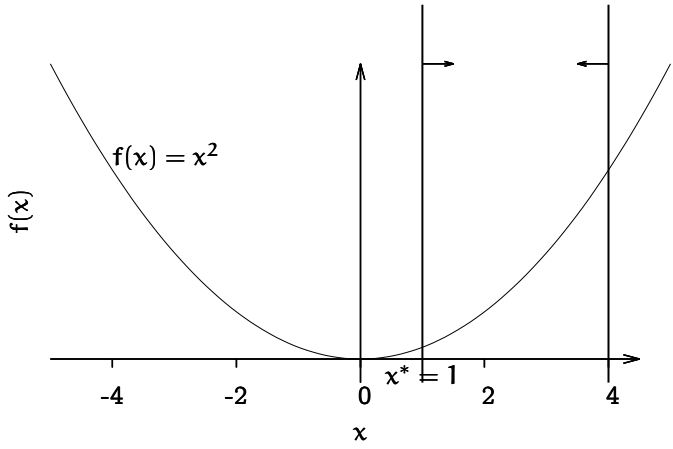


Figure 3.2: Illustration of Example 3.3

Definition 3.4 (Active constraints)**.** Consider $g : \mathbb{R}^n \to \mathbb{R}^p$ and $h : \mathbb{R}^n \to \mathbb{R}^m$. For $1 \leq i \leq p$, an inequality constraint

$$g_i(x) \leq 0 \tag{3.9}$$

is said to be *active* in x^* if

$$g_i(x^*) = 0, \tag{3.10}$$

and *inactive* in x^* if

$$g_i(x^*) < 0. \tag{3.11}$$

By extension, for $1 \leq i \leq m$, an equality constraint

$$h_i(x) = 0 \tag{3.12}$$

is said to be active at x^* if it is satisfied in x^*, i.e.,

$$h_i(x^*) = 0. \tag{3.13}$$

The set of indices of the active constraints in x^* is denoted by $\mathcal{A}(x^*)$.

This concept of active constraints is attractive because in x^* the active constraints can be considered equality constraints, while the inactive constraints can be ignored. In Example 3.2, the unconstrained optimization problem $\min_{x \in \mathbb{R}} x^2$ has exactly the same solution as (3.1)–(3.2). In Example 3.3, the constrained optimization problem $\min_{x \in \mathbb{R}} x^2$ subject to $x = 1$ has exactly the same solution as (3.5)–(3.6).

Theorem 3.5 (Active constraints). *Take a vector $x^* \in \mathbb{R}^n$. Consider the following optimization problem P_1*

$$\min_{x \in \mathbb{R}^n} f(x) \tag{3.14}$$

subject to

$$g(x) \leq 0, \tag{3.15}$$

$$x \in Y \subseteq \mathbb{R}^n, \tag{3.16}$$

where $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is continuous and Y is a subset of $\mathbb{R}^n$. If x^* is feasible, i.e., $g(x^*) \leq 0$, and if $\mathcal{A}(x^*) \subseteq \{1, \ldots, p\}$ is the set of indices of the active constraints in x^*, i.e.,

$$\mathcal{A}(x^*) = \{i | g_i(x^*) = 0\}, \tag{3.17}$$

we consider the following optimization problem P_2

$$\min_{x \in \mathbb{R}^n} f(x) \tag{3.18}$$

subject to

$$g_i(x) = 0, \quad i \in \mathcal{A}(x^*), \tag{3.19}$$

$$x \in Y \subseteq \mathbb{R}^n. \tag{3.20}$$

Here, x^* is a local minimum of P_1 if and only if x^* is a local minimum of P_2.

Proof. Sufficient condition. By continuity of g, for each inactive constraint j, there is a neighborhood of size ε_j around x^* such that the constraint j is strictly verified in the neighborhood. More formally, if $g_j(x^*) < 0$, then there exists $\varepsilon_j > 0$ such that

$$g_j(x) < 0 \quad \forall x \text{ such that } \|x - x^*\| < \varepsilon_j, \tag{3.21}$$

as illustrated in Figure 3.3. Consider two feasible neighborhoods around x^*. The first one is defined as

$$\mathcal{Y}_1(\varepsilon) = \{x | g(x) \leq 0, x \in Y \text{ and } \|x - x^*\| < \varepsilon\}, \tag{3.22}$$

and contains neighbors of x^* that are feasible for the problem P_1. The second is defined as

$$\mathcal{Y}_2(\varepsilon) = \{x | g_i(x) = 0 \; \forall i \in \mathcal{A}(x^*), x \in Y \text{ and } \|x - x^*\| < \varepsilon\}, \tag{3.23}$$

and contains neighbors of x^* that are feasible for the problem P_2.

Since x^* is a local minimum of P_1, according to Definition 1.5, there exists $\hat{\varepsilon} > 0$ such that $f(x^*) \leq f(x)$, $\forall x \in \mathcal{Y}_1(\hat{\varepsilon})$.

Consider the smallest neighborhood, and define

$$\tilde{\varepsilon} = \min(\hat{\varepsilon}, \min_{j \notin \mathcal{A}(x^*)} \varepsilon_j). \tag{3.24}$$

We show that

$$\mathcal{Y}_2(\tilde{\varepsilon}) \subseteq \mathcal{Y}_1(\hat{\varepsilon}). \tag{3.25}$$

Indeed, take any x in $\mathcal{Y}_2(\tilde{\varepsilon})$. In order to show that it belongs to $\mathcal{Y}_1(\hat{\varepsilon})$, we need to show that $g(x) \leq 0$, $x \in \mathcal{Y}$, and $\|x - x^*\| \leq \hat{\varepsilon}$. Since $\tilde{\varepsilon} \leq \hat{\varepsilon}$, we have $\|x - x^*\| < \tilde{\varepsilon} \leq \hat{\varepsilon}$, and the third condition is immediately verified. The second condition ($x \in \mathcal{Y}$) is inherited from the definition of $\mathcal{Y}_2(\tilde{\varepsilon})$. We need only to demonstrate that $g(x) \leq 0$. To do this, consider the constraints of $\mathcal{A}(x^*)$ separately from the others. By definition of $\mathcal{Y}_2$, we have $g_i(x) = 0$ for $i \in \mathcal{A}(x^*)$, which implies $g_i(x) \leq 0$. Take $j \notin \mathcal{A}(x^*)$. Since $\tilde{\varepsilon} \leq \varepsilon_j$, we have $g_j(x) < 0$ according to (3.21), which implies $g_j(x) \leq 0$. This completes the proof that $g(x) \leq 0$, so that $x \in \mathcal{Y}_1(\hat{\varepsilon})$.

As a result of (3.25), since x^* is the best element (in the sense of the objective function) of $\mathcal{Y}_1(\hat{\varepsilon})$ (according to Definition 1.5 of the local minimum), and since x^* belongs to $\mathcal{Y}_2(\tilde{\varepsilon})$, it is also the best element of this set, and a local minimum of P_2.

Necessary condition. Let X_1 be the set of feasible points of P_1 and X_2 the set of feasible points of P_2. We have $X_2 \subseteq X_1$. Let x^* be a local minimum of P_2. We assume by contradiction that it is not a local minimum of P_1. Then, for any $\varepsilon > 0$, there exists $x \in X_1$ such that $\|x - x^*\| \leq \varepsilon$ and $f(x) < f(x^*)$. Since x^* is a local minimum of P_2, x cannot be feasible for P_2, and $x \notin X_2 \subseteq X_1$, leading to the contradiction. □

We have managed to eliminate some constraints. Unfortunately, this required knowing the solution x^*. This simplification is thus relevant mainly in theory.

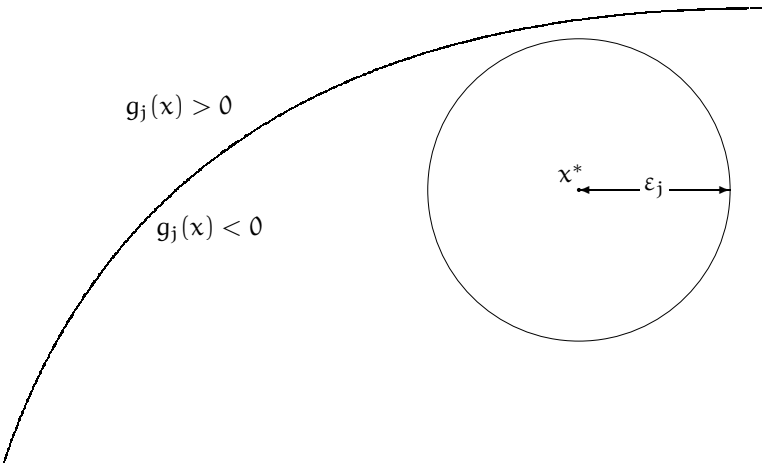


Figure 3.3: Strictly feasible neighborhood of x^* when the inequality constraint j is inactive

3.2 Linear independence of the constraints

The analysis of the structure of the constraints and their management in algorithms is complex. It is therefore necessary to introduce assumptions that are general enough not to be restrictive from an operational point of view, and that render it possible to avoid pathological cases. In particular, the linear independence of the constraints plays an important role. Even though this concept is relatively intuitive when the constraints are linear, it is necessary to define it strictly for non linear constraints.

Start with the case where all constraints are defined by affine functions (see Definition 2.25). In this case, when using the techniques of Section 1.2, it is always possible to express the optimization problem as

$$\min f(x) \tag{3.26}$$

subject to

$$Ax = b \tag{3.27}$$
$$x \geq 0 \tag{3.28}$$

by writing $h(x) = b - Ax$ in (1.72), with $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, and $g(x) = -x$ in (1.73).

Since the inequality constraints are simple, we analyze in more details the system of equations $Ax = b$. Like any linear system, three possibilities may arise:

- the system is incompatible, and there is no x such that $Ax = b$;
- the system is underdetermined, and there is an infinite number of x such that $Ax = b$;
- the system is non singular, and there is a unique x that satisfies $Ax = b$.

In an optimization context, incompatible and non singular systems have little relevance because they leave no degree of freedom to optimize any objective. We thus only consider underdetermined systems.

If the system is compatible, the rank of A (Definition B.29) gives us information on the relevance of the various constraints. If the rank is deficient, this means that certain rows of A form a linear combination of the others, and the corresponding constraints are redundant. This is formalized by Theorem 3.6 and illustrated by Example 3.7.

Theorem 3.6 (Redundant constraints). *Consider a compatible system of linear equality constraints $Ax = b$, with $A \in \mathbb{R}^{m\times n}$, $m \leq n$. If the rank of A is deficient, i.e., rank(A)=$r < m$, then there exists a matrix $\tilde{A} \in \mathbb{R}^{r\times n}$ of full rank (i.e., rank($\tilde{A}$) $= r$), composed exclusively of rows $\ell_1, \ldots, \ell_r$ of A such that*

$$\tilde{A}x = \tilde{b} \iff Ax = b, \tag{3.29}$$

where $\tilde{b}$ is composed of elements $\ell_1, \ldots, \ell_r$ of b.

Proof. Since the rank of A is r, this signifies that $m - r$ rows of A can be written as linear combinations of r other rows. Without loss of generality, we can assume that the last $m - r$ rows are linear combinations of the first r rows. By denoting a_k^T the kth row of A, we have

$$a_k = \sum_{j=1}^{r} \lambda_k^j a_j \quad k = r+1, \ldots, m \text{ and } \exists j \text{ t.q. } \lambda_k^j \neq 0. \tag{3.30}$$

Moreover, since by hypothesis the system $Ax = b$ is compatible, for each $k = r + 1, \ldots, m$ we have

$$\begin{aligned} b_k &= a_k^T x \\ &= \textstyle\sum_{j=1}^{r} \lambda_k^j a_j^T x \\ &= \textstyle\sum_{j=1}^{r} \lambda_k^j b_j. \end{aligned} \tag{3.31}$$

Denote $\tilde{A}$ the matrix composed of the first r rows of A, and $\tilde{b}$ the vector composed of the first r components of b. Then, $\ell_i = i$, $i = 1, \ldots, r$.

$\Longrightarrow$ Consider x such that $\tilde{A}x = \tilde{b}$. According to the definition of $\tilde{A}$, we have that x satisfies the first r equations of the system, i.e., $a_i^T x = b_i$ for $i = 1, \ldots, r$. Select k as an arbitrary index between $r+1$ and m, and demonstrate that x satisfies the

corresponding equation. We have

$$\begin{aligned} a_k^T x &= \sum_{j=1}^{r} \lambda_k^j a_j^T x && \text{according to (3.30)} \\ &= \sum_{j=1}^{r} \lambda_k^j b_j && \text{because } \tilde{A}x = \tilde{b} \\ &= b_k && \text{according to (3.31).} \end{aligned}$$

$\Longleftarrow$ Let x be such that $Ax = b$. x satisfies all equations of the system, particularly the first r ones. Then, it satisfies $\tilde{A}x = \tilde{b}$.

□

Example 3.7 (Redundant system). Take the constraints

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 - x_2 + x_4 &= 1 \\ x_1 - 5x_2 - 2x_3 + 3x_4 &= 1 \end{aligned} \tag{3.32}$$

i.e.,

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -5 & -2 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{3.33}$$

The rank of A is equal to 2, but there are 3 rows (the determinant of any squared submatrix of dimension 3 is zero). This means that one of the rows is a linear combination of the others. Since the system is compatible (for instance, $x_1 = 2/3$, $x_2 = 0$,$x_3 = 1/3$, $x_4 = 1/3$ is feasible), one of the constraints must be redundant. In this case, if a_i^T represents the ith row of A, we have

$$a_3 = -2a_1 + 3a_2. \tag{3.34}$$

We can remove the 3rd constraint, and the system

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 - x_2 + x_4 &= 1 \end{aligned} \tag{3.35}$$

is equivalent to the constraint system (3.32).

To generalize this result to non linear constraints $h(x) = 0$, we must linearize them by invoking Taylor's theorem (Theorem C.1) around a point x^+, i.e.,

$$h(x) = h(x^+) + \nabla h(x^+)^T (x - x^+) + o(\|(x - x^+\|).$$

In this case, by ignoring the term o of (C.1), $h(x) = 0$ can be approximated by

$$h(x^+) + \nabla h(x^+)^T x - \nabla h(x^+)^T x^+ = 0$$

or by

$$\nabla h(x^+)^T x = \nabla h(x^+)^T x^+ - h(x^+).$$

Therefore, the gradients of equality constraints play a similar role as the rows of the matrix A in (3.27). As for the inequality constraints, we saw in Section 3.1 that those that are inactive at x^+ can be ignored, and that those that are active can be considered equality constraints. Consequently, we can define the condition of linear independence as follows.

Definition 3.8 (Linear independence of the constraints)**.** Consider the optimization problem (1.71)–(1.73) $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$ and $g(x) \leq 0$, and x^+ a feasible point. The linear independence of the constraints is satisfied in x^+ if the gradients of the equality constraints and the gradients of the active inequality constraints in x^+ are linearly independent. By abuse of language, it is sometimes simply said that the constraints are linearly independent.

Example 3.9 (Linear independence of constraints)**.** Take an optimization problem in $\mathbb{R}^2$ with the inequality constraint

$$g(x) = x_1^2 + (x_2 - 1)^2 - 1 \leq 0 \tag{3.36}$$

and the equality constraint

$$h(x) = x_2 - x_1^2 = 0. \tag{3.37}$$

We have

$$\nabla g(x) = \begin{pmatrix} 2x_1 \\ 2x_2 - 2 \end{pmatrix} \text{ and } \nabla h(x) = \begin{pmatrix} -2x_1 \\ 1 \end{pmatrix}.$$

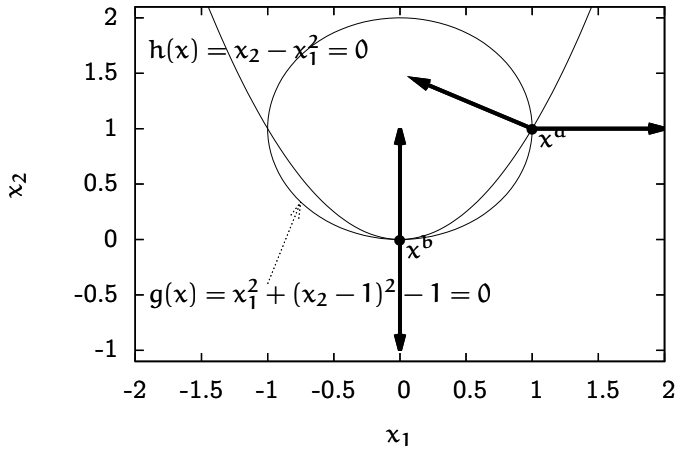


Figure 3.4: Linear independence of the constraints

Consider the point $x^a = (1,1)^T$, that is feasible, and for which the constraint (3.36) is active. We have

$$\nabla g(x^a) = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \text{ and } \nabla h(x^a) = \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

These two vectors are linearly independent, and the linear independence of the constraints is satisfied in x^a. Figure 3.4 represents the normalized vectors

$$\frac{\nabla g(x^a)}{\|\nabla g(x^a)\|} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } \frac{\nabla h(x^a)}{\|\nabla h(x^a)\|} = \begin{pmatrix} \frac{-2\sqrt{5}}{5} \\ \frac{\sqrt{5}}{5} \end{pmatrix}.$$

Consider the point $x^b = (0,0)^T$, that is also feasible, and for which the constraint (3.36) is active. We have

$$\nabla g(x^b) = \begin{pmatrix} 0 \\ -2 \end{pmatrix} \text{ and } \nabla h(x^b) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

These vectors are represented as normalized in Figure 3.4. They are linearly dependent because $\nabla g(x^b) = -2\nabla h(x^b)$, and the linear independence of the constraints is not satisfied in x^b.

3.3 Feasible directions

A major difficulty when we develop optimization algorithms is to move within the feasible set. The analysis of the feasible directions helps us in this task.

Definition 3.10 (Feasible direction)**.** Consider the general optimization problem (1.71)–(1.74), and the feasible point $x \in \mathbb{R}^n$. A direction d is said to be feasible in x if there exists $\eta > 0$ such that $x + \alpha d$ is feasible for any $0 < \alpha \leq \eta$.

In short, it is a direction that can be followed, at least a little bit, while staying within the feasible set. Some examples are provided in Figure 3.5, where the feasible set is the polygon represented by thin lines, feasible directions are represented with thick plain lines, and infeasible directions with thick dashed lines.

3.3.1 Convex constraints

When the set X of the constraints is convex, the identification of a feasible direction in $x \in X$ depends on the identification of a feasible point $y \in X$, other than x.

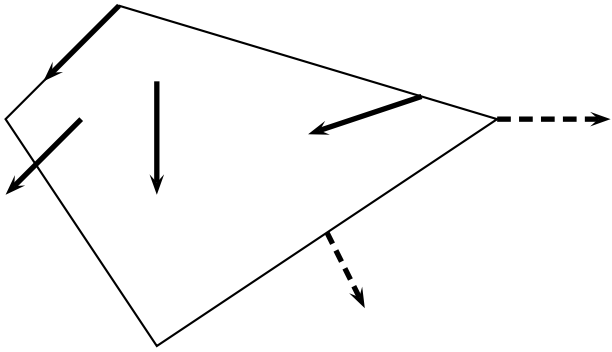


Figure 3.5: Feasible (plain) and infeasible (dashed) directions

Theorem 3.11 (Feasible direction in a convex set)**.** *Let X be a convex set, and consider $x, y \in X$, $y \neq x$. The direction $d = y - x$ is a feasible direction in x and $x + \alpha d = x + \alpha(y - x)$ is feasible for any $0 \leq \alpha \leq 1$.*

Proof. According to Definition B.2 of a convex set. □

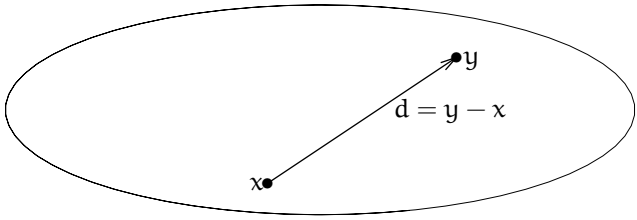


Figure 3.6: Feasible direction in a convex set

Corollary 3.12 (Feasible directions in an interior point)**.** *Let $X \subseteq \mathbb{R}^n$ be a subset of $\mathbb{R}^n$ and $x \in \mathbb{R}^n$ an interior point of X. Here, any direction $d \in \mathbb{R}^n$ is feasible in x.*

Proof. According to the definition of an interior point (Definition 1.15), there exists a (convex) neighborhood $V = \{z \text{ such that } \|z - x\| \leq \varepsilon\}$ such that $V \subseteq X$ and $\varepsilon > 0$. Consider an arbitrary direction d, and let $y = x + \varepsilon d/\|d\|$ be the point where the direction intersects the border of the neighborhood. Since $\|y - x\| = \varepsilon$, then $y \in V$. Since V is convex, Theorem 3.11 is invoked to demonstrate that d is feasible. □

This result is particularly important. The fact that all directions are feasible at an interior point gives freedom to algorithms in the selection of the direction. This is what motivates the method of interior points, as described in Chapter 18.

3.3.2 Constraints defined by equations-inequations

Here we consider the problem (1.71)–(1.73). Since we have at our disposal analytical expression of constraints, we characterize the set of feasible directions directly from these expressions. When the constraints are linear, the characterization is simple.

Theorem 3.13 (Feasible directions: linear case). *Consider the optimization problem (3.26)–(3.28)* $\min f(x)$ *subject to* $Ax = b$ *and* $x \geq 0$, *and let* x^+ *be a feasible point. A direction* d *is feasible in* x^+ *if and only if*

1. $Ad = 0$, *and*
2. $d_i \geq 0$ *when* $x_i^+ = 0$.

Proof. $\Longrightarrow$ Direct implication. Let d be a feasible direction in x^+. According to Definition 3.10, there exists $\eta > 0$ such that $x^+ + \alpha d$ is feasible for any $0 < \alpha \leq \eta$. It satisfies the constraint (3.27), i.e.,

$$b = A(x^+ + \alpha d) = Ax^+ + \alpha Ad = b + \alpha Ad.$$

We have that $Ad = 0$ because $\alpha > 0$. Consider i such that $x_i^+ = 0$. Since $x^+ + \alpha d$ is feasible, we have

$$x_i^+ + \alpha d_i = \alpha d_i \geq 0$$

and the second condition is satisfied.

$\Longleftarrow$ Inverse implication. Let d be a direction satisfying the two conditions. Consider the point $x^+ + \alpha d$. It satisfies the constraints (3.27) because $Ad = 0$. For the constraints (3.28), we consider three types of indices:

- Consider i such that $x_i^+ = 0$. In this case, $x_i^+ + \alpha d_i$ is non negative because $d_i \geq 0$ by hypothesis.
- Consider i such that $x_i^+ > 0$ and $d_i \geq 0$. The same conclusion.
- Consider i such that $x_i^+ > 0$ and $d_i < 0$. These are the only indices for which chances are that $x_i^+ + \alpha d_i$ is non feasible. In this case, we have to determine the step to take in direction d in order to stay feasible. Define

$$\eta = \min_{i | x_i^+ > 0 \text{ and } d_i < 0} -\frac{x_i}{d_i} > 0.$$

If we choose $\alpha \leq \eta$, we have

$$\alpha \leq \eta \leq -\frac{x_i^+}{d_i} \quad \forall i \text{ t.q. } x_i^+ > 0 \text{ and } d_i < 0,$$

and

$$x_i^+ + \alpha d_i \geq 0$$

because $d_i < 0$.

Then, $x^+ + \alpha d$ is feasible if $\alpha \leq \eta$. Since η is positive, d is a feasible direction according to Definition 3.10.

□

Corollary 3.14 (Combination of feasible directions: linear case)**.** *Consider the optimization problem (3.26)–(3.28)* $\min f(x)$ *subject to* $Ax = b$ *and* $x \geq 0$*. Let* x^+ *be a feasible point and* $d_1, \ldots, d_k$ *feasible directions in* x^+*. Then, the convex cone generated by these directions contains feasible directions. That is, any linear combination with non negative coefficients of these directions, i.e.,*

$$d = \sum_{j=1}^{k} \alpha_j d_j \quad \alpha_j \geq 0 \; \forall j, \tag{3.38}$$

is a feasible direction.

Proof. The two conditions of Theorem 3.13 are trivially satisfied for d. □

To switch to the non linear case, we must use the gradients of the constraints. Before this, we propose to interpret Theorem 3.13 in terms of gradients.

The first condition of the theorem concerns equality constraints. We have seen that the rows of the matrix A are the gradients of the constraints, i.e. $\nabla h_i(x) = a_i$, with

$$h_i(x) = a_i^T x - b_i = 0.$$

In the linear case, the first condition can be written as

$$\nabla h_i(x)^T d = 0 \quad i = 1, \ldots, m.$$

The inequality constraints can be written as

$$g_i(x) = -x_i \leq 0 \quad i = 1, \ldots, p,$$

and

$$\nabla g_i(x) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \text{ and } \nabla g_i(x)^T d = -d_i.$$

The second condition of the theorem can be expressed as follows: "*If the constraint* $g_i(x)$ *is active at* x^+*, then* $\nabla g_i(x^+)^T d \leq 0$." We should also note that if an inequality constraint is not active at x^+, it does not involve any condition on the direction for the latter to be feasible.

Unfortunately, the generalization of these results to the non linear case is not trivial. We develop it in two steps. We first see how to characterize feasible directions for an inequality constraint. We treat the equality constraint later.

We observe that the gradient of an inequality constraint at a point where it is active points toward the outside of the constraints, as shown by Example 3.15.

Example 3.15 (Constraint gradients)**.** Consider the subset of $\mathbb{R}^2$ defined by the constraint

$$\frac{1}{2}(x_1 - 1)^2 + \frac{1}{2}(x_2 - 1)^2 \leq \frac{1}{2},$$

and represented by Figure 3.7. Considering the formulation (1.73), we have

$$g(x) = \frac{1}{2}(x_1 - 1)^2 + \frac{1}{2}(x_2 - 1)^2 - \frac{1}{2},$$

and

$$\nabla g(x) = \begin{pmatrix} x_1 - 1 \\ x_2 - 1 \end{pmatrix}.$$

If we evaluate the gradient at different points along the border of the constraint, we obtain directions pointing towards the outside of the feasible domain, as shown in Figure 3.7.

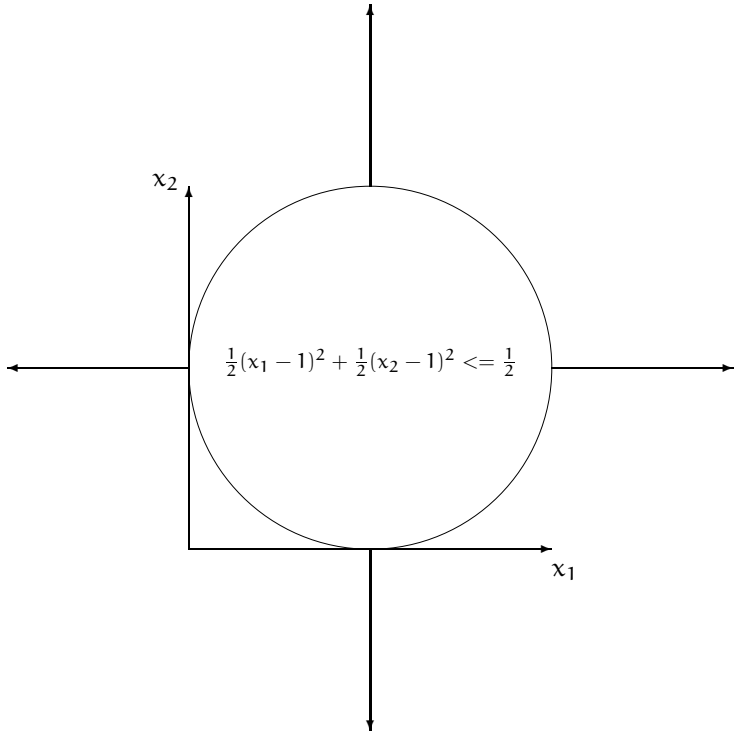


Figure 3.7: Constraint gradient

Intuitively, the gradient direction and the directions that form an acute angle with it cannot be considered as feasible directions.

Theorem 3.16 (Feasible directions: an inequality constraint). *Let* $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ *be a differentiable function, and consider* $x^+ \in \mathbb{R}^n$ *such that* $g(x^+) \leq 0$.

1. *If the constraint* $g(x) \leq 0$ *is inactive at* x^+, *all directions are feasible in* x.
2. *If the constraint is active at* x^+, *and* $\nabla g(x^+) \neq 0$, *a direction* d *is feasible in* x^+ *if*

$$\nabla g(x^+)^T d < 0. \tag{3.39}$$

Proof. 1. If the constraint is inactive at x^+, then x^+ is an interior point, and Corollary 3.12 applies.

2. Consider the case where the constraint is active at x^+, that is $g(x^+) = 0$. Let d be a direction satisfying (3.39). According to Definition 2.10, d is a descent direction for g in x^+. We can apply Theorem 2.11 to determine that there exists $\eta > 0$ such that

$$g(x^+ + \alpha d) < g(x^+) = 0, \quad \forall 0 < \alpha \leq \eta,$$

and conclude that d is feasible.

□

It is important to note that (3.39) is a sufficient condition in order to obtain a feasible direction when the constraint is active, but it is not a necessary condition. Indeed, in the linear case, we have a similar condition, with a non strict inequality. We still have to discuss the case where $\nabla g(x^+)^T d = 0$, which occurs when the gradient is zero, or when the direction d is perpendicular to the gradient. In this case, we invoke Theorem C.1 (Taylor's theorem) to obtain

$$g(x^+ + \alpha d) = g(x^+) + \alpha d^T \nabla g(x^+) + o(\alpha \|d\|) = o(\alpha).$$

In this case, nothing can guarantee that there exists α such that $x^+ + \alpha d$ is feasible. However, we can make the infeasibility as little as desired by choosing a sufficiently small α.

We now analyze the feasible directions for an equality constraint

$$h(x) = 0.$$

We can express this constraint in an equivalent manner as

$$\begin{aligned} h(x) &\leq 0 \\ -h(x) &\leq 0. \end{aligned}$$

If x^+ is feasible, these two inequality constraints are active. However, no direction d can simultaneously satisfy (3.39) for the two inequalities. This condition is unusable for equality constraints.

Since the equality constraints pose a problem, we address the problem in the other direction. Instead of positioning ourselves on a feasible point x^+ and wondering how to reach another one, we attempt to identify the ways to reach x^+ while remaining feasible. To do this, we introduce the concept of feasible sequences.

Definition 3.17 (Feasible sequences)**.** Consider the optimization problem (1.71)–(1.74), and a feasible point $x^+ \in \mathbb{R}^n$. A sequence $(x_k)_{k\in\mathbb{N}}$, with $x_k \in \mathbb{R}^n$ for any k is said to be a feasible sequence in x^+ if the following conditions are satisfied:

1. $\lim_{k\to\infty} x_k = x^+$,
2. there exists k_0 such that x_k is feasible for any $k \geq k_0$,
3. $x_k \neq x^+$ for all k.

The set of feasible sequences in x^+ is denoted by $\mathcal{S}(x^+)$.

Example 3.18 (Feasible sequence)**.** Consider the constraint in $\mathbb{R}^2$

$$h(x) = x_1^2 - x_2 = 0,$$

and the feasible point $x^+ = (0,0)^T$. The sequence defined by

$$x_k = \begin{pmatrix} \frac{1}{k} \\ \frac{1}{k^2} \end{pmatrix}$$

satisfies the three conditions of Definition 3.17 and belongs to $\mathcal{S}(x^+)$. It is illustrated in Figure 3.8.

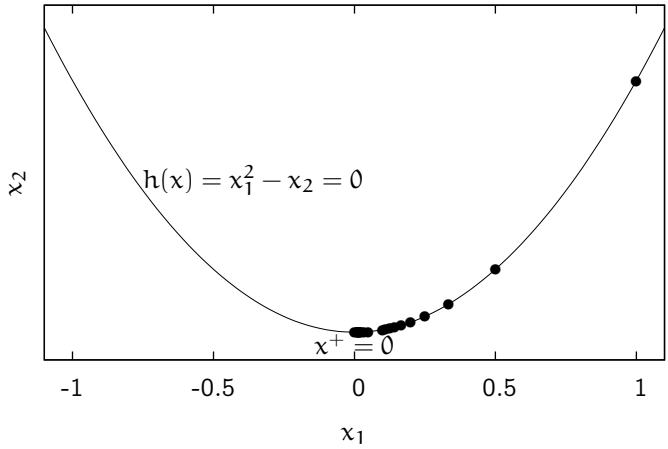


Figure 3.8: Example of a feasible sequence

Given that the sequence $(x_k)_k$ is feasible in x^+, we consider the directions connecting x_k to x^+ by normalizing them:

$$d_k = \frac{x_k - x^+}{\|x_k - x^+\|}. \tag{3.40}$$

We should keep in mind that these directions are generally not feasible directions. We are looking at what happens at the limit.

Example 3.19 (Feasible direction at the limit)**.** We consider once again Example 3.18. We have $x_k - x^+ = x_k$,

$$\|x_k - x^+\| = \frac{\sqrt{k^2+1}}{k^2}$$

and

$$d_k = \begin{pmatrix} \frac{k}{\sqrt{k^2+1}} \\ \frac{1}{\sqrt{k^2+1}} \end{pmatrix}.$$

At the limit, we obtain

$$d = \lim_{k\to\infty} d_k = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

These directions are illustrated in Figure 3.9. Note that

$$\nabla h(x^+) = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \text{ and } \nabla h(x^+)^T d = 0.$$

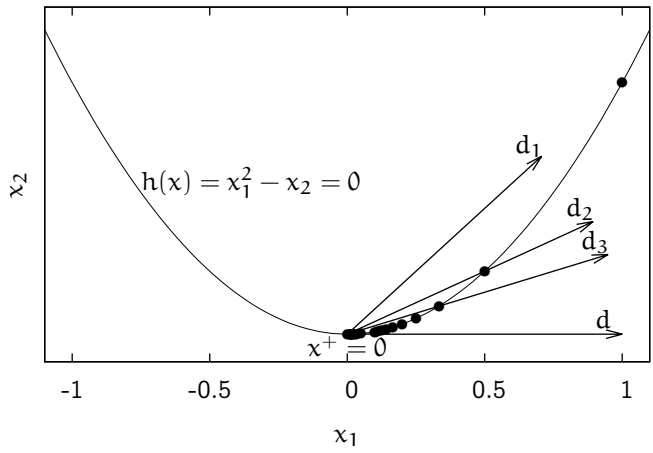


Figure 3.9: Example of a feasible direction at the limit

Unfortunately, it is not always possible to position oneself at the limit, as shown in Example 3.20 where the sequence is not convergent, and contains two adherent points. In this case, we should consider the limit of subsequences in order to identify the feasible directions at the limit.

Example 3.20 (Feasible direction at the limit). As for Example 3.18, we consider the constraint in $\mathbb{R}^2$:

$$h(x) = x_1^2 - x_2 = 0,$$

and the feasible point $x^+ = (0,0)^T$. The sequence defined by

$$x_k = \begin{pmatrix} \frac{(-1)^k}{k} \\ \frac{1}{k^2} \end{pmatrix}$$

satisfies the three conditions of Definition 3.17 and belongs to $\mathcal{S}(x^+)$. The calculation of the directions gives

$$d_k = \begin{pmatrix} \frac{(-1)^k k}{\sqrt{k^2+1}} \\ \frac{1}{\sqrt{k^2+1}}, \end{pmatrix}$$

and $\lim_{k\to\infty} d_k$ does not exist. However, if we consider the subsequences defined by the even and odd indices, respectively, we obtain

$$d' = \lim_{k\to\infty} d_{2k} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and

$$d'' = \lim_{k\to\infty} d_{2k+1} = \begin{pmatrix} -1 \\ 0 \end{pmatrix},$$

as illustrated in Figure 3.10. Note once again that $\nabla h(x^+)^T d' = \nabla h(x^+)^T d'' = 0$.

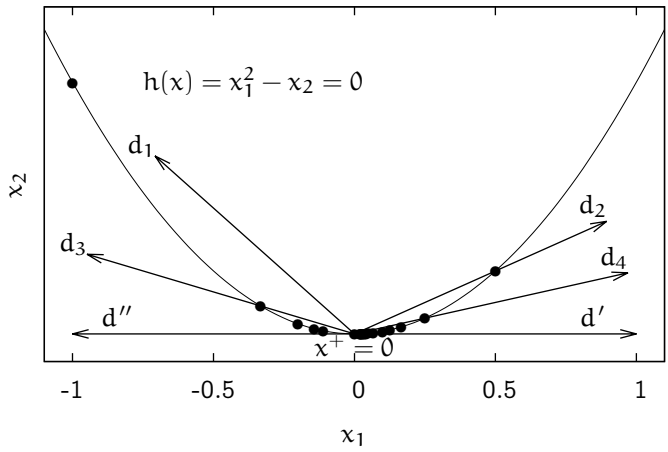


Figure 3.10: Example of a feasible sequence

We can now formally define the notion of a feasible sequence at the limit.

Definition 3.21 (Feasible direction at the limit)**.** Consider the optimization problem (1.71)–(1.74), and a feasible point $x^+ \in \mathbb{R}^n$. Let $(x_k)_{k\in\mathbb{N}}$ be a feasible sequence in x^+. The direction $d \neq 0$ is a feasible direction at the limit in x^+ for the sequence $(x_k)_{k\in\mathbb{N}}$ if there exists a subsequence $(x_{k_i})_{i\in\mathbb{N}}$ such that

$$\frac{d}{\|d\|} = \lim_{i\to\infty} \frac{x_{k_i} - x^+}{\|x_{k_i} - x^+\|}. \tag{3.41}$$

Note that any feasible direction d is also a feasible direction at the limit. Just take the feasible sequence $x_k = x^+ + \frac{1}{k}d$ in Definition 3.21.

Definition 3.22 (Tangent cone)**.** A feasible direction at the limit is also called a *tangent* direction. The set of all tangent directions in x^+ is called the *tangent cone* and denoted by $\mathcal{T}(x^+)$.

We can now make the connection between this concept and the constraint gradient. According to Theorem 3.16 and the associated comments, we consider all directions that form an obtuse angle with the active inequality constraint gradients and those that are orthogonal to the equality constraint gradients.

Definition 3.23 (Linearized cone)**.** Consider the optimization problem (1.71)–(1.74), and a feasible point $x^+ \in \mathbb{R}^n$. We call the linearized cone in x^+, denoted by $\mathcal{D}(x^+)$, the set of directions d such that

$$d^T \nabla g_i(x^+) \leq 0, \quad \forall i = 1, \ldots, p \text{ such that } g_i(x^+) = 0, \tag{3.42}$$

and

$$d^T \nabla h_i(x^+) = 0, \quad i = 1, \ldots, m, \tag{3.43}$$

and of all their multiples, i.e.,

$$\{\alpha d | \alpha > 0 \text{ and } d \text{ satisfies (3.42) and (3.43)}\}. \tag{3.44}$$

Theorem 3.24 (Feasible directions at the limit)**.** *Consider the optimization problem(1.71)–(1.74), and a feasible point $x^+ \in \mathbb{R}^n$. Any feasible direction at the limit in x^+ belongs to the linearized cone in x^+, that is*

$$\mathcal{T}(x^+) \subseteq \mathcal{D}(x^+). \tag{3.45}$$

Proof. Let d be a normalized feasible direction at the limit, and x_k a feasible sequence such that

$$d = \lim_{k\to\infty} \frac{x_k - x^+}{\|x_k - x^+\|}. \tag{3.46}$$

1. Consider an active inequality constraint, i.e.,

   $$g_i(x^+) = 0.$$

   For a sufficiently large k such that x_k is feasible, we can write

   $$g_i(x^+) + (x_k - x^+)^T \nabla g_i(x^+) + o(\|x_k - x^+\|) = g_i(x_k) \leq 0$$

   invoking Theorem C.1 (Taylor's theorem), and

   $$\frac{(x_k - x^+)^T \nabla g_i(x^+)}{\|x_k - x^+\|} + \frac{o(\|x_k - x^+\|)}{\|x_k - x^+\|} \leq 0.$$

   We now only have to position ourselves at the limit, and utilize (3.46) and Definition B.17 to obtain (3.42).
2. Consider an equality constraint,

   $$h_i(x^+) = 0.$$

   This constraint is equivalent to two inequality constraints

   $$\begin{aligned} h_i(x^+) &\leq 0 \\ -h_i(x^+) &\leq 0 \end{aligned}$$

   which are both active at x^+. According to the first point already demonstrated, we have $0 \leq d^T \nabla h_i(x^+)$ and $0 \leq -d^T \nabla h_i(x^+)$, and obtain (3.43).

□

Example 3.25. Illustration of Theorem 3.24 Returning to Example 3.20, we can observe in Figure 3.11 that the two feasible directions at the limit are orthogonal to the constraint gradient in x^+, i.e.,

$$\nabla h(x^+) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}.$$

Theorem 3.24 does not yet provide a characterization of feasible directions at the limit. Nevertheless, such a characterization is important, especially since the notion of linearized cone is easier to handle than the concept of feasible direction at the limit. Unfortunately, such a characterization does not exist in the general case. Therefore, it is useful to assume that any element of the linearized cone is a feasible direction at the limit. This hypothesis is called a constraint qualification.[1]

[1] Several constraint qualifications have been proposed in the literature. This one is sometimes called the Abadie Constraint Qualification, from the work by Abadie (1967).

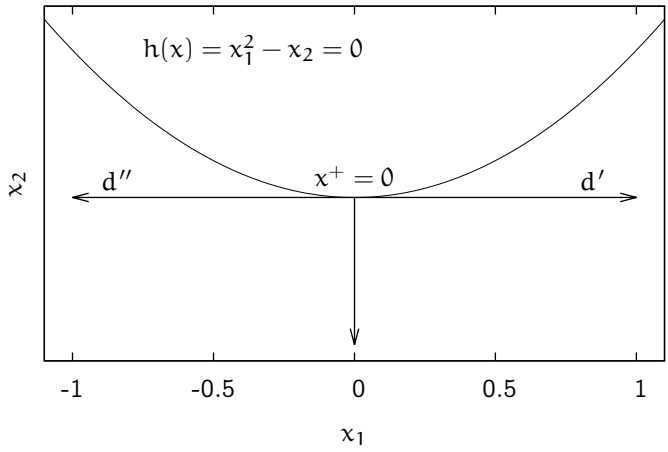


Figure 3.11: Gradient and feasible directions at the limit

Definition 3.26 (Constraint qualification)**.** Consider the optimization problem (1.71)–(1.74), and let x^+ be a feasible point. The constraint qualification condition is satisfied if any element of the linearized cone in x^+ is a feasible direction at the limit in x^+, that is if

$$\mathcal{T}(x^+) = \mathcal{D}(x^+). \tag{3.47}$$

This hypothesis, seemingly restrictive, is satisfied in a number of cases. In particular, when the constraints are defined solely by equations and inequations, each of the following is sufficient for a constraint qualification.

- If the constraints (1.71)–(1.73) are linear, the constraint qualification is satisfied at all feasible points.
- If the constraints are linearly independent in x^+ (Definition 3.8), the constraint qualification is satisfied at x^+.
- If there exists a vector $d \in \mathbb{R}^n$ such that

  1. $\nabla h_i(x^+)^T d = 0$, for any $i = 1, \ldots, m$,
  2. $\nabla g_i(x^+)^T d < 0$ for any $i = 1, \ldots, p$ such that $g_i(x^+) = 0$

  and such that the equality constraints are linearly independent in x^+, then the constraint qualification is satisfied at x^+ (Mangasarian and Fromovitz, 1967).
- If there is no equality constraint, the functions g_i are convex, and there exists a vector x^- such that

$$g_i(x^-) < 0 \text{ for any } i = 1, \ldots, p \text{ such that } g_i(x^+) = 0,$$

  the constraint qualification is satisfied at x^+ (Slater, 1950).

We develop the proof for the two first conditions.

Theorem 3.27 (Characterization of feasible directions at the limit – I). *Consider the optimization problem (1.71)–(1.73), and a feasible point $x^+ \in \mathbb{R}^n$ such that all active constraints in x^+ are linear. Every direction d such that $\|d\| = 1$ is feasible at the limit in x^+ if and only if it belongs to the linearized cone $\mathcal{D}(x^+)$, that is*

$$\mathcal{T}(x^+) = \mathcal{D}(x^+). \tag{3.48}$$

Proof. Theorem 3.24 shows that $\mathcal{T}(x^+) \subseteq \mathcal{D}(x^+)$. To demonstrate that $\mathcal{D}(x^+) \subseteq \mathcal{T}(x^+)$, consider a normalized direction[2] d that belongs to the linearized cone $\mathcal{D}(x^+)$. We need to create a feasible sequence $(x_k)_k$, such that (3.41) is satisfied.

For each inequality constraint i active at x^+, we have

$$g_i(x) = a_i^T x - b_i, \tag{3.49}$$

and for each equality constraint, we have

$$h_i(x) = \bar{a}_i^T x - \bar{b}_i. \tag{3.50}$$

Following the arguments developed in Theorem 3.5, there exists ε such that all constraints that are inactive at x^+ are also inactive in any point of the sphere of radius ε centered in x^+. Consider the sequence $(x_k)_k$ with

$$x_k = x^+ + \frac{\varepsilon}{k} d \quad k = 1, 2, \ldots \tag{3.51}$$

Each x_k is situated in the sphere mentioned above, and satisfies the inequality constraints that are inactive at x^+. For the inequality constraints that are active at x^+, we have

$$\begin{aligned} g_i(x_k) &= g_i(x_k) - g_i(x^+) && \text{because } g_i(x^+) = 0 \\ &= a_i^T x_k - b_i - a_i^T x^+ + b_i && \text{according to (3.49)} \\ &= a_i^T (x_k - x^+) \\ &= \frac{\varepsilon}{k} a_i^T d && \text{according to (3.51).} \end{aligned}$$

Since d is in the linearized cone at x^+, $\nabla g_i(x^+)^T d = a_i^T d \leq 0$, and x_k is feasible for any inequality constraint. For the equality constraint, we obtain in a similar manner

$$h_i(x_k) = \frac{\varepsilon}{k} \bar{a}_i^T d.$$

Since d is in the linearized cone at x^+, $\nabla h_i(x^+)^T d = \bar{a}_i^T d = 0$, and x_k is feasible for any equality constraint. The sequence $(x_k)_k$ is indeed a feasible sequence, in the sense of Definition 3.17.

[2] i.e., such that $\|d\| = 1$.

Finally, we now need only deduce from $\|d\| = 1$ that, for any k,

$$\frac{x_k - x^+}{\|x_k - x^+\|} = \frac{(\varepsilon/k)d}{\varepsilon/k} = d$$

to conclude that d is indeed a feasible direction at the limit. □

Theorem 3.28 (Characterization of feasible directions at the limit – II). *Consider the optimization problem(1.71)–(1.73), and a feasible point $x^+ \in \mathbb{R}^n$ for which the constraints are linearly independent. Any d such that $\|d\| = 1$ is feasible at the limit at x^+ if and only if it belongs to the linearized cone $\mathcal{D}(x^+)$, that is*

$$\mathcal{T}(x^+) = \mathcal{D}(x^+). \tag{3.52}$$

Proof. Theorem 3.24 shows that $\mathcal{T}(x^+) \subseteq \mathcal{D}(x^+)$. To demonstrate that $\mathcal{D}(x^+) \subseteq \mathcal{T}(x^+)$, consider a normalized direction d that belongs to the linearized cone $\mathcal{D}(x^+)$. We create a feasible sequence $(x_k)_k$, such that (3.41) is satisfied. We create it implicitly and not explicitly.

To simplify the notations, we first assume that all constraints are equality constraints. We consider the Jacobian matrix of constraints in x^+, $\nabla h(x^+)^T \in \mathbb{R}^{m\times n}$, for which the rows are constraint gradients in x^+ (see Definition 2.18). Since the constraints are linearly independent, the Jacobian matrix is of full rank. Consider a matrix $Z \in \mathbb{R}^{n\times(n-m)}$ for which the columns form a basis of the kernel of $\nabla h(x^+)^T$, i.e., such that $\nabla h(x^+)^T Z = 0$. We apply the theorem of implicit functions (Theorem C.6) to the parameterized function $F : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$F(\mu, x) = \begin{pmatrix} h(x) - \mu\nabla h(x^+)^T d \\ Z^T(x - x^+ - \mu d) \end{pmatrix}. \tag{3.53}$$

The assumptions of Theorem C.6 are satisfied for $\mu = 0$ and $x = x^+$. Indeed,

$$\nabla_x F(\mu, x) = \begin{pmatrix} \nabla h(x) & Z \end{pmatrix}$$

is non singular since the columns of Z are orthogonal to those of $\nabla h(x)$, and since the two submatrices are of full rank. Then, we have a function ϕ such that

$$x^+ = \phi(0) \tag{3.54}$$

and, for μ sufficiently close to zero,

$$F(\mu, \phi(\mu)) = \begin{pmatrix} h(\phi(\mu)) - \mu\nabla h(x^+)^T d \\ Z^T(\phi(\mu) - x^+ - \mu d) \end{pmatrix} = 0. \tag{3.55}$$

Since d is in the linearized cone, we deduce from the first part of (3.55)

$$h(\phi(\mu)) = \mu\nabla h(x^+)^T d = 0 \tag{3.56}$$

and $\phi(\mu)$ is feasible. We use ϕ to build a feasible sequence. To do so, we show that $\phi(\mu) \neq x^+$ when $\mu \neq 0$. Assume by contradiction that $\phi(\mu) = x^+$. In this case,

$$F(\mu, x^+) = \begin{pmatrix} h(x^+) - \mu \nabla h(x^+)^T d \\ Z^T(x^+ - x^+ - \mu d) \end{pmatrix} = \begin{pmatrix} -\mu \nabla h(x^+)^T d \\ -\mu Z^T d \end{pmatrix} = 0. \tag{3.57}$$

If $\mu \neq 0$, and since the matrices $\nabla h(x^+)^T$ and Z are of full rank, we deduce that $d = 0$, which is impossible since $\|d\| = 1$. Then, if $\mu \neq 0$, we necessarily have $\phi(\mu) \neq x^+$.

We are now able to generate a feasible sequence. To do so, we consider a sequence $(\mu_k)_k$ such that $\lim_{k\to\infty}(\mu_k)_k = 0$. Then, the sequence

$$x_k = \phi(\mu_k) \tag{3.58}$$

satisfies all conditions to be a feasible sequence (see Definition 3.17). We now need to demonstrate that d is a feasible direction at the limit.

For a sufficiently large k, such that μ_k is sufficiently close to zero, we use a Taylor series of h around x^+

$$\begin{aligned} h(x_k) &= h(x^+) + \nabla h(x^+)^T(x_k - x^+) + o(\|x_k - x^+\|) \\ &= \nabla h(x^+)^T(x_k - x^+) + o(\|x_k - x^+\|) \end{aligned}$$

in (3.55) to obtain

$$\begin{aligned} 0 &= F(\mu_k, x_k) \\ &= \begin{pmatrix} \nabla h(x^+)^T(x_k - x^+) + o(\|x_k - x^+\|) - \mu_k \nabla h(x^+)^T d \\ Z^T(x_k - x^+ - \mu_k d) \end{pmatrix} \\ &= \begin{pmatrix} \nabla h(x^+)^T(x_k - x^+ - \mu_k d) + o(\|x_k - x^+\|) \\ Z^T(x_k - x^+ - \mu_k d) \end{pmatrix} \\ &= \begin{pmatrix} \nabla h(x^+)^T \\ Z^T \end{pmatrix} (x_k - x^+ - \mu_k d) + o(\|x_k - x^+\|) \\ &= \begin{pmatrix} \nabla h(x^+)^T \\ Z^T \end{pmatrix} \left(\frac{x_k - x^+}{\|x_k - x^+\|} - \frac{\mu_k}{\|x_k - x^+\|} d \right) + \frac{o(\|x_k - x^+\|)}{\|x_k - x^+\|}. \end{aligned}$$

Then, since the matrices $\nabla h(x_k)^T$ and Z^T are of full rank, we have

$$\lim_{k\to\infty} \left(\frac{x_k - x^+}{\|x_k - x^+\|} - \frac{\mu_k}{\|x_k - x^+\|} d \right) = 0. \tag{3.59}$$

Define

$$\tilde{d} = \lim_{k\to\infty} \frac{x_k - x^+}{\|x_k - x^+\|} \text{ and } c = \lim_{k\to\infty} \frac{\mu_k}{\|x_k - x^+\|},$$

and (3.59) is written as

$$\tilde{d} = cd.$$

Since $\|\tilde{d}\| = \|d\| = 1$, we have $c = 1$ and

$$\lim_{k\to\infty} \frac{x_k - x^+}{\|x_k - x^+\|} = d.$$

Then, d is indeed a feasible direction at the limit.

To be completely accurate, we must consider a case with inequality constraints. For those that are active at x^+, the reasoning is identical, with the exception of (3.56) which becomes

$$g(\phi(\mu)) = \mu \nabla g(x^+)^T d \leq 0$$

from which we deduce the feasibility of $\phi(\mu)$. Inactive constraints do not pose a problem, since there is a sphere around x^+ such that all elements satisfy these constraints. Since Definition 3.17 is asymptotic, we can always choose k sufficiently large such that x_k belongs to this sphere. □

Feasible directions at the limit are an extension of the concept of a feasible direction. It enables us to identify in which direction an infinitesimal displacement continues to be feasible. Unfortunately, the definition is too complex to be operational. The linearized cone, based on the constraint gradients, is directly accessible to the calculation. We usually assume that the constraint qualification is satisfied.

3.4 Elimination of constraints

Optimization problems without constraint are simpler than those with constraints. We now analyze techniques to eliminate constraints.

We start with the optimization problem (3.26)–(3.28) $\min f(x)$ subject to $Ax = b$ and $x \geq 0$, where the constraints are linear. We assume that we have a system of constraints of full rank, obtained after eliminating any redundant constraint (see Theorem 3.6). It is then possible to simplify the problem by eliminating certain variables, as shown in Example 3.29.

Example 3.29 (Elimination of variables)**.** Consider the following optimization problem:

$$\min f(x_1, x_2, x_3, x_4) = x_1^2 + \sin(x_3 - x_2) + x_4 + 1 \tag{3.60}$$

subject to

$$\begin{array}{rcl} x_1 + x_2 + x_3 & = & 1 \\ x_1 - x_2 + x_4 & = & 1. \end{array} \tag{3.61}$$

We can rewrite the constraints in the following manner:

$$\begin{array}{rcl} x_3 & = & 1 - x_1 - x_2 \\ x_4 & = & 1 - x_1 + x_2. \end{array} \tag{3.62}$$

Thus, the optimization problem can be rewritten so as to depend only on two variables x_1 and x_2 :

$$\min f(x_1, x_2) = x_1^2 + \sin(-x_1 - 2x_2 + 1) - x_1 + x_2 + 2 \tag{3.63}$$

without constraint.

To generalize Example 3.29, we consider the constraints

$$Ax = b \tag{3.64}$$

with $A \in \mathbb{R}^{m\times n}$, $m \leq n$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ and $\operatorname{rank}(A) = m$. We choose m columns of A that are linearly independent corresponding to the variables that we wish to eliminate. Apply a permutation $P \in \mathbb{R}^{n\times n}$ of the columns of A in such a way that the m selected columns are the leftmost columns:

$$AP = (B\ N) \tag{3.65}$$

where $B \in \mathbb{R}^{m\times m}$ contains the m first columns of AP, and $N \in \mathbb{R}^{m\times(n-m)}$ contains the $n-m$ last ones. Recalling that $PP^T = I$, we write (3.64) in the following manner

$$Ax = AP(P^T x) = Bx_B + Nx_N = b \tag{3.66}$$

where $x_B \in \mathbb{R}^m$ contains the m first components of $P^T x$, and $x_N \in \mathbb{R}^{n-m}$ contains the $n-m$ last ones. Since the m first columns of AP are linearly independent, the matrix B is invertible. We can write

$$x_B = B^{-1}(b - Nx_N). \tag{3.67}$$

By adopting this convention, we consider the optimization problem with linear equality constraints

$$\min_{x_B, x_N} f\left(P\begin{pmatrix} x_B \\ x_N \end{pmatrix}\right) \tag{3.68}$$

subject to

$$Bx_B + Nx_N = b. \tag{3.69}$$

It is equivalent to the unconstrained problem

$$\min_{x_N} f\left(P\begin{pmatrix} B^{-1}(b - Nx_N) \\ x_N \end{pmatrix}\right). \tag{3.70}$$

The variables x_B are called basic variables, and the variables x_N non basic variables.

Example 3.30 (Elimination of variables – II)**.** In Example 3.29, we have

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

The variables to eliminate are x_3 and x_4. They correspond to the last two columns of the constraint matrix, and we choose the permutation matrix to make them the first two, that is

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

to obtain

$$AP = (B|N) = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & -1 \end{array}\right) \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad N = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

and

$$\begin{aligned} x_B = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} &= B^{-1}(b - Nx_N) \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) \\ &= \begin{pmatrix} 1 - x_1 - x_2 \\ 1 - x_1 + x_2 \end{pmatrix} \end{aligned}$$

which is exactly (3.62).

It is relatively easy to remove linear equality constraints. However, note that the calculation of the matrix B^{-1} can be tedious, especially when m is large, and it is sometimes preferable to explicitly maintain the constraints for the problem.

The elimination of non linear constraints can be problematic. An interesting example, proposed as an exercise by Fletcher (1983) and again by Nocedal and Wright (1999), illustrates this difficulty.

Example 3.31 (Elimination of non linear constraints)**.** Consider the problem

$$\min_x f(x_1, x_2) = x_1^2 + x_2^2$$

subject to

$$(x_1 - 1)^3 = x_2^2$$

shown in Figure 3.12.

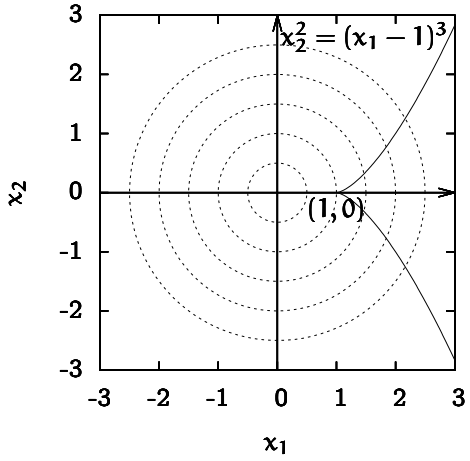


Figure 3.12: The problem in Example 3.31

The solution to this problem is $(1, 0)$. If we eliminate x_2, we obtain an optimization problem without constraint

$$\min_{x_1} \tilde{f}(x_1) = x_1^2 + (x_1 - 1)^3.$$

However, this new problem has no solution since $\tilde{f}$ is unbounded, i.e.,

$$\lim_{x_1 \to -\infty} \tilde{f}(x_1) = -\infty,$$

as shown in Figure 3.13. The problem is that the substitution can only be performed if $x_1 \geq 1$, since x_2^2 must necessarily be non negative. This implicit constraint in the original problem should be explicitly incorporated in the problem after elimination. It plays a crucial role since it is active at the solution.

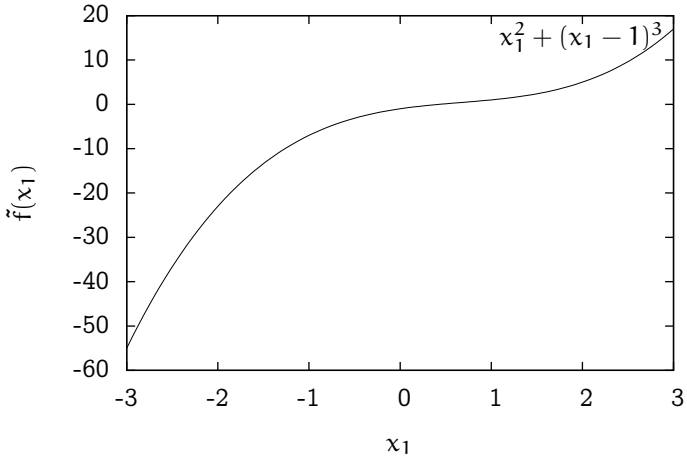


Figure 3.13: The problem without constraint in Example 3.31

One must thus be cautious when eliminating non linear constraints.

3.5 Linear constraints

When the constraints are linear, a more detailed analysis can be performed. We first give a geometric description of the constraints. Then, geometric concepts have their algebraic counterparts.

3.5.1 Polyhedron

We analyze the linear constraints (3.27)–(3.28) from a geometrical point of view. The central concept in this context is the *polyhedron*.

Definition 3.32 (Polyhedron)**.** A polyhedron is a set of points of $\mathbb{R}^n$ delimited by hyperplanes, i.e.,

$$\{x \in \mathbb{R}^n | Ax \leq b\}, \tag{3.71}$$

with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

By employing the techniques discussed in Section 1.2, it is always possible to transform an optimization problem with general linear constraints into a problem with the constraints $Ax \leq b$. Thus, the set of feasible points in an optimization problem with linear constraints is a polyhedron. To make the most of the technique of elimination of variables mentioned above, it is helpful to use the representation of a polyhedron called representation in *standard form*.

Definition 3.33 (Polyhedron represented in standard form)**.** A polyhedron represented in standard form is a polyhedron defined in the following manner

$$\{x \in \mathbb{R}^n | Ax = b,\ x \geq 0\}, \tag{3.72}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Note that according to Theorem 3.6, the matrix A is assumed to be of full rank without loss of generality.

The identification of vertices or extreme points of a polyhedron is possible thanks to the technique of elimination of variables described above. We begin by formally defining a vertex.

Definition 3.34 (Vertex)**.** Let $\mathcal{P}$ be a polyhedron. A vector $x \in \mathcal{P}$ is a vertex of $\mathcal{P}$ if it is impossible to find two vectors y and z in $\mathcal{P}$, different from x such that x is a convex combination (Definition B.3) of y and z, i.e., such that there exists a real number $0 < \lambda < 1$ such that

$$x = \lambda y + (1 - \lambda) z. \tag{3.73}$$

The Definition 3.34 is illustrated by Figure 3.14, where x is a vertex. If we choose $y \in \mathcal{P}$, it is impossible to find a z in $\mathcal{P}$ such that x is a convex combination of y and z. On the other hand, $\tilde{x}$ is not a vertex, and represents a convex combination of $\tilde{y}$ and $\tilde{z}$.

We can identify the vertices of a polyhedron represented in standard form by using the following procedure:

1. Choose m variables to eliminate.
2. Identify the matrix B that contains the corresponding columns of A.
3. Take $x_N = 0$.
4. In this case, (3.67) is expressed as $x_B = B^{-1}b$. If $x_B \geq 0$, then $x = (x_B^T\ x_N^T)^T$ is a vertex of the polyhedron.

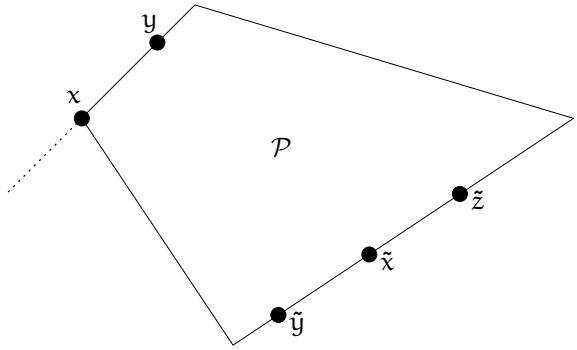


Figure 3.14: Illustration of Definition 3.34

We formalize this result with the following theorem.

Theorem 3.35 (Identification of vertices). *Let* $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ *be a polyhedron represented in standard form, with* $A \in \mathbb{R}^{m \times n}$ *of full rank and* $b \in \mathbb{R}^m$ *and* $n \geq m$. *Consider* m *linearly independent columns of* A, *and call* B *the matrix containing these* m *columns, and* N *the matrix containing the remaining* $n - m$ *columns, such that*

$$AP = (B|N) \tag{3.74}$$

where P *is the appropriate permutation matrix. Consider the vector*

$$x = P \begin{pmatrix} B^{-1}b \\ 0_{\mathbb{R}^{n-m}} \end{pmatrix}. \tag{3.75}$$

If $B^{-1}b \geq 0$, *then* x *is a vertex of* $\mathcal{P}$.

Proof. Without loss of generality, and to simplify the notations in the proof, we assume that the m columns chosen are the m first ones, such that the permutation matrix $P = I$. We assume by contradiction that there exists $y, z \in \mathcal{P}$, $y \neq x$, $z \neq x$ and $0 \leq \lambda \leq 1$ such that

$$x = \lambda y + (1 - \lambda)z. \tag{3.76}$$

After decomposition, we obtain

$$x_B = \lambda y_B + (1 - \lambda)z_B, \tag{3.77}$$

and

$$x_N = \lambda y_N + (1 - \lambda)z_N. \tag{3.78}$$

Since y and z are in $\mathcal{P}$, we have $y_N \geq 0$ and $z_N \geq 0$. Since $0 \leq \lambda \leq 1$, the only way for $x_N = 0$ is that $y_N = z_N = 0$. Then,

$$y_B = B^{-1}(b - Ny_N) = B^{-1}b = x_B, \tag{3.79}$$

and

$$z_B = B^{-1}(b - Nz_N) = B^{-1}b = x_B. \tag{3.80}$$

We obtain $x = y = z$, which contradicts the fact that y and z are different from x, and proves the result. □

It then appears that the vertices can be characterized by the set of active constraints.

Theorem 3.36 (Vertices and active constraints). *Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$. Let $x^* \in \mathcal{P}$, and*

$$\mathcal{A}(x^*) = \{i | x_i^* = 0\}$$

be the set of indices of the active constraints. x^* is a vertex of $\mathcal{P}$ if and only if the linear manifold

$$\mathcal{L}(x^*) = \{x \in \mathbb{R}^n | Ax = b \text{ and } x_i = 0 \; \forall i \in \mathcal{A}(x^*)\} \tag{3.81}$$

is zero-dimensional, i.e., $\mathcal{L}(x^*) = \{x^*\}$.

Proof. $\Longrightarrow$ Direct implication. Let x^* be a vertex. Assume by contradiction that $\mathcal{L}(x^*)$ is not zero-dimensional. There is then a straight line in $\mathcal{L}(x^*)$ characterized by the equation

$$x^* + \lambda d, \quad \lambda \in \mathbb{R}$$

with $d \in \mathbb{R}^n, d \neq 0$ and $Ad = 0$ such that $d_i = 0$ for any $i \in \mathcal{A}(x_0)$ (see Theorem 3.13). For every i such that $d_i \neq 0$, we define

$$\alpha_i = -\frac{x_i^*}{d_i}.$$

According to Definition (3.81) of linear manifold, $\alpha_i \neq 0$, for any i. Indeed, if $d_i \neq 0$, then $i \notin \mathcal{A}(x^*)$, and $x_i^* > 0$. We are now able to find two points of the polyhedron such that x^* is a convex combination of these points, contradicting the fact that it is a vertex.
Consider $\alpha_1 = \min_i\{\alpha_i | \alpha_i > 0\}$. If no α_i is positive, we take $\alpha_1 = 1$. Similarly, $\alpha_2 = \max_i\{\alpha_i | \alpha_i < 0\}$. If no α_i is negative, we take $\alpha_2 = -1$. Then, the points $y = x^* + \alpha_1 d$ and $z = x^* + \alpha_2 d$ belong by construction to the polyhedron $\mathcal{P}$. Moreover $x^* = \lambda y + (1 - \lambda)z$, with

$$\lambda = \frac{-\alpha_2}{\alpha_1 - \alpha_2}.$$

Since $\alpha_1 > 0$ and $\alpha_2 < 0$, we have $0 < \lambda < 1$, and x^* is a convex combination of y and z.

$\Longleftarrow$ Inverse implication. Consider $x^* \in \mathcal{P}$ such that $\mathcal{L}(x^*) = \{x^*\}$. We assume by contradiction that x^* is not a vertex of the polyhedron $\mathcal{P}$. There then exists y

and z in $\mathcal{P}$ such that $x^* = \frac{1}{2}(y+z)$, by arbitrarily taking $\lambda = \frac{1}{2}$ in Definition 3.34. For all indices i such that $x_i^* = 0$, the corresponding indices of y and z are also necessarily zero, because $y \geq 0$ and $z \geq 0$. Then, y and z belong to $\mathcal{L}(x^*)$, which contradicts the fact that x^* is the only element.

□

The characterization of vertices by active constraints is particularly useful when developing algorithms. A more explicit representation than linear manifold is desirable. This is the concept of a feasible basic solution. Before introducing this notion, we demonstrate that a non empty polyhedron represented in standard form always contains at least one vertex.

Theorem 3.37 (Existence of a vertex). *Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$. If $\mathcal{P}$ is non empty, it has at least one vertex.*

Proof. We construct a finite number of points belonging to linear varieties (defined by (3.81)) of decreasing dimension. The last one is the vertex of $\mathcal{P}$, then proving its existence.

Since $\mathcal{P}$ is non empty, there exists $x_0 \in \mathcal{P}$. If $\dim \mathcal{L}(x_0) = 0$, x_0 is a vertex according to Theorem 3.36. Otherwise, there exists a straight line contained in $\mathcal{L}(x_0)$ characterized by

$$x_0 + \lambda d, \quad \lambda \in \mathbb{R}$$

with $d \in \mathbb{R}^n, d \neq 0$ and $Ad = 0$ such that $d_i = 0$ for any $i \in \mathcal{A}(x_0)$. For each i such that $d_i \neq 0$, we define

$$\alpha_i = -\frac{(x_0)_i}{d_i}.$$

According to Definition (3.81) of linear manifold, $\alpha_i \neq 0$, for all i. Indeed, if $d_i \neq 0$, then $i \notin \mathcal{A}(x_0)$, and $(x_0)_i > 0$. Without loss of generality, we can assume that there exists at least one $\alpha_i > 0$ (if this is not the case, they are all non positive, and we can utilize the same approach using the straight line defined by $-d$). We define

$$\alpha^* = \min_{i | d_i > 0} \alpha_i$$

and j an index for which the minimum is reached, i.e., $\alpha^* = \alpha_j$. The point $x_1 = x_0 + \alpha^* d$ belongs to the polyhedron by construction. Moreover, $(x_1)_j = 0$ and $(x_0)_j > 0$. Then, the dimension of $\mathcal{L}(x_1)$ is strictly inferior to that of $\mathcal{L}(x_0)$. We now need only repeat the procedure to obtain, after a certain number of iterations k at most equal to the dimension of $\mathcal{L}(x_0)$, a point x_k such that $\dim \mathcal{L}(x_k) = 0$. According to Theorem 3.36, x_k is a vertex of $\mathcal{P}$. This proof is illustrated in Figure 3.15, where the linear manifold $\{x | Ax = b\}$ is shown. In this example, $\mathcal{L}(x_0)$ is the represented plane, $\mathcal{L}(x_1)$ is the straight line corresponding to the second coordinate axis, and $\mathcal{L}(x_2) = \{x_2\}$. □

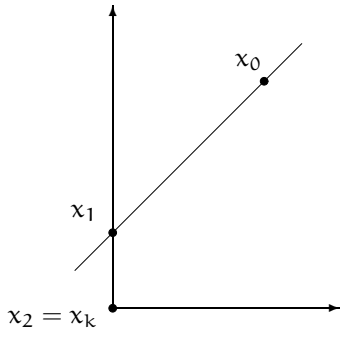


Figure 3.15: Illustration of the proof of Theorem 3.37

3.5.2 Basic solutions

The notion of a vertex is a purely geometric concept. By invoking Theorem 3.35, it is possible to characterize it algebraically. In this case, we speak of a *feasible basic solution*.

Definition 3.38 (Basic solution)**.** Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$. A vector $x \in \mathbb{R}^n$ such that $Ax = b$ is along with a set of indices $j_1, \ldots, j_m$ said to be a basic solution of $\mathcal{P}$ if

1. the matrix $B = (A_{j_1} \cdots A_{j_m})$ composed of columns $j_1, \ldots, j_m$ of the matrix A is non singular and
2. $x_i = 0$ if $i \neq j_1, \ldots, j_m$.

If, moreover, $x_B = B^{-1}b \geq 0$, the vector x is called a feasible basic solution.

It is common to say that the variables $j_1, \ldots, j_m$ in Definition 3.38 are *basic* variables, and that the others are *non basic* variables. Example 3.39 identifies the basic solutions of a polygon, written in the form of a polyhedron represented in standard form.

Example 3.39 (Basic solutions)**.** Consider a polyhedron represented in standard form

$$\mathcal{P} = \left\{ x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \mid Ax = b, x \geq 0 \right\} \tag{3.82}$$

with

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{3.83}$$

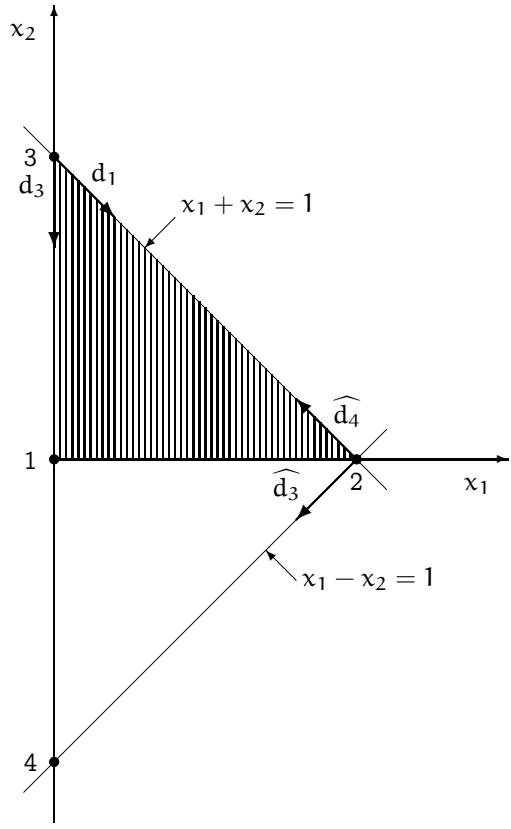


Figure 3.16: Feasible domain of Example 3.39

In order to view it in $\mathbb{R}^2$, we represent the polygon

$$\tilde{\mathcal{P}} = \left\{ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mid x_1 + x_2 \leq 1, x_1 - x_2 \leq 1, x_1 \geq 0, x_2 \geq 0 \right\} \tag{3.84}$$

in Figure 3.16. Note that if $(x_1, x_2, x_3, x_4)^T \in \mathcal{P}$, then $(x_1, x_2)^T \in \tilde{\mathcal{P}}$. Furthermore, if $(x_1, x_2)^T \in \tilde{\mathcal{P}}$, then $(x_1, x_2, 1 - x_1 - x_2, 1 - x_1 + x_2)^T \in \mathcal{P}$. The variables x_3 and x_4 are slack variables (Definition 1.4).

Each basic solution is obtained by selecting 2 variables out of 4 to be in the basis. There is a total of 6 possible selections of basic variables.

1. Basic solution with x_1 and x_2 in the basis ($j_1 = 1$, $j_2 = 2$).

$$B = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; B^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}; x_B = B^{-1} b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

This basic solution is feasible and corresponds to point 2 in Figure 3.16.

2. Basic solution with x_1 and x_3 in the basis ($j_1 = 1$, $j_2 = 3$).

$$B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}; B^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}; x_B = B^{-1}b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

   This basic solution is feasible and also corresponds to point 2 in Figure 3.16.

3. Basic solution with x_1 and x_4 in the basis ($j_1 = 1$, $j_2 = 4$).

$$B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}; B^{-1} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}; x_B = B^{-1}b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

   This basic solution is feasible and also corresponds to point 2 in Figure 3.16.

4. Basic solution with x_2 and x_3 in the basis ($j_1 = 2$, $j_2 = 3$).

$$B = \begin{pmatrix} 1 & 1 \\ -1 & 0 \end{pmatrix}; B^{-1} = \begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix}; x_B = B^{-1}b = \begin{pmatrix} -1 \\ 2 \end{pmatrix}; x = \begin{pmatrix} 0 \\ -1 \\ 2 \\ 0 \end{pmatrix}.$$

   This basic solution is not feasible because $B^{-1}b \not\geq 0$. It corresponds to point 4 in Figure 3.16.

5. Basic solution with x_2 and x_4 in the basis ($j_1 = 2$, $j_2 = 4$).

$$B = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}; B^{-1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}; x_B = B^{-1}b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}; x = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 2 \end{pmatrix}.$$

   This basic solution is feasible and corresponds to point 3 in Figure 3.16.

6. Basic solution with x_3 and x_4 in the basis ($j_1 = 3$, $j_2 = 4$).

$$B = B^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; x_B = B^{-1}b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; x = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

   This basic solution is feasible and corresponds to point 1 in Figure 3.16.

The notion of a basic solution (Definition 3.38) enables us to analyze the polyhedron in terms of active constraints of the optimization problem (Definition 3.4). Let x be a feasible basic solution such that $x_B = B^{-1}b > 0$. We say that it is non degenerate. In this case, there are exactly n active constraints in x: the m equality constraints and the $n - m$ non basic variables which are 0, and which make the

constraints of type $x_i \geq 0$ active. The constraints $x_i \geq 0$ corresponding to the basic variables are all inactive because $x_B > 0$. According to Theorem 3.5, the feasible basic solution is defined by n equations. If A is of full rank and $x_B > 0$, there is then a bijective relationship between the vertices of the polyhedron and the feasible basic solutions. This equivalence between an algebraic and a geometric concept is useful when developing algorithms.

Theorem 3.40 (Equivalence between vertices and feasible basic solutions)**.** *Let* $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ *be a polyhedron. The point* $x^* \in \mathcal{P}$ *is a vertex of* $\mathcal{P}$ *if and only if it is a feasible basic solution.*

Proof. $\Longrightarrow$ Let x^* be a vertex of $\mathcal{P}$. We assume that x^* is not a feasible basic solution. We can assume without loss of generality that the matrix A is of full rank (by removing redundant constraints). As x^* is not a feasible basic solution, there are strictly more than m non zero components in x^*. Consider m linearly independent columns of A, corresponding to non zero components of x^*, which form an invertible matrix B, and where the remaining $n - m$ columns form a matrix N. Here, x^* can be decomposed (see Section 3.4) into one basic component x_B and one non basic component x_N such that

$$x_B = B^{-1}(b - Nx_N).$$

Since x^* is not a feasible basic solution, there exists at least one component k of x_N that is not zero. We construct the direction d for which the basic component is

$$d_B = -B^{-1}A_k,$$

where A_k is the kth column of A, and the non basic component for all zero components, except the kth one which equals 1. Therefore,

$$Ad = Bd_B + Nd_n = -BB^{-1}A_k + \sum_{j \text{non basic}} A_j d_j = -A_k + A_k = 0.$$

Then, for all α,

$$A(x^* + \alpha d) = Ax^* + \alpha Ad = Ax^* = b.$$

Since $x^*_k > 0$, it is possible to choose $\alpha_1 > 0$ and $\alpha_2 > 0$ sufficiently small so that $x_1 = x^* + \alpha_1 d$ and $x_2 = x^* - \alpha_2 d$ are in $\mathcal{P}$. We take

$$\lambda = \frac{\alpha_2}{\alpha_1 + \alpha_2}.$$

We have $0 < \lambda < 1$ and $x^* = \lambda x_1 + (1 - \lambda)x_2$, which contradicts the fact that x^* is a vertex of the polyhedron.

$\Longleftarrow$ Theorem 3.35.

□

It is important to note that Theorem 3.40 does not guarantee a bijective relationship between the vertices of the polyhedron and the feasible basic solutions in all

cases. Indeed, when some of the components of $x_B = B^{-1}b$ are zero, there are more than n active constraints, and the feasible basic solution is defined by more than n equations in a space of n dimensions. We say in this case that we are dealing with a degenerate feasible basic solution.

Definition 3.41 (Degenerate feasible basic solution)**.** Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$. A basic solution $x \in \mathbb{R}^n$ is said to be degenerate if more than n constraints are active at x, i.e., if more than $n - m$ components of x are zero.

In the presence of degeneracy, a vertex may correspond to multiple feasible basic solutions. In Example 3.39, three constraints are active at vertex 2 (Figure 3.16), even though we only need two constraints to characterize it. Then, the first three basic solutions identified in the example all correspond to this vertex and are degenerate.

3.5.3 Basic directions

If x is a feasible basic solution, the feasible directions in x (there are infinitely many) can be characterized by a finite number of directions, called basic directions, and which correspond to the edges of the polyhedron of the constraints adjacent to the vertex corresponding to the feasible basic solution x.

To define these basic directions, we consider a feasible basic solution

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix} = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix} \tag{3.85}$$

where we assume, without loss of generality, that the indices of the basic variables are the m first ones, and that B consists of the m first columns of the matrix A. Consider a non basic variable, for instance the variable with index p, and define a direction that gives positive values to this non basic variable, all the while maintaining the other non basic variables at zero. Then

$$d = \begin{pmatrix} d_B \\ d_N \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_m \\ d_{m+1} \\ \vdots \\ d_{p-1} \\ d_p \\ d_{p+1} \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_m \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.86}$$

Since part d_N of the direction is defined, we now need only define d_B. For this, we invoke Theorem 3.13. To ensure that such a direction is feasible, the first condition

is that $Ad = 0$. Then, by denoting A_j the jth column of A, we obtain

$$Ad = Bd_B + Nd_N = Bd_B + \sum_{j=m+1}^{n} A_j d_j = Bd_B + A_p = 0, \tag{3.87}$$

that is,

$$d_B = -B^{-1}A_p. \tag{3.88}$$

Definition 3.42 (Basic direction)**.** Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$ and let $x \in \mathbb{R}^n$ be a feasible basic solution of $\mathcal{P}$. A direction d is called the pth basic direction in x if p is the index of a non basic variable, and

$$d_p = P \begin{pmatrix} d_{B_p} \\ d_{N_p} \end{pmatrix} \tag{3.89}$$

where P is the permutation matrix corresponding to the basic solution x, $d_{B_p} = -B^{-1}A_p$, and d_{N_p} is such that

$$P^T e_p = \begin{pmatrix} 0 \\ d_{N_p} \end{pmatrix}, \tag{3.90}$$

i.e., that all the elements of d_{N_p} are zero, except the one corresponding to the variable p, which is 1.

Note that these directions are not always feasible, as discussed later. But first, we illustrate the concept with Example 3.39.

Example 3.43 (Basic directions)**.** Consider the polygon in Example 3.39, and the feasible basic solution where x_2 and x_4 are in the basis. Then

$$x = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 2 \end{pmatrix} \text{ and } P = \left(\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right).$$

The basic direction corresponding to the non basic variable x_1 is

$$d_1 = P \begin{pmatrix} -B^{-1}A_1 \\ 1 \\ 0 \end{pmatrix} = P \begin{pmatrix} -\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ 1 \\ 0 \end{pmatrix} = P \begin{pmatrix} -1 \\ -2 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 0 \\ -2 \end{pmatrix},$$

and the basic direction corresponding to the non basic variable x_3 is

$$d_3 = P \begin{pmatrix} -B^{-1}A_3 \\ 0 \\ 1 \end{pmatrix} = P \begin{pmatrix} -\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 \\ 1 \end{pmatrix} = P \begin{pmatrix} -1 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 1 \\ -1 \end{pmatrix}.$$

These two directions are shown in Figure 3.16, from the feasible basic solution 3.

We now consider the feasible basic solution where x_1 and x_4 are in the basis (point 2 in Figure 3.16). Then,

$$x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ and } P = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right).$$

The basic direction corresponding to the non basic variable x_2 is

$$\tilde{d}_2 = P \begin{pmatrix} -B^{-1}A_2 \\ 1 \\ 0 \end{pmatrix} = P \begin{pmatrix} -\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \\ 1 \\ 0 \end{pmatrix} = P \begin{pmatrix} -1 \\ 2 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 0 \\ 2 \end{pmatrix},$$

and the basic direction corresponding to the non basic variable x_3 is

$$\tilde{d}_3 = P \begin{pmatrix} -B^{-1}A_3 \\ 0 \\ 1 \end{pmatrix} = P \begin{pmatrix} -\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 \\ 1 \end{pmatrix} = P \begin{pmatrix} -1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

These directions are not represented in Figure 3.16. Finally, consider the feasible basic solution where x_1 and x_2 are in the basis. Then,

$$x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ and } P = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right) = I.$$

The basic direction corresponding to the non basic variable x_3 is

$$\widehat{d}_3 = \begin{pmatrix} -B^{-1}A_3 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ 1 \\ 0 \end{pmatrix},$$

and the basic direction corresponding to the non basic variable x_4 is

$$\widehat{d}_4 = \begin{pmatrix} -B^{-1}A_4 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ 0 \\ 1 \end{pmatrix}.$$

These two directions are shown in Figure 3.16. Note that $\widehat{d}_4$ is a feasible direction, whereas $\widehat{d}_3$ is not.

Theorem 3.44 (Feasible basic directions). *Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$, and let $x \in \mathbb{R}^n$ be a feasible basic solution of $\mathcal{P}$. If x is non degenerate (in the sense of Definition 3.41), then any basic direction is a feasible direction in x.*

Proof. Let k be an arbitrary index of a non basic variable. According to Definition 3.42, we have $Ad_k = 0$. Moreover, since x is non degenerate, only its non basic components are zero. The corresponding components of d_k are non negative by definition. Theorem 3.13 can be applied to prove the feasibility of d_k. □

The following theorem enables us to consider only the feasible basic directions in order to characterize any feasible direction.

Theorem 3.45 (Combination of basic directions). *Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be a polyhedron represented in standard form, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ and $n \geq m$, and let $x \in \mathbb{R}^n$ be a feasible basic solution of $\mathcal{P}$. Any feasible direction d in x can be written as a linear combination of the basic directions, i.e.,*

$$d = \sum_{j \in \mathcal{N}} (d)_j d_j, \tag{3.91}$$

where $\mathcal{N}$ is the set of indices of the non basic variables, $d_j \in \mathbb{R}^n$ the jth basic direction, and $(d)_j \in \mathbb{R}$, the jth component of d.

Proof. Consider a feasible direction d, and assume without loss of generality that the basic variables are the m first ones. According to Theorem 3.13, we have

$$Ad = Bd_B + Nd_N = 0$$

and

$$d_B = -B^{-1}Nd_N = -\sum_{j=m+1}^{n} (d)_j B^{-1} A_j \tag{3.92}$$

where $(d)_j \in \mathbb{R}$ is the jth component of the vector d. By decomposing d_N in the canonical basis, we can also write

$$d_N = \sum_{j=m+1}^{n} (d)_j e_{j-m} \tag{3.93}$$

where $e_k \in \mathbb{R}^{n-m}$ is a vector for which all the components are zero, except the kth one, which is 1. According to Definition 3.42, (3.92) and (3.93) are written as

$$d = \begin{pmatrix} d_B \\ d_N \end{pmatrix} = \sum_{j=m+1}^{n} (d)_j \begin{pmatrix} -B^{-1}A_j \\ e_{j-m} \end{pmatrix} = \sum_{j=m+1}^{n} (d)_j d_j, \tag{3.94}$$

where d_j is the jth basic direction. We obtain (3.91). □

The proof of Theorems 3.27 and 3.28 is inspired by Nocedal and Wright (1999). That of Theorems 3.36 and 3.37 is inspired by de Werra et al. (2003).

3.6 Exercises

Exercise 3.1. Take the feasible set defined by the constraints

$$\begin{array}{rcrcrcrcr} x_1 & - & x_2 & + & 2x_3 & - & 2x_4 & + & 3x_5 & = & 3 \\ x_1 & & & + & x_3 & & & + & 2x_5 & = & 1 \\ & & x_2 & - & x_3 & + & 2x_4 & - & x_5 & = & -2. \end{array}$$

1. Identify a feasible point.
2. Verify whether the constraints are linearly independent.

Exercise 3.2. Take the feasible set defined by the constraints

$$\begin{array}{rcl} h(x_1, x_2) & = & \begin{pmatrix} (x_1-1)^2 + x_2^2 - 1 \\ (x_1-2)^2 + x_2^2 - 4 \end{pmatrix} = 0, \\ g(x_1, x_2) & = & -e^{\sin(x_1)+\cos(x_2)} - x_1^2 - x_2^2 \leq 0. \end{array}$$

1. Determine the set of feasible points.
2. For each one, determine the active constraints.
3. For each one, verify whether the condition of independence of the constraints (Definition 3.8) is satisfied.

Exercise 3.3. Take the feasible set defined by the constraints

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & & & \leq & 3 \\ x_1 & & & + & x_3 & \leq & 7 \\ x_1 & & & & & \geq & 0 \\ & & x_2 & & & \geq & 0 \\ & & & & x_3 & \geq & 0. \end{array}$$

1. Take the point $x = (3\ 0\ 4)^T$. Characterize the linearized cone in x.
2. Express the constraints in standard form.
3. Identify the basic solutions, and among them, those that are feasible.
4. For each point corresponding to a feasible basic solution,
   (a) characterize the linearized cone,
   (b) identify the basic directions,
   (c) verify whether the basic directions are in the linearized cone.

Exercise 3.4. Take the feasible set defined by the constraints

$$\begin{array}{rcrcl} -x_1 & + & x_2 & \leq & 1 \\ x_1 & + & 2x_2 & \leq & 4. \end{array}$$

1. Provide a graphic representation of this feasible set.

2. Express the constraints in standard form.
3. List the basic solutions, and represent them on the graph.
4. For each feasible basic solution, list the basic directions and represent them on the graph.

Exercise 3.5. Consider the feasible set defined by the following constraints:

$$\begin{array}{rcrcr} x_1 & - & x_2 & \geq & -2 \\ 2x_1 & + & x_2 & \leq & 8 \\ x_1 & + & x_2 & \leq & 5 \\ x_1 & + & 2x_2 & \leq & 10 \\ & & x_1 & \geq & 0 \\ & & x_2 & \geq & 0 \end{array}$$

1. Provide a graphic representation of this feasible set.
2. Enumerate the vertices of $\mathcal{D}$.
3. List the basic solutions and represent them on the graph.
4. For each feasible basic solution, list the basic directions and represent them on the graph.
5. Reformulate the same set of constraints using a minimum number of constraints (use the graphical representation to identify them).

Exercise 3.6. Take the feasible set defined by the constraints

$$\begin{array}{rcl} (x_1+1)^2 + x_2^2 & \leq & 1 \\ (x_1-1)^2 + x_2^2 & \leq & 1. \end{array}$$

For each x and d below,

1. verify that x is feasible,
2. specify whether the direction d is feasible in x (justify!),
3. specify whether the direction d is feasible at the limit in x (justify!).

x		d		x		d	
0	0	-1	-1	1	0	-1	0
0	0	1	1	0	1	-1	1
1	0	0	1	0	1	-1	-1
1	0	0	-1	0	1	1	1
1	0	1	0	0	1	1	-1

Exercise 3.7. Take the optimization problem $\min_{x\in\mathbb{R}^2} x_1+x_2$ subject to $x_1^2+x_2^2=2$, and the point $\bar{x} = (-\sqrt{2},0)^T$. Identify the feasible directions at the limit in $\bar{x}$ by employing the following sequences:

$$\begin{pmatrix} -\sqrt{2-\frac{1}{k^2}} \\ \frac{1}{k} \end{pmatrix} \text{ and } \begin{pmatrix} -\sqrt{2-\frac{1}{k^2}} \\ -\frac{1}{k} \end{pmatrix}.$$

First verify that they are indeed feasible sequences.

Chapter 4

Introduction to duality

There are those who are subject to constraints and others who impose them. We now take the point of view of the second category in order to analyze an optimization problem from a different angle.

Contents

In Section 3.4, we attempted to explicitly eliminate the constraints of the optimization problem by expressing one set of variables as a function of the others. In this chapter, we use another technique to remove constraints. This technique, called constraint relaxation, plays an important role both theoretically and algorithmically. In addition, it enables us to introduce the concept of duality.

4.1 Constraint relaxation

Here we introduce the concept of constraint relaxation with a simple example.

Example 4.1 (The mountaineer and the billionaire)**.** A billionaire decides to offer a mountaineer a prize linked to the altitude he manages to climb, at a rate of €1 per meter. However, for reasons only he knows, the billionaire requires the mountaineer to stay in the Alps. The mountaineer immediately takes on the optimal strategy. He climbs Mont Blanc and pockets €4807 (Figure 4.1(a)).

However, the mountaineer loves freedom and does not easily accept constraints. After some negotiation, the billionaire allows the climber to go elsewhere than in the Alps if he so desires, but must then pay a fine. A problem arises for the billionaire. If the fine is too low, the climber will want to go to the Himalayas and climb Mount

(a) Mont Blanc (4,807 m) (b) Mount Everest (8,848 m)

Figure 4.1: Solutions for Example 4.1

Everest, culminating at 8,848 meters (Figure 4.1(b)). The billionaire thus decides to set the fine at €4,041. In this case, climbing Everest would give the mountaineer $8{,}848 - 4{,}041 = €4{,}807$, which is exactly the same amount as if he decided to climb Mont Blanc. Therefore, the mountaineer has no interest in violating the constraint, and the final solution to the problem is the same as with the constraint.

We can model the problem of the climber by calling his position (longitude/latitude) x and the corresponding altitude $f(x)$. The first problem is a constrained optimization problem:

$$\max_x f(x)$$

subject to

$$x \in \text{Alps}\,.$$

The fine imposed by the billionaire is denoted by $a(x)$, and depends on the position x. In particular, $a(x) = 0$ if $x \in$ Alps. The optimization problem is now without constraints and can be expressed as

$$\max_x f(x) - a(x)\,.$$

Although somewhat imaginative, Example 4.1 shows us that an optimization problem can be seen from two points of view. From the viewpoint of the one solving the problem (the mountaineer) and of the one who defines the rules of the game (the billionaire). If we would like a constraint relaxation in order to remove them, we must put ourselves in the place of the billionaire, so that the new rules are consistent with the old ones. We now apply the same approach to another simple optimization problem.

János von Neumann was born on December 28, 1903, in Budapest. Originally, his name did not have the noble prefix "von". In 1913, his father bought a title of nobility. János took the name John von Neumann when he became an American in 1937. Although holder of a chemistry degree from ETH (Swiss Federal Institute of Technology) in Zürich, he quickly turned to mathematics. The results by Gödel on incompleteness gave him the incentive to abandon his work on axiomatization of set theory. Within the framework of quantum mechanics, he unified the theories of Schrödinger and Heisenberg. He is considered the father of game theory. It is in this context that he developed the principle of duality. One of his most famous quotes is "If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is!" Von Neumann died on February 8, 1957, in Washington, DC.

Figure 4.2: John von Neumann

Example 4.2 (Constraint relaxation)**.** Consider the optimization problem

$$\min_{x \in \mathbb{R}^2} 2x_1 + x_2 \tag{4.1}$$

subject to

$$\begin{aligned} 1 - x_1 - x_2 &= 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned} \tag{4.2}$$

for which the solution is $x^* = (0,1)^T$ with an optimal value 1. We now relax the constraint $1 - x_1 - x_2 = 0$ and introduce a fine that is proportional to the violation of the constraint, with a proportionality factor λ. This way, the fine is zero when the constraint is satisfied. We obtain the following problem:

$$\min_{x \in \mathbb{R}^2} 2x_1 + x_2 + \lambda(1 - x_1 - x_2) \tag{4.3}$$

subject to

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0\,. \end{aligned} \tag{4.4}$$

We examine different values of λ.

- If $\lambda = 0$, (4.3) becomes $2x_1 + x_2$ and the solution to the problem is $x^* = (0,0)^T$ with an optimal value of 0 (Figure 4.3). This solution violates the constraint of the original problem, and the optimal value is lower. It is a typical case where the penalty value is ill-suited, and where it becomes interesting to violate the constraint.

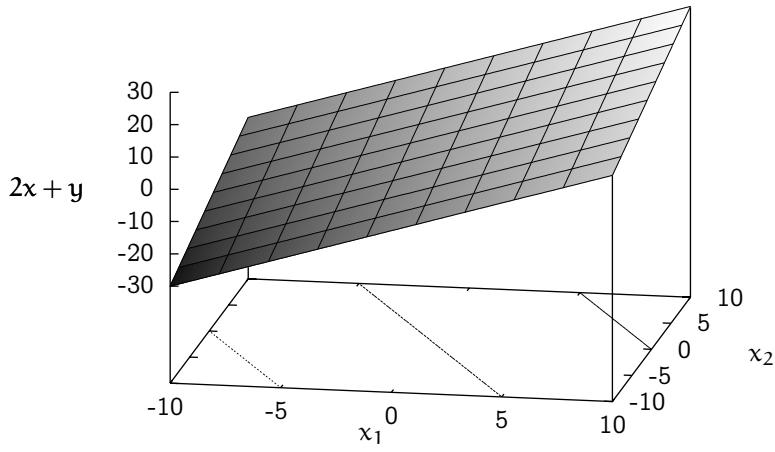


Figure 4.3: Objective function of Example 4.2: $\lambda = 0$

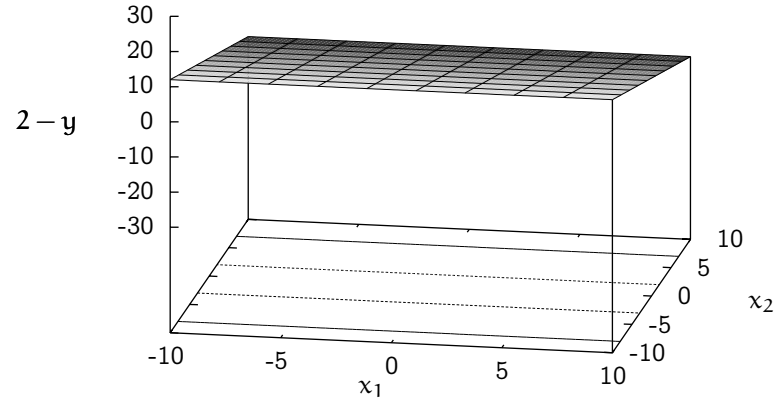


Figure 4.4: Objective function of Example 4.2: $\lambda = 2$

- If $\lambda = 2$, (4.3) becomes $2 - x_2$ and the problem is unbounded, because the more the value of x_2 increases, the more the objective function decreases (Figure 4.4). It is imperative to avoid such values of the penalty parameter, which generate unbounded problems.
- Finally, if $\lambda = 1$, (4.3) becomes $x_1 + 1$, and each x such that $x_1 = 0$ is a solution to the problem, with an optimal value of 1 (Figure 4.5). In this case, regardless of the value of x_2, there is no way to get a better value than the optimal value of the initial problem. The penalty parameter acts as a deterrent, and there is nothing to be gained by violating the constraint.

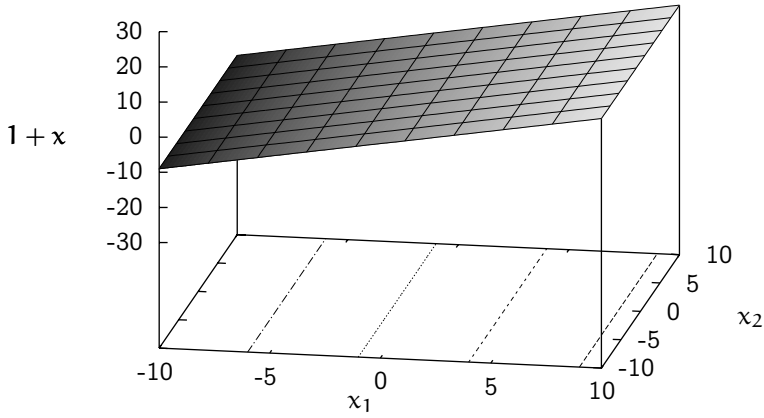


Figure 4.5: Objective function of Example 4.2: $\lambda = 3$

Consider the optimization problem (1.71)–(1.73). We generalize this idea to incorporate constraints in the objective function. The function thus obtained is called *Lagrangian* or the *Lagrangian function*.

Definition 4.3 (Lagrangian function)**.** Consider the optimization problem (1.71)–(1.73) $\min f(x)$ subject to $h(x) = 0$ and $g(x) \leq 0$, and consider the vectors $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p$. The function $L : \mathbb{R}^{n+m+p} \to \mathbb{R}$ defined by

$$\begin{aligned} L(x, \lambda, \mu) &= f(x) + \lambda^T h(x) + \mu^T g(x) \\ &= f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{p} \mu_j g_j(x) \end{aligned} \tag{4.5}$$

is called *Lagrangian* or the *Lagrangian function* of the problem (1.71)–(1.73).

As we did in Example 4.2, we can minimize the Lagrangian function for each fixed value of the parameters λ and μ. Indeed, the Lagrangian function now depends only on x. The function that associates a set of parameters to the optimal value of the associated problem is called a dual function.

Definition 4.4 (Dual function)**.** Consider the optimization problem (1.71)–(1.73) and its Lagrangian function $L(x, \lambda, \mu)$ defined by (4.5). The function $q : \mathbb{R}^{m+p} \to \mathbb{R}$ defined by

$$q(\lambda, \mu) = \min_{x \in \mathbb{R}^n} L(x, \lambda, \mu) \tag{4.6}$$

is the dual function of the problem (1.71)–(1.73). The parameters λ and μ are called dual variables. In this context, the variables x are called primal variables.

If we take Example 4.1, $-q(\lambda, \mu)$ represents the mountaineer's prize[1] if the billionaire imposes a fine for violation of the constraints $\lambda^T h(x) + \mu^T g(x)$.

For inequality constraints, since only non negative values of $g(x)$ should be avoided and result in a fine, it is essential that $\mu \geq 0$. Indeed, the term $\mu^T g(x)$ is non negative, and thus penalizing, only when $g(x) > 0$.

Theorem 4.5 (Bound from dual function). *Let x^* be the solution to the optimization problem (1.71)–(1.73), and let $q(\lambda, \mu)$ be the dual function to the same problem. Consider $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p$, $\mu \geq 0$. Then,*

$$q(\lambda, \mu) \leq f(x^*), \tag{4.7}$$

and the dual function provides lower bounds on the optimal value of the problem.

Proof.

$$\begin{aligned}
q(\lambda, \mu) &= \min_{x \in \mathbb{R}^n} L(x, \lambda, \mu) && \text{according to (4.6)} \\
&\leq L(x^*, \lambda, \mu) \\
&= f(x^*) + \lambda^T h(x^*) + \mu^T g(x^*) && \text{according to (4.5)} \\
&= f(x^*) + \mu^T g(x^*) && h(x^*) = 0 \\
&\leq f(x^*) && g(x^*) \leq 0 \text{ and } \mu \geq 0.
\end{aligned}$$

□

Corollary 4.6 (Objective functions of the primal and dual). *Let x be a feasible solution of the optimization problem (1.71)–(1.73), and let $q(\lambda, \mu)$ be the dual function to the same problem. Consider $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p$, $\mu \geq 0$. Then,*

$$q(\lambda, \mu) \leq f(x). \tag{4.8}$$

Proof. Denote x^* the optimal solution of the primal problem. As x is primal feasible, we have $f(x^*) \leq f(x)$. The results follows from Theorem (4.5). □

If we take the point of view of the billionaire, the problem is to define these fines in such a manner that the mountaineer wins as little as possible with the new system. He tries to optimize the dual function, ensuring that the considered parameters λ and $\mu \geq 0$ do not generate an unbounded problem. This optimization problem is called the dual problem.

[1] The sign of q is changed because the problem with the mountaineer is one of maximization and not minimization.

Definition 4.7 (Dual problem)**.** Consider the optimization problem (1.71)–(1.73) and its dual function $q(\lambda, \mu)$ defined by (4.6). Let $X_q \subseteq \mathbb{R}^{m+p}$ be the domain of q, i.e.,

$$X_q = \{\lambda, \mu \mid q(\lambda, \mu) > -\infty\}. \tag{4.9}$$

The optimization problem

$$\max_{\lambda,\mu} q(\lambda, \mu) \tag{4.10}$$

subject to

$$\mu \geq 0 \tag{4.11}$$

and

$$(\lambda, \mu) \in X_q \tag{4.12}$$

is the dual problem of the problem (1.71)–(1.73). In this context, the original problem (1.71)–(1.73) is called the primal problem.

Example 4.8 (Dual problem)**.** Take again Example 4.2:

$$\min_{x \in \mathbb{R}^2} 2x_1 + x_2 \tag{4.13}$$

subject to

$$\begin{aligned} h_1(x) &= 1 - x_1 - x_2 = 0 && (\lambda) \\ g_1(x) &= -x_1 \leq 0 && (\mu_1) \\ g_2(x) &= -x_2 \leq 0 && (\mu_2). \end{aligned} \tag{4.14}$$

The Lagrangian function of this problem is

$$\begin{aligned} L(x_1, x_2, \lambda, \mu_1, \mu_2) &= 2x_1 + x_2 + \lambda(1 - x_1 - x_2) - \mu_1 x_1 - \mu_2 x_2 \\ &= (2 - \lambda - \mu_1)x_1 + (1 - \lambda - \mu_2)x_2 + \lambda. \end{aligned}$$

In order for the dual function to be bounded, the coefficients of x_1 and x_2 have to be zero, and

$$2 - \lambda - \mu_1 = 0, \qquad 1 - \lambda - \mu_2 = 0,$$

or

$$\mu_1 = 2 - \lambda, \qquad \mu_2 = 1 - \lambda. \tag{4.15}$$

Therefore, we can eliminate μ_1 and μ_2 so that

$$X_q = \{\lambda | \lambda \leq 1\},$$

and the dual function becomes

$$q(\lambda) = \lambda$$

The dual problem is written as

$$\max \lambda$$

subject to

$$\lambda \leq 1,$$

for which the solution is $\lambda^* = 1$. According to the equalities (4.15), we have $\mu_1^* = 1$ and $\mu_2^* = 0$.

As a direct consequence of Theorem 4.5, the optimal value of this problem can never exceed the optimal value of the original problem. This result is called the weak duality theorem.

Theorem 4.9 (Weak duality). *Let x^* be the optimal solution to the primal problem (1.71)–(1.73) and let (λ^*, μ^*) be the optimal solution to the associated dual problem (4.10)–(4.12). Then*

$$q(\lambda^*, \mu^*) \leq f(x^*). \tag{4.16}$$

Proof. This theorem is a special case of Theorem 4.5 for $\lambda = \lambda^*$ and $\mu = \mu^*$. □

Corollary 4.10 (Duality and feasibility). *Consider the primal problem (1.71)–(1.73) and the associated dual problem (4.10)–(4.12).*

- *If the primal problem is unbounded, then the dual problem is not feasible.*
- *If the dual problem is unbounded, then the primal problem is not feasible.*

Proof. If the optimal value of the primal problem is $-\infty$, there is no dual variable (λ, μ) that satisfies (4.16) and the dual problem is not feasible. Similarly, if the optimal value of the dual problem is $+\infty$, there is no primal variable x that satisfies (4.16) and the primal problem is not feasible.

□

Corollary 4.11 (Optimality of the primal and the dual). *Let x^* be a feasible solution of the primal problem (1.71)–(1.73) and let (λ^*, μ^*) be a feasible solution of the associated dual problem (4.10)–(4.12). If $q(\lambda^*, \mu^*) = f(x^*)$, then x^* is optimal for the primal, and (λ^*, μ^*) is optimal for the dual.*

Proof. Consider any x feasible for the primal. From Theorem 4.5, we have

$$f(x) \geq q(\lambda^*, \mu^*) = f(x^*),$$

proving the optimality of x^*. Similarly, consider any (λ, μ) feasible for the dual. From the same theorem, we have

$$q(\lambda, \mu) \leq f(x^*) = q(\lambda^*, \mu^*),$$

proving the optimality of (λ^*, μ^*). □

Corollary 4.12 (Duality and feasibility (II)). *Consider the primal problem (1.71)–(1.73) and the associated dual problem (4.10)–(4.12).*

- *If the primal problem is infeasible, then the dual problem is either unbounded or infeasible.*
- *If the dual problem is infeasible, then the primal problem is either unbounded or infeasible.*

Proof. We show the contrapositive. If the dual problem is bounded and feasible, it has an optimal solution. From Corollary 4.11, the primal problem has also an optimal solution, as is therefore feasible. The second statement is shown in a similar way. □

The dual problem has interesting geometric properties. Indeed, the objective function to maximize is concave, and the domain X_q is convex.

Theorem 4.13 (Concavity-convexity of a dual problem). *Let (4.10)–(4.12) be the dual problem of an optimization problem. The objective function (4.10) is concave, and the domain of the dual function (4.9) is convex.*

Proof. Consider $x \in \mathbb{R}^n$, $\gamma = (\lambda, \mu)$ and $\bar{\gamma} = (\bar{\lambda}, \bar{\mu}) \in \mathbb{R}^{m+p}$, such that $\mu, \bar{\mu} \geq 0$, $\gamma, \bar{\gamma} \in X_q$ and $\gamma \neq \bar{\gamma}$. Consider also $\alpha \in \mathbb{R}$ such that $0 \leq \alpha \leq 1$. According to Definition 4.3, we have

$$L\big(x, \alpha\gamma + (1-\alpha)\bar{\gamma}\big) = \alpha L(x, \gamma) + (1-\alpha) L(x, \bar{\gamma})\,.$$

Taking the minimum, we obtain

$$\min_x L\big(x, \alpha\gamma + (1-\alpha)\bar{\gamma}\big) \geq \alpha \min_x L(x, \gamma) + (1-\alpha) \min_x L(x, \bar{\gamma}) \tag{4.17}$$

or

$$q\big(\alpha\gamma + (1-\alpha)\bar{\gamma}\big) \geq \alpha q(\gamma) + (1-\alpha) q(\bar{\gamma})\,, \tag{4.18}$$

which demonstrates the concavity of q (Definition 2.3). Since γ and $\bar{\gamma}$ are in X_q, then $q(\gamma) > -\infty$ and $q(\bar{\gamma}) > -\infty$. According to (4.18), we also have $q(\alpha\gamma + (1-\alpha)\bar{\gamma}) > -\infty$ and this way $\alpha\gamma + (1-\alpha)\bar{\gamma}$ is in X_q, proving the convexity of X_q (Definition B.2). □

Giuseppe Lodovico Lagrangia, born in Turin on January 25, 1736, is often considered a French mathematician, despite his Italian origin, and is known under the name Joseph-Louis Lagrange. It was he himself who in his youth took the French version of the name. In 1766, he succeeded Euler as director of the mathematics section of the Academy of Sciences in Berlin. He was the first professor of analysis at the Ecole Polytechnique in Paris (founded 1794 under the name "Ecole Centrale des Travaux Publics." He was a member of the Bureau des Longitudes, created on June 25, 1795. Napoleon presented him with the Legion of Honor in 1808 and the Grand Cross of the Imperial Order of Reunion on April 3, 1813, a few days before his death. He contributed significantly to diverse areas, such as calculus, astronomy, analytical mechanics, probability, fluid mechanics and number theory. He oversaw the introduction of the metric system, working with Lavoisier. He died on April 10, 1813, and is buried in the Pantheon, in Paris. The funeral oration was given by Laplace.

Figure 4.6: Joseph-Louis Lagrange

4.2 Duality in linear optimization

We now analyze the dual problem in the context of linear optimization. We consider the following (primal) problem:

$$\min_x c^T x \tag{4.19}$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned} \tag{4.20}$$

and we have

$$h(x) = b - Ax \qquad \text{and} \qquad g(x) = -x\,.$$

Therefore, the Lagrangian function (4.5) can be written as

$$\begin{aligned} L(x, \lambda, \mu) &= c^T x + \lambda^T (b - Ax) - \mu^T x \\ &= \left(c - A^T\lambda - \mu\right)^T x + \lambda^T b\,. \end{aligned} \tag{4.21}$$

The Lagrangian function is linear in x. The only possibility for it to be bounded is if it is constant, i.e.,

$$c - A^T\lambda - \mu = 0\,.$$

In this case, the dual function is $q(\lambda, \mu) = \lambda^T b$ and the dual problem is written as

$$\max_{\lambda,\mu} \lambda^T b \tag{4.22}$$

subject to

$$\begin{aligned} \mu &\geq 0 \\ \mu &= c - A^T\lambda\,. \end{aligned} \tag{4.23}$$

By eliminating μ, renaming λ x, and changing the maximization to a minimization, we obtain

$$\min_x -b^T x \tag{4.24}$$

subject to

$$A^T x \leq c\,. \tag{4.25}$$

This is another linear optimization problem. We calculate its dual problem. Since there are no equality constraints, we have

$$\begin{aligned} L(x,\mu) &= -b^T x + \mu^T(A^T x - c) \\ &= (-b + A\mu)^T x - \mu^T c\,. \end{aligned}$$

Again, for this linear function to be bounded, it has to be constant, i.e., $-b+A\mu = 0$. The dual function is $q(\mu) = -\mu^T c$ and the dual problem is written as

$$\max_\mu -\mu^T c$$

subject to

$$\begin{aligned} \mu &\geq 0 \\ A\mu &= b\,. \end{aligned}$$

By replacing μ by x, and transforming the maximization into a minimization, we obtain the original problem (4.19)–(4.20). The dual of the dual problem is the primal problem. We can now generalize these results.

Theorem 4.14 (Dual of a linear problem). *Consider the following linear problem:*

$$\min_x c_1^T x_1 + c_2^T x_2 + c_3^T x_3 \tag{4.26}$$

subject to

$$\begin{aligned} A_1x_1 + B_1x_2 + C_1x_3 &= b_1 \\ A_2x_1 + B_2x_2 + C_2x_3 &\leq b_2 \\ A_3x_1 + B_3x_2 + C_3x_3 &\geq b_3 \\ x_1 &\geq 0 \\ x_2 &\leq 0 \\ x_3 &\in \mathbb{R}^{n_3}\,, \end{aligned} \tag{4.27}$$

where $x_1 \in \mathbb{R}^{n_1}$, $x_2 \in \mathbb{R}^{n_2}$, $x_3 \in \mathbb{R}^{n_3}$, $b_1 \in \mathbb{R}^m$, $b_2 \in \mathbb{R}^{p_i}$ *and* $b_3 \in \mathbb{R}^{p_s}$. *The matrices* A_i, B_i, C_i, $i = 1,2,3$, *have appropriate dimensions. The dual of this problem is*

$$\max_\gamma \gamma^T b = \gamma_1^T b_1 + \gamma_2^T b_2 + \gamma_3^T b_3 \tag{4.28}$$

subject to

$$\begin{aligned} & & (\gamma_1 \in \mathbb{R}^m) \\ & & \gamma_2 \leq 0 \\ & & \gamma_3 \geq 0 \\ (A_1^T\gamma_1 + A_2^T\gamma_2 + A_3^T\gamma_3 =) & & A^T\gamma \leq c_1 \\ (B_1^T\gamma_1 + B_2^T\gamma_2 + B_3^T\gamma_3 =) & & B^T\gamma \geq c_2 \\ (C_1^T\gamma_1 + C_2^T\gamma_2 + C_3^T\gamma_3 =) & & C^T\gamma = c_3 \end{aligned} \tag{4.29}$$

with $\gamma = \begin{pmatrix} \gamma_1^T & \gamma_2^T & \gamma_3^T \end{pmatrix}^T \in \mathbb{R}^{m+p_i+p_s}$ *and* $A = \begin{pmatrix} A_1^T & A_2^T & A_3^T \end{pmatrix}^T \in \mathbb{R}^{(m+p_i+p_s)\times n_1}$. *The matrices* B *and* C *are defined in a similar manner.*

Proof. The Lagrangian function is written as

$$\begin{aligned} L(x, \lambda, \mu_2, \mu_3, \mu_{x_1}, \mu_{x_2}) = {} & c_1^T x_1 + c_2^T x_2 + c_3^T x_3 \\ & + \lambda^T (b_1 - A_1 x_1 - B_1 x_2 - C_1 x_3) \\ & + \mu_2^T (A_2 x_1 + B_2 x_2 + C_2 x_3 - b_2) \\ & + \mu_3^T (b_3 - A_3 x_1 - B_3 x_2 - C_3 x_3) \\ & - \mu_{x_1}^T x_1 \\ & + \mu_{x_2}^T x_2 \end{aligned}$$

with $\mu_2,\ \mu_3,\ \mu_{x_1},\ \mu_{x_2} \geq 0$. By combining the terms, we obtain

$$\begin{aligned} L(x, \lambda, \mu_2, \mu_3, \mu_{x_1}, \mu_{x_2}) = {} & \lambda^T b_1 - \mu_2^T b_2 + \mu_3^T b_3 \\ & + \left(c_1 - A_1^T\lambda + A_2^T\mu_2 - A_3^T\mu_3 - \mu_{x_1}\right)^T x_1 \\ & + \left(c_2 - B_1^T\lambda + B_2^T\mu_2 - B_3^T\mu_3 + \mu_{x_2}\right)^T x_2 \\ & + \left(c_3 - C_1^T\lambda + C_2^T\mu_2 - C_3^T\mu_3\right)^T x_3 . \end{aligned}$$

Define $\gamma_1 = \lambda$, $\gamma_2 = -\mu_2$ and $\gamma_3 = \mu_3$. We immediately deduce that $\gamma_1 \in \mathbb{R}^m$, $\gamma_2 \leq 0$ and $\gamma_3 \geq 0$. We obtain the Lagrangian function

$$\begin{aligned} L(x, \gamma, \mu_{x_1}, \mu_{x_2}) = {} & \gamma^T b \\ & + \left(c_1 - A^T\gamma - \mu_{x_1}\right)^T x_1 \\ & + \left(c_2 - B^T\gamma + \mu_{x_2}\right)^T x_2 \\ & + \left(c_3 - C^T\gamma\right)^T x_3 . \end{aligned}$$

This is a linear function. For it to be bounded, is needs to be constant and

$$\begin{aligned} \mu_{x_1} &= c_1 - A^T\gamma \\ \mu_{x_2} &= B^T\gamma - c_2 \\ C^T\gamma &= c_3 . \end{aligned}$$

We now need only use $\mu_{x_1} \geq 0$ and $\mu_{x_2} \geq 0$ to obtain the result. □

Note that the problem (4.26)–(4.27) combines all the possibilities of writing the constraints of a linear problem: equality, lower inequality, upper inequality, non positivity, and non negativity. The result can be summarized as follows:

- For each constraint of the primal there is a dual variable

Constraint of the primal	Dual variable
$=$	free
$\leq$	≤ 0
$\geq$	≥ 0

- For each primal variable there is a dual constraint

Primal variable	Dual constraint
≥ 0	$\leq$
≤ 0	$\geq$
free	$=$

Theorem 4.15 (The dual of the dual is the primal)**.** *Consider a (primal) linear optimization problem. If the dual is converted into a minimization problem, and we calculate its dual, we obtain a problem equivalent to the primal problem.*

Proof. In the problem (4.28)–(4.29) of Theorem 4.14, we replace γ by $-x$ and the maximization by a minimization:

$$\min_{x_1,x_2,x_3} x_1^T b_1 + x_2^T b_2 + x_3^T b_3$$

subject to

$$\begin{aligned} x_1 &\in \mathbb{R}^m \\ x_2 &\geq 0 \\ x_3 &\leq 0 \\ A_1^T x_1 + A_2^T x_2 + A_3^T x_3 &\geq -c_1 \\ B_1^T x_1 + B_2^T x_2 + B_3^T x_3 &\leq -c_2 \\ C_1^T x_1 + C_2^T x_2 + C_3^T x_3 &= -c_3 \,. \end{aligned}$$

According to Theorem 4.14, the dual of this problem is

$$\max_{\gamma_1,\gamma_2,\gamma_3} -c_1\gamma_1 - c_2\gamma_2 - c_3\gamma_3$$

subject to

$$\begin{aligned} A_1\gamma_1 + B_1\gamma_2 + C_1\gamma_3 &= b_1 \\ A_2\gamma_1 + B_2\gamma_2 + C_2\gamma_3 &\leq b_2 \\ A_3\gamma_1 + B_3\gamma_2 + C_3\gamma_3 &\geq b_3 \\ \gamma_1 &\geq 0 \\ \gamma_2 &\leq 0 \\ \gamma_3 &\in \mathbb{R}^{n_3} \,. \end{aligned}$$

We now need only replace γ by x, and convert the maximization into a minimization to obtain (4.26)–(4.27) and prove the result. □

Example 4.16 (Dual of a linear problem)**.** Consider the linear optimization problem

$$\min x_1 + 2x_2 + 3x_3$$

subject to

$$\begin{aligned} -x_1 \qquad + 3x_2 &= 5 \\ 2x_1 - x_2 + 3x_3 &\geq 6 \\ x_3 &\leq 4 \\ x_1 &\geq 0 \\ x_2 &\leq 0 \\ x_3 &\in \mathbb{R}\,. \end{aligned}$$

The dual problem is

$$\max 5\gamma_1 + 6\gamma_2 + 4\gamma_3$$

subject to

$$\begin{aligned} \gamma_1 &\in \mathbb{R} \\ \gamma_2 &\geq 0 \\ \gamma_3 &\leq 0 \\ -\gamma_1 + 2\gamma_2 &\leq 1 \\ 3\gamma_1 - \gamma_2 &\geq 2 \\ 3\gamma_2 + \gamma_3 &= 3\,. \end{aligned}$$

This is also a linear problem. We write it as a minimization problem and rename the variables x.

$$\min -5x_1 - 6x_2 - 4x_3$$

subject to

$$\begin{aligned} x_1 &\in \mathbb{R} \\ x_2 &\geq 0 \\ x_3 &\leq 0 \\ x_1 - 2x_2 &\geq -1 \\ -3x_1 + x_2 &\leq -2 \\ -3x_2 - x_3 &= -3\,. \end{aligned}$$

We can calculate its dual :

$$\max -\gamma_1 - 2\gamma_2 - 3\gamma_3$$

subject to

$$\begin{aligned} \gamma_1 \qquad - 3\gamma_2 &= -5 \\ -2\gamma_1 + \gamma_2 - 3\gamma_3 &\leq -6 \\ -\gamma_3 &\geq -4 \\ \gamma_1 &\geq 0 \\ \gamma_2 &\leq 0 \\ \gamma_3 &\in \mathbb{R}. \end{aligned}$$

It is easy to verify that this problem is equivalent to the original problem.

We conclude this chapter with an important result in linear optimization, called the *strong duality* theorem.

Theorem 4.17 (Strong duality). *Consider a linear optimization problem and its dual. If one problem has an optimal solution, so does the other one, and the optimal value of their objective functions are the same.*

Proof. Consider $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^m$. Consider the primal problem

$$\min c^T x$$

subject to

$$Ax = b, \quad x \geq 0,$$

and the dual problem

$$\max b^T \lambda$$

subject to

$$A^T \lambda \leq c.$$

Assume that the dual problem has an optimal solution λ^*. Therefore, for each λ that is dual feasible, we have $b^T\lambda \leq b^T\lambda^*$. Equivalently, for any $\varepsilon > 0$, we have $b^T\lambda < b^T\lambda^* + \varepsilon$. Therefore, there is no λ which verifies both the dual constraints $A^T\lambda \leq c$ and $b^T\lambda \geq b^T\lambda^* + \varepsilon$ or, equivalently, $-b^T\lambda \leq -b^T\lambda^* - \varepsilon$. In other words, the system of $n+1$ linear inequalities, with m variables

$$\begin{pmatrix} A^T \\ -b^T \end{pmatrix} \lambda \leq \begin{pmatrix} c \\ -b^T\lambda^* - \varepsilon \end{pmatrix}$$

is incompatible. According to Farkas' lemma (Lemma C.10), there exists a vector

$$\begin{pmatrix} y \\ r \end{pmatrix},$$

where $y \in \mathbb{R}^n$, $y \geq 0$, and $r \in \mathbb{R}$, $r \geq 0$, such that

$$\begin{pmatrix} y^T & r \end{pmatrix} \begin{pmatrix} A^T \\ -b^T \end{pmatrix} = 0,$$

that is

$$Ay - rb = 0, \tag{4.30}$$

and

$$\begin{pmatrix} y^T & r \end{pmatrix} \begin{pmatrix} c \\ -b^T\lambda^* - \varepsilon \end{pmatrix} < 0,$$

that is

$$c^T y - rb^T\lambda^* - r\varepsilon < 0. \tag{4.31}$$

We distinguish two cases.

$r = 0$ In this case, (4.30) is $Ay = 0$ and (4.31) is $c^T y < 0$. Applying Farkas' lemma to the compatible system $A^T\lambda \leq c$ (it is verified at least by λ^*), we obtain that $c^T y \geq 0$, for each $y \geq 0$ such that $Ay = 0$, contradicting (4.30)–(4.31). Therefore $r \neq 0$.

$r > 0$ Divide (4.30) by r, and define $x^* = y/r$ to obtain

$$Ax^* - b = 0. \tag{4.32}$$

As $y \geq 0$ and $r > 0$, x^* is feasible for the primal problem. Dividing also (4.31) by r, we obtain

$$c^T x^* - b^T\lambda^* - \varepsilon < 0. \tag{4.33}$$

Denote $\delta = c^T x^* - b^T\lambda^*$. By Corollary (4.6), as x^* is primal feasible and λ^* dual feasible, we know that $\delta = c^T x^* - b^T\lambda^* \geq 0$. Therefore, (4.33) is written as

$$0 \leq \delta < \varepsilon.$$

As this must be true for any arbitrary small ε, we obtain $\delta = 0$, and $c^T x^* = b^T\lambda^*$. From Corollary (4.11), x^* is the optimal solution of the primal problem.

As the dual of the dual is the primal (Theorem (4.15)), the result holds in the other direction as well. □

Another proof, based on the optimality conditions, is presented in Theorem 6.33. Note that the strong duality result does not hold in general for all optimization problems. Yet, it holds if the objective function is convex and the constraints linear (see Bertsekas, 1999, Proposition 5.2.1).

4.3 Exercises

Exercise 4.1. Consider the optimization problem

$$\min_{x \in \mathbb{R}^2} x_1^2 + x_2^2 \qquad \text{subject to } x_1 = 1\,.$$

1. Write the Lagrangian of this problem.
2. Write the dual function.
3. Write and solve the dual problem.

Exercise 4.2. Same questions as for Exercise 4.1 for the problem

$$\min_{x \in \mathbb{R}^2} \frac{1}{2}(x_1^2 + x_2^2) \qquad \text{s.c. } x_1 \geq 1\,.$$

Exercise 4.3. Consider a matrix $A \in \mathbb{R}^{n \times n}$ such that $A^T = -A$, and the vector $c \in \mathbb{R}^n$. Consider the optimization problem

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to

$$\begin{aligned} Ax &\geq -c \\ x &\geq 0\,. \end{aligned}$$

Demonstrate that this problem is self-dual, i.e., that the dual problem is equivalent to the primal problem.

Exercise 4.4. Consider the optimization problem

$$\min_{x \in \mathbb{R}^2} -3x_1 + 2x_2$$

subject to

$$\begin{aligned} x_1 - x_2 &\leq 2 \\ -x_1 + x_2 &\leq -3 \\ x_1,\, x_2 &\geq 0\,. \end{aligned}$$

1. Write the Lagrangian.
2. Write the dual function.
3. Write the dual problem.
4. Represent graphically the feasible set of the primal problem.
5. Represent graphically the feasible set of the dual problem.

Exercise 4.5. Same questions for the following problem.

$$\min_{x \in \mathbb{R}^2} -x_1 - x_2$$

subject to

$$\begin{aligned} -x_1 + x_2 &\leq 1 \\ x_1 - \frac{1}{2} x_2 &\leq 0 \\ x_1,\, x_2 &\geq 0\,. \end{aligned}$$

Part II

Optimality conditions

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

Albert Einstein

Before developing algorithms that enable us to identify solutions to an optimization problem, we must be able to decide whether a given point is optimal or not. These optimality conditions have three key roles in the development of algorithms:

1. they provide a theoretical analysis of the problem,
2. they directly inspire ideas for algorithms,
3. they render it possible to determine a stopping criterion for iterative algorithms.

We analyze them in detail and in a gradual manner, starting with the simplest ones.

Chapter 5

Unconstrained optimization

Contents

5.1 Necessary optimality conditions

Consider the problem of unconstrained optimization (1.71) $\min_{x \in \mathbb{R}^n} f(x)$ and a local minimum x^*, as defined in Definition 1.5. We attempt to characterize the minimum by using the results developed in Chapter 2. The first optimality condition is a generalization of a well-known result, attributed to Fermat.

Theorem 5.1 (Necessary optimality conditions). *Let x^* be a local minimum of a function $f : \mathbb{R}^n \to \mathbb{R}$. If f is differentiable in an open neighborhood V of x^*, then,*

$$\nabla f(x^*) = 0\,. \tag{5.1}$$

If, in addition, f is twice differentiable on V, then

$$\nabla^2 f(x^*) \quad \text{is positive semidefinite}\,. \tag{5.2}$$

Condition (5.1) is said to be a first-order necessary condition, and condition (5.2) is said to be a second-order necessary condition.

Proof. We recall that $-\nabla f(x^*)$ is the direction of the steepest descent in x^* (Theorem 2.13) and assume by contradiction that $\nabla f(x^*) \neq 0$. We can then use Theorem 2.11 with the descent direction $d = -\nabla f(x^*)$ to obtain η such that

$$f\big(x^* - \alpha \nabla f(x^*)\big) < f(x^*)\,, \qquad \forall \alpha \in \,]0, \eta]\,,$$

which contradicts the optimality of x^* and demonstrates the first-order condition. To demonstrate the second-order condition, we invoke Taylor's theorem (Theorem C.2) in x^*, with an arbitrary direction d and an arbitrary step $\alpha > 0$ such that $x^* + \alpha d \in V$. As

$$f(x^* + \alpha d) - f(x^*) = \alpha d^T \nabla f(x^*) + \frac{1}{2}\alpha^2 d^T \nabla^2 f(x^*) d + o\big(\|\alpha d\|^2\big),$$

we have

$$\begin{aligned} f(x^* + \alpha d) - f(x^*) &= \frac{1}{2}\,\alpha^2 d^T \nabla^2 f(x^*) d + o\big(\|\alpha d\|^2\big) && \text{from (5.1)} \\ &= \frac{1}{2}\,\alpha^2 d^T \nabla^2 f(x^*) d + o(\alpha^2) && \|d\| \text{ does not depend on } \alpha \\ &\geq 0 && x^* \text{ is optimal}. \end{aligned}$$

When we divide by α^2, we get

$$\frac{1}{2}\, d^T \nabla^2 f(x^*) d + \frac{o(\alpha^2)}{\alpha^2} \geq 0\,. \tag{5.3}$$

Intuitively, as the second term can be made as small as desired, the result must hold. More formally, let us assume by contradiction that $d^T \nabla^2 f(x^*) d$ is negative and that its value is -2ε, with $\varepsilon > 0$. According to the Landau notation $o(\cdot)$ (Definition B.17), for all $\varepsilon > 0$, there exists η such that

$$\frac{\big|o(\alpha^2)\big|}{\alpha^2} < \varepsilon\,, \qquad \forall 0 < \alpha \leq \eta\,,$$

and

$$\frac{1}{2}\, d^T \nabla^2 f(x^*) d + \frac{o(\alpha^2)}{\alpha^2} \leq \frac{1}{2}\, d^T \nabla^2 f(x^*) d + \frac{\big|o(\alpha^2)\big|}{\alpha^2} < -\frac{1}{2}\, 2\varepsilon + \varepsilon = 0\,,$$

which contradicts (5.3) and proves that $d^T \nabla^2 f(x^*) d \geq 0$. Since d is an arbitrary direction, $\nabla^2 f(x^*)$ is positive semidefinite (Definition B.8). □

From a geometrical point of view, the second-order condition means that f is locally convex in x (Theorem 2.21).

Example 5.2 (Affine function)**.** Consider an affine function (see Definition 2.25):

$$f(x) = c^T x + d, \tag{5.4}$$

where $c \in \mathbb{R}^n$ is a vector of constants and $d \in \mathbb{R}$. Then, $\nabla f(x) = c$ and $\nabla^2 f(x) = 0$. Therefore, the necessary optimality conditions are verified for every x if $c = 0$, and for no x if $c \neq 0$. The geometric interpretation is that an affine function is bounded only if it is constant. We have used this property in Section 4.1 to derive the dual problem.

Example 5.3 (Necessary optimality condition – I)**.** Consider the function

$$f(x_1, x_2) = 100\,\big(x_2 - x_1^2\big)^2 + \big(1 - x_1\big)^2$$

illustrated in Figure 5.1 (see Section 11.6 for a discussion of this function). The point $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$ is a local minimum of the function. We have

$$\nabla f(x_1, x_2) = \begin{pmatrix} 400\,x_1^3 - 400\,x_1 x_2 + 2x_1 - 2 \\ 200\,x_2 - 200\,x_1^2 \end{pmatrix},$$

which is indeed zero in $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$. Moreover,

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} 1200\,x_1^2 - 400\,x_2 + 2 & -400\,x_1 \\ -400\,x_1 & 200 \end{pmatrix},$$

which, in $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$, is:

$$\nabla^2 f(1, 1) = \begin{pmatrix} 802 & -400 \\ -400 & 200 \end{pmatrix},$$

for which the eigenvalues are positive (0.39936 and 1,001.6) and the Hessian matrix is positive semidefinite. Note that the conditioning of f in $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$ is high (2,508) and that the function is ill-conditioned at the solution (Section 2.5).

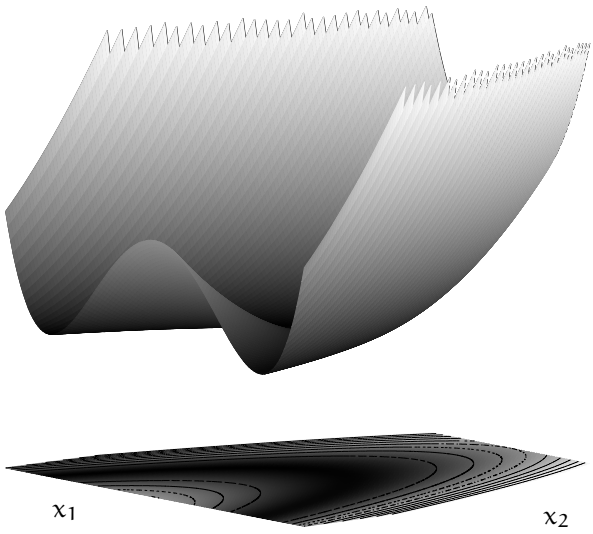


Figure 5.1: Function of Example 5.3

It is important to emphasize that the necessary optimality conditions are not sufficient, as shown by Examples 5.4 and 5.5.

Example 5.4 (Necessary optimality condition – II)**.** Consider the function

$$f(x_1, x_2) = -x_1^4 - x_2^4$$

illustrated in Figure 5.2. The point $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ satisfies the necessary optimality conditions. Indeed,

$$\nabla f(x_1, x_2) = \begin{pmatrix} -4x_1^3 \\ -4x_2^3 \end{pmatrix}$$

is zero in $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$. Moreover,

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} -12\,x_1^2 & 0 \\ 0 & -12\,x_2^2 \end{pmatrix}$$

is positive semidefinite in $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$. However, this is not a local minimum. To demonstrate this, consider a non zero arbitrary direction $d = \begin{pmatrix} d_1 & d_2 \end{pmatrix}^T$ and take a step $\alpha > 0$ of any length from the point $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$. We have

$$0 = f(0,0) > f(\alpha d_1, \alpha d_2) = -(\alpha d_1)^4 - (\alpha d_2)^4$$

and $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ turns out to be a local maximum. From a geometrical point of view, the function is in fact concave and not convex in $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$.

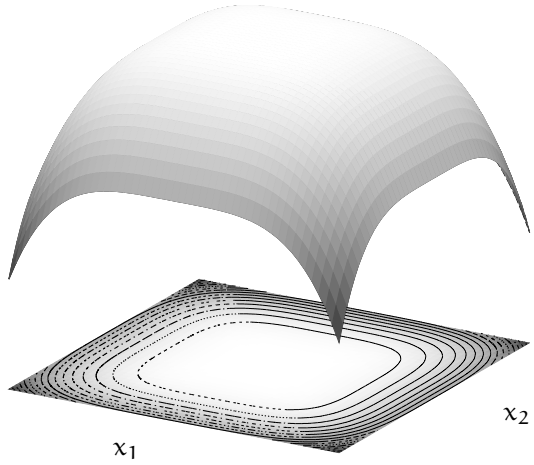


Figure 5.2: Function of Example 5.4

Example 5.5 (Necessary optimality condition – III)**.** Consider the function

$$f(x_1, x_2) = 50\,x_1^2 - x_2^3$$

illustrated in Figure 5.3. The point $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ satisfies the necessary optimality conditions. Indeed,

$$\nabla f(x_1, x_2) = \begin{pmatrix} 100\,x_1 \\ -3x_2^2 \end{pmatrix}$$

is zero in $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$. Moreover,

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} 100 & 0 \\ 0 & -6x_2 \end{pmatrix}$$

is positive semidefinite in $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$. However, it is not a local minimum. To show this, consider the direction $d = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$ and take any one step $\alpha > 0$ from the point $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$. We have

$$0 = f(0,0) > f(0,\alpha) = -\alpha^3$$

and $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ is not a local minimum. However, if we consider the direction $d = \begin{pmatrix} 0 & -1 \end{pmatrix}^T$, we obtain

$$0 = f(0,0) < f(0,-\alpha) = \alpha^3 .$$

Then, $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ is not a local maximum either. From a geometrical viewpoint, the function is neither concave nor convex in $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$.

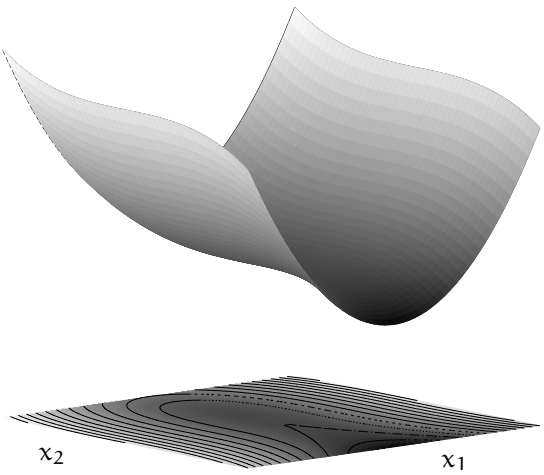


Figure 5.3: Function of Example 5.5

In practice, the second-order necessary condition is difficult to check, as this requires calculations of the second derivatives and analyses of the eigenvalues of the Hessian matrix. The first-order necessary optimality condition plays a central role in optimization. The vectors x that satisfy this condition are called *critical* points or *stationary* points. Among them, there are local minima, local maxima, and points that are neither (Example 5.5). The latter are called *saddle points*.

Definition 5.6 (Critical point). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Any vector $x \in \mathbb{R}^n$ such that $\nabla f(x) = 0$ is said to be a critical point or stationary point of f.

5.2 Sufficient optimality conditions

Theorem 5.7 (Sufficient optimality conditions). *Consider a function $f : \mathbb{R}^n \to \mathbb{R}$ twice differentiable in an open subset V of $\mathbb{R}^n$ and let $x^* \in V$ satisfy the conditions*

$$\nabla f(x^*) = 0 \tag{5.5}$$

and

$$\nabla^2 f(x^*) \quad \text{is positive definite.} \tag{5.6}$$

In this case, x^* is a local minimum of f.

Proof. We assume by contradiction that there exists a direction d and $\eta > 0$ such that, for any $0 < \alpha \leq \eta$, $f(x^* + \alpha d) < f(x^*)$. With an identical approach to the proof of Theorem 5.1, we have

$$\frac{f(x^* + \alpha d) - f(x^*)}{\alpha^2} = \frac{1}{2} d^T \nabla^2 f(x^*) d + \frac{o(\alpha^2)}{\alpha^2}$$

and

$$\frac{1}{2} d^T \nabla^2 f(x^*) d + \frac{o(\alpha^2)}{\alpha^2} < 0$$

or

$$\frac{1}{2} d^T \nabla^2 f(x^*) d + \frac{o(\alpha^2)}{\alpha^2} + \varepsilon = 0$$

with $\varepsilon > 0$. According to the definition of the Landau notation $o(\cdot)$ (Definition B.17), there exists $\bar{\eta}$ such that

$$\frac{|o(\alpha^2)|}{\alpha^2} < \varepsilon, \qquad \forall \alpha, 0 < \alpha \leq \bar{\eta},$$

and then, for any $\alpha \leq \min(\eta, \bar{\eta})$, we have

$$-\frac{o(\alpha^2)}{\alpha^2} \leq \frac{|o(\alpha^2)|}{\alpha^2} < \varepsilon,$$

such that

$$\frac{1}{2} d^T \nabla^2 f(x^*) d = -\frac{o(\alpha^2)}{\alpha^2} - \varepsilon < 0,$$

which contradicts the fact that $\nabla^2 f(x^*)$ is positive definite. □

Example 5.8 (Optimality conditions)**.** Consider the function

$$f(x_1, x_2) = \frac{1}{2} x_1^2 + x_1 \cos x_2$$

illustrated in Figure 5.4. We use the optimality conditions to identify the minima of this function. We have

$$\nabla f(x_1, x_2) = \begin{pmatrix} x_1 + \cos x_2 \\ -x_1 \sin x_2 \end{pmatrix}.$$

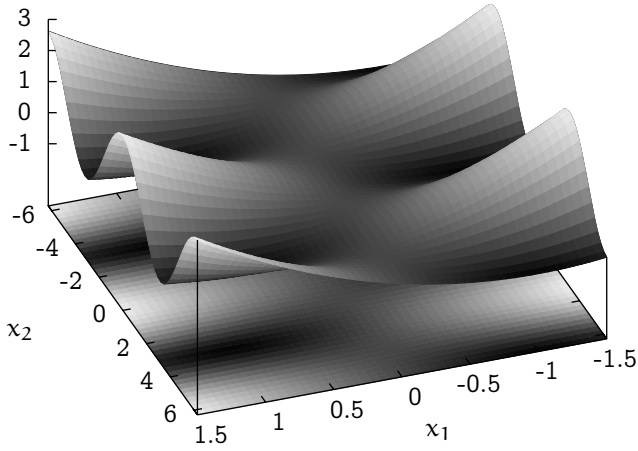


Figure 5.4: Function of Example 5.8: the surface

This gradient is zero for $x_k^* = \left((-1)^{k+1}, k\pi\right)^T$, $k \in \mathbb{Z}$, and for $\bar{x}_k = \left(0, \frac{\pi}{2} + k\pi\right)^T$, $k \in \mathbb{Z}$, as illustrated in Figure 5.5. We also have

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} 1 & -\sin x_2 \\ -\sin x_2 & -x_1 \cos x_2 \end{pmatrix}.$$

By evaluating this matrix in x_k^*, we get for any k

$$\nabla^2 f(x_k^*) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Since this matrix is positive definite, each point x_k^* satisfies the sufficient optimality conditions and is a local minimum of the function.

By evaluating the Hessian matrix in $\bar{x}_k$, we get for any k

$$\nabla^2 f(\bar{x}_k) = \begin{pmatrix} 1 & (-1)^{k+1} \\ (-1)^{k+1} & 0 \end{pmatrix}.$$

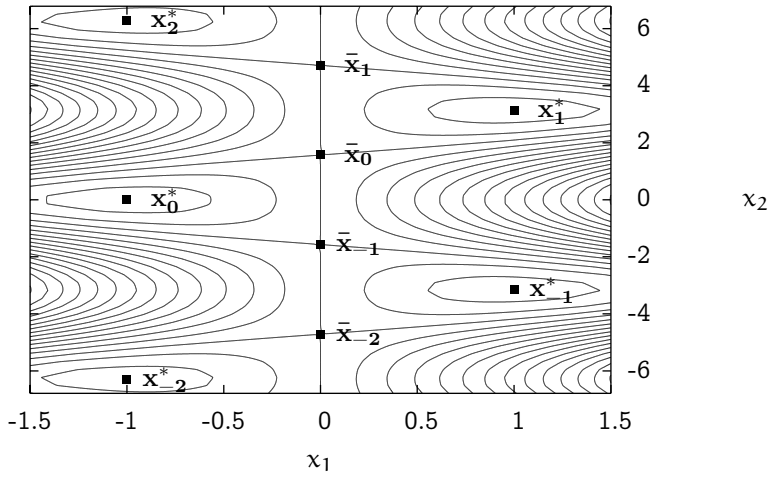


Figure 5.5: Function of Example 5.8: stationary points

Regardless of k, this matrix is not positive semidefinite. Therefore, there is no $\bar{x}_k$ that satisfies the necessary optimality conditions. None of them can then be a local minimum.

We now present a sufficient condition for a local minimum to also be a global minimum.

Theorem 5.9 (Sufficient global optimality conditions)**.** *Consider a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ and let $x^* \in \mathbb{R}^n$ be a local minimum of f. If f is a convex function, then x^* is a global minimum of f. If, moreover, f is strictly convex, x^* is the only global minimum of f.*

Proof. We assume by contradiction that there exists another local minimum $x^+ \neq x^*$, such that $f(x^+) < f(x^*)$. By the convexity of f (Definition 2.1), we have

$$f\big(\alpha x^* + (1-\alpha)x^+\big) \leq \alpha f(x^*) + (1-\alpha) f(x^+),$$

where $0 \leq \alpha \leq 1$. Since $f(x^+) < f(x^*)$, we have for each $\alpha \in [0, 1[$

$$f\big(\alpha x^* + (1-\alpha)x^+\big) < \alpha f(x^*) + (1-\alpha) f(x^*) = f(x^*)\,. \tag{5.7}$$

It means that any point strictly between x^* and x^+ is also strictly better than x^*. Consider an arbitrary $\varepsilon > 0$, and demonstrate that Definition 1.5 of the local minimum is contradicted. If $\varepsilon \geq \|x^* - x^+\|$, (1.75) is not satisfied for $x = x^+$, when taking $\alpha = 1$ in (5.7). If $\varepsilon < \|x^* - x^+\|$, consider $0 < \eta < 1$ such that $\|\eta x^* + (1-\eta)x^+\| = \varepsilon$. In this case, (1.75) is not satisfied for $x = \alpha x^* + (1-\alpha)x^+$ with $\eta \leq \alpha < 1$ according to (5.7). Since $\eta < 1$, such α always exist.

We now consider a strictly convex function, and assume that x^* and y^* are two distinct global minima, and then $x^* \neq y^*$ and $f(x^*) = f(y^*)$. According to Definition 2.2, we have

$$f\big(\alpha x^* + (1-\alpha)y^*\big) < \alpha f(x^*) + (1-\alpha) f(y^*) = f(x^*) = f(y^*)\,, \quad \forall \alpha \in \,]0,1[\,,$$

which contradicts that x^* and y^* are global minima. □

Pierre de Fermat was born in Beaumont-de-Lomagne close to Montauban on August 20, 1601, and died in Castres on January 12, 1665. With the exception of a few isolated articles, Fermat never published and never gave any publicity to his methods. Some of his most important results were written in the margin of books, the most often without proof. For instance, his "observations on Diophante," an important part of his work on number theory, was published by his son on the basis of margin notes in a copy of *Arithmetica*. Fermat's conjecture is probably the most famous of his intuitions. It affirms that when $n \geq 3$, there exists no non zero integer numbers x, y and z such that $x^n + y^n = z^n$. He wrote the following note in the margin of *Arithmetica* by Diophante: "I have a marvelous demonstration, but this margin is too narrow to contain it." This conjecture, called Fermat's last theorem, was proven by Wiles (1995). Fermat's body was transferred from Castres to the Augustinian Convent in Toulouse in 1675.

Figure 5.6: Pierre de Fermat

We conclude this chapter with a discussion of the optimality conditions for quadratic problems (Definition 2.28).

Theorem 5.10 (Optimality conditions for quadratic problems)**.** *We consider the problem*

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} x^T Q x + g^T x + c\,, \tag{5.8}$$

where $Q \in n \times n$ is a symmetric matrix, $g \in \mathbb{R}^n$ and $c \in \mathbb{R}$.

1. *If Q is not positive semidefinite, then the problem (5.8) has no solution, i.e., there is no $x \in \mathbb{R}^n$ that is a local minimum of (5.8).*
2. *If Q is positive definite, then*

$$x^* = -Q^{-1} g \tag{5.9}$$

   is the only global minimum of (5.8).
3. *If Q is positive semidefinite, but not positive definite,*

either the problem is not bounded or there is an infinite number of global minima. More precisely, we consider the Schur decomposition of Q *(see Theorem C.5):*

$$Q = U\Lambda U^T, \tag{5.10}$$

where U *is an orthogonal matrix (composed of the eigenvectors of* Q *organized in columns) and* Λ *is a diagonal matrix with the eigenvalues of* Q *as entries. As* Q *is positive semidefinite but not positive definite, it means that it is rank deficient, so that* r *eigenvalues are positive and* $n-r$ *are zero. We can sort the indices in such a way that the* r *first eigenvalues on the diagonal are positive, and the* $n-r$ *last are zero:*

$$\Lambda = \begin{pmatrix} \Lambda_r & 0 \\ 0 & \Lambda_{n-r} \end{pmatrix} = \begin{pmatrix} \Lambda_r & 0 \\ 0 & 0 \end{pmatrix} = \operatorname{diag}(\lambda_1, \dots, \lambda_r, 0, \dots, 0). \tag{5.11}$$

We decompose the vectors x *and* g *as follows:*

$$U^T x = \begin{pmatrix} y_r \\ y_{n-r} \end{pmatrix} \quad \text{and} \quad U^T g = \begin{pmatrix} g_r \\ g_{n-r} \end{pmatrix}, \tag{5.12}$$

where $y_r, g_r \in \mathbb{R}^r$ *and* $y_{n-r}, g_{n-r} \in \mathbb{R}^{n-r}$*. Therefore, if* $g_{n-r} \neq 0$*, the problem is unbounded. If* $g_{n-r} = 0$*, then for any* $y_{n-r} \in \mathbb{R}^{n-r}$*,*

$$x^* = U \begin{pmatrix} -\Lambda_r^{-1} g_r \\ y_{n-r} \end{pmatrix} \tag{5.13}$$

is a global minimum of (5.8).

Proof. We have $\nabla f(x) = Qx + g$ and $\nabla^2 f(x) = Q$.

1. We assume by contradiction that there exists a local minimum x^* of (5.8). According to (5.2) of Theorem 5.1, $\nabla^2 f(x) = Q$ is positive semidefinite, which contradicts the hypothesis.
2. Since Q is positive definite, the point x^* in (5.9) is indeed definite and
$$\nabla f(x^*) = -QQ^{-1}g + g = 0.$$
The sufficient optimality conditions (5.5) and (5.6) are satisfied and x^* is a local minimum of f. Moreover, according to Theorem 2.21, f is strictly convex. According to Theorem 5.9, x^* is the only global minimum.
3. Using the Schur decomposition, and the fact that U is orthogonal (so that $UU^T = I$), we write the objective function of (5.8) as
$$f(x) = \frac{1}{2} x^T Q x + g^T x + c = \frac{1}{2} x^T U \Lambda U^T x + g^T U U^T x + c. \tag{5.14}$$
Using (5.12), we obtain
$$f(y_r, y_{n-r}) = \frac{1}{2} y_r^T \Lambda_r y_r + g_r^T y_r + g_{n-r}^T y_{n-r} + c. \tag{5.15}$$

The gradient is

$$\nabla f(y) = \begin{pmatrix} \Lambda_r y_r + g_r \\ g_{n-r} \end{pmatrix}.$$

If $g_{n-r} \neq 0$, the gradient is different from zero for any value of y, and the necessary optimality condition is never verified. Now, if $g_{n-r} = 0$, the variables y_{n-r} do not affect the objective function. We fix y_{n-r} to any arbitrary value and solve the problem for y_r:

$$\min f(y_r) = \frac{1}{2} y_r^T \Lambda_r y_r + g_r^T y_r. \tag{5.16}$$

As Λ_r is positive definite, the first result of this theorem applies, and

$$y_r^* = -\Lambda_r^{-1} g_r. \tag{5.17}$$

We obtain (5.13) using (5.12).

□

The last result of Theorem 5.10 has a geometric interpretation. The Schur decomposition actually identifies one subspace where the quadratic function is strictly convex (the subspace corresponding to the positive eigenvalues), and one subspace where the function is linear (the subspace corresponding to zero eigenvalues). In this latter subspace, in order to guarantee that the function is bounded, the linear part must be constant, which corresponds to the condition $g_{n-r} = 0$ (see Example 5.2).

5.3 Exercises

For the following optimization problems :

1. Calculate the gradient and the Hessian of the objective function.
2. Identify the critical points.
3. Eliminate those that do not satisfy the necessary optimality conditions.
4. Identify those that satisfy the sufficient optimality conditions.

Exercise 5.1. $\min_{x \in \mathbb{R}^2} x_1^2 + x_2^2$.

Exercise 5.2. $\min_{x \in \mathbb{R}^2} \frac{1}{3} x_1^3 + x_2^3 - x_1 - x_2$.

Exercise 5.3. $\min_{x \in \mathbb{R}} x^2 + \frac{1}{x-2}$.

Exercise 5.4. $\min_{x \in \mathbb{R}^2} x_1^6 - 3x_1^4 x_2^2 + 3x_1^2 x_2^4 - x_2^6$.

Exercise 5.5. $\min_{x \in \mathbb{R}^2} f(x)$, where f is defined by one of the functions of Exercise 2.2.

Chapter 6

Constrained optimization

Contents

The development of optimality conditions in the presence of constraints is based on the same intuition as in the unconstrained case: it is impossible to descend from a minimum. However, we can no longer apply the optimality conditions described in Chapter 5, as illustrated by the following example.

Example 6.1 (Necessary optimality condition without constraint)**.** Consider the problem

$$\min f(x) = x^2$$

subject to

$$x \geq 1\,.$$

The solution to the problem is $x = 1$. And yet, $f'(1) = 2 \neq 0$.

Here, instead of verifying that no direction is a descent direction, we must only take into account the feasible directions and, if there is none, the feasible directions at the limit (see Definition 3.21 and the discussions of Section 3.3). Theorem 6.2 expresses that if x^* is a local minimum, no feasible direction at the limit is a descent direction.

Theorem 6.2 (Necessary optimality conditions for general constraints). *Let x^* be a local minimum for the optimization problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$, $g(x) \leq 0$ and $x \in X$ defined in Section 1.4. Here,*

$$\nabla f(x^*)^T d \geq 0 \tag{6.1}$$

for any feasible direction d at the limit in x^*.

Proof. We assume by contradiction that there exists a feasible direction at the limit d such that $\nabla f(x^*)^T d < 0$ and let us consider a feasible (sub-)sequence $(x_k)_{k\in\mathbb{N}}$ of Definition 3.21. According to Taylor's theorem (Theorem C.1) with $d = x_k - x^*$, we have

$$f(x_k) = f(x^*) + (x_k - x^*)^T \nabla f(x^*) + o(\|x_k - x^*\|) . \tag{6.2}$$

Since $d = \lim_k (x_k - x^*)$, and $\nabla f(x^*)^T d < 0$, there exists an index K such that $(x_k - x^*)^T \nabla f(x^*) < 0$ for all $k \geq K$. In addition, the term $o(\|x_k - x^*\|)$ can be made as small as desired by making k sufficiently large (see Theorem 2.11 or Theorem 5.1 for a more formal analysis of this result). Therefore, there exists an index k large enough that $f(x_k) < f(x^*)$, which contradicts the local optimality of x^*. □

This general result does not take into account a possible structure in the constraints. We now propose optimality conditions for specific problems.

6.1 Convex constraints

We now consider the optimization problem $\min f(x)$ subject to $x \in X$, where X is a closed non empty convex set. We obtain a specific version of Theorem 6.2.

Theorem 6.3 (Necessary optimality conditions for convex constraints). *Let x^* be a local minimum to the optimization problem*

$$\min_{x\in X} f(x) ,$$

where $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is differentiable at X and X is a non empty convex set. Then, $\forall x \in X$,

$$\nabla f(x^*)^T (x - x^*) \geq 0 . \tag{6.3}$$

Proof. We assume by contradiction that (6.3) is not satisfied. In this case, according to Definition 2.10, the direction $d = x - x^*$ is a descent direction. According to Theorem 2.11, there exists $\eta > 0$ such that

$$f(x^* + \alpha d) < f(x^*) , \qquad \forall \alpha \in [0, \eta] . \tag{6.4}$$

Moreover, according to Theorem 3.11, d is a feasible direction and $x^* + \alpha d$ is feasible for any $0 < \alpha \leq 1$. Then, for each $0 < \alpha \leq \min(\eta, 1)$, we have $x^* + \alpha d \in X$ and $f(x^* + \alpha d) < f(x^*)$. This contradicts the local optimality of x^*. □

The condition (6.3) signifies geometrically that any feasible direction should form an acute angle with the gradient, as illustrated in Figure 6.1. When the convex set has a particular structure, the necessary optimality conditions can be simplified, as shown in Example 6.4.

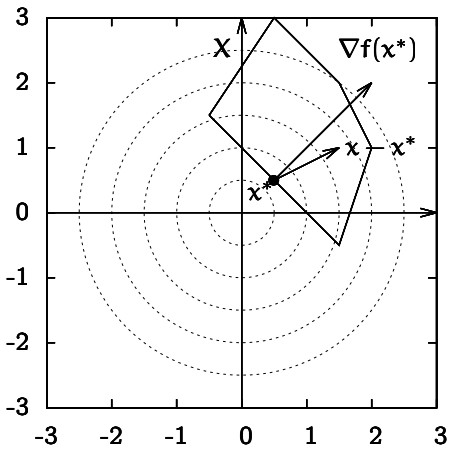


Figure 6.1: Illustration of the necessary optimality condition

Example 6.4 (Bound constraints). Consider the optimization problem

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

with

$$X = \{x \mid \ell_i \leq x_i \leq u_i\,,\ i = 1, \ldots, n\}\,, \tag{6.5}$$

where $\ell_i \neq u_i$, for any $i = 1, \ldots, n$. Let x^* be a local minimum of this problem. Since the condition (6.3) should be satisfied for all $x \in \mathbb{R}^n$, we select some specific values to derive necessary conditions. Each time, we select an arbitrary index i and choose $x \in \mathbb{R}^n$ such that $x_j = x_j^*$ for all $j \neq i$. For such x, the condition (6.3) simplifies to

$$\frac{\partial f(x^*)}{\partial x_i}\,(x_i - x_i^*) \geq 0\,. \tag{6.6}$$

We now need to specify x_i and verify that $\ell_i \leq x_i \leq u_i$ in order to obtain a feasible point and apply the necessary optimality condition. We consider three cases.

1. $x_i^* = \ell_i$. If we choose

$$x_i = \ell_i + \frac{u_i - \ell_i}{2} = \frac{u_i + \ell_i}{2}\,,$$

x_i is located exactly halfway between ℓ_i and u_i, and x is feasible. Moreover, $x_i - x_i^* = (u_i - \ell_i)/2 > 0$ because $u_i > \ell_i$. The condition (6.6) implies that

$$\frac{\partial f(x^*)}{\partial x_i} \geq 0 .$$

2. $x_i^* = u_i$. If we choose

$$x_i = u_i - \frac{u_i - \ell_i}{2} = \frac{u_i + \ell_i}{2},$$

x is feasible. Moreover, $x_i - x_i^* = -(u_i - \ell_i)/2 < 0$. The condition (6.6) implies that

$$\frac{\partial f(x^*)}{\partial x_i} \leq 0 .$$

3. $\ell_i < x_i^* < u_i$. If we choose

$$x_i = \frac{u_i + x_i^*}{2},$$

x is feasible. Moreover, $x_i - x_i^* = (u_i - x_i^*)/2 > 0$ because $x_i^* < u_i$. The condition (6.6) implies that

$$\frac{\partial f(x^*)}{\partial x_i} \geq 0 .$$

If we choose

$$x_i = \frac{\ell_i + x_i^*}{2},$$

x is feasible. Moreover, $x_i - x_i^* = (\ell_i - x_i^*)/2 < 0$ because $\ell_i < x_i^*$. The condition (6.6) implies that

$$\frac{\partial f(x^*)}{\partial x_i} \leq 0 .$$

By combining these two results, we get

$$\frac{\partial f(x^*)}{\partial x_i} = 0 .$$

Then, in the case of bound constraints defined by (6.5), the necessary optimality conditions can be written as

$$\begin{aligned}
&\frac{\partial f(x^*)}{\partial x_i} \geq 0, && \text{if } x_i^* = \ell_i \\
&\frac{\partial f(x^*)}{\partial x_i} \leq 0, && \text{if } x_i^* = u_i \\
&\frac{\partial f(x^*)}{\partial x_i} = 0, && \text{if } \ell_i < x_i^* < u_i
\end{aligned}$$

for any i such that $\ell_i < u_i$. Finally, let us note that, in the case where $\ell_i = u_i$, each feasible x is such that $x_i = \ell_i = u_i = x_i^*$ and the condition (6.6) is trivially satisfied, regardless of the value of $\partial f(x^*)/\partial x_i$.

Figure 6.2(b) illustrates the problem

$$\min f(x) = x_1^2 + x_2^2$$

subject to

$$0.7 \leq x_1 \leq 2$$
$$-1 \leq x_2 \leq 1\,.$$

The solution is $x^* = \begin{pmatrix} 0.7 & 0 \end{pmatrix}^T$ and $\nabla f(x^*) = \begin{pmatrix} 1.4 & 0 \end{pmatrix}^T$. Since $x_1^* = \ell_1 = 0.7$, we have $\partial f(x^*)/\partial x_1 \geq 0$. Since $\ell_2 < x_2^* < u_2$, we have $\partial f(x^*)/\partial x_2 = 0$.

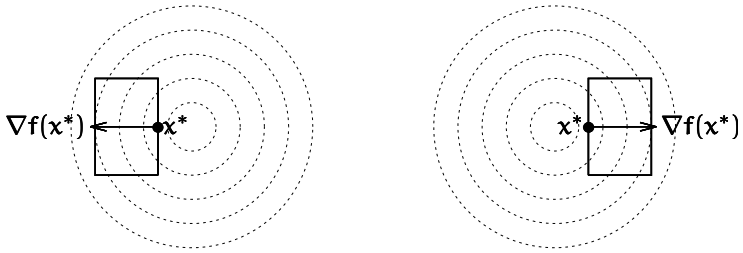
Figure 6.2(a) illustrates the problem

$$\min f(x) = x_1^2 + x_2^2$$

subject to

$$-2 \leq x_1 \leq -0.7$$
$$-1 \leq x_2 \leq 1\,.$$

The solution is $x^* = \begin{pmatrix} -0.7 & 0 \end{pmatrix}^T$ and $\nabla f(x^*) = \begin{pmatrix} -1.4 & 0 \end{pmatrix}^T$. Since $x_1^* = u_1 = -0.7$, we have $\partial f(x^*)/\partial x_1 \leq 0$. Since $\ell_2 < x_2^* < u_2$, we have $\partial f(x^*)/\partial x_2 = 0$.



(a) Upper bound active (b) Lower bound active

Figure 6.2: Illustration of the necessary optimality condition for bound constraints

Theorem 6.5 (Sufficient optimality conditions for convex constraints – I). *Consider the optimization problem*

$$\min_{x \in X} f(x)\,,$$

where X *is a closed non empty convex set, and* $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ *is differentiable and convex at* X*. Then, (6.3) is a sufficient condition for* x^* *to be a global minimum of* f *in* X*.*

Proof. According to Theorem 2.16, we have

$$f(x) - f(x^*) \geq (x - x^*)^T \nabla f(x^*), \qquad \forall x \in X.$$

If (6.3) is satisfied, then

$$f(x) - f(x^*) \geq 0, \qquad \forall x \in X,$$

and x^* is a global minimum (Definition 1.7) □

Example 6.6 (Projection on a convex set)**.** Let X be a closed non empty convex set for $\mathbb{R}^n$ and let us take $z \in \mathbb{R}^n$. The projection of z over X, denoted by $[z]^P$, is defined as the unique solution of the following optimization problem:

$$\min_x f(x) = \frac{1}{2}(x - z)^T(x - z) \qquad \text{subject to} \quad x \in X.$$

Since f is convex and $\nabla f(x) = x - z$, a necessary and sufficient condition for x^* to be the projection on z over X is

$$(x^* - z)^T(x - x^*) \geq 0, \qquad \forall x \in X. \tag{6.7}$$

Note that if $z \in X$, then (6.7) implies that $x^* = z$.

It is interesting to characterize the optimality condition (6.3) by using the projection operator.

Theorem 6.7 (Optimality conditions for convex constraints – II)**.** *Consider the optimization problem*

$$\min_{x \in X} f(x),$$

where X is a closed non empty convex set and $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is differentiable. If x^* is a local minimum, then

$$x^* = \left[x^* - \alpha \nabla f(x^*)\right]^P, \qquad \forall \alpha > 0. \tag{6.8}$$

If, moreover, f is convex, (6.8) is sufficient for x^* to optimize f over X.

Proof. Consider $z(\alpha) = x^* - \alpha \nabla f(x^*)$. According to (6.7), we have $\left[z(\alpha)\right]^P = x^*$ for all $\alpha > 0$ if and only if

$$\left(x^* - z(\alpha)\right)^T (x - x^*) \geq 0, \qquad \forall x \in X, \ \forall \alpha > 0,$$

or

$$\left(x^* - x^* + \alpha \nabla f(x^*)\right)^T (x - x^*) \geq 0, \qquad \forall x \in X, \ \forall \alpha > 0.$$

The latter equation is equivalent to the optimality condition (6.3). □

6.2 Lagrange multipliers: necessary conditions

Theorem 6.2 is based on the notion of feasible directions at the limit, which form the tangent cone (Definition 3.22). As we discussed in the last part of Section 3.3, this notion is too complex and the linearized cone (Definition 3.23) is much easier to handle. The most common cases, where the linearized cone is equivalent to the tangent cone, are optimization problems with linear constraints (Theorem 3.27) and linearly independent constraints (Theorem 3.28). Therefore, we present the results for only these cases. It is also possible to develop optimality conditions by considering the linearized cone from a general point of view. The details are described in Mangasarian (1979) and Nocedal and Wright (1999). The necessary optimality conditions are generally called *Karush-Kuhn-Tucker conditions* or KKT conditions. In fact, for many years, they were called *Kuhn-Tucker conditions*, following the article by Kuhn and Tucker (1951). It later turned out that Karush (1939) had already formulated them independently. John (1948) proposed a generalization a decade later (Theorem 6.12).

Note that the theory of Lagrange multipliers extends beyond the optimality conditions presented in this book and that they can also be adapted to non differentiable optimization. We refer the interested reader to Bertsekas (1982) and Rockafellar (1993).

In this text, we adopt the approach of Bertsekas (1999), who first presents these conditions in the case of linear constraints, and then for problems including linearly independent equality constraints. The proof provides intuitions that are reused in the development of algorithms. Subsequently, we generalize the result for problems that also include inequality constraints.

6.2.1 Linear constraints

Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{6.9}$$

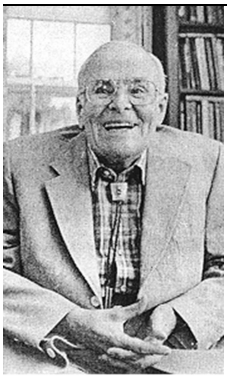
subject to

$$Ax = b \tag{6.10}$$

with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. According to Theorem 3.6, the matrix A can be considered of full rank without loss of generality. In this case, the Karush-Kuhn-Tucker conditions are formulated in the following manner.

Theorem 6.8 (Karush-Kuhn-Tucker: linear case)**.** *Let x^* be a local minimum of the problem $\min_{x \in \mathbb{R}^n} f(x)$ subject to $Ax = b$, where $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and $A \in \mathbb{R}^{m \times n}$ is of full rank. There thus exists a single vector $\lambda^* \in \mathbb{R}^m$ such that*

$$\nabla_x L(x^*, \lambda^*) = \nabla f(x^*) + A^T \lambda^* = 0\,, \tag{6.11}$$



Albert William Tucker was born on November 25, 1905 in Ontario, Canada, and died in Hightstown, New Jersey on January 25, 1995. He received his PhD in 1932 from Princeton University, where he spent most of his career. He is particularly known for his work on linear programming and game theory. After Dantzig visited von Neumann in 1948, Tucker drove Dantzig back to the train station in Princeton. It was during this short ride that Dantzig exposed linear programming to Tucker. In the context of game theory, von Neumann is generally cited as the inventor of the duality theorem, and Tucker, Kuhn and Wales as the first to propose a rigorous proof. In 1950, Tucker proposed the example of the prisoner's dilemma to illustrate the difficulty of non zero-sum games. John Nash, Nobel Laureate in Economics 1994, was one of his students at Princeton. The story goes that von Neumann did not agree with the approach of Nash's game theory, while Tucker encouraged him to develop his ideas and supervised his doctoral thesis.

Figure 6.3: Albert William Tucker

where L *is the Lagrangian function (Definition 4.3). If* f *is twice differentiable, then*

$$y^T \nabla^2_{xx} L(x^*, \lambda^*) y \geq 0\,, \quad \forall y \in \mathbb{R}^n \text{ such that } Ay = 0\,. \tag{6.12}$$

Proof. We employ the technique for the elimination of constraints described in Section 3.4 to convert the optimization problem with constraints into an optimization problem without constraint (3.70). To simplify the proof, we can assume that the variables are arranged in a way that the m variables to eliminate are the m first ones. Then, $P = I$ in (3.70) and the minimization problem is

$$\min_{x_N \in \mathbb{R}^{n-m}} g(x_N) = f\begin{pmatrix} B^{-1}(b - Nx_N) \\ x_N \end{pmatrix}. \tag{6.13}$$

If $x^* = (x^*_B, x^*_N)$ is a local minimum of the problem with constraints, then x^*_N is a local minimum of the problem without constraints and the necessary condition (5.1) applies to (6.13). By using chain rule differentiation (see (C.6) of Theorem C.3) we obtain:

$$\nabla g(x^*_N) = -N^T B^{-T} \nabla_B f(x^*) + \nabla_N f(x^*) = 0\,, \tag{6.14}$$

where $\nabla_B f(x^*)$ and $\nabla_N f(x^*)$ represent the gradient of f with regard to the variables x_B and x_N, respectively. If we define

$$\lambda^* = -B^{-T} \nabla_B f(x^*)\,, \tag{6.15}$$

which can also be written as

$$\nabla_B f(x^*) + B^T \lambda^* = 0\,, \tag{6.16}$$

the condition (6.14) is expressed as

$$\nabla_N f(x^*) + N^T \lambda^* = 0\,. \tag{6.17}$$

Equations (6.16) and (6.17) form (6.11), which proves the first-order result.

To demonstrate the second-order result, we first derive (6.11) to obtain

$$\nabla^2_{xx} L(x,\lambda) = \nabla^2 f(x)\,. \tag{6.18}$$

We consider a vector $y \in \mathbb{R}^n$ such that $Ay = 0$ and then $By_B + Ny_N = 0$ or $y_B = -B^{-1}Ny_N$. Then, if $d \in \mathbb{R}^{m-n}$ is an arbitrary vector, the vector

$$y = \begin{pmatrix} y_B \\ y_N \end{pmatrix} = \begin{pmatrix} -B^{-1}Nd \\ d \end{pmatrix} \tag{6.19}$$

is such that $Ay = 0$. According to the necessary optimality conditions (5.2) in the unconstrained case and Definition B.8 of a positive semidefinite matrix, we have that

$$d^T \nabla^2 g(x^*_N) d \geq 0\,, \qquad \forall d \in \mathbb{R}^{m-n}\,. \tag{6.20}$$

However, when deriving (6.14), we obtain

$$\begin{aligned} \nabla^2 g(x^*_N) = {} & N^T B^{-T} \nabla^2_{BB} f(x^*) B^{-1} N \\ & - N^T B^{-T} \nabla^2_{BN} f(x^*) \\ & - \nabla^2_{NB} f(x^*) B^{-1} N \\ & + \nabla^2_{NN} f(x^*)\,, \end{aligned} \tag{6.21}$$

where $\nabla^2 f(x^*)$ is decomposed into

$$\nabla^2 f(x^*) = \begin{pmatrix} \nabla^2_{BB} f(x^*) & \nabla^2_{BN} f(x^*) \\ \nabla^2_{NB} f(x^*) & \nabla^2_{NN} f(x^*) \end{pmatrix}. \tag{6.22}$$

Then,

$$\begin{aligned} d^T \nabla^2 g(x^*_N) d = {} & d^T N^T B^{-T} \nabla^2_{BB} f(x^*) B^{-1} N d \\ & - d^T N^T B^{-T} \nabla^2_{BN} f(x^*) d \\ & - d^T \nabla^2_{NB} f(x^*) B^{-1} N d \\ & + d^T \nabla^2_{NN} f(x^*) d && \text{from (6.21)} \\ = {} & y_B^T \nabla^2_{BB} f(x^*) y_B \\ & + y_B^T \nabla^2_{BN} f(x^*) y_N \\ & + y_N^T \nabla^2_{NB} f(x^*) y_B \\ & + y_N^T \nabla^2_{NN} f(x^*) y_N && \text{from (6.19)} \\ = {} & y^T \nabla^2 f(x^*) y && \text{from (6.22)} \\ = {} & y^T \nabla^2_{xx} L(x^*) y && \text{from (6.18)} \end{aligned}$$

and (6.12) is equivalent to (6.20). □

Example 6.9 (Karush-Kuhn-Tucker: linear case)**.** We consider the following optimization problem:

$$\min f(x_1, x_2, x_3, x_4) = x_1^2 + x_2^2 + x_3^2 + x_4^2$$

subject to

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 - x_2 + x_4 &= 1\,. \end{aligned}$$

The solution to this problem is

$$x^* = \begin{pmatrix} 2/3 \\ 0 \\ 1/3 \\ 1/3 \end{pmatrix} \quad \text{and} \quad \nabla f(x^*) = \begin{pmatrix} 4/3 \\ 0 \\ 2/3 \\ 2/3 \end{pmatrix}.$$

By decomposing

$$A = \begin{pmatrix} B & N \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{array}\right),$$

we obtain

$$\lambda^* = -B^{-T}\nabla_B f(x^*) = -\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{pmatrix}\begin{pmatrix} 4/3 \\ 0 \end{pmatrix} = \begin{pmatrix} -2/3 \\ -2/3 \end{pmatrix},$$

and (6.11) is expressed as

$$\begin{pmatrix} 4/3 \\ 0 \\ 2/3 \\ 2/3 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} -2/3 \\ -2/3 \end{pmatrix} = \begin{pmatrix} 4/3 \\ 0 \\ 2/3 \\ 2/3 \end{pmatrix} + \begin{pmatrix} -4/3 \\ 0 \\ -2/3 \\ -2/3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Any vector of the form

$$y = \begin{pmatrix} -\frac{1}{2}y_3 - \frac{1}{2}y_4 \\ -\frac{1}{2}y_3 + \frac{1}{2}y_4 \\ y_3 \\ y_4 \end{pmatrix}$$

is such that $Ay = 0$ and (6.12) is written as

$$\begin{aligned} y^T\nabla^2 f(x^*)y &= \begin{pmatrix} -\frac{1}{2}y_3 - \frac{1}{2}y_4 \\ -\frac{1}{2}y_3 + \frac{1}{2}y_4 \\ y_3 \\ y_4 \end{pmatrix}^T \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} -\frac{1}{2}y_3 - \frac{1}{2}y_4 \\ -\frac{1}{2}y_3 + \frac{1}{2}y_4 \\ y_3 \\ y_4 \end{pmatrix} \\ &= 3y_3^2 + 3y_4^2 \geq 0\,. \end{aligned}$$

6.2.2 Equality constraints

We consider here the problem with equality constraints (1.71)–(1.72). In this case, the Karush-Kuhn-Tucker conditions are formulated in the following way. Thanks to the Lagrangian function (Definition 4.3), their expression presents similarities with conditions without constraint.

Theorem 6.10 (Karush-Kuhn-Tucker: equality constraints). *Let x^* be a local minimum of the problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$, with $f : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ continuously differentiable. If the constraints are linearly independent in x^* (in the sense of Definition 3.8), there exists a unique vector $\lambda^* \in \mathbb{R}^m$ such that*

$$\nabla L(x^*, \lambda^*) = 0\,, \tag{6.23}$$

where L is the Lagrangian function (Definition 4.3). If f and h are twice differentiable, then

$$y^T \nabla^2_{xx} L(x^*, \lambda^*) y \geq 0\,, \qquad \forall y \in \mathcal{D}(x^*)\,, \tag{6.24}$$

where $\mathcal{D}(x^*)$ is the linearized cone[1] in x^* (Definition 3.23). Moreover,

$$\lambda^* = -\left(\nabla h(x^*)^T \nabla h(x^*)\right)^{-1} \nabla h(x^*)^T \nabla f(x^*)\,. \tag{6.25}$$

Proof. We generate a sequence of optimization problems without constraint approaching the original problem. The idea is to penalize the violation of constraints by introducing a penalty term. The objective function of the problem without constraints is defined by

$$F_k(x) = f(x) + \frac{k}{2}\left\|h(x)\right\|^2 + \frac{\alpha}{2}\left\|x - x^*\right\|^2, \tag{6.26}$$

where $k \in \mathbb{N}$, x^* is a local minimum to the problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$ and $\alpha > 0$ is arbitrary. Since x^* is a local minimum (Definition 1.5), there exists ε such that

$$f(x^*) \leq f(x)\,, \quad \forall x \text{ such that } h(x) = 0 \text{ and } x \in S_\varepsilon\,, \tag{6.27}$$

where S_ε is the sphere defined by $S_\varepsilon = \left\{x \middle| \|x - x^*\| \leq \varepsilon\right\}$. According to the Weierstrass theorem (Theorem 1.14), the problem

$$\min F_k(x) \tag{6.28}$$

subject to

$$x \in S_\varepsilon \tag{6.29}$$

has a solution in S_ε, denoted by x_k. One should keep in mind that the problem (6.28)–(6.29) is subject to constraints. Nevertheless, we demonstrate that, for a sufficiently

[1] Since the constraints are linearly independent, the constraint qualification is satisfied and the linearized cone corresponds to the tangent cone.

large k, the solution lies strictly inside S_ε and is therefore a solution to the problem (6.28) without constraint (according to Theorem 3.5). The role of S_ε is to ensure that no local minima other than x^* are found.

We have

$$F_k(x_k) \leq f(x^*)\,. \tag{6.30}$$

Indeed, $F_k(x_k) \leq F_k(x^*)$ because x_k is the solution to (6.28)-(6.29) and $x^* \in S_\varepsilon$. Also, according to (6.26),

$$F_k(x^*) = f(x^*) + \frac{k}{2}\left\|h(x^*)\right\|^2 + \frac{\alpha}{2}\left\|x^* - x^*\right\|^2 = f(x^*),$$

because $h(x^*) = 0$. Then, when $k \to \infty$, the value of $F_k(x_k)$ remains bounded. We show by contradiction that this implies that

$$\lim_{k\to\infty} \left\|h(x_k)\right\| = 0\,. \tag{6.31}$$

Indeed, if (6.31) is not satisfied, then the term $\frac{k}{2}\left\|h(x_k)\right\|^2$ tends towards $+\infty$. Since $F(x_k)$ remains bounded, this signifies that either $f(x_k)$, or $\|x_k - x^*\|^2$ tends towards $-\infty$. However $f(x_k)$ is bounded from below by $f(x^*)$ over S_ε (according to (6.27)) and $\|x_k - x^*\|^2$ is positive, which leads to a contradiction and proves (6.31).

Let $\widehat{x}$ be a limit point of the sequence $(x_k)_k$ (Definition B.20). According to (6.31), we have $h(\widehat{x}) = 0$ and $\widehat{x}$ is feasible for the original problem and thus $f(x^*) \leq f(\widehat{x})$. Moreover, according to (6.30)

$$\lim_{k\to\infty} F_k(x_k) = f(\widehat{x}) + \alpha\left\|\widehat{x} - x^*\right\|^2 \leq f(x^*)\,. \tag{6.32}$$

Then,

$$f(\widehat{x}) + \alpha\left\|\widehat{x} - x^*\right\|^2 \leq f(\widehat{x})\,. \tag{6.33}$$

As a result, $\alpha\left\|\widehat{x}-x^*\right\|^2 = 0$ and $\widehat{x} = x^*$. The sequence $\left(x_k\right)_k$ converges to x^*. According to Definition B.19, there exists $\widehat{k}$ such that

$$\|x_k - x^*\| \leq 0.9\,\varepsilon < \varepsilon\,, \qquad \forall k \geq \widehat{k}\,, \tag{6.34}$$

where ε is the radius of the sphere S_ε involved in the definition of the local minimum (6.27). The point x_k is inside S_ε when k is sufficiently large.

According to Theorem 1.16, x_k is a local minimum of the unconstrained problem (6.28). We can apply the necessary optimality conditions of an unconstrained problem, given by Theorem 5.1:

$$\nabla F_k(x_k) = \nabla f(x_k) + k\nabla h(x_k)h(x_k) + \alpha(x_k - x^*) = 0 \tag{6.35}$$

and $\nabla^2 F_k(x_k)$ is positive semidefinite, with

$$\nabla^2 F_k(x_k) = \nabla^2 f(x_k) + k\sum_{i=1}^{m} h_i(x_k)\nabla^2 h_i(x_k) + k\nabla h(x_k)\nabla h(x_k)^T + \alpha I\,. \tag{6.36}$$

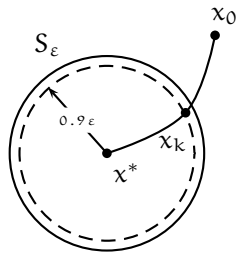


Figure 6.4: Illustration of the proof of Theorem 6.10

By multiplying (6.35) with $\nabla h(x_k)^T$, we get

$$\nabla h(x_k)^T \nabla f(x_k) + k \nabla h(x_k)^T \nabla h(x_k) h(x_k) + \alpha \nabla h(x_k)^T (x_k - x^*) = 0\,. \tag{6.37}$$

Since the constraints are linearly independent by hypothesis, the matrix $\nabla h(x^*)^T \nabla h(x^*)$ is of full rank and invertible. By continuity of ∇h (indeed, h is continuously differentiable), there exists a k sufficiently large such that $\nabla h(x_k)^T \nabla h(x_k)$ is also invertible. By multiplying (6.37) by $\left(\nabla h(x_k)^T \nabla h(x_k)\right)^{-1}$, we obtain

$$k h(x_k) = -\left(\nabla h(x_k)^T \nabla h(x_k)\right)^{-1} \nabla h(x_k)^T \left(\nabla f(x_k) + \alpha(x_k - x^*)\right). \tag{6.38}$$

When $k \to \infty$, we define

$$\lambda^* = \lim_{k \to \infty} k h(x_k) \tag{6.39}$$

to obtain (6.25). By letting $k \to \infty$ in (6.35), we get

$$\nabla f(x^*) + \nabla h(x^*) \lambda^* = 0\,. \tag{6.40}$$

According to Definition 4.3, (6.40) is equivalent to $\nabla_x L(x^*, \lambda^*) = 0$. Since $0 = h(x^*) = \nabla_\lambda L(x^*, \lambda^*)$, we get (6.23).

To demonstrate the second-order condition (6.24), let us consider y in the linearized cone (and thus $\nabla h(x^*)^T y = 0$) and, for a sufficiently large k, let us consider its projection y_k on the kernel of $\nabla h(x_k)^T$. According to the theorem of projection on the kernel of a matrix (Theorem C.7), we have

$$y_k = y - \nabla h(x_k) \left(\nabla h(x_k)^T \nabla h(x_k)\right)^{-1} \nabla h(x_k)^T y\,. \tag{6.41}$$

We also have

$$\lim_{k \to \infty} y_k = y - \nabla h(x^*) \left(\nabla h(x^*)^T \nabla h(x^*)\right)^{-1} \nabla h(x^*)^T y = y\,, \tag{6.42}$$

because $\nabla h(x^*)^T y = 0$. Then, from (6.36),

$$\begin{aligned} y_k^T \nabla^2 F(x_k) y_k = \; & y_k^T \left(\nabla^2 f(x_k) + k \sum_{i=1}^{m} h_i(x_k) \nabla^2 h_i(x_k) \right) y_k \\ & + k y_k^T \nabla h(x_k) \nabla h(x_k)^T y_k + \alpha y_k^T y_k. \end{aligned}$$

As $\nabla h(x_k)^T y_k = 0$,

$$y_k^T \nabla^2 F(x_k) y_k = y_k^T \left(\nabla^2 f(x_k) + k \sum_{i=1}^{m} h_i(x_k) \nabla^2 h_i(x_k) \right) y_k + \alpha y_k^T y_k.$$

As $\nabla^2 F_k(x_k)$ is positive semidefinite, this last quantity is non negative. By letting $k \to \infty$ and using $\lambda^* = \lim_{k\to\infty} k h(x_k)$, we get

$$y^T \left(\nabla^2 f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 h_i(x^*) \right) y + \alpha y^T y \geq 0\,. \tag{6.43}$$

According to Definition 4.3 of the Lagrangian function, (6.43) is equivalent to

$$y^T \nabla^2_{xx} L(x^*, \lambda^*) y + \alpha y^T y \geq 0\,. \tag{6.44}$$

If (6.24) is not satisfied, (6.44) is not valid for all $\alpha > 0$. Indeed, if $y^T \nabla^2_{xx} L(x^*, \lambda^*) y < 0$, (6.44) is not satisfied for the values of α such that

$$\alpha < -\frac{y^T \nabla^2_{xx} L(x^*, \lambda^*) y}{y^T y}\,.$$

Since α can be arbitrarily chosen, this concludes the proof. □

Note that for linear constraints, $h(x) = Ax - b$, $\nabla h(x) = A^T$ and (6.23) is written as

$$\begin{pmatrix} \nabla f(x^*) + A^T \lambda^* \\ Ax - b \end{pmatrix} = 0\,,$$

which is equivalent to (6.11) of Theorem 6.8.

Example 6.11 (Karush-Kuhn-Tucker: equality constraints)**.** Consider the optimization problem

$$\min_{x \in \mathbb{R}^2} x_1 + x_2 \tag{6.45}$$

subject to

$$h(x) = \begin{pmatrix} x_1^2 + (x_2 - 1)^2 - 1 \\ -x_1^2 + x_2 \end{pmatrix} = 0\,. \tag{6.46}$$

The set of constraints is illustrated in Figure 6.5. We have

$$\nabla f(x) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \text{and} \qquad \nabla h(x) = \begin{pmatrix} 2x_1 & -2x_1 \\ 2x_2 - 2 & 1 \end{pmatrix}. \tag{6.47}$$

The Lagrangian function of the problem is

$$L(x, \lambda) = x_1 + x_2 + \lambda_1 \left(x_1^2 + (x_2 - 1)^2 - 1 \right) + \lambda_2 (-x_1^2 + x_2) \tag{6.48}$$

and

$$\nabla L(x, \lambda) = \begin{pmatrix} 1 + 2\lambda_1 x_1 - 2\lambda_2 x_1 \\ 1 + 2\lambda_1 (x_2 - 1) + \lambda_2 \\ x_1^2 + (x_2 - 1)^2 - 1 \\ -x_1^2 + x_2 \end{pmatrix}. \tag{6.49}$$

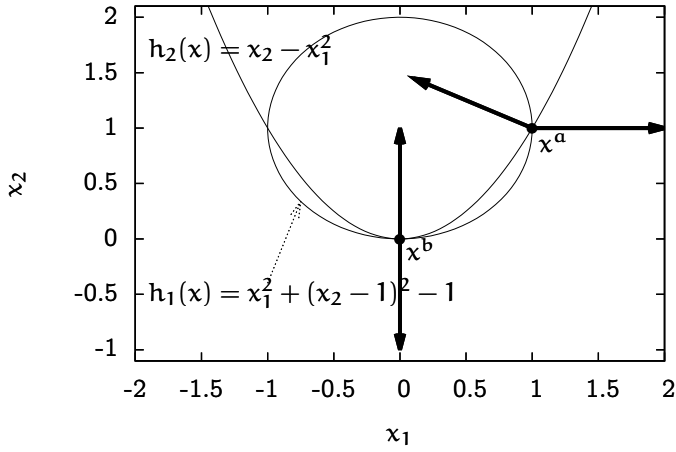


Figure 6.5: Illustration of the KKT conditions

The point $x^a = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$ is a local minimum of the problem. The constraints are linearly independent in x^a because the matrix

$$\nabla h(x^a) = \begin{pmatrix} 2 & -2 \\ 0 & 1 \end{pmatrix} \tag{6.50}$$

is of full rank. Thanks to the relations (6.25) and (6.47), we get

$$\lambda^* = \begin{pmatrix} -3/2 \\ -1 \end{pmatrix}. \tag{6.51}$$

The condition $\nabla L(x^a, \lambda^*) = 0$ is satisfied. Note that the linearized cone is empty in x^a and that the second-order condition is trivially satisfied.

The point $x^b = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ is also a local minimum (in fact, we are dealing with a global minimum of the problem). We have

$$\nabla L(x^b, \lambda) = \begin{pmatrix} 1 \\ 1 - 2\lambda_1 + \lambda_2 \\ 0 \\ 0 \end{pmatrix}, \tag{6.52}$$

which cannot be zero for any λ. The necessary condition is not satisfied in this case. Indeed, the constraints are not linearly independent in x^b because the matrix

$$\nabla h(x^b) = \begin{pmatrix} 0 & 0 \\ -2 & 1 \end{pmatrix} \tag{6.53}$$

is not of full rank.

We now present the result of John (1948) for equality constraints, which constitutes a generalization of Theorem 6.10.

Theorem 6.12 (Fritz John: equality constraints). *Let x^* be a local minimum of the problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x)=0$, where $f:\mathbb{R}^n\to\mathbb{R}$ and $h:\mathbb{R}^n\to\mathbb{R}^m$ are continuously differentiable. Then, there exists $\mu_0^*\in\mathbb{R}$ and a vector $\lambda^*\in\mathbb{R}^m$ such that*

$$\mu_0^*\nabla f(x^*)+\nabla h(x^*)\lambda^*=0 \tag{6.54}$$

and $\mu_0^*, \lambda_1^*,\dots,\lambda_m$, are not all zero.

Proof. In the case where the constraints are linearly independent, Theorem 6.10 applies and (6.54) is trivially obtained with $\mu_0^*=1$. In the case where the constraints are linearly dependent, then there exist $\lambda_1^*,\dots,\lambda_m^*$, not all zero, such that

$$\sum_{i=1}^{m}\lambda_i^*\nabla h_i(x^*)=0,$$

and (6.54) is obtained when $\mu_0=0$. □

6.2.3 Equality and inequality constraints

We now present the necessary Karush-Kuhn-Tucker optimality conditions for a general case including equality and inequality constraints. As is often the case, the approach consists in returning to an already studied case, in this case the problem with only equality constraints.

Theorem 6.13 (Karush-Kuhn-Tucker). *Let x^* be a local minimum of the problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x)=0$, $g(x)\le 0$, where $f:\mathbb{R}^n\to\mathbb{R}$, $h:\mathbb{R}^n\to\mathbb{R}^m$ and $g:\mathbb{R}^n\to\mathbb{R}^p$ are continuously differentiable. If the constraints are linearly independent in x^* (in the sense of Definition 3.8), there exists a unique vector $\lambda^*\in\mathbb{R}^m$ and a unique vector $\mu^*\in\mathbb{R}^p$ such that*

$$\nabla_x L(x^*,\lambda^*,\mu^*)=0, \tag{6.55}$$

$$\mu_j^*\ge 0, \qquad j=1,\dots,p, \tag{6.56}$$

and

$$\mu_j^* g_j(x^*)=0, \qquad j=1,\dots,p, \tag{6.57}$$

where L is the Lagrangian function (Definition 4.3). If f, h and g are twice differentiable, then

$$\begin{aligned} &y^T\nabla^2_{xx}L(x^*,\lambda^*,\mu^*)y\ge 0, \quad \forall y\ne 0 \text{ such that}\\ &\quad y^T\nabla h_i(x^*)=0, \quad i=1,\dots,m\\ &\quad y^T\nabla g_i(x^*)=0, \quad i=1,\dots,p \quad \text{such that } g_i(x^*)=0. \end{aligned} \tag{6.58}$$

Proof. We consider the active inequality constraints at the solution as equality constraints and let us ignore the other constraints in order to obtain the problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$, $g_i(x) = 0$, for any $i \in \mathcal{A}(x^*)$, where $\mathcal{A}(x^*)$ is the set of active constraints in x^* (Definition 3.4). According to Theorem 3.5 where $Y = \{x \mid h(x) = 0\}$, x^* is a local minimum of the optimization problem with equality constraints. According to Theorem 6.10, there exist Lagrange multipliers $\lambda^* \in \mathbb{R}^m$ and μ_i^*, with $i \in \mathcal{A}(x^*)$ such that

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* + \sum_{i\in\mathcal{A}(x^*)} \mu_i^* \nabla g_i(x^*) = 0\,. \tag{6.59}$$

We associate a zero multiplier to each non active inequality constraint in x^* to obtain

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* + \sum_{i=1}^{p} \mu_i^* \nabla g_i(x^*) = 0 \tag{6.60}$$

with $\mu_i^* = 0$ if $i \notin \mathcal{A}(x^*)$. We thus get (6.55). Similarly, the second-order condition of Theorem 6.10 implies that

$$\begin{aligned} &y^T \nabla^2_{xx} L(x^*, \lambda^*, \mu^*) y \geq 0\,, \quad \forall y \text{ such that} \\ &\quad y^T \nabla h_i(x^*) = 0\,, \quad i = 1, \ldots, m \\ &\quad y^T \nabla g_i(x^*) = 0\,, \quad i = 1, \ldots, p \quad \text{such that } g_i(x^*) = 0 \end{aligned} \tag{6.61}$$

and (6.58) is satisfied. We note that (6.57) is trivially satisfied. Indeed, if the constraint $g_j(x^*) \leq 0$ is active, we have $g_j(x^*) = 0$. If on the other hand it is not, we have $\mu_j^* = 0$. We now need only demonstrate (6.56).

We take the same proof as Theorem 6.10, by defining the penalty function for inequality constraints with

$$g_i^+(x) = \max\{0, g_i(x)\}\,, \qquad i = 1, \ldots, p\,. \tag{6.62}$$

In this case, the function (6.26) becomes

$$F_k(x) = f(x) + \frac{k}{2}\left\|h(x)\right\|^2 + \frac{k}{2}\sum_{i=1}^{p} g_j^+(x)^2 + \frac{\alpha}{2}\left\|x - x^*\right\|^2. \tag{6.63}$$

Since $g_j^+(x)^2$ is differentiable and

$$\nabla g_j^+(x)^2 = 2 g_j^+(x) \nabla g_j(x)\,, \tag{6.64}$$

we can invoke the same development as in the proof of Theorem 6.10. Since we have obtained (6.39), we have

$$\mu_i^* = \lim_{k\to\infty} k g_i^+(x_k)\,, \qquad i = 1, \ldots, p\,. \tag{6.65}$$

And since $g_i^+(x) \geq 0$, we get (6.56). □

Example 6.14 (Karush-Kuhn-Tucker: inequality constraints – I)**.** Consider the problem

$$\min_{x \in \mathbb{R}^2} x_1 + x_2 \tag{6.66}$$

subject to

$$\begin{aligned} (x_1 - 3)^2 + x_2^2 &\leq 9 \\ x_1^2 + (x_2 - 3)^2 &\leq 9 \\ x_1^2 &\leq 1 + x_2 \,, \end{aligned} \tag{6.67}$$

illustrated in Figure 6.6. Re-arranging the equations of the constraints, we obtain

$$\begin{aligned} g_1(x) &= x_1^2 - 6x_1 + x_2^2 \\ g_2(x) &= x_1^2 - 6x_2 + x_2^2 \\ g_3(x) &= x_1^2 - x_2 - 1 \,, \end{aligned} \tag{6.68}$$

and the Lagrangian function is written as

$$\begin{aligned} L(x, \mu) = x_1 + x_2 &+ \mu_1 (x_1^2 - 6x_1 + x_2^2) \\ &+ \mu_2 (x_1^2 - 6x_2 + x_2^2) + \mu_3 (x_1^2 - x_2 - 1) \,. \end{aligned} \tag{6.69}$$

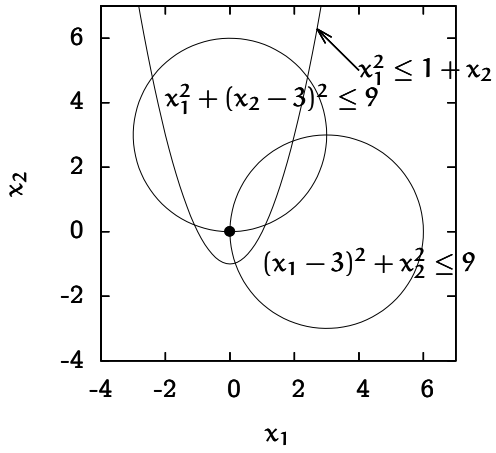


Figure 6.6: Karush-Kuhn-Tucker optimality conditions

The point $x^* = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ is a local minimum of this problem. The constraints $g_1(x) \leq 0$ and $g_2(x) \leq 0$ are active in x^*, whereas the constraint $g_3(x) \leq 0$ is not. The point x^* is also a local minimum of the problem where the two active inequality constraints are replaced by equality constraints, and the active constraint is ignored, that is

$$\min_{x \in \mathbb{R}^2} x_1 + x_2 \tag{6.70}$$

subject to

$$\begin{aligned} (x_1 - 3)^2 + x_2^2 &= 9 \\ x_1^2 + (x_2 - 3)^2 &= 9 \,, \end{aligned} \tag{6.71}$$

or, equivalently

$$\begin{aligned} h_1(x) &= x_1^2 - 6x_1 + x_2^2 = 0 \\ h_2(x) &= x_1^2 - 6x_2 + x_2^2 = 0\,. \end{aligned} \tag{6.72}$$

The gradient of the constraints is written as

$$\nabla h(x) = \begin{pmatrix} 2x_1 - 6 & 2x_1 \\ 2x_2 & 2x_2 - 6 \end{pmatrix}. \tag{6.73}$$

According to the KKT conditions (Theorem 6.10), λ^* is given by (6.25) and we have

$$\lambda^* = -\left(\begin{pmatrix} -6 & 0 \\ 0 & -6 \end{pmatrix}^T \begin{pmatrix} -6 & 0 \\ 0 & -6 \end{pmatrix}\right)^{-1} \begin{pmatrix} -6 & 0 \\ 0 & -6 \end{pmatrix}^T \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1/6 \\ 1/6 \end{pmatrix}.$$

Then, the necessary first-order KKT condition for the initial problem is written as

$$\nabla L_x(x^*, \mu^*) = \begin{pmatrix} 1 - 6\mu_1^* + 2(\mu_1^* + \mu_2^* + \mu_3^*)x_1^* \\ 1 - 6\mu_2^* + 2(\mu_1^* + \mu_2^*)x_2^* - \mu_3^* \end{pmatrix} = 0\,,$$

where L is defined by (6.69), $x^* = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ and

$$\mu^* = \begin{pmatrix} \lambda_1^* \\ \lambda_2^* \\ 0 \end{pmatrix} = \begin{pmatrix} 1/6 \\ 1/6 \\ 0 \end{pmatrix}.$$

Since the linearized cone is empty in x^*, the necessary second-order KKT condition is trivially satisfied.

Example 6.15 (Karush-Kuhn-Tucker: inequality constraints – II)**.** Consider the optimization problem

$$\min_{x \in \mathbb{R}^2} \frac{1}{2}(x_1^2 - x_2^2) \tag{6.74}$$

subject to

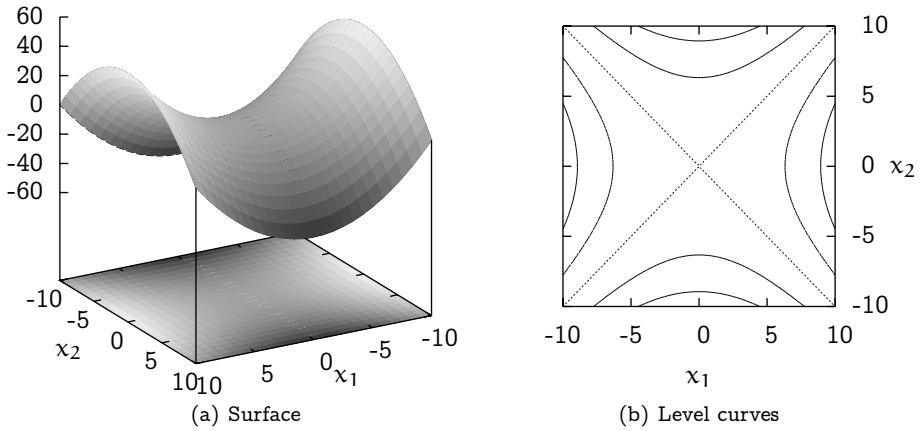
$$x_2 \leq 1\,, \tag{6.75}$$

for which the objective function is illustrated in Figure 6.7. We have

$$L(x, \mu) = \frac{1}{2}x_1^2 - \frac{1}{2}x_2^2 + \mu(x_2 - 1)\,.$$

The point $x^* = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$ is a local minimum of the problem. The constraint is active in this point. The first-order condition (6.55) is expressed as

$$\nabla_x L(x^*, \mu^*) = \begin{pmatrix} x_1^* \\ -x_2^* + \mu^* \end{pmatrix} = \begin{pmatrix} 0 \\ -1 + \mu^* \end{pmatrix} = 0$$



(a) Surface (b) Level curves

Figure 6.7: Objective function of Examples 6.15 and 6.21

and is satisfied with $\mu^* = 1$, which is positive. The second-order condition (6.58) is written as

$$\begin{pmatrix} y_1 & y_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = y_1^2 - y_2^2 \geq 0 \tag{6.76}$$

for any y such that

$$\nabla g(x^*)^T \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = y_2 = 0$$

and is satisfied. Note that if we choose a feasible direction y, for instance $y = \begin{pmatrix} 0 & -1 \end{pmatrix}^T$, the condition (6.76) is not satisfied. It is only valid for y orthogonal to active constraints.

Example 6.16 (Physical interpretation of KKT conditions)**.** We consider the optimization problem

$$\min_{x \in \mathbb{R}^2} x_1$$

subject to

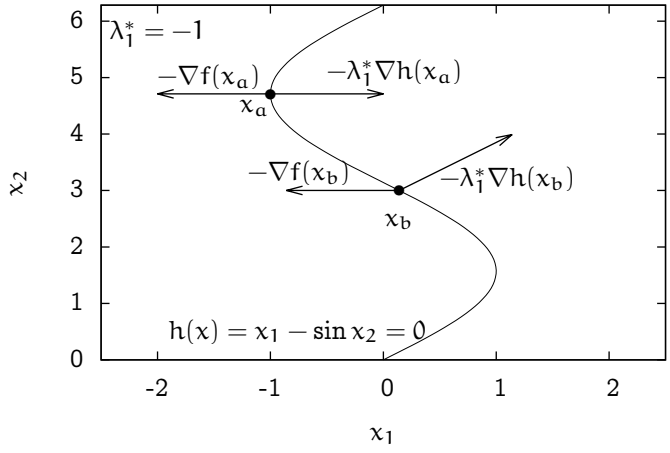
$$h_1(x) = x_1 - \sin x_2 = 0\,.$$

The equality constraint is represented in Figure 6.8(a). The point $x_a = \begin{pmatrix} -1 & 3\pi/2 \end{pmatrix}^T$ is a local minimum of the problem. We have

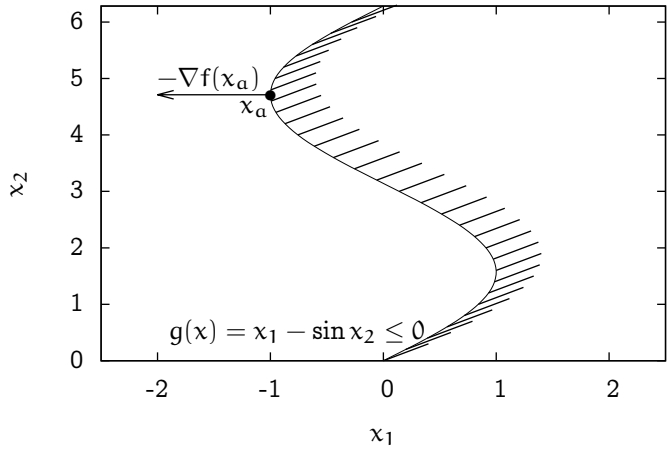
$$\nabla f(x) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad \nabla h(x) = \begin{pmatrix} 1 \\ -\cos x_2 \end{pmatrix}$$

and the necessary optimality condition is written as

$$-\nabla f(x_a) - \lambda_1^* \nabla h(x_a) = 0\,.$$



(a) Equality constraint



(b) Inequality constraint

Figure 6.8: Interpretation of the KKT constraints

Using the fact that $-\nabla f(x_a)$ is the direction with the steepest descent (Theorem 2.13), we can interpret this condition as an equilibrium between the "force" $-\nabla f(x_a)$, which drives the solution to lower values of the objective function, and the force $-\lambda_1 \nabla h(x_a)$, which maintains the solution on the constraint. If x_a is optimal, this signifies that the forces are balanced and that their sum is zero. In our example, since the "force" $-\nabla f(x_a)$ acts in the same direction as $-\nabla h(x_a)$, the multiplier λ_1^* should be negative so that the two "forces" can compensate each other.

If we now take the point $x_b = \begin{pmatrix} \sin(3) & 3 \end{pmatrix}^T$, the "forces" $-\nabla f(x_b)$ and $-\lambda \nabla h(x_b)$ are not balanced. This is not only the case when $\lambda = \lambda_1^*$, as shown in Figure 6.8(a), but for all λ.

We now consider the problem

$$\min_{x\in\mathbb{R}^2} x_1$$

subject to

$$g_1(x) = x_1 - \sin x_2 \leq 0\,.$$

The inequality constraint is represented in Figure 6.8(b). The point $x_a = (-1, 3\pi/2)$ is not a local minimum of the problem. In fact, the direction $-\nabla f(x_a)$ is feasible and the associated "force" drives the solution towards feasible values.

Note that the constraint is active in x_a and that the equation

$$-\nabla f(x_a) - \mu_1 \nabla g_1(x_a) = 0$$

is satisfied for $\mu_1 = -1$. The interpretation of the forces signifies that the "force" $-\mu_1 \nabla g_1(x_a)$ drives the solution to the right and prevents it from going inside the feasible domain, which is incompatible with the definition of the inequality constraint. The condition (6.56) in Theorem 6.13, $\mu^* \geq 0$ signifies that, for an inequality constraint, the "force" can only act in a single direction, so as to prevent the points from leaving the feasible domain, but not from going inside. For the equality constraints, the "force" can act in two directions and there is no condition for the sign of the multipliers.

Example 6.17 (Slack variables). Consider problem (P1)

$$\min_{x\in\mathbb{R}^n} f(x) \tag{6.77}$$

subject to

$$g_i(x) \leq 0,\ i = 1, \ldots, m. \tag{6.78}$$

The Lagrangian of (P1) is

$$L(x, \mu) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x). \tag{6.79}$$

The first derivative is

$$\frac{\partial L}{\partial x_j}(x, \mu) = \frac{\partial f}{\partial x_j}(x) + \sum_{i=1}^{m} \mu_i \frac{\partial g_i}{\partial x_j}(x), \tag{6.80}$$

for $j = 1, \ldots, n$, and the second derivative is

$$\frac{\partial^2 L}{\partial x_j \partial x_k}(x, \mu) = \frac{\partial^2 f}{\partial x_j \partial x_k}(x) + \sum_{i=1}^{m} \mu_i \frac{\partial g_i}{\partial x_j \partial x_k}(x), \tag{6.81}$$

for $j, k = 1, \ldots, n$.

Let x^* be a local optimum of problem P1. Therefore, the first order necessary optimality (KKT) conditions say that, under appropriate assumptions, there is a unique $\mu^* \in \mathbb{R}^m$, $\mu^* \geq 0$, such that

$$\begin{aligned} \frac{\partial L}{\partial x_j}(x^*, \mu^*) = \frac{\partial f}{\partial x_j}(x^*) + \sum_{i=1}^{m} \mu_i^* \frac{\partial g_i}{\partial x_j}(x^*) &= 0, \\ \mu_i^* g_i(x^*) &= 0 \; i = 1, \ldots, m. \end{aligned} \tag{6.82}$$

Assume that the first p constraints are active at x^*, and the others not, that is

$$g_i(x^*) = 0, \; i = 1, \ldots, p, \tag{6.83}$$

and

$$g_i(x^*) < 0, \; i = p+1, \ldots, m. \tag{6.84}$$

Therefore, we obtain

$$\frac{\partial L}{\partial x_j}(x^*, \mu^*) = \frac{\partial f}{\partial x_j}(x^*) + \sum_{i=1}^{p} \mu_i^* \frac{\partial g_i}{\partial x_j}(x^*) = 0, \tag{6.85}$$

and

$$\begin{aligned} \mu_i^* &\geq 0, \; i = 1, \ldots, p, \\ \mu_i^* &= 0, \; i = p+1, \ldots, m. \end{aligned} \tag{6.86}$$

Moreover, for each $d \in \mathbb{R}^m$ such that, for each $i = 1, \ldots, p$

$$\sum_{k=1}^{n} d_k \frac{\partial g_i}{\partial x_k}(x^*) = 0, \tag{6.87}$$

we have

$$\sum_{j=1}^{n} \sum_{k=1}^{n} \left(\frac{\partial^2 f}{\partial x_j \partial x_k}(x^*) + \sum_{i=1}^{p} \mu_i^* \frac{\partial g_i}{\partial x_j \partial x_k}(x^*) \right) d_j d_k \geq 0. \tag{6.88}$$

Consider now problem (P2), obtained from problem (P1) by transforming the inequality constraints into equality constraints using slack variables, as suggested in Section 1.2.2:

$$\min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} f(x) \tag{6.89}$$

subject to

$$h_i(x, y) = g_i(x) + y_i^2 = 0, \; i = 1, \ldots, m. \tag{6.90}$$

For each $i = 1, \ldots, m$, the first derivatives of the constraint are

$$\frac{\partial h_i}{\partial x_k} = \frac{\partial g_i}{\partial x_k}, \; k = 1, \ldots, n, \tag{6.91}$$

$$\frac{\partial h_i}{\partial y_i} = 2y_i, \tag{6.92}$$

and

$$\frac{\partial h_i}{\partial y_\ell} = 0,\ \ell = 1, \dots, m, \ell \neq i. \tag{6.93}$$

The Lagrangian of (P2) is

$$L(x, y, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i (g_i(x) + y_i^2). \tag{6.94}$$

The first derivatives are

$$\frac{\partial L}{\partial x_j}(x, y, \lambda) = \frac{\partial f}{\partial x_j}(x) + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial x_j}(x), \tag{6.95}$$

for $j = 1, \dots, n$, and

$$\frac{\partial L}{\partial y_i}(x, y, \lambda) = 2\lambda_i y_i, \tag{6.96}$$

for $i = 1, \dots, m$. The second derivatives are

$$\frac{\partial^2 L}{\partial x_j \partial x_k}(x, y, \lambda) = \frac{\partial^2 f}{\partial x_j \partial x_k}(x) + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i}{\partial x_j \partial x_k}(x), \tag{6.97}$$

for $j, k = 1, \dots, n$,

$$\frac{\partial^2 L}{\partial y_i^2}(x, y, \lambda) = 2\lambda_i, \tag{6.98}$$

for $i = 1, \dots, m$,

$$\frac{\partial^2 L}{\partial y_i \partial y_\ell}(x, y, \lambda) = 0, \tag{6.99}$$

for $i, \ell = 1, \dots, m$, $i \neq \ell$, and

$$\frac{\partial^2 L}{\partial x_j \partial y_i}(x, y, \lambda) = 0, \tag{6.100}$$

for $j = 1, \dots, n$, $i = 1, \dots, m$.

Let x^* and y^* be local optima of problem P2. Therefore, the first order necessary optimality (KKT) conditions say that there is a unique $\lambda^* \in \mathbb{R}^m$ such that

$$\frac{\partial L}{\partial x_j}(x^*, y^*, \lambda^*) = \frac{\partial f}{\partial x_j}(x^*) + \sum_{i=1}^{m} \lambda_i^* \frac{\partial g_i}{\partial x_j}(x^*) = 0, \tag{6.101}$$

and

$$2\lambda_i y_i^* = 0\ i = 1, \dots, m. \tag{6.102}$$

Moreover, for each

$$d = \begin{pmatrix} d_x \\ d_y \end{pmatrix} \in \mathbb{R}^{n+m}$$

such that, for each $i = 1, \dots, m$

$$\sum_{k=1}^{n} (d_x)_k \frac{\partial g_i}{\partial x_k}(x^*) + 2(d_y)_i y_i^* = 0, \tag{6.103}$$

we have

$$\sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial^2 f}{\partial x_j \partial x_k}(x^*)+\sum_{i=1}^{m}\lambda_i^*\frac{\partial g_i}{\partial x_j \partial x_k}(x^*)\right)(d_x)_j(d_x)_k+2\sum_{i=1}^{m}\lambda_i^*(d_y)_i^2\geq 0. \quad (6.104)$$

Now, assume that the constraints are numbered so that $y_i^*=0$ for $i=1,\ldots,p$, and $y_i^*\neq 0$ for $i=p+1,\ldots,m$. As x^* verifies the constraints (6.90), we have also $g_i(x^*)=0$ for $i=1,\ldots,p$, and $g_i(x^*)<0$ for $i=p+1,\ldots,m$. Then, from (6.102), we have $\lambda_i^*=0$, $i=p+1,\ldots,m$. The first order condition (6.101) becomes

$$\frac{\partial L}{\partial x_j}(x^*,y^*,\lambda^*)=\frac{\partial f}{\partial x_j}(x^*)+\sum_{i=1}^{p}\lambda_i^*\frac{\partial g_i}{\partial x_j}(x^*)=0. \quad (6.105)$$

For the second order conditions, for each

$$d=\begin{pmatrix} d_x \\ d_y \end{pmatrix}\in\mathbb{R}^{n+m}$$

such that, for each $i=1,\ldots,p$

$$\sum_{k=1}^{n}(d_x)_k\frac{\partial g_i}{\partial x_k}(x^*)=0, \quad (6.106)$$

and for each $i=p+1,\ldots,m$

$$\sum_{k=1}^{n}(d_x)_k\frac{\partial g_i}{\partial x_k}(x^*)+2(d_y)_i y_i^*=0, \quad (6.107)$$

we have

$$\sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial^2 f}{\partial x_j \partial x_k}(x^*)+\sum_{i=1}^{p}\lambda_i^*\frac{\partial g_i}{\partial x_j \partial x_k}(x^*)\right)(d_x)_j(d_x)_k+2\sum_{i=1}^{p}\lambda_i^*(d_y)_i^2\geq 0. \quad (6.108)$$

In particular, consider d such that $d_x=0$ and $(d_y)_i=0$, $i=p+1,\ldots,m$. It clearly verifies conditions (6.106) and (6.107). We have for any value of $(d_y)_i$, $i=1,\ldots,p$,

$$2\sum_{i=1}^{p}\lambda_i^*(d_y)_i^2\geq 0. \quad (6.109)$$

In particular, select k between 1 and p, and set $(d_y)_i=0$ for each $i\neq k$, and $(d_y)_k=1$. Therefore, (6.109) implies $\lambda_k^*\geq 0$, for any $k=1,\ldots,p$.

Based on these results, we can prove that (x^*,μ^*) verifies the KKT conditions of problem P1, if and only if (x^*,y^*,λ^*) verifies the KKT conditions of problem P2, where $\lambda^*=\mu^*$ and $y_i^*=\sqrt{-g_i(x^*)}$, $i=1,\ldots,m$.

P1 $\Longrightarrow$ P2 Consider (x^*,μ^*) that verifies the KKT conditions of problem P1, such that $g_i(x^*)=0$ for $i=1,\ldots,p$ and $g_i(x^*)<0$ for $i=p+1,\ldots,m$. Define y^* such that

$$\begin{array}{rcll} y_i^* & = & 0 & i=1,\ldots,p, \\ y_i^* & = & \sqrt{-g_i(x^*)} & i=p+1,\ldots,m, \end{array} \quad (6.110)$$

and define $\lambda^*=\mu^*$. Then (x^*,y^*,λ^*) verifies the KKT conditions of problem P2.

- Constraints (6.90) are trivially verified from the definition of y^*.
- The first order conditions (6.105) are exactly the same as (6.85).
- The second order KKT conditions are also trivially verified. Consider a direction d that verifies (6.106) and (6.107). As (6.106) is equivalent to (6.87), we deduce from (6.88) that

$$\sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial^2 f}{\partial x_j \partial x_k}(x^*)+\sum_{i=1}^{p}\lambda_i^*\frac{\partial g_i}{\partial x_j \partial x_k}(x^*)\right)(d_x)_j(d_x)_k \geq 0. \tag{6.111}$$

  Now, (6.108) results from (6.111) and (6.86), which says that $\lambda_i^* \geq 0$, for each i.

P2 $\Longrightarrow$ P1 Consider (x^*, y^*, λ^*) that verifies the KKT conditions of problem P2, such that $y^* = 0$ for $i = 1, \ldots, p$ and $y_i^* \neq 0$ for $i = p+1, \ldots, m$. Then (x^*, μ^*), where $\mu^* = \lambda^*$, verifies the KKT conditions of problem P1.

- The constraints (6.78) are direct consequences of (6.90).
- The first order conditions (6.105) are exactly the same as (6.85).
- The conditions (6.86) on the Lagrange multipliers are verified, as for P2, $\lambda_i^* \geq 0$, for $i = 1, \ldots, p$ and $\lambda_i^* = 0$, $i = p+1, \ldots, m$ (see the discussion above).
- Consider d that verifies (6.87). Define $d_x = d$, and define d_y such that $(d_y)_i = 0$, $i = 1, \ldots, p$, and

$$(d_y)_i = -\frac{1}{2y_i^*}\sum_{k=1}^{n}(d_x)_k\frac{\partial g_i}{\partial x_k}(x^*) \tag{6.112}$$

  for $i = p+1, \ldots, m$. By definition, d_x and d_y verify (6.106) and (6.107). Therefore, (6.108) holds. As $(d_y)_i = 0$, $i = 1, \ldots, p$, we obtain (6.88).

6.3 Lagrange multipliers: sufficient conditions

Similarly to the approach presented in Section 6.2, we start with problems with equality constraints. We then generalize the result for general problems.

The demonstration of the sufficient optimality condition utilizes what is called an *augmented Lagrangian*, which is also used for algorithms.

Definition 6.18 (Augmented Lagrangian)**.** Consider the optimization problem with equality constraints (1.71)–(1.72) $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$ and let us take a parameter $c \in \mathbb{R}$, $c > 0$, called penalty parameter. The Lagrangian function of the problem

$$\min_{x\in\mathbb{R}^n} f(x) + \frac{c}{2}\left\|h(x)\right\|^2 \qquad \text{subject to} \qquad h(x) = 0 \tag{6.113}$$

is called the augmented Lagrangian function of the problem (1.71)–(1.72) and is expressed as

$$\begin{aligned} L_c(x,\lambda) &= L(x,\lambda) + \frac{c}{2}\left\|h(x)\right\|^2 \\ &= f(x) + \lambda^T h(x) + \frac{c}{2}\left\|h(x)\right\|^2 . \end{aligned} \tag{6.114}$$

The idea of the objective function (6.113) is to penalize points x that violate the constraints, hence the name penalty parameter for c .

6.3.1 Equality constraints

Thanks to the Lagrangian function, the sufficient conditions are similar to those for the unconstrained case. However, we should note the role of the linearized cone.

Theorem 6.19 (Sufficient optimality conditions: equality constraints). *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *and* $h : \mathbb{R}^n \to \mathbb{R}^m$ *be twice differentiable functions. Consider* $x^* \in \mathbb{R}^n$ *and* $\lambda^* \in \mathbb{R}^m$ *such that*

$$\nabla L(x^*,\lambda^*) = 0 \tag{6.115}$$

and

$$y^T \nabla^2_{xx} L(x^*,\lambda^*) y > 0 , \qquad \forall y \in \mathcal{D}(x^*),\ y \neq 0 , \tag{6.116}$$

where L *is the Lagrangian function of the optimization problem* $\min_{x\in\mathbb{R}^n} f(x)$ *subject to* $h(x) = 0$ *and* $\mathcal{D}(x^*)$ *is the linearized cone in* x^**. Then,* x^* *is a strict local minimum of the optimization problem.*

Proof. We first note that any solution to the augmented problem (6.113) is also a solution to the original problem. We go back to a problem of unconstrained optimization thanks to the augmented Lagrangian function, by showing that x^* is a strict local minimum of the problem

$$\min_{x\in\mathbb{R}^n} L_c(x,\lambda^*) \tag{6.117}$$

for sufficiently large c. Indeed,

$$\begin{aligned} \nabla_x L_c(x^*,\lambda^*) &= \nabla f(x^*) + \nabla h(x^*)\big(\lambda^* + c h(x^*)\big) && \text{by derivation of (6.114)} \\ &= \nabla f(x^*) + \nabla h(x^*)\lambda^* && \text{because } x^* \text{ is feasible} \\ &= \nabla_x L(x^*,\lambda^*) && \text{according to Definition 4.3} \\ &= 0 && \text{from (6.115)} . \end{aligned}$$

Similarly, we obtain

$$\nabla^2_{xx} L_c(x^*,\lambda^*) = \nabla^2_{xx} L(x^*,\lambda^*) + c\nabla h(x^*)\nabla h\big(x^*\big)^T . \tag{6.118}$$

By applying the theorem for the formation of a positive definite matrix (Theorem C.18), there exists $\hat{c}$ such that (6.118) is positive definite for all $c > \hat{c}$. According to

Theorem 5.7, x^* is a strict local minimum of the unconstrained problem (6.117) for sufficiently large c.

According to Definition 1.6, there exists $\varepsilon > 0$ such that

$$L_c(x^*, \lambda^*) < L_c(x, \lambda^*), \quad \forall x \in \mathbb{R}^n, \; x \neq x^* \text{ such that } \|x - x^*\| < \varepsilon. \tag{6.119}$$

According to Definition 6.18 of L_c, we get

$$f(x^*) < f(x), \quad \forall x \in \mathbb{R}^n, \; x \neq x^* \text{ such that } \|x - x^*\| < \varepsilon \text{ and } h(x) = 0. \tag{6.120}$$

According to Definition 1.6, x^* is a strict local minimum of the problem. □

Notes

- Theorem 6.10 can be demonstrated by using the same constraint elimination technique as for Theorem 6.8. The logic behind the demonstration is the same, but the proof is more technical (see Bertsekas, 1999).
- No constraint qualifications appear in the sufficient optimality conditions, neither linear independence nor any other.

6.3.2 Inequality constraints

Theorem 6.20 (Sufficient optimality conditions). *Let* $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$ *and* $g : \mathbb{R}^n \to \mathbb{R}^p$ *be twice differentiable functions. Consider* $x^* \in \mathbb{R}^n$, $\lambda^* \in \mathbb{R}^m$ *and* $\mu^* \in \mathbb{R}^p$ *such that*

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0 \tag{6.121}$$
$$h(x^*) = 0 \tag{6.122}$$
$$g(x^*) \leq 0 \tag{6.123}$$
$$\mu^* \geq 0 \tag{6.124}$$
$$\mu_j^* g_j(x^*) = 0, \qquad j = 1, \ldots, p \tag{6.125}$$
$$\mu_j^* > 0, \qquad \forall j \in \mathcal{A}(x^*) \tag{6.126}$$

$$\begin{aligned} &y^T \nabla_{xx}^2 L(x^*, \lambda^*, \mu^*) y > 0, \quad \forall y \neq 0 \text{ such that} \\ &\quad y^T \nabla h_i(x^*) = 0, \; i = 1, \ldots, m \\ &\quad y^T \nabla g_i(x^*) = 0, \; i = 1, \ldots, p \text{ such that } g_i(x^*) = 0, \end{aligned} \tag{6.127}$$

where L *is the Lagrangian function of the optimization problem* $\min_{x \in \mathbb{R}^n} f(x)$ *subject to* $h(x) = 0$ *and* $g(x) \leq 0$. *Then,* x^* *is a strict local minimum of the optimization problem.*

Proof. We use slack variables (Definition 1.4) to obtain the following optimization problem with equality constraints

$$\min_{x\in\mathbb{R}^n,\, z\in\mathbb{R}^p} f(x) \tag{6.128}$$

subject to

$$\begin{aligned} h_i(x) &= 0\,, \qquad && i = 1,\dots,m \\ g_i(x) + z_i^2 &= 0\,, && i = 1,\dots,p\,, \end{aligned} \tag{6.129}$$

and let us define

$$z_i^* = \sqrt{-g_i(x^*)} \tag{6.130}$$

such that $g_i(x^*) + \left(z_i^*\right)^2 = 0$ is trivially satisfied.

The Lagrangian function of this problem is

$$\widehat{L}(x,z,\lambda,\mu) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{i=1}^{p} \mu_i \left(g_i(x) + z_i^2\right) \tag{6.131}$$

and

$$\nabla_x \widehat{L}(x,z,\lambda,\mu) = \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x) + \sum_{i=1}^{p} \mu_i \nabla g_i(x) = \nabla_x L(x,\lambda,\mu) \tag{6.132}$$

and

$$\frac{\partial \widehat{L}(x,z,\lambda,\mu)}{\partial z_i} = 2\mu_i z_i\,, \qquad i = 1,\dots,p\,. \tag{6.133}$$

Moreover, by expressing $\widehat{x} = \begin{pmatrix} x \\ z \end{pmatrix}$, we have

$$\nabla^2_{\widehat{x}\widehat{x}} \widehat{L}(x,z,\lambda,\mu) = \left(\begin{array}{c|cccc} \nabla^2_{xx} L(x,\lambda,\mu) & \multicolumn{4}{c}{0} \\ \hline & 2\mu_1 & 0 & \cdots & 0 \\ & 0 & 2\mu_2 & \cdots & 0 \\ 0 & \vdots & \vdots & \ddots & \vdots \\ & 0 & 0 & \cdots & 2\mu_p \end{array}\right). \tag{6.134}$$

From the hypothesis (6.125) and as per (6.130), we have $\mu_i^* z_i^* = 0$. Moreover, from the hypothesis (6.121), we have

$$\nabla \widehat{L}_{\widehat{x}}(x^*, z^*, \lambda^*, \mu^*) = 0\,. \tag{6.135}$$

Consider a non zero vector

$$\begin{pmatrix} y \\ w \end{pmatrix} \in \mathbb{R}^{m+p}\,,$$

in the linearized cone at

$$\begin{pmatrix} x^* \\ z^* \end{pmatrix}$$

for the problem (6.128)–(6.129), i.e., such that

$$y^T \nabla h_i(x^*) = 0, \qquad i = 1, \dots, m, \tag{6.136}$$

and

$$y^T \nabla g_i(x^*) + 2 z_i^* w_i = 0, \qquad i = 1, \dots, p. \tag{6.137}$$

Note that if $i \in \mathcal{A}(x^*)$, then $z_i = 0$ and (6.137) is written as $y^T \nabla g_i(x^*) = 0$. The vector y always corresponds to the conditions of the hypothesis (6.127). We have

$$\begin{aligned}
&\begin{pmatrix} y^T & w^T \end{pmatrix} \nabla^2_{\widehat{x}\widehat{x}} \widehat{L}(x^*, z^*, \lambda^*, \mu^*) \begin{pmatrix} y \\ w \end{pmatrix} \\
&= y^T \nabla^2_{xx} L(x^*, \lambda^*, \mu^*) y + 2 \sum_{i=1}^{p} \mu_i^* w_i^2 && \text{from (6.134)} \\
&= y^T \nabla^2_{xx} L(x^*, \lambda^*, \mu^*) y + 2 \sum_{i \in \mathcal{A}(x^*)} \mu_i^* w_i^2 && \text{from (6.125)}.
\end{aligned}$$

From the hypothesis (6.127), the first term is positive if $y \neq 0$. From the hypothesis (6.124), each term of the sum of the second term is non negative. If $y = 0$, there is necessarily an i such that $w_i \neq 0$. In order for (6.137) to be satisfied, we have that $z_i^* = 0$ and then $i \in \mathcal{A}(x^*)$. From (6.126), the corresponding term $\mu_i^* w_i^2$ is positive.

The sufficient optimality conditions of Theorem 6.19 are satisfied for x^*, z^*, λ^* and μ^* and

$$\begin{pmatrix} x^* \\ z^* \end{pmatrix}$$

is a strict local minimum of (6.128)–(6.129). Consequently, x^* is a strict local minimum of the initial problem. □

The condition (6.126) is called the *strict complementarity condition*. The following example illustrates its importance.

Example 6.21 (Importance of the strict complementarity condition)**.** Consider the problem

$$\min_{x \in \mathbb{R}^2} \frac{1}{2}(x_1^2 - x_2^2) \tag{6.138}$$

subject to

$$x_2 \leq 0 \tag{6.139}$$

for which the objective function is illustrated in Figure 6.7. We demonstrate that all sufficient optimality conditions except (6.126) are satisfied for $x_* = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ and $\mu^* = 0$. We have

$$L(x, \mu) = \frac{1}{2}(x_1^2 - x_2^2) + \mu x_2. \tag{6.140}$$

Then,

$$\nabla L_x(x, \mu) = \begin{pmatrix} x_1 \\ -x_2 + \mu \end{pmatrix}$$

and (6.121) is satisfied when $x_* = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ and $\mu^* = 0$. The other conditions are trivially satisfied and (6.126) is not satisfied because the constraint is active in x^* and $\mu^* = 0$.

We now consider the point $(0, -\alpha)$, with $\alpha > 0$. It is feasible and the objective function is $-\frac{1}{2}\alpha^2$, which is strictly smaller than the value in x^*, which is therefore not a local minimum.

To understand why it is not a local optimum, let us take the proof of Theorem 6.20 and transform the problem into a problem with equality constraints:

$$\min_{x \in \mathbb{R}^2} \frac{1}{2}(x_1^2 - x_2^2) \tag{6.141}$$

subject to

$$x_2 + z^2 = 0\,. \tag{6.142}$$

We have

$$z^* = \sqrt{-g(x_*)} = 0\,. \tag{6.143}$$

The Lagrangian function of this problem is

$$\widehat{L}(x, z, \mu) = \frac{1}{2}(x_1^2 - x_2^2) + \mu(x_2 + z^2)\,. \tag{6.144}$$

The Hessian $\nabla^2_{\widehat{x}\widehat{x}} \widehat{L}(x^*, z^*, \mu^*)$ used in the proof is

$$\nabla^2_{\widehat{x}\widehat{x}} \widehat{L}(x, z, \mu) = \left(\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & -1 & 0 \\ \hline 0 & 0 & 2\mu \end{array}\right)$$

and

$$\nabla^2_{\widehat{x}\widehat{x}} \widehat{L}(x^*, z^*, \mu^*) = \left(\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & -1 & 0 \\ \hline 0 & 0 & 0 \end{array}\right),$$

and it is singular. As in the proof, let us take a vector belonging to the linearized cone at

$$\begin{pmatrix} x^* \\ z^* \end{pmatrix}$$

of the problem with equality constraints, i.e.,

$$\begin{pmatrix} y \\ w \end{pmatrix} = \left(\begin{array}{c} 0 \\ 0 \\ \hline \gamma \end{array}\right).$$

For all γ, we have

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

and the sufficient condition for the problem with equality constraints is not satisfied, which prevents us from proving Theorem 6.20.

We conclude this section with two examples of the use of optimality conditions to identify critical points. They both lead to the resolution of a system of equations, the topic of the next section of this book.

Example 6.22 (Identification of critical points – I)**.** Consider the problem

$$\min_{x \in \mathbb{R}^2} 3x_1^2 + x_2^2$$

subject to

$$2x_1 + x_2 = 1$$

illustrated in Figure 6.9. The necessary optimality condition (6.23) is written as

$$\begin{aligned} 6x_1 + 2\lambda &= 0 \\ 2x_2 + \lambda &= 0 \\ 2x_1 + x_2 - 1 &= 0\,. \end{aligned}$$

This is a system with three linear equations, with three unknowns, for which the solution is

$$\begin{aligned} x_1^* &= \frac{2}{7} \\ x_2^* &= \frac{3}{7} \\ \lambda^* &= -\frac{6}{7}\,. \end{aligned}$$

We now need only ensure that this point satisfies the sufficient second-order conditions in order to determine that it is indeed a solution.

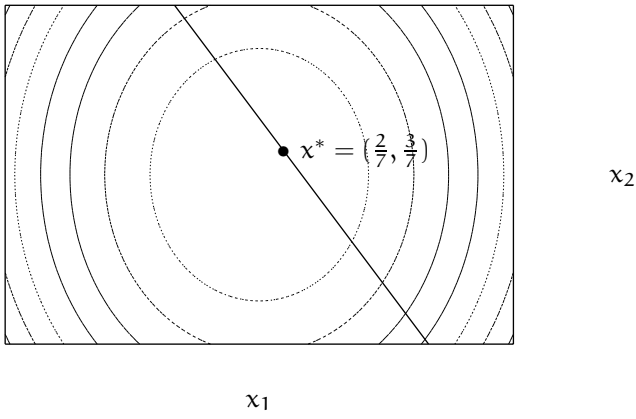


Figure 6.9: Problem for Example 6.22

Example 6.23 (Identification of critical points – II)**.** Consider the problem

$$\min_{x \in \mathbb{R}^2} 3x_1^2 + x_2^2$$

subject to

$$x_1^2 + 4x_1 - x_2 + 3 = 0$$

illustrated in Figure 6.10. The necessary optimality condition (6.23) is expressed as

$$\begin{aligned} 6x_1 + 2\lambda x_1 + 4\lambda &= 0 \\ 2x_2 - \lambda &= 0 \\ x_1^2 + 4x_1 + 3 &= 0\,. \end{aligned}$$

This is a system of three non linear equations, with three unknowns. One solution is $x_1 = -1$, $x_2 = 1.5$, and $\lambda = 3$. It is not necessarily straightforward to calculate it.

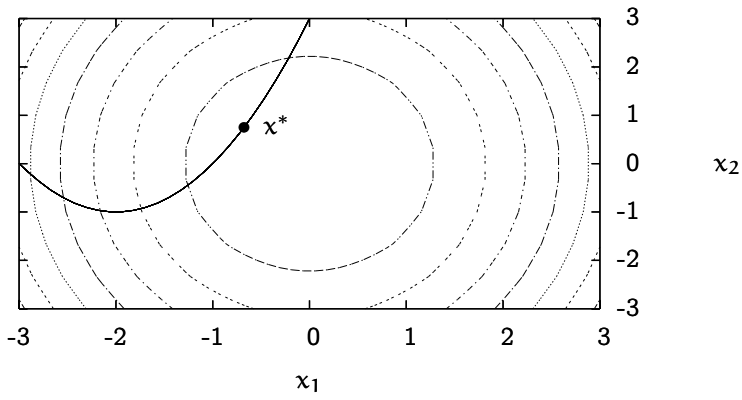


Figure 6.10: Problem of Example 6.23

6.4 Sensitivity analysis

When the data of an optimization problem is slightly disturbed, the solution to the perturbed problem generally does not differ fundamentally from that of the unperturbed problem. We first analyze this relation for problems with equality constraints.

Theorem 6.24 (Sensitivity analysis: equality constraints)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *and* $h : \mathbb{R}^n \to \mathbb{R}^m$ *be twice continuously differentiable functions. Consider the optimization problem (1.71)–(1.72)*

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

subject to

$$h(x) = 0\,.$$

Moreover, let x^* be a local minimum and let λ^* satisfy the sufficient optimality conditions (6.115) and (6.116), such that the constraints are linearly independent in x^*, according to Definition 3.8. Consider a perturbation of the data characterized by $\delta \in \mathbb{R}^m$, and the perturbed optimization problem

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

subject to

$$h(x) = \delta.$$

There thus exists a sphere $S \subset \mathbb{R}^m$ centered in 0 such that if $\delta \in S$, there exist $x(\delta)$ and $\lambda(\delta)$ satisfying the sufficient optimality conditions of the perturbed problem. The functions $x : \mathbb{R}^m \to \mathbb{R}^n : \delta \rightsquigarrow x(\delta)$ and $\lambda : \mathbb{R}^m \to \mathbb{R}^m : \delta \rightsquigarrow \lambda(\delta)$ are continuously differentiable in S, with $x(0) = x^*$ and $\lambda(0) = \lambda^*$. Moreover, for all $\delta \in S$, we have

$$\nabla p(\delta) = -\lambda(\delta)\,, \tag{6.145}$$

where

$$p(\delta) = f\big(x(\delta)\big)\,. \tag{6.146}$$

Proof. We note that

$$\gamma = \begin{pmatrix} x \\ \lambda \end{pmatrix} \in \mathbb{R}^{n+m}$$

and consider the function $F : \mathbb{R}^{m+n+m} \to \mathbb{R}^{n+m}$ defined by

$$F(\delta, \gamma) = \begin{pmatrix} \nabla f(x) + \nabla h(x)\lambda \\ h(x) - \delta \end{pmatrix} = \begin{pmatrix} \nabla_x L(x, \lambda) \\ h(x) - \delta \end{pmatrix}. \tag{6.147}$$

We first demonstrate that the gradient matrix

$$\nabla_\gamma F(\delta, \gamma^*) = \begin{pmatrix} \nabla^2_{xx} L(x^*, \lambda^*) & \nabla h(x^*) \\ \nabla h(x^*)^T & 0 \end{pmatrix}$$

is non singular. We assume by contradiction that this is not the case. There then exist $y \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$, non zero, such that

$$\nabla_\gamma F(\delta, \gamma^*) \begin{pmatrix} y \\ z \end{pmatrix} = 0\,,$$

i.e.,

$$\nabla^2_{xx} L(x^*, \lambda^*) y + \nabla h(x^*) z = 0 \tag{6.148}$$

$$\nabla h(x^*)^T y = 0\,. \tag{6.149}$$

We have

$$\begin{aligned} y^T \nabla^2_{xx} L(x^*, \lambda^*) y &= -y^T \nabla h(x^*) z && \text{from (6.148)} \\ &= 0 && \text{from (6.149)}. \end{aligned}$$

Since the sufficient optimality condition (6.116) is satisfied, $\nabla^2_{xx} L(x^*, \lambda^*)$ is positive definite and $y = 0$. Then, according to (6.148), $\nabla h(x^*) z = 0$. By assumption, the constraints are linearly independent at x^* and the matrix $\nabla h(x^*)$ is of full rank. Then, $z = 0$, which contradicts the fact that y and z are non zero. The matrix $\nabla_\gamma F(\delta, \gamma^*)$ is indeed non singular and we can apply the theorem of implicit functions (Theorem C.6): there exist neighborhoods $\mathcal{V}_0 \subseteq \mathbb{R}^m$ around $\delta^+ = 0$ and $\mathcal{V}_{\gamma^*} \subseteq \mathbb{R}^{n+m}$ around $\gamma^+ = \gamma^*$, as well as a continuous function

$$\phi : \mathcal{V}_0 \to \mathcal{V}_{\gamma^*} : \delta \rightsquigarrow \gamma(\delta) = \begin{pmatrix} x(\delta) \\ \lambda(\delta) \end{pmatrix}$$

such that

$$F\big(\delta, \gamma(\delta)\big) = 0\,, \qquad \forall \delta \in \mathcal{V}_0\,,$$

i.e.,

$$\begin{pmatrix} \nabla f\big(x(\delta)\big) + \nabla h\big(x(\delta)\big)\lambda(\delta) \\ h\big(x(\delta)\big) - \delta \end{pmatrix} = 0\,. \tag{6.150}$$

We now demonstrate that, for δ sufficiently close to 0, the sufficient optimality conditions of the perturbed problem are satisfied. Assuming (by contradiction) that this is not the case, there exists a sequence $(\delta_k)_k$, such that $\lim_{k\to\infty} \delta_k = 0$ and a sequence $(y_k)_k$, with $y_k \in \mathbb{R}^m$, $\|y_k\| = 1$ and $\nabla h\big(x(\delta_k)\big)^T y_k = 0$, for all k, such that

$$y_k^T \nabla^2_{xx} L\big(x(\delta_k), \lambda(\delta_k)\big) y_k \le 0\,, \quad \forall k\,.$$

Consider a subsequence of y_k converging toward $\bar{y} \neq 0$. When we take the limit, we obtain by continuity of $\nabla^2_{xx} L$ (as a result of the continuity of $\nabla^2_{xx} f$ and $\nabla^2_{xx} h_i$, $i = 1, \dots, m$) that

$$\bar{y}^T \nabla^2_{xx} L(x^*, \lambda^*) \bar{y} \le 0\,,$$

which contradicts the sufficient optimality condition in x^* and λ^* (6.116).

By differentiating the second row of (6.150), we obtain

$$\nabla_\delta h\big(x(\delta)\big) = \nabla x(\delta) \nabla h\big(x(\delta)\big) = I\,. \tag{6.151}$$

When multiplying the first row of (6.150) by $\nabla x(\delta)$, we get

$$0 = \nabla x(\delta) \nabla f\big(x(\delta)\big) + \nabla x(\delta) \nabla h\big(x(\delta)\big)\lambda(\delta) = \nabla x(\delta) \nabla f\big(x(\delta)\big) + \lambda(\delta)\,,$$

where the second equality comes from (6.151). Therefore,

$$\nabla p(\delta) = \nabla_\delta f\big(x(\delta)\big) = \nabla x(\delta) \nabla f\big(x(\delta)\big) = -\lambda(\delta)\,,$$

which demonstrates (6.145). □

Example 6.25 (Sensitivity)**.** Consider the problem $\min_{x\in\mathbb{R}} x^2$ subject to $x = 1$, for which the solution is $x^* = 1$, $\lambda^* = -2$. We consider the perturbed problem $\min_{x\in\mathbb{R}} x^2$ subject to $x = 1 + \delta$, for which the solution is $x(\delta) = 1 + \delta$ and $\lambda(\delta) = -2\delta - 2$. We have $f\big(x(\delta)\big) = \delta^2 + 2\delta + 1$ and

$$\frac{df}{d\delta}\big(x(\delta)\big) = 2\delta + 2 = -\lambda(\delta)\,.$$

The quantity $\nabla f\big(x(\delta)\big)$ represents the marginal modification of the objective function for a perturbation δ of the constraints. When δ is small, we use Taylor's theorem (Theorem C.1) to obtain

$$p(\delta) = p(0) + \delta^T \nabla p(0) + o\big(\|d\|\big)\,.$$

Neglecting the last term, we obtain

$$f\big(x(\delta)\big) \approx f(x^*) - \delta^T \lambda^*\,.$$

Note that if $p(\delta)$ is linear, we have exactly

$$f\big(x(\delta)\big) = f(x^*) - \delta^T \lambda^*\,. \tag{6.152}$$

This result has significant practical consequences. Indeed, it becomes possible to measure the impact of a perturbation of the constraint on the objective function, without re-optimizing.

Example 6.26 (Sensitivity analysis)**.** We consider a company manufacturing two products. Each unit of the first product brings in €6,000, while each unit of the second product brings in €5,000. A total of 10 machine-hours and 15 tons of raw material are available daily. Each unit of the first product requires 2 machine-hours and 1 ton of raw material. Each unit of the second product requires 1 machine-hour and 3 tons of raw material. In thousands of euros, the optimal production that the company should consider is obtained by solving the optimization problem

$$\max_{x_1,x_2} 6x_1 + 5x_2$$

subject to

$$\begin{aligned} 2x_1 + x_2 &\leq 10 \\ x_1 + 3x_2 &\leq 15 \\ x_1\,,\ x_2 &\geq 0\,. \end{aligned}$$

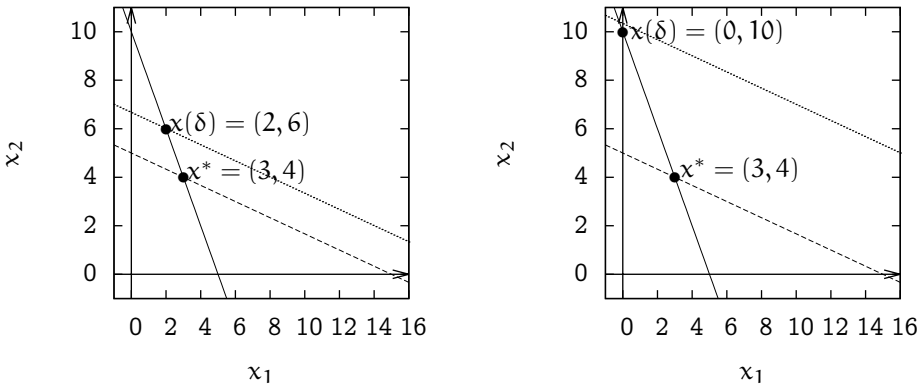
We omit the non negativity constraints to maintain a simple formulation (these constraints are inactive at the solution). We first express the problem in the form (1.71)–(1.72) by changing the maximization problem into a minimization problem, and by including slack variables (see Section 1.2):

$$\min_{x_1,x_2,x_3,x_4} f(x) = -6x_1 - 5x_2$$

subject to

$$h_1(x) = 2x_1 + x_2 + x_3^2 - 10 = 0$$
$$h_2(x) = x_1 + 3x_2 + x_4^2 - 15 = 0\,,$$

where x_3 and x_4 are the slack variables. The solution to the problem is $x^* = \begin{pmatrix} 3 & 4 & 0 & 0 \end{pmatrix}^T$ and $\lambda^* = \begin{pmatrix} 13/5 & 4/5 \end{pmatrix}^T$, enabling the company to bring in €38,000 per day.



(a) $\delta = 5$ (b) $\delta = 16$

Figure 6.11: Graphical analysis of Example 6.26

In order to increase its production, the company wishes to invest by purchasing an additional quantity δ of raw material per day. In this case, the constraints would become

$$h_1(x) = 2x_1 + x_2 + x_3^2 - 10 = 0$$
$$h_2(x) = x_1 + 3x_2 + x_4^2 - 15 = \delta\,.$$

To determine what this investment would bring in, we use (6.152) with

$$\delta = \begin{pmatrix} 0 \\ \delta \end{pmatrix}$$

to obtain

$$f\big(x(\delta)\big) = f(x^*) - \delta^T\lambda^* = -38 - \frac{4}{5}\,\delta\,. \tag{6.153}$$

Therefore, the purchase of 5 tons of raw material per day would enable the company to bring in €4,000. If this purchase costs less than €4,000, it is worth going through with the investment. Otherwise, the investment is of no use.

Note that this result is only valid for small values of δ. If $\delta = 16$, such that 31 tons of raw material are available each day, the company no longer has enough machine-hours to use up all of the raw material. Therefore, the second constraint is no longer active (here, x_4 is positive). The company should thus produce only the second product, which consumes half as many machine-hours. In this case, the

purchase of additional raw material would not enable the company to earn more, and the company should thus rather invest in new machines.

The inspiration for this example came from de Werra et al. (2003).

Corollary 6.27 (Sensitivity analysis). *Let* $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^p$ *and* $h : \mathbb{R}^n \to \mathbb{R}^m$ *be twice differentiable functions. Consider the optimization problem (1.71)–(1.73)*

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

subject to

$$h(x) = 0$$
$$g(x) \leq 0\,.$$

Moreover, let x^* *be a local minimum and let* λ^*, μ^* *satisfy the sufficient optimality conditions (6.121)–(6.126), such that the constraints are linearly independent in* x^**, according to Definition 3.8. Consider perturbations* $\delta_h \in \mathbb{R}^m$ *and* $\delta_g \in \mathbb{R}^p$*, and the perturbed optimization problem*

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

subject to

$$h(x) = \delta_h$$
$$g(x) \leq \delta_g.$$

Then, there exists a sphere $S \subset \mathbb{R}^{m+p}$ *centered in* 0 *such that if* $\delta = \begin{pmatrix} \delta_h^T & \delta_g^T \end{pmatrix}^T \in S$*, there are* $x(\delta)$*,* $\lambda(\delta)$ *and* $\mu(\delta)$ *are satisfying the sufficient optimality conditions of the perturbed problem. The functions* $x : \mathbb{R}^{m+p} \to \mathbb{R}^n$ *,* $\lambda : \mathbb{R}^{m+p} \to \mathbb{R}^m$ *and* $\mu : \mathbb{R}^{m+p} \to \mathbb{R}^p$ *are continuously differentiable in* S*, with* $x(0,0) = x^*$*,* $\lambda(0,0) = \lambda^*$ *and* $\mu(0,0) = \mu^*$*. Moreover, for all* $\delta \in S$*, we have*

$$\begin{aligned} \nabla_{\delta_h} p(\delta) &= -\lambda(\delta) \\ \nabla_{\delta_g} p(\delta) &= -\mu(\delta)\,, \end{aligned} \tag{6.154}$$

with

$$p(\delta) = f\big(x(\delta)\big)\,. \tag{6.155}$$

Proof. From Theorem 3.5, x^*, λ^* and μ^* satisfy the optimality conditions of the problem

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

subject to

$$\begin{aligned} h(x) &= 0 \\ g_i(x) &= 0\,, \qquad \forall i \in \mathcal{A}(x^*)\,. \end{aligned}$$

From Theorem 6.24, there exist $x(\delta)$, $\lambda(\delta)$ and $\mu(\delta)$ satisfying the sufficient optimality conditions of the problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{6.156}$$

subject to

$$h(x) = \delta_h \tag{6.157}$$

$$g_i(x) = \left(\delta_g\right)_i, \qquad \forall i \in \mathcal{A}(x^*). \tag{6.158}$$

The functions $x : \mathbb{R}^{m+p} \to \mathbb{R}^n$, $\lambda : \mathbb{R}^{m+p} \to \mathbb{R}^m$ and $\mu_i : \mathbb{R}^{m+p} \to \mathbb{R}$ are continuously differentiable in S, with $x(0,0) = x^*$, $\lambda(0,0) = \lambda^*$ and $\mu_i(0,0) = \mu_i^*$, $i \in \mathcal{A}(x^*)$. Moreover, for all $\delta \in S$, we have

$$\nabla_{\delta_h} p(\delta) = -\lambda(\delta)$$

$$\left(\nabla_{\delta_g} p(\delta)\right)_i = -\mu_i(\delta), \qquad i \in \mathcal{A}x^*.$$

We now need only verify the result for inequality constraints that are inactive in x^*, i.e.,

$$g_i(x^*) < 0.$$

In this case, if δ_i is sufficiently close to 0, $g_i\big(x(\delta)\big) < \delta_i$, and the constraint i is also inactive on the solution to the perturbed problem (by continuity of g_i). Then, according to Theorem 3.5, the perturbed problem is equivalent to the problem (6.156)–(6.158). If we take $\mu_i(\delta) = 0$ for all $i \notin \mathcal{A}(x^*)$, $x(\delta)$, $\lambda(\delta)$ and $\mu(\delta)$ satisfying the sufficient optimality conditions. Moreover, regardless of the value of δ_i (small enough for the constraint of the problem to remain inactive), the value of $x(\lambda)$ remains constant, since it is determined by the problem (6.156)–(6.158), which does not depend on δ_i, if $i \notin \mathcal{A}(x^*)$. Therefore,

$$\frac{\partial p}{\partial \delta_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial \delta_i} = 0 = -\mu_i(\delta),$$

which concludes the proof. □

We emphasize the importance of the condition requiring that the inactive constraints in the initial problem remain so in the perturbed problem. This is illustrated for Example 6.26 in Figure 6.11. When $\delta = 5$, the solution $x(\delta)$ is such that the constraints $x_1 \geq 0$ and $x_2 \geq 0$, inactive in x^*, remain inactive in $x(\delta)$. However, when $\delta = 16$, the constraint $x_1 \geq 0$ becomes active in $x(\delta)$ and we leave the domain of application of the theorem of sensitivity.

6.5 Linear optimization

We now analyze in greater detail the optimality conditions for the linear optimization problem

$$\min_{x \in \mathbb{R}^n} c^T x \tag{6.159}$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0\,, \end{aligned} \tag{6.160}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, for which the Lagrangian function is

$$L(x,\lambda,\mu) = c^T x + \lambda^T(b - Ax) - \mu^T x\,, \tag{6.161}$$

with $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^n$. By directly applying Theorem 6.13, the necessary first-order optimality condition is expressed as

$$\begin{aligned} \nabla L(x,\lambda,\mu) = c - A^T\lambda - \mu &= 0 \\ \mu &\geq 0\,. \end{aligned} \tag{6.162}$$

These conditions represent exactly the constraints (4.23) of the dual problem described in Section 4.2. The second-order conditions are trivial, because $\nabla^2_{xx} L(x,\lambda,\mu) = 0$, for all (x,λ,μ). The complementarity condition (6.57) is simply written as

$$\mu_i x_i = 0\,, \qquad i = 1,\ldots,n, \tag{6.163}$$

or, equivalently,

$$(c_i - \sum_{j=1}^{m} a_{ji}\lambda_j)x_i = 0\,, \qquad i = 1,\ldots,n. \tag{6.164}$$

We show below that this condition happens to also be sufficient for optimality.

We can also utilize the necessary conditions of Theorem 5.1. In particular, if we consider the j^{th} basic direction d_j (Definition 3.42), the directional derivative of the objective function in the direction d_j is given by

$$\nabla f(x^*)^T d_j = c^T d_j = c_B^T (d_j)_B + c_N^T (d_j)_N = -c_B^T B^{-1} A_j + c_j\,. \tag{6.165}$$

In the context of linear optimization, this quantity is often called *reduced cost*.

Definition 6.28 (Reduced costs)**.** Consider the linear optimization problem (6.159)–(6.160) and let x be a feasible basic solution of the constraint polyhedron. The reduced cost of x_j is

$$\bar{c}_j = c_j - c_B^T B^{-1} A_j\,, \qquad j = 1,\ldots,n\,. \tag{6.166}$$

In matrix form, we have

$$\bar{c} = c - A^T B^{-T} c_B. \tag{6.167}$$

The reduced costs can be decomposed into their basic and non basic components, as follows:

$$\bar{c}_B = c_B - B^T B^{-T} c_B = 0, \tag{6.168}$$

and

$$\bar{c}_N = c_N - N^T B^{-T} c_B. \tag{6.169}$$

Therefore, for any basis B, the basic components of the reduced costs is always 0. This, together with the geometric interpretation of the non basic components, is formalized in the next theorem.

Theorem 6.29 (Reduced costs). *Consider the linear problem (6.159)–(6.160) and let x be a basic solution of the constraint polyhedron. When j is the index of a non basic variable, the j^{th} reduced cost is the directional derivative of the objective function in the j^{th} basic direction. When j is the index of a basic variable, the j^{th} reduced cost is zero.*

Proof. In the case where j is non basic, the proof can be found above (see (6.165)). For basic j, we see that $B^{-1}B = I$ and $B^{-1}A_j$ is the j^{th} column of the identity matrix. Then, $c_B^T B^{-1} A_j = c_j$ and the reduced cost is zero. □

The concept of reduced costs now enables us to state the optimality conditions for linear optimization.

Theorem 6.30 (Necessary optimality conditions, linear optimization). *Consider the linear problem (6.159)–(6.160) and let x^* be a non degenerate basic solution of the constraint polyhedron. If x^* is the solution to (6.159)–(6.160), then $\bar{c} \geq 0$.*

Proof. Consider the basic direction d_k. According to Theorem 3.44, the non degeneracy of x^* ensures that d_k is feasible. Therefore, given the convexity of the set of constraints, the necessary condition of Theorem 6.3 applies and

$$\nabla f(x^*)^T d_k = \bar{c} \geq 0,$$

by using (6.166) and Theorem 6.29. □

Theorem 6.31 (Sufficient conditions, linear optimization). *Consider the linear problem (6.159)–(6.160) and let x^* be a feasible basic solution of the constraint polyhedron. If $\bar{c} \geq 0$, then x^* is optimal.*

Proof. Let y be an arbitrary feasible point and $w = y - x^*$. Since the feasible set is convex, w is a feasible direction (Theorem 3.11). If d_j is the j^{th} basic direction[2] (Definition 3.42), we have

$$\begin{aligned} c^T w &= \sum_{j \in \mathcal{N}} (w)_j c^T d_j && \text{from Theorem 3.45} \\ &= \sum_{j \in \mathcal{N}} (w)_j (-c_B^T B^{-1} A_j + c_j) && \text{from Definition 3.42} \\ &= \sum_{j \in \mathcal{N}} (w)_j \bar{c}_j && \text{from Definition 6.28.} \end{aligned}$$

[2] In this proof, d_j is a vector of $\mathbb{R}^n$, while $(w)_j$ is a scalar, representing the jth entry of the vector w.

Since x^* is a basic solution, $j \in \mathcal{N}$ implies that $x_j^* = 0$ according to Definition 3.38. Therefore, $(w)_j = y_j - x_j^* = y_j \geq 0$ by feasibility of y. Then, as the reduced costs are non negative, we obtain

$$c^T y - c^T x^* = c^T w = \sum_{j \in \mathcal{N}} (w)_j \bar{c}_j \geq 0,$$

which proves the optimality of x^*. □

Note that the sufficient conditions do not assume that x^* is non degenerate. To understand why the necessary conditions may not hold when x^* is degenerate, we go back to the example illustrated in Figure 3.16. In this example, the vertex number 2 corresponds to a degenerate solution, and the basic direction $\widehat{d}_3$ is not feasible. Therefore, if this vertex happens to be the optimal solution of an optimization problem, it is not necessary for the basic direction $\widehat{d}_3$ to correspond to an ascent direction. It does not matter if it is a descent direction, as the direction is not feasible anyway. If it happens to be a descent direction, the reduced cost is negative, and the necessary condition is not verified although we are at the optimal solution.

We now characterize the optimal solution of the dual problem given the optimal solution of the primal. We obtain an important result for linear optimization, called *strong duality*, that the optimal value of the primal coincides with the optimal value of the dual. Moreover, this result provides a direct link between the reduced costs and the dual variables.

Corollary 6.32 (Optimality of the dual). *Consider the primal linear problem (6.159)–(6.160) and let* B *be a basis such that* $B^{-1}b \geq 0$ *and* $\bar{c} \geq 0$. *Consider also the dual problem*

$$\max_{\lambda \in \mathbb{R}^m} \lambda^T b \tag{6.170}$$

subject to

$$A^T \lambda \leq c. \tag{6.171}$$

Then, the primal vector x^* *with basic variables*

$$x_B^* = B^{-1} b \tag{6.172}$$

and non basic variables $x_N^* = 0$, *is optimal for the primal problem, the dual vector*

$$\lambda^* = B^{-T} c_B \tag{6.173}$$

is optimal for the dual problem, and the objective functions are equal, that is

$$(\lambda^*)^T b = c^T x^*. \tag{6.174}$$

Proof. The optimality of x^* is guaranteed by Theorem 6.31. We have also

$$\begin{aligned}(\lambda^*)^T b &= c_B^T B^{-1} b && \text{from (6.173)}\\ &= c_B^T x_B^* && \text{from (6.172)}\\ &= c^T x^* && \text{as } x_N^* = 0,\end{aligned}$$

proving (6.174).

The vector λ^* is feasible for the dual. Indeed, from (6.167), we have

$$A^T \lambda^* = A^T B^{-T} c_B = c - \bar{c}.$$

As $\bar{c} \geq 0$, we obtain (6.171).

Consider now any dual feasible λ. By the weak duality theorem (Theorem 4.9),

$$\lambda^T b \leq c^T x^* = (\lambda^*)^T b,$$

which proves the optimality of λ^* for the dual problem. □

The above result leads to an important result called *strong duality*. Consider x^* an optimal solution of the primal problem. If x^* is non degenerate, the condition $\bar{c} \geq 0$ is a sufficient and necessary condition for its optimality. If it is degenerate, it can be shown that there exists a basis B such that $x_B^* = B^{-1} b \geq 0$ and $\bar{c} = c - A^T B^{-T} c_B \geq 0$. The idea is that the simplex algorithm described in Chapter 16, combined with appropriate rules attributed to Bland (1977) terminates in a finite number of iterations with an optimal basis and non negative reduced costs.

Theorem 6.33 (Strong duality). *Consider the primal linear problem*

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to

$$\begin{aligned} Ax &= b\\ x &\geq 0, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, *and the dual problem*

$$\max_{\lambda \in \mathbb{R}^m} \lambda^T b$$

subject to

$$A^T \lambda \leq c.$$

If either the primal or the dual problem has an optimal solution, so does the other, and the optimal objective values are equal.

Proof. From the discussion above, if x^* is a solution to the primal problem, there exists a basis B such that $x_B^* = B^{-1} b \geq 0$ and $\bar{c} = c - A^T B^{-T} c_B \geq 0$. Therefore, Corollary 6.32 applies, the optimal solution of the dual is $B^{-T} c_B$, and the objective functions are equal.

If λ^* is a solution to the dual problem, the fact that the primal problem is the dual of the dual (Theorem 4.15) is used to prove the result. □

Note that another proof of this result, exploiting Farkas' lemma, is presented in Theorem 4.17. Finally, we show that the complementarity condition (6.164) is a sufficient and necessary optimality condition.

Theorem 6.34 (Complementarity slackness). *Consider the primal linear problem*

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to

$$Ax = b$$
$$x \geq 0,$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, *and the dual problem*

$$\max_{\lambda \in \mathbb{R}^m} \lambda^T b$$

subject to

$$A^T \lambda \leq c.$$

Consider x^* *primal feasible and* λ^* *dual feasible.* x^* *is optimal for the primal and* λ^* *is optimal for the dual if and only if*

$$(c_i - \sum_{j=1}^{m} a_{ji}\lambda_j)x_i = 0, \qquad i = 1, \ldots, n. \tag{6.175}$$

Proof. Conditions (6.175) are KKT necessary optimality conditions (see Theorem 6.13 and the discussion at the beginning of the section). To show that they are sufficient, consider the equation

$$(c - A^T\lambda^*)^T x^* = \sum_{i=1}^{n} (c_i - \sum_{j=1}^{m} a_{ji}\lambda_j)x_i = 0.$$

Therefore,

$$c^T x^* = (\lambda^*)^T A x^*.$$

As x^* is primal feasible, we have $Ax^* = b$ and

$$c^T x^* = (\lambda^*)^T b.$$

Consequently, the objective function of the primal at x^* equals the objective function of the dual at λ^*. We apply Theorem 4.11 to prove the optimality of x^* and λ^*. □

Conditions (6.175) are called *complementarity slackness* conditions because the activity of the constraints must be complementary. At the optimal solution, if a

primal variable is positive, that is if $x_i > 0$, the corresponding dual constraint must be active, that is $c_i - \sum_{j=1}^{m} a_{ji}\lambda_j$. Symmetrically, if a dual constraint is inactive, that is if $c_i > \sum_{j=1}^{m} a_{ji}\lambda_j$, the corresponding primal variable must be equal to 0.

6.6 Quadratic optimization

We now consider a case of quadratic optimization with equality constraints:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + c^T x \tag{6.176}$$

subject to

$$Ax = b\,, \tag{6.177}$$

where $Q \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. The Lagrangian function is

$$L(x,\lambda) = \frac{1}{2} x^T Q x + c^T x + \lambda^T (b - Ax)\,, \tag{6.178}$$

with $\lambda \in \mathbb{R}^m$. By directly applying Theorem 6.13, the necessary first-order optimality condition is written as

$$\nabla_x L(x,\lambda) = Qx + c - A^T\lambda = 0\,. \tag{6.179}$$

By combining (6.177) and (6.179), we obtain the linear system

$$\begin{pmatrix} Q & -A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -c \\ b \end{pmatrix}. \tag{6.180}$$

We demonstrate the case where this system has a unique solution.

Lemma 6.35. *Consider the quadratic problem (6.176)–(6.177), with A of full rank. Let $Z \in \mathbb{R}^{n\times(n-m)}$ be a matrix for which the columns form a basis of the null space of A, i.e., $AZ = 0$, and Z is of full rank. If the reduced Hessian matrix Z^TQZ is positive definite, then the system (6.180) is non singular and has a unique solution (x^*, λ^*).*

Proof. Consider x and λ such that

$$\begin{pmatrix} Q & -A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

i.e., $Qx = A^T\lambda$ and $Ax = 0$. We demonstrate that x and λ are zero in order to prove that the matrix is non singular. Since $Ax = 0$, we have

$$0 = \begin{pmatrix} x^T & \lambda^T \end{pmatrix} \begin{pmatrix} Q & -A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = x^T Q x\,.$$

Since Z is of full rank, there exists y such that $x = Zy$. Therefore,

$$y^T Z^T Q Z y = 0 .$$

Since $Z^T Q Z$ is positive definite, then $y = 0$. As a result, $x = Zy = 0$ and the first equation is written as

$$Qx - A^T \lambda = -A^T \lambda = 0 .$$

Since A is of full rank, then $\lambda = 0$. □

We calculate the analytical solution to this problem.

Lemma 6.36. *Consider the quadratic problem (6.176)–(6.177) with* $Q = I$ *and* $b = 0$, *i.e.,*

$$\min \frac{1}{2} x^T x + c^T x$$

subject to

$$Ax = 0$$

where A *is of full rank. The solution to this problem is*

$$x^* = A^T \left(AA^T\right)^{-1} Ac - c \tag{6.181}$$

$$\lambda^* = \left(AA^T\right)^{-1} Ac . \tag{6.182}$$

Proof. The system (6.180) is written as

$$\begin{pmatrix} I & -A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} -c \\ 0 \end{pmatrix} .$$

By multiplying the first equation

$$x^* - A^T \lambda^* = -c , \tag{6.183}$$

by A, we obtain

$$Ax^* - AA^T \lambda^* = -Ac .$$

Since $Ax^* = 0$ and A is of full rank, we obtain (6.182). We now need only introduce (6.182) in (6.183) to obtain (6.181). □

Lemma 6.37. *Consider the quadratic problem (6.176)–(6.177) with* $Q = I$, *i.e.,*

$$\min_x \frac{1}{2} x^T x + c^T x$$

subject to

$$Ax = b$$

where A is of full rank. The solution to this problem is

$$x^* = A^T(AA^T)^{-1}(Ac + b) - c \tag{6.184}$$

$$\lambda^* = (AA^T)^{-1}(Ac + b) \tag{6.185}$$

Proof. Consider x_0 such that $Ax_0 = b$ and let $y = x - x_0$, i.e., $x = y + x_0$. The problem becomes

$$\min_y \frac{1}{2}(y^T y + x_0^T x_0 + 2y^T x_0) + c^T y + x^T x_0$$

subject to

$$Ay + Ax_0 = b\,.$$

By removing the constant terms of the objective function and using $Ax_0 = b$, we obtain

$$\min_y \frac{1}{2} y^T y + (c + x_0)^T y$$

subject to

$$Ay = 0\,.$$

According to Theorem 6.36, the solution to this problem is

$$y^* = A^T(AA^T)^{-1}A(c + x_0) - (c + x_0)$$

$$\lambda^* = (AA^T)^{-1}A(c + x_0)\,.$$

We now need only use $Ax_0 = b$ and $y^* = x^* - x_0$ to obtain the result. □

Theorem 6.38 (Analytical solution of a quadratic problem)**.** *Consider the quadratic problem (6.176)–(6.177) $\min_{x\in\mathbb{R}^n} \frac{1}{2}x^T Qx + c^T x$ subject to $Ax = b$, where A is of full rank. If the matrix Q is positive definite, then the system (6.180) is non singular and has a unique solution (x^*, λ^*) given by*

$$x^* = Q^{-1}(A^T\lambda^* - c) \tag{6.186}$$

and

$$\lambda^* = (AQ^{-1}A^T)^{-1}(AQ^{-1}c + b)\,. \tag{6.187}$$

Proof. Let $Z \in \mathbb{R}^{n\times(n-m)}$ be a matrix where the columns form a basis of the null space of A, i.e., such that $AZ = 0$. Since Q is positive definite, then so is $Z^T QZ$, and Theorem 6.35 applies to demonstrate the non singularity of the system and the unicity of the solution. Let L be an lower triangular matrix such that $Q = LL^T$ and let us take $y = L^T x$. The problem (6.176)–(6.177) is thus written as

$$\min_y \frac{1}{2} y^T L^{-1} L L^T L^{-T} y + c^T L^{-T} y = \frac{1}{2} y^T y + c^T L^{-T} y$$

subject to

$$AL^{-T}y = b\,.$$

The solution to this problem is given by Theorem 6.37, by replacing c with $L^{-1}c$ and A with AL^{-T}. Then,

$$\lambda^* = \left(AL^{-T}L^{-1}A^T\right)^{-1}(AL^{-T}L^{-1}c + b) = (AQ^{-1}A^T)^{-1}(AQ^{-1}c + b)$$

and

$$y^* = L^{-1}A^T\lambda^* - L^{-1}c\,.$$

We now need only take $y^* = L^T x^*$ to obtain the result. □

The presentation of the proof of Theorems 6.10, 6.13, 6.19, 6.20, and 6.24 was inspired by Bertsekas (1999). That of the proof of Theorem 6.31 was inspired by Bertsimas and Tsitsiklis (1997).

6.7 Exercises

Exercise 6.1.

Identify the local optima of the following optimization problems, and verify the optimality conditions.

1. $\min_{x\in\mathbb{R}^n} \|x\|_2^2$, subject to $\sum_{i=1}^n x_i = 1$.
2. $\min_{x\in\mathbb{R}^n} \sum_{i=1}^n x_i$, subject to $\|x\|_2^2 = 1$.
3. $\min_{x\in\mathbb{R}^2} -x_1^2 - x_2^2$, subject to $(x_1/2)^2 + (x_2/2)^2 \leq 1$ (Hint: plot the level curves and the constraints).
4. $\min_{x\in\mathbb{R}^2} -x_1^2 - x_2^2$, subject to $-x_1^2 + x_2^2 \leq 1$, and $-5 \leq x_1 \leq 5$ (Hint: plot the level curves and the constraints).
5. The Indiana Jones problem (Section 1.1.6): $\min_{x\in\mathbb{R}^2} x_1^2 + x_2^2$, subject to $x_1x_2 - hx_1 - \ell x_2 = 0, x_1 \geq \ell, x_2 \geq h$.

Exercise 6.2.

An electricity company must supply a town that consumes 100 MWh daily. Three plants are used to generate the energy: a gas plant, producing at the cost of €800/MWh, a coal plant, producing at the cost of €1,500/MWh, and a hydroelectric plant producing at the cost of €300/MWh. The amount of available water limits the production of the latter plant to 40 MWh per day. Moreover, due to ecological concerns, the two other plants are limited to produce no more than 80 MWh per day each.

1. Formulate a linear optimization problem that would optimize the costs of the company.
2. Formulate the dual problem.

3. Prove, using the optimality conditions, that the optimal solution is to produce 60 MWh per day with the gas plant, 40 MWh per day with the hydroelectric plant, and not to use the coal plant.
4. Deduce the optimal values of the dual variables.
5. Use sensitivity analysis to propose profitable investments to the company.

Exercise 6.3. Consider the optimization problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to

$$\sum_{i=1}^{n} x_i = 1 \text{ and } x \geq 0.$$

Let x^* be a local minimum of f.

1. Prove that, if $x_i^* > 0$, then

$$\frac{\partial f(x^*)}{\partial x_i} \leq \frac{\partial f(x^*)}{\partial x_j} \ \forall j. \tag{6.188}$$

   (Hint: refer to Example 6.4).
2. Show that, if f is convex, condition (6.188) is sufficient. (Hint: Define $\Delta = \min_i \partial f(x^*)/\partial x_i$).

Exercise 6.4 (Slack variables)**.** Consider problem (P1)

$$\min_{x\in\mathbb{R}^n} f(x) \tag{6.189}$$

subject to

$$g_i(x) \leq 0,\ i = 1,\ldots,m,$$

and problem (P2)

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

subject to

$$\begin{aligned} g_i(x) + y_i &= 0,\ i = 1,\ldots,m, \\ y_i &\geq 0,\ i = 1,\ldots,m. \end{aligned}$$

1. Write the necessary optimality conditions (KKT) for problem (P1), both first and second order.
2. Write the necessary optimality conditions (KKT) for problem (P2), both first and second order.
3. Prove that (x^*, μ^*) verifies the KKT conditions of problem P1, if and only if (x^*, y^*, λ^*) verifies the KKT conditions of problem P2, where $\lambda^* = \mu^*$ and $y_i^* = -g_i(x^*)$, $i = 1,\ldots,m$ (Hint: refer to Example 6.16).

Part III

Solving equations

Equations are more important to me, because politics is for the present, but an equation is something for eternity.

Albert Einstein

We have seen that the necessary optimality conditions enable us to identify the critical points (Definition 5.6) that are candidates for the solution to an optimization problem. In the case of optimization without constraint, we use condition (5.1). For constrained optimization, we use conditions (6.11), (6.23), and (6.55)–(6.57). One way to address the problem is to solve the system of equations defined by these conditions. This is how the problem in Example 5.8 was solved, as well as Example 6.22. In these two cases, the system of equations is easily solved. This is not always the case, as illustrated in Example 6.23.

We now consider numerical methods enabling us to solve such systems of non linear equations. Even though these are not directly utilized for optimization, they are the basis for the main algorithms.

Chapter 7

Newton's method

Contents

Newton's method plays a crucial role in the context of solving non linear equations and, by extension, in that of non linear optimization. Isaac Newton was inspired by a method from Vieta, and the method was later on improved by Raphson (and sometimes called "Newton-Raphson.") We refer the reader to Deuflhard (2012) for a historical perspective of the method. We introduce it for the simple problem of solving one equation of one unknown, i.e., by deriving numerical methods to find $x \in \mathbb{R}$ such that $F(x) = 0$.

7.1 Equation with one unknown

Let $F : \mathbb{R} \to \mathbb{R}$ be a real differentiable function of one variable. In order to solve the equation $F(x) = 0$, the main idea of Newton's method consists in simplifying the problem. Since a non linear equation is complicated to solve, it is replaced by a linear equation. The concept of replacing a difficult problem with a simpler one is used throughout this book. We use the term *model* when referring to a function that is a simplification of another.

To obtain this simplified equation, we invoke Taylor's Theorem C.1 which ensures that a differentiable function can be approximated at a point by a straight line and that the magnitude of the error decreases with the distance to this point.

Isaac Newton was born prematurely and fatherless on December 25, 1642, in Woolsthorpe, England. (As 11 days were dropped in September 1752 to adjust the calendar, the date of his birth in the "new style" calendar, that is, January 4, 1643, is sometimes reported.) He is considered as the father of modern analysis, especially thanks to his study on differentiable functions, and infinitesimal calculus (that he called "fluxions"). His most famous work, published in *Philosophiae naturalis principia mathematica*, concerns the theory of gravitation and associated principles (inertia, action-reaction, tides, etc.) He is considered as the founder of celestial mechanics. Newton claimed that the fall of an apple inspired in him the concept of gravitation. Some dispute him being the father of these findings, rather attributing the fundamental ideas to Robert Hooke. Newton accused Leibniz (apparently wrongfully) of having plagiarized his work. He was the first British scientist to be knighted, on April 16, 1705, by Queen Anne. He died on March 20, 1727, in London. One of his most famous quotes is "If I have seen further than others, it is by standing upon the shoulders of giants." He is buried in Westminster Abbey, with the following inscription on his grave: "Hic depositum est, quod mortale fuit Isaaci Newtoni". (Here lies that which was mortal of Isaac Newton).

Figure 7.1: Sir Isaac Newton

Example 7.1 (Linear model)**.** Take the function

$$F(x) = x^2 - 2$$

and the point $\widehat{x} = 2$. According to Taylor's theorem, for any $d \in \mathbb{R}$, we have

$$\begin{aligned} F(\widehat{x} + d) &= F(\widehat{x}) + dF'(\widehat{x}) + o\big(|d|\big) \\ &= \widehat{x}^2 - 2 + 2\widehat{x}d + o\big(|d|\big) \\ &= 2 + 4d + o\big(|d|\big)\,. \end{aligned}$$

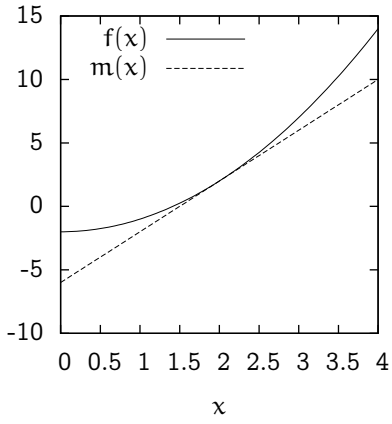
The linear model is obtained by ignoring the error $o\big(|d|\big)$:

$$m(\widehat{x} + d) = 2 + 4d\,.$$

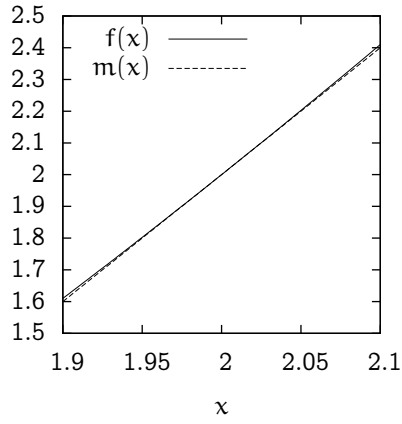
Defining $x = \widehat{x} + d$, we get

$$m(x) = 2 + 4(x - 2) = 4x - 6\,.$$

The function and the model are presented in Figure 7.2(a). The zoom in Figure 7.2(b) illustrates the good agreement between the model and the function around $\widehat{x} = 2$.



(a) $\widehat{x} = 2$



(b) Zoom

Figure 7.2: Linear model of $x^2 - 2$

We can now provide a general definition of the linear model of a non linear function.

Definition 7.2 (Linear model of a function with one variable)**.** Let $F : \mathbb{R} \to \mathbb{R}$ be a differentiable function. The linear model of F in $\widehat{x}$ is a function $m_{\widehat{x}} : \mathbb{R} \to \mathbb{R}$ defined by

$$m_{\widehat{x}}(x) = F(\widehat{x}) + (x - \widehat{x})F'(\widehat{x})\,. \tag{7.1}$$

From a first approximation $\widehat{x}$, the main idea of Newton's method in order to find the root of the function f consists in

1. calculating the linear model in $\widehat{x}$,
2. calculating the root x^+ of this linear model,
3. if x^+ is not the root of f, considering x^+ as a new approximation and starting over.

According to Definition 7.2, the root of the linear model is the solution to

$$F(\widehat{x}) + (x - \widehat{x})F'(\widehat{x}) = 0\,, \tag{7.2}$$

i.e., if $F'(\widehat{x}) \neq 0$,

$$x^+ = \widehat{x} - \frac{F(\widehat{x})}{F'(\widehat{x})}\,, \tag{7.3}$$

which summarizes the first two steps presented above.

We also need to specify the third step. How do we conclude that x^+ is a root of the function, i.e., $F(x^+) = 0$, and that we can stop the iterations? Seemingly innocuous, this question is far from simple to answer. Indeed, computers operating in finite arithmetic are not capable of representing all real numbers (that represent an uncountable infinity). Therefore, it is possible, and even common, that the method never generates a point x^+ such that $F(x^+) = 0$ exactly. Then, we must often settle for a solution x^+ such that $F(x^+)$ is "sufficiently close" to 0. In practice, the user provides a measure of this desired proximity, denoted by ε and the algorithm is interrupted when

$$\left|F(x^+)\right| \leq \varepsilon\,. \tag{7.4}$$

A typical value for ε is $\sqrt{\varepsilon_M}$, where ε_M is the *machine epsilon*, that is, an upper bound on the relative error due to rounding in floating point arithmetic. A simple way to compute ε_M is Algorithm 7.1. The loop stops when ε_M is so small that, when added to 1, the result is also 1.

Algorithm 7.1: Machine epsilon

```
Objective
    Find the machine epsilon ε_M.
Initialization
    ε_M := 1.
while 1 + ε_M ≠ 1 do
    ε_M := ε_M/2.
```

Typical values are

- $\varepsilon_M = 5.9605\ 10^{-8}$ for single precision floating point (so that $\varepsilon = \sqrt{\varepsilon_M} = 2.4414\ 10^{-4}$), and
- $\varepsilon_M = 1.1102\ 10^{-16}$ for double precision floating point (so that $\varepsilon = \sqrt{\varepsilon_M} = 1.0537\ 10^{-8}$).

We now have all the elements in order to write Newton's algorithm to solve an equation with one unknown (Algorithm 7.2).

Abu Ja'far Muhammad ibn Musa Al-Khwarizmi was a Persian mathematician born before AD 800. Only a few details about his life can be gleaned from Islamic literature. His name appears to indicate that he was from the State of Khwarazm or Khorezm (currently Khiva in Uzbekhistan). However, other sources suggest that he was born between the Tigris and Euphrates in the Baghdad area. Al Khwarizmi was an astronomer in the House of Wisdom (Dar al-Hikma) of caliph Abd Allah al Mahmoun. He is primarily known for his treatise *al Kitab almukhtasar fi hisab al-jabr w'al muqabala* (that can be translated as "The Compendious Book on Calculation by Completion and Balancing"), which provides the origin of the word *algebra* (al-Jabr, used in the sense of transposition, became algebra). He explained in Arabic the system of Indian decimal digits applied to arithmetic operations. The Latin translation of this work, entitled *Algoritmi de numero Indorum* gave rise to the word *algorithm*. Al Khwarizmi died after AD 847.

Figure 7.3: Al Khwarizmi

Algorithm 7.2: Newton's method: one variable

Objective
  Find (an approximation of) a solution to the equation $F(x) = 0$.
Input
  The function $F : \mathbb{R} \to \mathbb{R}$.
  The derivative of the function $F' : \mathbb{R} \to \mathbb{R}$.
  A first approximation of the solution $x_0 \in \mathbb{R}$.
  The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.
Output
  An approximation $x^* \in \mathbb{R}$ to the solution.
Initialization
  $k := 0$.
Repeat
  $x_{k+1} := x_k - F(x_k)/F'(x_k)$,
  $k := k + 1$.
Until $|F(x_k)| \leq \varepsilon$
$x^* = x_k$

Example 7.3 (Newton's method: one variable – I)**.** Take the equation

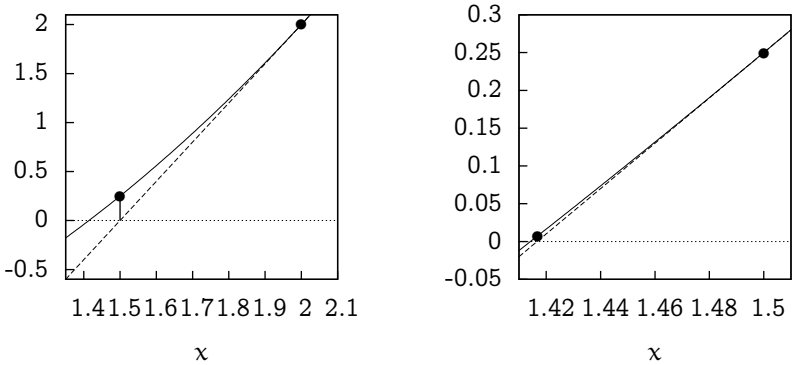
$$F(x) = x^2 - 2 = 0\,.$$

We have $F'(x) = 2x$. We apply Newton's method (Algorithm 7.2) with $x_0 = 2$, and $\varepsilon = 10^{-15}$. The iterations are listed in Table 7.1. The first two iterations are portrayed in Figure 7.4. Figure 7.4(a) represents the first iteration, where $x_0 = 2$.

The linear model at x_0 is represented by a dotted line. It intersects the x-axis at $x_1 = 1.5$. Figure 7.4(b) represents the second iteration, where $x_1 = 1.5$. The linear model at x_1 is represented by a dotted line. It intersects the x-axis at $x_2 = 1.4666$.

Table 7.1: Iterations with Newton's method for Example 7.3

k	x_k	$F(x_k)$	$F'(x_k)$
0	+2.00000000E+00	+2.00000000E+00	+4.00000000E+00
1	+1.50000000E+00	+2.50000000E-01	+3.00000000E+00
2	+1.41666667E+00	+6.94444444E-03	+2.83333333E+00
3	+1.41421569E+00	+6.00730488E-06	+2.82843137E+00
4	+1.41421356E+00	+4.51061410E-12	+2.82842712E+00
5	+1.41421356E+00	+4.44089210E-16	+2.82842712E+00



(a) First iteration (b) Second iteration

Figure 7.4: Newton's method for Example 7.3

According to Example 7.3, Newton's method seems quite fast, as only 5 iterations were necessary to converge. We characterize this speed below. Before that, however, we illustrate by other examples that the method does not always work that well.

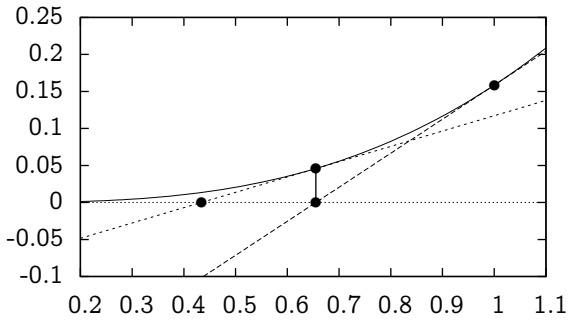
Example 7.4 (Newton's method: one variable – II)**.** Take the equation

$$F(x) = x - \sin x = 0\,.$$

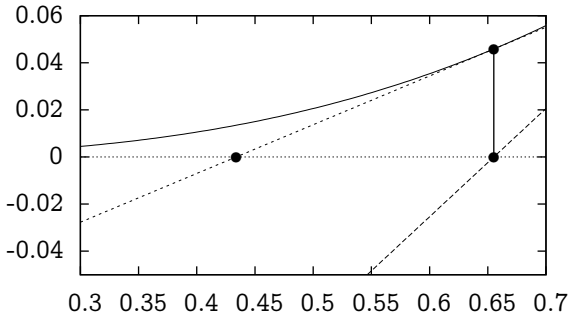
We have $F'(x) = 1 - \cos x$. We apply Newton's method (Algorithm 7.2) with $x_0 = 1$ and $\varepsilon = 10^{-15}$. The iterations are listed in Table 7.2. The number of iterations is much larger than for the previous example. Note how the derivative $F'(x_k)$ is getting closer and closer to 0 as the iterations proceed. Actually, the root of this equation is $x^* = 0$, and the value of the derivative at the root is 0. As Newton's method divides by $F'(x_k)$ at each iteration, the fact that $F'(x^*) = 0$ is the source of the slow behavior of the method. The first two iterations are portrayed in Figure 7.5(a). The linear model at the starting point $x_0 = 1$ is represented by a dotted line and intersects the x-axis at 0.65, which is the first iterate.

Table 7.2: Iterations with Newton's method for Example 7.4

k	x_k	$F(x_k)$	$F'(x_k)$
0	+1.00000000E+00	+1.58529015E-01	+4.59697694E-01
1	+6.55145072E-01	+4.58707860E-02	+2.07040452E-01
2	+4.33590368E-01	+1.34587380E-02	+9.25368255E-02
3	+2.88148401E-01	+3.97094846E-03	+4.12282985E-02
4	+1.91832312E-01	+1.17439692E-03	+1.83434616E-02
⋮			
25	+3.84171966E-05	+9.44986548E-15	+7.37940486E-10
26	+2.56114682E-05	+2.79996227E-15	+3.27973648E-10
27	+1.70743119E-05	+8.29617950E-16	+1.45766066E-10



(a) Two iterations



(b) Zoom

Figure 7.5: Newton's method for Example 7.4

The linear model at that point is also represented by a dotted line, and intersects the x-axis at 0.43, which is the second iterate. Figure 7.5(b) is a zoom on the same figure.

Even though Newton's method has managed to provide the desired precision in 5 iterations for Example 7.3, more than 5 times as many iterations are necessary for Example 7.4. In the following Example, we see that the method may sometimes not work at all.

Example 7.5 (Newton's method: one variable – III)**.** Take equation

$$F(x) = \arctan x = 0\,.$$

We have $F'(x) = 1/(1+x^2)$. We apply Newton's method (Algorithm 7.2) with $x_0 = 1.5$ and $\varepsilon = 10^{-15}$. The first 10 iterations are listed in Table 7.3. We note that the absolute value of x_k increases with each iteration, that the value of $F(x_k)$ seems to oscillate, and that the value of $F'(x_k)$ closes in on 0. Therefore, not only does the algorithm not approach the solution, but when the derivative approaches 0, the main iteration cannot be performed due to the division by 0. The first three iterations are portrayed in Figure 7.6.

Table 7.3: The ten first iterations with Newton's method for Example 7.5

k	x_k	$F(x_k)$	$F'(x_k)$
0	+1.50000000E+00	+9.82793723E-01	+3.07692308E-01
1	-1.69407960E+00	-1.03754636E+00	+2.58404230E-01
2	+2.32112696E+00	+1.16400204E+00	+1.56552578E-01
3	-5.11408784E+00	-1.37769453E+00	+3.68271300E-02
4	+3.22956839E+01	+1.53984233E+00	+9.57844131E-04
5	-1.57531695E+03	-1.57016153E+00	+4.02961851E-07
6	+3.89497601E+06	+1.57079607E+00	+6.59159364E-14
7	-2.38302890E+13	-1.57079633E+00	+1.76092712E-27
8	+8.92028016E+26	+1.57079633E+00	+1.25673298E-54
9	-1.24990460E+54	-1.57079633E+00	+6.40097701E-109
10	+2.45399464E+108	+1.57079633E+00	+1.66055315E-217

We now analyze in detail the aspects that influence the efficiency of the method. The main result can be stated as follows:

- if the function is not too non linear,
- if the derivative of F at the solution is not too close to 0,
- if x_0 is not too far from the root,
- then Newton's method converges quickly toward the solution.

The central idea of the analysis is to measure the error that is committed when the non linear function is replaced by the linear model. Intuitively, if the function is almost linear, the error is small. While if the function is highly non linear, the error is more significant. We use here the Lipschitz continuity of the derivative of F to characterize the non linearity, as discussed in Section 2.4 (Definition 2.27). Theorem 7.6 considers a linear model at $\widehat{x}$, and provides an upper bound on the error

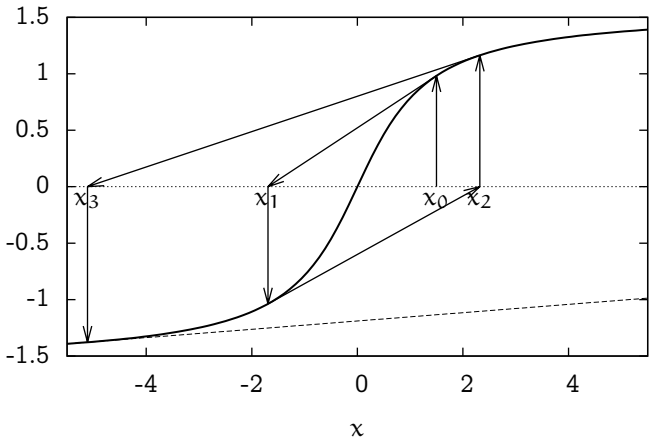


Figure 7.6: Newton's method for Example 7.5

at a point x^+. This bound depends on the distance between $\widehat{x}$ and x^+, and on the Lipschitz constant that characterizes the nonlinearity of the function.

Theorem 7.6 (Error of the linear model: one variable)**.** *Consider an open interval* $X \subseteq \mathbb{R}$ *and a function* F *for which the derivative is Lipschitz continuous over* X*, where* M *is the Lipschitz constant. So, for all* $\widehat{x}$*,* $x^+ \in X$*,*

$$\left|F(x^+) - m_{\widehat{x}}(x^+)\right| \leq M\frac{(x^+ - \widehat{x})^2}{2}\,. \tag{7.5}$$

Proof. We have

$$\begin{aligned}
\int_{\widehat{x}}^{x^+} \left(F'(z) - F'(\widehat{x})\right) dz &= \int_{\widehat{x}}^{x^+} F'(z)\,dz - F'(\widehat{x})\int_{\widehat{x}}^{x^+} dz && \text{linearity of the integral}\\
&= F(x^+) - F(\widehat{x}) - F'(\widehat{x})(x^+ - \widehat{x})\\
&= F(x^+) - m_{\widehat{x}}(x^+) && \text{from (7.1)}\,.
\end{aligned}$$

We take $z = \widehat{x} + t(x^+ - \widehat{x})$ and $dz = (x^+ - \widehat{x})\,dt$ to obtain

$$F(x^+) - m_{\widehat{x}}(x^+) = \int_0^1 \left(F'\left(\widehat{x} + t(x^+ - \widehat{x})\right) - F'(\widehat{x})\right)(x^+ - \widehat{x})\,dt\,.$$

Therefore,

$$
\begin{aligned}
&\left|F(x^+) - m_{\widehat{x}}(x^+)\right| \\
&= \left| \int_0^1 \left(F'\left(\widehat{x} + t(x^+ - \widehat{x})\right) - F'(\widehat{x})\right)(x^+ - \widehat{x})\, dt \right| \\
&\leq \int_0^1 \left|\left(F'\left(\widehat{x} + t(x^+ - \widehat{x})\right) - F'(\widehat{x})\right)\right| |x^+ - \widehat{x}|\, dt \qquad \text{from Theorem C.12} \\
&= |x^+ - \widehat{x}| \int_0^1 \left|\left(F'\left(\widehat{x} + t(x^+ - \widehat{x})\right) - F'(\widehat{x})\right)\right| dt \\
&\leq |x^+ - \widehat{x}| \int_0^1 M\left|t(x^+ - \widehat{x})\right| dt \qquad \text{from Definition 2.27} \\
&= M\left|x^+ - \widehat{x}\right|^2 \int_0^1 t\, dt \\
&= \frac{M}{2}\left(x^+ - \widehat{x}\right)^2 .
\end{aligned}
$$

□

We now use this bound on the error to demonstrate the convergence of Newton's method.

Theorem 7.7 (Convergence of Newton's method: one variable). *Consider an open interval $X \subseteq \mathbb{R}$ and a continuously differentiable function F such that its derivative is Lipschitz continuous over X, and where the Lipschitz constant is M. Assume that there exists $\rho > 0$ such that*

$$\left|F'(x)\right| \geq \rho, \qquad \forall x \in X. \tag{7.6}$$

Assume that there exists $x^* \in X$ such that $F(x^*) = 0$. There then exists $\eta > 0$ such that, if

$$|x_0 - x^*| < \eta \tag{7.7}$$

with $x_0 \in X$, the sequence $\left(x_k\right)_k$ defined by

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}, \qquad k = 0, 1, \ldots, \tag{7.8}$$

is well defined and converges toward x^*. Moreover,

$$|x_{k+1} - x^*| \leq \frac{M}{2\rho}\left|x_k - x^*\right|^2 . \tag{7.9}$$

Proof. We provide a proof by induction. For $k = 0$, x_1 is well defined as $F'(x_0) \neq 0$ by assumption (7.6), as $x_0 \in X$. We have

$$\begin{aligned} x_1 - x^* &= x_0 - \frac{F(x_0)}{F'(x_0)} - x^* && \text{from (7.8)} \\ &= x_0 - x^* - \frac{F(x_0) - F(x^*)}{F'(x_0)} && \text{because } F(x^*) = 0 \\ &= \frac{1}{F'(x_0)} \left(F(x^*) - m_{x_0}(x^*) \right) && \text{from (7.1)}. \end{aligned}$$

Then

$$\begin{aligned} \left|x_1 - x^*\right| &\leq \frac{1}{\left|F'(x_0)\right|} \left|F(x^*) - m_{x_0}(x^*)\right| \\ &\leq \frac{M}{2\left|F'(x_0)\right|} \left|x_0 - x^*\right|^2 && \text{from (7.5)} \\ &\leq \frac{M}{2\rho} |x_0 - x^*|^2 && \text{from (7.6)}, \end{aligned}$$

which proves the result for $k = 0$.

We now need technical constants. Take τ such that $0 < \tau < 1$ and let r be the radius of the largest interval contained in X and centered in x^*. We then create

$$\eta = \min\left(r, \tau \frac{2\rho}{M}\right). \tag{7.10}$$

Therefore, based on the hypothesis (7.7), we have

$$|x_0 - x^*| \leq \eta \leq \tau \frac{2\rho}{M} \tag{7.11}$$

and

$$|x_1 - x^*| \leq \frac{M}{2\rho} \left|x_0 - x^*\right|^2 \leq \frac{M}{2\rho} \tau \frac{2\rho}{M} |x_0 - x^*| = \tau |x_0 - x^*| < \eta,$$

where the last inequality is the result of the fact that $\tau < 1$ and $|x_0 - x^*| < \eta$. Since $|x_1 - x^*| < \eta$, we also have that $|x_1 - x^*| < r$ (according to (7.10)) and $x_1 \in X$. x_1 thus satisfies the same assumptions as x_0. We can now apply the recurrence using the same arguments for x_2, x_3, and so forth. □

We now comment a summarized version of the result of Theorem 7.6:

If the function is not too non linear This assumption is related to the Lipschitz continuity. The closer M is to 0, the less non linear is the function.

If the derivative of F is not too close to 0 This is hypothesis (7.6). If this assumption is not satisfied, the method may not be well defined (division by zero), may not converge, or may converge slowly, as illustrated by Example 7.4.

If x_0 is not too far from the root This is hypothesis (7.7). If x_0 is too far from the root, the method may not converge, as shown in Example 7.5. It is interesting

to take a close look at Definition (7.10) of η, assuming that r (a technical parameter) is sufficiently large such that $\eta = 2\rho\tau/M$. If the function is close to being linear, then M is small and η is large. It means that the set of starting points such that the method converges is large, and we can afford to start from a point x_0 farther away from x^*. In practice, as x^* is not known, it increases the chance of finding a valid starting point.

Newton's method converges quickly toward the solution The speed is characterized by (7.9). At each iteration, the new distance to the solution is of the order of the square of the former. For instance, if the initial error is of the order of 10^{-1}, it only takes three iterations for it to become of the order of 10^{-8}. This is illustrated in Example 7.3, for which the iterations are described in Table 7.1. The method is said to converge q-quadratically.

Definition 7.8 (q-quadratic convergence)**.** Take a sequence $(x_k)_k$ in $\mathbb{R}^n$ that converges toward x^*. The sequence is said to converge q-quadratically toward x^* if there exists $c \geq 0$ and $\widehat{k} \in \mathbb{N}$ such that

$$\|x_{k+1} - x^*\| \leq c\|x_k - x^*\|^2, \qquad \forall k \geq \widehat{k}. \tag{7.12}$$

In Definition 7.8, the prefix q signifies *quotient*. In practice, the other types of convergence are rarely used, and the prefix could be omitted. More details can be found in Ortega and Rheinboldt (1970).

7.2 Systems of equations with multiple unknowns

We now generalize Newton's method for systems of non linear equations with multiple unknowns. The concepts are exactly the same. We start with the definition of the linear model. Although in our context, the function F maps $\mathbb{R}^n$ into $\mathbb{R}^n$, we provide the most general definition for a function from $\mathbb{R}^n$ to $\mathbb{R}^m$.

Definition 7.9 (Linear model of a function with n variables)**.** Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a continuously differentiable function. The linear model of F in $\widehat{x}$ is a function $m_{\widehat{x}} : \mathbb{R}^n \to \mathbb{R}^m$ defined by

$$m_{\widehat{x}}(x) = F(\widehat{x}) + \nabla F(\widehat{x})^T(x - \widehat{x}) = F(\widehat{x}) + J(\widehat{x})(x - \widehat{x}), \tag{7.13}$$

where $\nabla F(\widehat{x}) \in \mathbb{R}^{n\times m}$ is the gradient matrix of F in $\widehat{x}$ (Definition 2.17) and $J(\widehat{x}) = \nabla F(\widehat{x})^T$ is the Jacobian matrix, of dimensions $m \times n$ (Definition 2.18).

As in the case with one variable, we determine a bound for the error committed when replacing the function F by the linear model and finding a result similar to that of Theorem 7.6. The proof is essentially the same.

Theorem 7.10 (Error of the linear model: n variables). *Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a continuously differentiable function over an open convex set $X \subset \mathbb{R}^n$. The Jacobian matrix of F is Lipschitz continuous over X (Definition 2.27, where M is the Lipschitz constant, and the matrix norm is induced by the vector norm; Definition B.27). So, for all $\widehat{x}$, $x^+ \in X$,*

$$\left\|F(x^+) - m_{\widehat{x}}(x^+)\right\| \leq M\frac{\left\|x^+ - \widehat{x}\right\|^2}{2}. \tag{7.14}$$

Proof. The structure of the proof is identical to that of Theorem 7.6. We have

$$\begin{aligned}
&F(x^+) - m_{\widehat{x}}(x^+) \\
&\quad = F(x^+) - F(\widehat{x}) - J(\widehat{x})(x^+ - \widehat{x}) && \text{from (7.13)} \\
&\quad = \int_0^1 J\big(\widehat{x} + t(x^+ - \widehat{x})\big)(x^+ - \widehat{x})\,dt - J(\widehat{x})(x^+ - \widehat{x}) && \text{Theorem C.11} \\
&\quad = \int_0^1 \left\{J\big(\widehat{x} + t(x^+ - \widehat{x})\big) - J(\widehat{x})\right\}(x^+ - \widehat{x})\,dt\,.
\end{aligned}$$

Then,

$$\begin{aligned}
&\left\|F(x^+) - m_{\widehat{x}}(x^+)\right\| \\
&\quad \leq \int_0^1 \left\|J\big(\widehat{x} + t(x^+ - \widehat{x})\big) - J(\widehat{x})\right\| \left\|x^+ - \widehat{x}\right\| dt && \text{Theorem C.12} \\
&\quad \leq \int_0^1 M\left\|t(x^+ - \widehat{x})\right\| \left\|x^+ - \widehat{x}\right\| dt && \text{Definition 2.27} \\
&\quad = M\left\|x^+ - \widehat{x}\right\|^2 \int_0^1 t\,dt \\
&\quad = M\frac{\left\|x^+ - \widehat{x}\right\|^2}{2}.
\end{aligned}$$

□

Newton's method for systems of equations is also essentially the same as for a single variable. It is described by Algorithm 7.3. System (7.16) solved at step 13 of the algorithm is often called the *Newton equations*. Note that we have intentionally not written this step as

$$d_{k+1} = -J(x_k)^{-1}F(x_k).$$

Indeed, from a numerical point of view, the calculation of d_{k+1} must be performed by solving the system of linear equations, not by inverting the Jacobian matrix.

Algorithm 7.3: Newton's method: n variables

Objective

To find (an approximation of) a solution to the system of equations

$$F(x) = 0\,. \tag{7.15}$$

Input

The function $F : \mathbb{R}^n \to \mathbb{R}^n$.

The Jacobian matrix of the function $J : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.

A first approximation of the solution $x_0 \in \mathbb{R}^n$.

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation $x^* \in \mathbb{R}^n$ of the solution.

Initialization

$k = 0$.

Repeat

Calculate d_{k+1} solution of

$$J(x_k)d_{k+1} = -F(x_k)\,. \tag{7.16}$$

$x_{k+1} := x_k + d_{k+1}$.

$k := k + 1$.

Until $\left\|F(x_k)\right\| \leq \varepsilon$

$x^* = x_k$

Example 7.11 (Newton's method: n variables)**.** Consider the system of equations

$$\begin{aligned} (x_1+1)^2 + x_2^2 &= 2 \\ e^{x_1} + x_2^3 &= 2\,. \end{aligned} \tag{7.17}$$

We apply Newton's method with

$$F(x) = \begin{pmatrix} (x_1+1)^2 + x_2^2 - 2 \\ e^{x_1} + x_2^3 - 2 \end{pmatrix} \quad \text{and} \quad J(x) = \begin{pmatrix} 2(x_1+1) & 2x_2 \\ e^{x_1} & 3x_2^2 \end{pmatrix}.$$

If $x_0 = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$, we have

$$F(x_0) = \begin{pmatrix} 3 \\ e-1 \end{pmatrix} \approx \begin{pmatrix} 3 \\ 1.7182 \end{pmatrix}$$

and

$$J(x_0) = \begin{pmatrix} 4 & 2 \\ e & 3 \end{pmatrix}.$$

The iterations of Newton's method are described in Table 7.4, with $\varepsilon = 10^{-15}$, where the first column reports the iteration number, the second column the current iterate, the third the value of the function at the current iterate, and the last its norm. The quadratic convergence of the method is well illustrated in this example. Indeed, the value of x_k converges rapidly to the solution $(0, 1)^T$, and the values of $\|F(x_k)\|$ decrease rapidly toward zero.

Table 7.4: Iterations of Newton's method for Example 7.11

k	x_k	$F(x_k)$	$\|F(x_k)\|$
0	1.00000000e+00	3.00000000e+00	3.45723768e+00
	1.00000000e+00	1.71828182e+00	
1	1.52359213e-01	7.56629795e-01	1.15470870e+00
	1.19528157e+00	8.72274931e-01	
2	-1.08376809e-02	5.19684443e-02	1.14042557e-01
	1.03611116e+00	1.01513475e-01	
3	-8.89664601e-04	1.29445248e-03	3.94232975e-03
	1.00153531e+00	3.72375572e-03	
4	-1.37008875e-06	3.13724882e-06	8.07998556e-06
	1.00000293e+00	7.44606181e-06	
5	-5.53838974e-12	1.05133679e-11	2.88316980e-11
	1.00000000e+00	2.68465250e-11	
6	-1.53209346e-16	-2.22044604e-16	2.22044604e-16
	1.00000000e+00	0.00000000e+00	

We now analyze the impact of the starting point on the solution of Newton's method.

Example 7.12 (Newton fractal)**.** Consider the system of equations

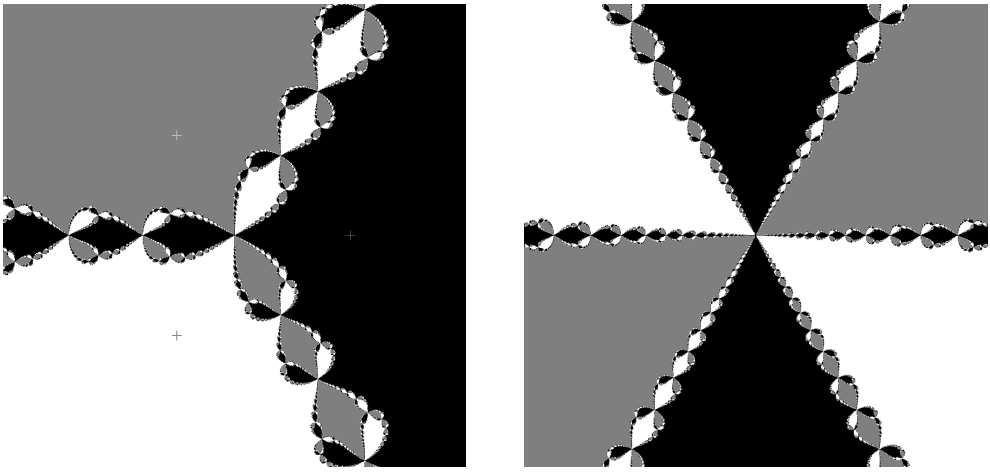
$$F(x) = \begin{pmatrix} x_1^3 - 3x_1x_2^2 - 1 \\ x_2^3 - 3x_1^2x_2 \end{pmatrix} = 0\,.$$

It has three roots:

$$x^*(b) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad x^*(g) = \begin{pmatrix} -1/2 \\ \sqrt{3}/2 \end{pmatrix}, \qquad x^*(w) = \begin{pmatrix} -1/2 \\ -\sqrt{3}/2 \end{pmatrix}.$$

We apply Newton's method to this problem, starting from different points. To visualize the process, we take on the following convention:

- if Newton's method, when starting from the point x_0, converges toward the solution $x^*(b)$, the point x_0 is colored in black;
- if Newton's method, when starting from the point x_0, converges toward the solution $x^*(g)$, the point x_0 is colored in gray;
- if Newton's method, when starting from the point x_0 converges toward the solution $x^*(w)$, the point x_0 is colored in white.



(a) $-2 \leq x_1 \leq 2$, $-2 \leq x_2 \leq 2$ (b) $-0.001 \leq x_1 \leq 0.001$, $-0.001 \leq x_2 \leq 0.001$

Figure 7.7: Newton's method: relation between the starting point and the solution

The result is presented in Figure 7.7(a), where the three roots are represented by a $+$ sign. We see that there is no direct relationship between the position of the starting point and the root identified by the method. For example, look at the gray areas at the bottom right of Figure 7.7(a). Although these starting points are closer to the roots $x^*(b)$ and $x^*(w)$, Newton's method converges towards $x^*(g)$ when started from these areas. But the most noticeable feature of this figure is the shape of the borders between each region. This type of configuration is called a *fractal* (see Mandelbrot, 1982). The zoom presented in Figure 7.7(b) shows that two points that are very close may be colored differently. This is an illustration of a chaotic system, which exhibits a significantly different outcome when the starting conditions are perturbed just a little bit.

We now generalize Theorem 7.7 for the case of n equations and n variables.

Theorem 7.13 (Convergence of Newton's method: n variables)**.** *Consider an open convex set $X \subseteq \mathbb{R}^n$ and a function $F : X \to \mathbb{R}^n$. We assume that there exists $x^* \in X$, a sphere $B(x^*, r)$ centered in x^* with radius r and a constant $\rho > 0$ such that $F(x^*) = 0$, $B(x^*, r) \subset X$, $J(x^*)$ is invertible,*

$$\left\| J(x^*)^{-1} \right\| \leq \frac{1}{\rho} \tag{7.18}$$

and J is Lipschitz continuous over $B(x^*, r)$, where M is the Lipschitz constant. There thus exists $\eta > 0$ such that if

$$x_0 \in B(x^*, \eta)\,, \tag{7.19}$$

the sequence $(x_k)_k$ *defined by*

$$x_{k+1} = x_k - J(x_k)^{-1} F(x_k), \qquad k = 0, 1, \dots, \tag{7.20}$$

is well defined and converges toward x^*. *Moreover,*

$$\|x_{k+1} - x^*\| \leq \frac{M}{\rho} \|x_k - x^*\|^2. \tag{7.21}$$

Proof. In order for the sequence to be well defined, the matrix $J(x_k)$ always has to be invertible. By assumption, it is the case at x^*. We choose η such that $J(x)$ is invertible for all x in a sphere $B(x^*, \eta)$ of radius η around x^*. We take

$$\eta = \min\Big(r, \frac{\rho}{2M}\Big). \tag{7.22}$$

We first demonstrate that $J(x_0)$ is invertible, by using the theorem about the inverse of a perturbed matrix (Theorem C.16), with $A = J(x^*)$ and $B = J(x_0)$. The hypothesis (C.28) on which the theorem is based is satisfied. Indeed,

$$\begin{aligned}
\left\|J(x^*)^{-1}\big(J(x_0) - J(x^*)\big)\right\| &\leq \left\|J(x^*)^{-1}\right\| \|J(x_0) - J(x^*)\| \\
&\leq \frac{1}{\rho} \|J(x_0) - J(x^*)\| && \text{from (7.18)} \\
&\leq \frac{M}{\rho} \|x_0 - x^*\| && \text{Lipschitz} \\
&\leq \frac{M}{\rho} \eta && \text{from (7.19)} \\
&\leq \frac{1}{2} && \text{from (7.22)}.
\end{aligned}$$

Therefore, $J(x_0)$ is invertible, and x_1 given by (7.20) is well defined. By using this result, Theorem C.16 and by noting that if $y \leq 1/2$, then $1/(1-y) \leq 2$, we obtain

$$\begin{aligned}
\left\|J(x_0)^{-1}\right\| &\leq \frac{\left\|J(x^*)^{-1}\right\|}{1 - \left\|J(x^*)^{-1}\big(J(x_0) - J(x^*)\big)\right\|} \\
&\leq 2\left\|J(x^*)^{-1}\right\| \leq \frac{2}{\rho}.
\end{aligned} \tag{7.23}$$

We have

$$\begin{aligned}
x_1 - x^* &= x_0 - J(x_0)^{-1} F(x_0) - x^* && \text{according to (7.20)} \\
&= x_0 - J(x_0)^{-1}\big(F(x_0) - F(x^*)\big) - x^* && \text{because } F(x^*) = 0 \\
&= J(x_0)^{-1}\big(F(x^*) - F(x_0) - J(x_0)(x^* - x_0)\big) \\
&= J(x_0)^{-1}\big(F(x^*) - m_{x_0}(x^*)\big) && \text{from (7.13)}.
\end{aligned}$$

Consequently,

$$\begin{aligned}\|x_1 - x^*\| &\leq \left\|J(x_0)^{-1}\right\| \left\|F(x^*) - m_{x_0}(x^*)\right\| \\ &\leq \frac{2}{\rho} \left\|F(x^*) - m_{x_0}(x^*)\right\| && \text{from (7.23)} \\ &\leq \frac{2}{\rho} M \frac{\left\|x_0 - x^*\right\|^2}{2} && \text{from (7.14)},\end{aligned}$$

which proves (7.21) for $k = 0$. Since

$$\begin{aligned}\|x_0 - x^*\| &\leq \eta && \text{from (7.19)} \\ &\leq \frac{\rho}{2M} && \text{from (7.22)},\end{aligned}$$

we have

$$\|x_1 - x^*\| \leq \frac{2}{\rho} M \frac{\rho}{2M} \frac{\|x_0 - x^*\|}{2} = \frac{1}{2} \|x_0 - x^*\|$$

and $x_1 \in B(x^*, \eta)$. The same reasoning can be applied recursively to prove the result for $k = 1, 2, 3, \ldots$ □

Newton's method constitutes an effective tool that is central in optimization problems. However, it has two undesirable features:

1. it must be started close to the solution (which is not known in practice) and, therefore, does not work from any starting point (assumption (7.19));
2. it requires calculating the matrix of the derivatives at each iteration, which can involve a great deal of calculations in solving real problems.

Techniques that permit us to address the first issue are called *globalization techniques*. A global algorithm exhibits convergence when started from any point. We study such methods directly in the context of optimization in later chapters, and refer the interested reader to Dennis and Schnabel (1996) for a comprehensive description of these techniques in the context of solving systems of equations. In Chapter 8, we address the second issue by presenting methods based on the same idea as Newton's method, but without using the derivatives. Such methods are called *quasi-Newton methods*.

The presentation of the proof of Theorems 7.6, 7.7, and 7.13 is inspired by Dennis and Schnabel (1996).

7.3 Project

The general organization of the projects is described in Appendix D.

Objective

To analyze the impact of the starting point on the convergence of Newton's method, with inspiration taken from Example 7.12.

Approach

To create drawings similar to those of Figure 7.7, we use the following convention:

- Associate a specific color for each solution. For instance, when we have three solutions, the RGB codes $(255, 0, 0)$, $(0, 255, 0)$ and $(0, 0, 255)$ could be utilized.
- Define a maximum number of iterations K.
- Apply Newton's method from a starting point x_0.
- If the method converges in k iterations toward the first solution, associate the color $\big(255 - (255\ k/K), 0, 0\big)$ to the point x_0. Similarly, associate the color $\big(0, 255 - (255\ k/K), 0\big)$ and $\big(0, 0, 255 - (255\ k/K)\big)$ if the algorithm converges toward the second or third solution, respectively.
- If the method does not converge, associate the color black $(0, 0, 0)$ to the point x_0.

Algorithm

Algorithm 7.3.

Problems

Exercise 7.1. The system

$$\begin{aligned} x_2 &= x_1^2 \\ x_1^2 + (x_2 - 2)^2 &= 4 \end{aligned}$$

has three roots: $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$, $\begin{pmatrix} -\sqrt{3} & 3 \end{pmatrix}^T$, $\begin{pmatrix} \sqrt{3} & 3 \end{pmatrix}^T$. Note that there are three intersections between a circle and a parabola (draw the sketch). Note also that the Jacobian is singular when $x_1 = 0$.

Exercise 7.2. The system

$$\begin{aligned} 3x_1^2 + 2x_2^2 &= 35 \\ 4x_1^2 - 3x_2^2 &= 24 \end{aligned}$$

has four solutions: $\begin{pmatrix} -3 & -2 \end{pmatrix}^T$, $\begin{pmatrix} -3 & 2 \end{pmatrix}^T$, $\begin{pmatrix} 3 & -2 \end{pmatrix}^T$, $\begin{pmatrix} 3 & 2 \end{pmatrix}^T$. Note that there are three intersections between an ellipse and a hyperbole (draw the sketch).

Exercise 7.3. The system

$$\begin{aligned} x_1^2 - x_1 x_2 + x_2^2 &= 21 \\ x_1^2 + 2x_1 x_2 - 8x_2^2 &= 0 \end{aligned}$$

has four solutions: $\begin{pmatrix} -2\sqrt{7} & -\sqrt{7} \end{pmatrix}^T$, $\begin{pmatrix} 2\sqrt{7} & \sqrt{7} \end{pmatrix}^T$, $\begin{pmatrix} -4 & 1 \end{pmatrix}^T$, $\begin{pmatrix} 4 & -1 \end{pmatrix}^T$.

Warning: when implementing these systems, one must not confuse the Jacobian and the gradient matrix. Each row of the Jacobian corresponds to an equation and each column to a variable.

Chapter 8

Quasi-Newton methods

"You cannot have your cake and eat it too," says a popular proverb. In this chapter, however, we try! The method developed here has an effectiveness close to that of Newton's method, without requiring the calculation of derivatives.

Contents

When conditions so permit, Newton's method proves to be fast. However, it requires that the Jacobian matrix be explicitly calculated at each iteration. There are a number of cases where the function F is not specified by formulas, but rather by experiments or determined by software. In these cases, the analytical expression of the derivative is unavailable. Even if the problem happens to have an analytical formulation, the calculation of the derivatives can be prohibitive or even impossible when the analytical calculation and implementation of the derivatives require excessively long work, into which errors can easily slip.

In this chapter, we see that it is possible to use the ideas from Newton's method, without using the derivatives. This is of course done at the expense of performance. However, this expense is often small compared with what we gain by not having to calculate Jacobian matrices. We introduce the main ideas regarding the simple problem of one equation with one unknown, before generalizing for systems of equations.

8.1 Equation with one unknown

The main idea is based on the definition of the derivative:

$$F'(x) = \lim_{s \to 0} \frac{F(x+s) - F(x)}{s}. \tag{8.1}$$

Sister Caasi Newton, or Quasi Newton, is the twin sister of Sir Isaac Newton. Caasi Newton tried to follow in the footsteps of her illustrious brother, but was never able to understand the complex concept of derivatives. Her striking resemblance to her brother and the complete absence of any writings cast doubt on her existence.

Figure 8.1: Sister Caasi Newton.

To obtain a good approximation of the value of the derivative, we simply choose a value of s that is close enough to zero and obtain

$$a_s(x) = \frac{F(x+s) - F(x)}{s}. \tag{8.2}$$

Geometrically, the derivative at x is the slope of the tangent to the function at x. The above approximation replaces the tangent by a secant intersecting the function at x and $x+s$, as illustrated in Figure 8.2. The model obtained from this approximation is therefore called the *secant linear model* of the function.

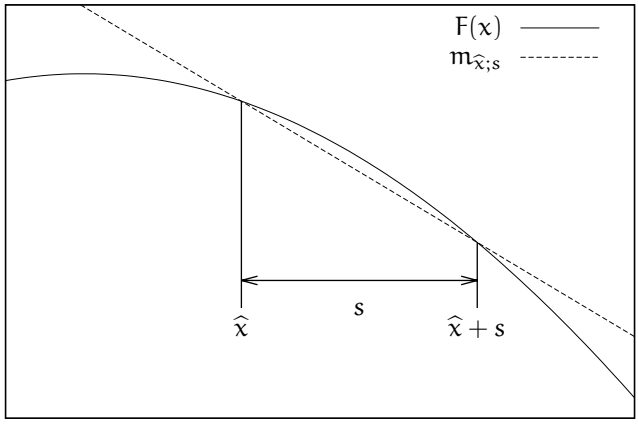


Figure 8.2: Secant linear model

Definition 8.1 (Secant linear model of a function with one variable)**.** Let $F : \mathbb{R} \to \mathbb{R}$ be a differentiable function. The secant linear model of F in $\widehat{x}$ is a function $m_{\widehat{x};s} : \mathbb{R} \to \mathbb{R}$ defined by

$$m_{\widehat{x};s}(x) = F(\widehat{x}) + \frac{F(\widehat{x}+s) - F(\widehat{x})}{s}(x - \widehat{x}), \tag{8.3}$$

where $s \neq 0$.

We can now utilize the same principle as in Newton's method, replacing the derivative in (7.3) by its secant approximation and obtain

$$x^+ = \widehat{x} - \frac{F(\widehat{x})}{a_s(\widehat{x})}\,. \tag{8.4}$$

To obtain an algorithm, we now need only define the value of s. As said above, a natural idea is to choose s small so as to obtain a good approximation of the derivative. For example, s can be defined as

$$s = \begin{cases} \tau\widehat{x} & \text{if } |\widehat{x}| \geq 1 \\ \tau & \text{otherwise}\,, \end{cases} \tag{8.5}$$

where τ is small, for instance equal to 10^{-7}. For a more sophisticated calculation of τ, taking into account the epsilon machine and the precision obtained when calculating F, we refer the reader to Dennis and Schnabel (1996, Algorithm A5.6.3). The algorithm based on this definition of s is called the *finite difference Newton's method* and is presented as Algorithm 8.1.

Algorithm 8.1: Finite difference Newton's method: one variable

Objective
  To find (an approximation of) a solution to the equation
  $$F(x) = 0\,.$$
Input
  The function $F : \mathbb{R} \to \mathbb{R}$.
  A first approximation of the solution $x_0 \in \mathbb{R}$.
  A parameter $\tau > 0$.
  The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.
Output
  An approximation of the solution $x^* \in \mathbb{R}$.
Initialization
  $k := 0$.
Repeat
  if $|x_k| \geq 1$ **then**
    $s := \tau x_k$
  else
    $s := \tau$
  $x_{k+1} := x_k - \dfrac{sF(x_k)}{F(x_k + s) - F(x_k)}$.
  $k := k + 1$.
Until $\left|F(x_k)\right| \leq \varepsilon$
$x^* = x_k$.

The iterations of this algorithm applied to Example 7.3, with $\tau = 10^{-7}$, are described in Table 8.1. The difference with the iterations of Newton's method (Table 7.1) are almost imperceptible. What is even more interesting is that a higher value of τ may still enable the algorithm to converge, even if this convergence is slow. Table 8.2 contains the iterations of the algorithm applied to Example 7.3, with $\tau = 0.1$. The first two iterations of this algorithm are illustrated in Figure 8.3. Intuitively, we expect it to work well, with a relatively large s, when the function is not too linear.

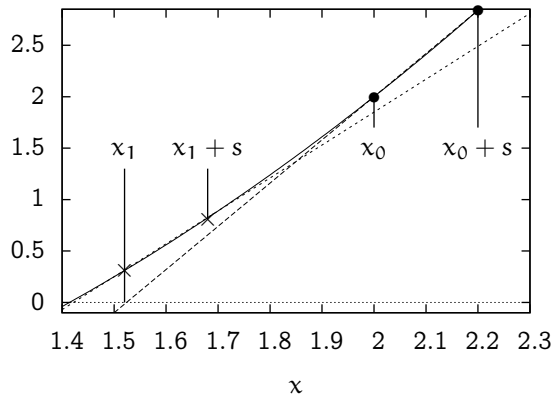


Figure 8.3: Finite difference Newton's method for Example 7.3

Table 8.1: Iterations for finite difference Newton's method ($\tau = 10^{-7}$) for Example 7.3

k	x_k	$F(x_k)$
0	+2.00000000E+00	+2.00000000E+00
1	+1.50000003E+00	+2.50000076E-01
2	+1.41666667E+00	+6.94446047E-03
3	+1.41421569E+00	+6.00768206E-06
4	+1.41421356E+00	+4.81081841E-12
5	+1.41421356E+00	+4.44089210E-16

In practice, there is no reason to take $\tau = 0.1$ because, even if this choice provides results, it slows down the convergence. The only motivation to take a larger s would be to save on function evaluations. This is the idea of the *secant method*, that uses a step based on the last two iterates, that is

$$s = x_{k-1} - x_k \,,$$

in such a way that (8.2) is written as

$$a_s(x_k) = \frac{F(x_{k-1}) - F(x_k)}{x_{k-1} - x_k} \,.$$

Therefore, no additional evaluation of the function is required, because $F(x_{k-1})$ has already been calculated during the previous iteration. The secant method is described

Table 8.2: Iterations for finite difference Newton's method ($\tau = 0.1$) for Example 7.3

k	x_k	$F(x_k)$
0	+2.00000000E+00	+2.00000000E+00
1	+1.52380952E+00	+3.21995465E-01
2	+1.42318594E+00	+2.54582228E-02
3	+1.41466775E+00	+1.28485582E-03
4	+1.41423526E+00	+6.13706622E-05
5	+1.41421460E+00	+2.92283950E-06
6	+1.41421361E+00	+1.39183802E-07
7	+1.41421356E+00	+6.62780186E-09
8	+1.41421356E+00	+3.15609761E-10
9	+1.41421356E+00	+1.50284230E-11
10	+1.41421356E+00	+7.15427717E-13
11	+1.41421356E+00	+3.41948692E-14
12	+1.41421356E+00	+1.33226763E-15
13	+1.41421356E+00	+4.44089210E-16

as Algorithm 8.2. Note that this technique does not work at the first iteration ($k = 0$), as x_{k-1} is not defined. For the first iteration, an arbitrary value for a_0 is therefore selected.

Table 8.3 shows the iterations of the secant method, with $a_0 = 1$. Figure 8.4 illustrates the first two iterations of the method for this example. At the iterate $x_0 = 2$, a first arbitrary linear model, with slope 1, is first considered. It intersects the x-axis at 0, which becomes the next iterate x_1. Then the secant method can start. The secant intersecting the function at x_0 and x_1 is considered. It intersects the x-axis at 1, which becomes iterate x_2. The next iteration is illustrated in Figure 8.5. The secant intersecting the function at x_1 and x_2, crosses the x-axis at $x_3 = 2$. Interestingly, by coincidence, it happens to be the same value as x_0. But it does not mean that iteration 3 is the same as iteration 0. Indeed, between the two, the algorithm has collected information about the function, and accumulated it into a_k. If $a_0 = 1$ is an arbitrary value, not containing information about F, the value $a_3 = 3$ used for the next secant model has been calculated using explicit measures of the function F. As a consequence, the secant intersecting the function at x_2 and x_3 crosses the x-axis at x_4, that happens not to be too far from the zero of the function. Therefore, the convergence of the method from iteration 4 is pretty fast, as can be seen in Table 8.3. Indeed, during the last iterations, x_k and x_{k-1} are closer and closer and $s = x_{k-1} - x_k$ is smaller and smaller. Geometrically, it means that the secant is closer and closer to the actual tangent, and the method becomes similar to the finite difference Newton's method. The rate of convergence is fast, and is characterized as *superlinear*.

Algorithm 8.2: Secant method: one variable

Objective

To find (an approximation of) a solution to the equation

$$F(x) = 0\,.$$

Input

The function $F : \mathbb{R} \to \mathbb{R}$.

A first approximation of the solution $x_0 \in \mathbb{R}$.

A first approximation of the derivative a_0 (by default: $a_0 = 1$).

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the solution $x^* \in \mathbb{R}$.

Initialization

$k := 0$.

Repeat

Update the current iterate

$$x_{k+1} := x_k - \frac{F(x_k)}{a_k}.$$

Update the approximation of the derivative

$$a_{k+1} := \frac{F(x_k) - F(x_{k+1})}{x_k - x_{k+1}}.$$

$k := k + 1$.

Until $\left|F(x_k)\right| \leq \varepsilon$

$x^* = x_k$

Definition 8.2 (Superlinear convergence)**.** Consider a sequence $\left(x_k\right)_k$ in $\mathbb{R}^n$ that converges toward x^*. The sequence is said to converge superlinearly toward x^* if

$$\lim_{k\to\infty} \frac{\left\|x_{k+1} - x^*\right\|}{\left\|x_k - x^*\right\|} = 0. \tag{8.6}$$

Table 8.3: Iterations for the secant method ($a_0 = 1$) for Example 7.3

k	x_k	$F(x_k)$	$a_s(x_k)$
0	+2.00000000E+00	+2.00000000E+00	+1.00000000E+00
1	+0.00000000E+00	-2.00000000E+00	+2.00000000E+00
2	+1.00000000E+00	-1.00000000E+00	+1.00000000E+00
3	+2.00000000E+00	+2.00000000E+00	+3.00000000E+00
4	+1.33333333E+00	-2.22222222E-01	+3.33333333E+00
5	+1.40000000E+00	-4.00000000E-02	+2.73333333E+00
6	+1.41463415E+00	+1.18976800E-03	+2.81463415E+00
7	+1.41421144E+00	-6.00728684E-06	+2.82884558E+00
8	+1.41421356E+00	-8.93145558E-10	+2.82842500E+00
9	+1.41421356E+00	+8.88178420E-16	+2.82842706E+00

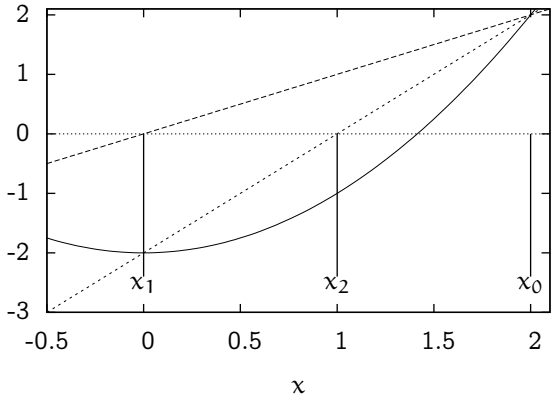


Figure 8.4: Iterations 0 and 1 of the secant method for Example 7.3

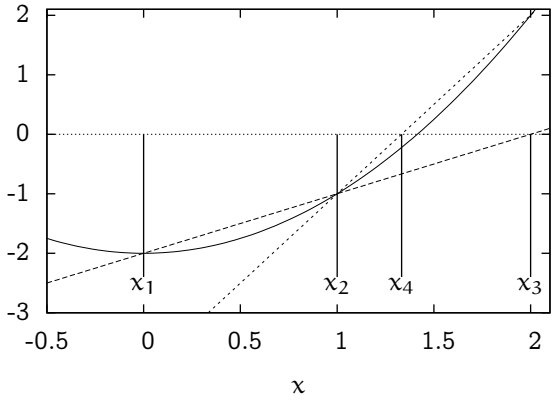


Figure 8.5: Iterations 2 and 3 of the secant method for Example 7.3

8.2 Systems of equations with multiple unknowns

We now generalize the concepts of Section 8.1 for systems of n equations with n unknowns. Again, the ideas are based on a *linear model*.

Algorithm 8.3: Finite difference Newton's method: n variables

Objective

To find (an approximation of) a solution to the system of equations

$$F(x) = 0\,. \tag{8.7}$$

Input

The function $F : \mathbb{R}^n \to \mathbb{R}^n$.
A first approximation of the solution $x_0 \in \mathbb{R}^n$.
A parameter $\tau > 0$.
The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the solution $x^* \in \mathbb{R}^n$.

Initialization

$k := 0$.

Repeat

  for $j = 1, \ldots, n$ **do**
    if $|(x_k)_j| \geq 1$ **then**
      $s_j := \tau (x_k)_j$
    else if $0 \leq (x_k)_j \leq 1$ **then**
      $s_j := \tau$
    else
      $s_j := -\tau$
  Form the matrix A_k with columns

$$\left(A_k\right)_j := \frac{F(x_k + s_j e_j) - F(x_k)}{s_j}\,, \qquad j = 1, \ldots, n\,,$$

  where $\left(A_k\right)_j$ is the j^{th} column of A_k, and $e_j \in \mathbb{R}^n$ is the j^{th} canonical vector, composed of 0, except at the j^{th} place containing 1 instead.
  Calculate d_{k+1} solution of $A_k d_{k+1} = -F(x_k)$.
  $x_{k+1} := x_k + d_{k+1}$.
  $k := k + 1$.

Until $\left\|F(x_k)\right\| \leq \varepsilon$

$x^* = x_k$.

Definition 8.3 (Linear secant model for a function with n variables)**.** Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a Lipschitz continuous function and $A \in \mathbb{R}^{m \times n}$ a matrix. The linear secant model of F in $\widehat{x}$ is a function $m_{\widehat{x};A} : \mathbb{R}^n \to \mathbb{R}^m$ defined by

$$m_{\widehat{x};A}(x) = F(\widehat{x}) + A(x - \widehat{x}) . \tag{8.8}$$

When $m = n$, Definition 8.3 is similar to Definition 7.9, where $J(\widehat{x})$ is replaced by A. As we did for problems with one variable, we now consider two methods to determine A: the approximation of $J(\widehat{x})$ by finite difference and the secant method based on previous iterates.

Algorithm 8.3 describes the method based on finite difference approximation. The comments related to the problems with one variable remain valid. When τ is small, the differences with the original Newton method are small (compare Table 7.4 and Table 8.4). When τ is large, the method still works, but with a much slower convergence speed. Table 8.5 describes the iterations for $\tau = 0.1$. We note that the choice of $\tau = 0.1$ is given only as an illustration. In practice, if the finite difference method is adopted, a small value of τ should be used (see Dennis and Schnabel, 1996, for more details).

Table 8.4: Iterations for the finite difference Newton's method for Example 7.11 ($\tau = 10^{-7}$)

k	x_k	$F(x_k)$	$\|F(x_k)\|$
0	+1.00000000e+00	+3.00000000e+00	+3.45723769e+00
	+1.00000000e+00	+1.71828183e+00	
1	+1.52359228e-01	+7.56629845e-01	+1.15470878e+00
	+1.19528158e+00	+8.72274977e-01	
2	-1.08376852e-02	+5.19684806e-02	+1.14042632e-01
	+1.03611119e+00	+1.01513541e-01	
3	-8.89667761e-04	+1.29445824e-03	+3.94234579e-03
	+1.00153532e+00	+3.72377069e-03	
4	-1.37016733e-06	+3.13751967e-06	+8.08060994e-06
	+1.00000294e+00	+7.44662523e-06	
5	-5.68344146e-12	+1.09472431e-11	+2.98662028e-11
	+1.00000000e+00	+2.77875500e-11	
6	-9.93522913e-17	+0.00000000e+00	+4.44089210e-16
	+1.00000000e+00	+4.44089210e-16	

The main disadvantage of this method is that it uses $n + 1$ evaluations of the function per iteration. This turns out to be prohibitive when n is large. Therefore, we use the same idea as in the case involving a single variable: force the linear model in x_k to interpolate the function F in x_k and in x_{k-1}. We immediately observe that $m_{x_k;A_k}(x_k) = F(x_k)$ by Definition 8.3. We now need only impose

$$m_{x_k;A_k}(x_{k-1}) = F(x_k) + A_k(x_{k-1} - x_k) = F(x_{k-1}) \tag{8.9}$$

Table 8.5: Iterations for the finite difference Newton's method for Example 7.11 ($\tau = 0.1$)

k	x_k	$F(x_k)$	$\|F(x_k)\|$
0	+1.00000000e+00	+3.00000000e+00	+3.45723769e+00
	+1.00000000e+00	+1.71828183e+00	
1	+1.64629659e-01	+8.02103265e-01	+1.21852778e+00
	+1.20238971e+00	+9.17300554e-01	
2	-1.45741083e-02	+8.85985792e-02	+1.88972898e-01
	+1.05713499e+00	+1.66916290e-01	
3	-5.72356301e-03	+8.21459268e-03	+2.52536228e-02
	+1.00976678e+00	+2.38802414e-02	
4	-4.76896360e-04	+1.48824845e-03	+3.51842725e-03
	+1.00122016e+00	+3.18817297e-03	
⋮			
14	-2.17152295e-13	+5.45341550e-13	+1.36591792e-12
	+1.00000000e+00	+1.25233157e-12	
15	-2.49137961e-14	+6.26165786e-14	+1.56919411e-13
	+1.00000000e+00	+1.43884904e-13	
16	-2.79620466e-15	+7.10542736e-15	+1.79018084e-14
	+1.00000000e+00	+1.64313008e-14	
17	-2.35536342e-16	+8.88178420e-16	+1.98602732e-15
	+1.00000000e+00	+1.77635684e-15	
18	-5.13007076e-17	+0.00000000e+00	+0.00000000e+00
	+1.00000000e+00	+0.00000000e+00	

or

$$A_k(x_k - x_{k-1}) = F(x_k) - F(x_{k-1}).$$

This equation is called the *secant equation*.

Definition 8.4 (Secant equation). A linear model satisfies the secant equation in x_k and x_{k-1} if the matrix A defining it is such that

$$A(x_k - x_{k-1}) = F(x_k) - F(x_{k-1}). \tag{8.10}$$

By taking

$$\begin{aligned} d_{k-1} &= x_k - x_{k-1} \\ y_{k-1} &= F(x_k) - F(x_{k-1}), \end{aligned} \tag{8.11}$$

it is written as

$$Ad_{k-1} = y_{k-1}. \tag{8.12}$$

Given x_k, x_{k-1}, $F(x_k)$ and $F(x_{k-1})$, the linear secant model is based on a matrix A satisfying the system of equations (8.10) or (8.12). This system of n linear

equations has n^2 unknowns (the elements of A). Therefore, when $n > 1$, it is always underdetermined and has an infinite number of solutions. From a geometrical point of view, there are infinitely many hyperplanes passing through the two points.

The idea proposed by Broyden (1965) is to choose, among the infinite number of linear models verifying the secant equation, the one that is the closest to the model established during the previous iteration, thereby conserving to the largest possible extent what has already been calculated. We now calculate the difference between two successive models, that is $m_{x_{k-1};A_{k-1}}(x)$, the model of the function in the previous iterate x_{k-1}, and $m_{x_k;A_k}(x)$, the model of the function in the current iterate x_k.

Lemma 8.5. *Let $m_{x_k;A_k}(x)$ and $m_{x_{k-1};A_{k-1}}(x)$ be linear secant models of a function $F : \mathbb{R}^n \to \mathbb{R}^n$ in x_k and x_{k-1}, respectively. If these models satisfy the secant equation, we can characterize the difference between the two models by*

$$m_{x_k;A_k}(x) - m_{x_{k-1};A_{k-1}}(x) = (A_k - A_{k-1})(x - x_{k-1}). \tag{8.13}$$

Proof. The proof exploits the definition (8.8) of the secant model, and the secant equation (8.10).

$$\begin{aligned} m_{x_k;A_k}(x) - m_{x_{k-1};A_{k-1}}(x) = {} & F(x_k) + A_k(x - x_k) \\ & - F(x_{k-1}) - A_{k-1}(x - x_{k-1}) && \text{from (8.8)} \\ = {} & F(x_k) + A_k(x - x_k) \\ & - F(x_{k-1}) - A_{k-1}(x - x_{k-1}) \\ & + A_k x_{k-1} - A_k x_{k-1} \\ = {} & F(x_k) - F(x_{k-1}) - A_k(x_k - x_{k-1}) \\ & + (A_k - A_{k-1})(x - x_{k-1}) \\ = {} & (A_k - A_{k-1})(x - x_{k-1}) && \text{from (8.10)}. \end{aligned}$$

□

We now need only establish which matrix A_k minimizes this difference.

Theorem 8.6 (Broyden update). *Let $m_{x_{k-1};A_{k-1}}(x)$ be the linear secant model of a function $F : \mathbb{R}^n \to \mathbb{R}^n$ in x_{k-1} and let us take $x_k \in \mathbb{R}^n$, $x_k \neq x_{k-1}$. The linear secant model of F in x_k that satisfies the secant equation (8.10) and is as close as possible to $m_{x_{k-1};A_{k-1}}(x)$ is*

$$m_{x_k;A_k}(x) = F(x_k) + A_k(x - x_k), \tag{8.14}$$

with

$$A_k = A_{k-1} + \frac{(y_{k-1} - A_{k-1}d_{k-1})d_{k-1}^T}{d_{k-1}^T d_{k-1}}, \tag{8.15}$$

where $d_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = F(x_k) - F(x_{k-1})$.

Proof. According to Lemma 8.5, the difference between the two linear models is

$$(A_k - A_{k-1})(x - x_{k-1}) . \tag{8.16}$$

The secant equation imposes the behavior of the linear model solely in the direction d_{k-1}. Therefore, the degrees of freedom should be explored in the directions that are orthogonal to d_{k-1}. For all x, we can decompose

$$x - x_{k-1} = \alpha d_{k-1} + s , \tag{8.17}$$

where $s \in \mathbb{R}^n$ is such that $d_{k-1}^T s = 0$. Therefore, (8.16) is written as

$$\alpha(A_k - A_{k-1})d_{k-1} + (A_k - A_{k-1})s . \tag{8.18}$$

The secant equation imposes that the first term should be

$$\alpha(A_k - A_{k-1})d_{k-1} = \alpha(y_{k-1} - A_{k-1}d_{k-1}) .$$

It does not depend on A_k and no degrees of freedom are available here. However, d_{k-1} is not involved in the second term, and the secant equation is irrelevant for this term. We choose A_k such that this second term disappears and that the gap between the two models is minimal. This is the case if $A_k - A_{k-1}$ is defined by

$$A_k - A_{k-1} = u d_{k-1}^T , \tag{8.19}$$

because $d_{k-1}^T s = 0$. In this way, the choice of A_k depends on the choice of u. Once again, it is the secant equation that enables its definition. We have

$$u d_{k-1}^T d_{k-1} = (A_k - A_{k-1})d_{k-1} = y_{k-1} - A_{k-1}d_{k-1} .$$

Therefore,

$$u = \frac{y_{k-1} - A_{k-1}d_{k-1}}{d_{k-1}^T d_{k-1}} \tag{8.20}$$

and (8.15) is obtained directly from (8.19) and (8.20). □

We now show that this update indeed generates the matrix satisfying the secant equation that is the closest to A_{k-1}.

Theorem 8.7 (Broyden optimality). *Consider $A_{k-1} \in \mathbb{R}^{n\times n}$, d_{k-1} and $y_{k-1} \in \mathbb{R}^n$, $d_{k-1} \neq 0$. Let $\mathcal{S} = \{A \mid Ad_{k-1} = y_{k-1}\}$ be the set of matrices satisfying the secant equation. Then (8.15), i.e.,*

$$A_k = A_{k-1} + \frac{(y_{k-1} - A_{k-1}d_{k-1})d_{k-1}^T}{d_{k-1}^T d_{k-1}}$$

is the solution to

$$\min_{A\in\mathcal{S}} \|A - A_{k-1}\|_2$$

and the unique solution to

$$\min_{A\in\mathcal{S}} \|A - A_{k-1}\|_F .$$

Proof. Let A be an arbitrary matrix in $\mathcal{S}$. We have

$$\begin{aligned}
\left\|A_k - A_{k-1}\right\|_2 &= \left\| \frac{(y_{k-1} - A_{k-1}d_{k-1})d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_2 && \text{from (8.15)} \\
&= \left\| \frac{(Ad_{k-1} - A_{k-1}d_{k-1})d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_2 && \text{because } A \in \mathcal{S} \\
&= \left\| \frac{(A - A_{k-1})d_{k-1}d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_2 \\
&\leq \left\|A - A_{k-1}\right\|_2 \left\| \frac{d_{k-1}d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_2 && \text{from (C.22)} \\
&= \left\|A - A_{k-1}\right\|_2 && \text{from (C.25)}.
\end{aligned}$$

Similarly, we have

$$\begin{aligned}
\left\|A_k - A_{k-1}\right\|_F &= \left\| \frac{(y_{k-1} - A_{k-1}d_{k-1})d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_F && \text{from (8.15)} \\
&= \left\| \frac{(Ad_{k-1} - A_{k-1}d_{k-1})d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_F && \text{because } A \in \mathcal{S} \\
&= \left\| \frac{(A - A_{k-1})d_{k-1}d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_F \\
&\leq \left\|A - A_{k-1}\right\|_F \left\| \frac{d_{k-1}d_{k-1}^T}{d_{k-1}^T d_{k-1}} \right\|_2 && \text{from (C.23)} \\
&= \left\|A - A_{k-1}\right\|_F && \text{from (C.25)}.
\end{aligned}$$

The uniqueness follows from the strict convexity of the Frobenius norm and the convexity of the set $\mathcal{S}$. □

Algorithm 8.4 describes the secant method for n variables. Table 8.6 describes the iterations for the secant method for Example 7.11. It is noteworthy that the method converges, but toward another solution than that of Newton's method. Table 8.7 compares matrix A_k for some iterations with the corresponding Jacobian matrix. Clearly, these matrices are different for the first iterations of the algorithm. We can see, even for the last iterations, that matrix A_k is a poor approximation of the Jacobian matrix. This is one of the strengths of the secant method: it is not necessary to have an asymptotically good approximation of the Jacobian matrix for the method to work well.

Algorithm 8.4: Secant method: n variables

Objective

To find (an approximation of) a solution to the system of equations

$$F(x) = 0\,. \tag{8.21}$$

Input

The function $F : \mathbb{R}^n \to \mathbb{R}^n$.

A first approximation of the solution $x_0 \in \mathbb{R}^n$.

A first approximation of the Jacobian matrix A_0 (by default $A_0 = I$).

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the solution $x^* \in \mathbb{R}^n$.

Initialization

$x_1 := x_0 - A_0^{-1} F(x_0)$.

$d_0 := x_1 - x_0$.

$y_0 := F(x_1) - F(x_0)$.

$k := 1$.

Repeat

Broyden update:

$$A_k := A_{k-1} + \frac{(y_{k-1} - A_{k-1} d_{k-1}) d_{k-1}^T}{d_{k-1}^T d_{k-1}}\,.$$

Calculate d_k solution of $A_k d_k = -F(x_k)$.

$x_{k+1} := x_k + d_k$.

$y_k := F(x_{k+1}) - F(x_k)$.

$k := k + 1$.

Until $\|F(x_k)\| \leq \varepsilon$

$x^* = x_k$.

Table 8.6: Iterations for the secant method for Example 7.11

k	x_k	$F(x_k)$	$\|F(x_k)\|$
0	1.00000000e+00	3.00000000e+00	3.45723768e+00
	1.00000000e+00	1.71828182e+00	
1	-2.00000000e+00	-4.84071214e-01	2.28706231e+00
	-7.18281828e-01	-2.23524698e+00	
2	-1.66450025e+00	-8.68008706e-01	1.51117836e+00
	8.30921595e-01	-1.23702099e+00	
3	-2.42562564e-01	2.72598221e+00	7.74156513e+00
	2.03771213e+00	7.24574714e+00	
4	-1.24155582e+00	-1.34676047e+00	1.83898030e+00
	7.71291329e-01	-1.25223192e+00	
5	-5.80521668e-01	-1.64577514e+00	2.13825933e+00
	4.22211781e-01	-1.36512898e+00	
⋮			
15	-1.71374738e+00	-1.15696833e-07	1.85422885e-07
	1.22088678e+00	-1.44899582e-07	
16	-1.71374741e+00	-2.43091768e-10	3.89249065e-10
	1.22088682e+00	-3.04008596e-10	
17	-1.71374741e+00	8.17124146e-14	1.30803685e-13
	1.22088682e+00	1.02140518e-13	
18	-1.71374741e+00	-2.22044604e-16	2.22044604e-16
	1.22088682e+00	0.00000000e+00	

Table 8.7: Jacobian matrix and Broyden matrix for the secant method for Example 7.11

k	$J(x_k)$		A_k	
0	4.00000000e+00	2.00000000e+00	1.00000000e+00	0.00000000e+00
	2.71828182e+00	3.00000000e+00	0.00000000e+00	1.00000000e+00
1	-2.00000000e+00	-1.43656365e+00	1.12149881e+00	6.95897342e-02
	1.35335283e-01	1.54778635e+00	5.61032855e-01	1.32133752e+00
2	-1.32900051e+00	1.66184319e+00	1.00559588e+00	-4.65603572e-01
	1.89285227e-01	2.07129209e+00	3.95856681e-01	5.58620104e-01
3	1.51487487e+00	4.07542427e+00	2.12000014e+00	4.80185068e-01
	7.84614657e-01	1.24568122e+01	3.35797853e+00	3.07255629e+00
4	-4.83111643e-01	1.54258265e+00	2.63710364e+00	1.13571564e+00
	2.88934337e-01	1.78467094e+00	3.83878676e+00	3.68207545e+00
⋮				
17	-1.42749482e+00	2.44177364e+00	-1.06423011e+00	2.70386672e+00
	1.80189282e-01	4.47169389e+00	6.34731480e-01	4.79964153e+00
18	-1.42749482e+00	2.44177364e+00	-1.06870996e+00	2.71006685e+00
	1.80189282e-01	4.47169389e+00	6.34731480e-01	4.79964153e+00

8.3 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of the present project is to solve a fixed point problem, i.e., given a function $T : \mathbb{R}^n \to \mathbb{R}^n$, to identify $x \in \mathbb{R}^n$ such that $T(x) = x$. This is of course equivalent to solving the system of equations $F(x) = 0$ defined by $F(x) = T(x) - x$. Even if the example that we consider is relatively simple, we assume that the derivatives are unavailable.

Approach

- Implement the Banach fixed-point algorithm $x_{k+1} = T(x_k)$, as well as the secant method (Algorithm 8.4).
- From the starting point $x = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^T$, solve the problem with the fixed-point algorithm.
- Solve the system of equations $T(x) - x = 0$ by using the secant method (Algorithm 8.4) from the same starting point.
- Solve the system of equations $x - T(x) = 0$ by using the secant method (Algorithm 8.4) from the same starting point.
- Compare the obtained solutions.
- Compare the number of iterations required to obtain the solution.

Algorithms

The simplest algorithm to solve fixed-point problems consists in applying *Banach iterations*, i.e., $x_{k+1} = T(x_k)$. The secant method is used to solve $F(x) = T(x) - x = 0$.

Problem

Exercise 8.1. Find $x^* \in \mathbb{R}^7$ such that $T(x^*) = x^*$, with

$$T(x) = \begin{pmatrix} 1 + x_1 - (x_1^2/4) \\ (1/2)x_2 + (3/10)x_4 + (1/2)x_6 \\ 1 + x_3 - (x_3^2/3) \\ (1/4)x_2 + (2/5)x_4 + (1/5)x_6 \\ \sqrt{2x_5} \\ (1/4)x_2 + (3/10)x_4 + (3/10)x_6 \\ 8/(2 + x_7) \end{pmatrix}.$$

Part IV

Unconstrained optimization

All constraint, except what wisdom lays on evil men, is evil.

William Cowper

We discuss here the description of algorithms for solving unconstrained optimization problems. The chosen approach is the following:

1. First, in Chapter 9, we study quadratic problems, because they often appear as subproblems in various algorithms.
2. Based on the necessary optimality conditions described in Chapter 5, we use in Chapter 10 Newton's method and its variants presented in Part III to solve the system of equations (5.1). We show with examples that this approach does not always work.
3. In Chapter 11, we define a class of methods called *descent methods*, specifically designed for minimization problems. We demonstrate that Newton's method can, once adapted, be part of this class.
4. The methods known as *trust region methods*, described in Chapter 12, constitute an interesting alternative to descent methods. Again, we show that Newton's method can be adapted also to this context.
5. Finally, we describe quasi-Newton methods, similar to the methods presented in Chapter 8 in the context of optimization.

Chapter 9

Quadratic problems

Contents

Before developing algorithms for general non linear problems, let us study the case of quadratic problems (Definition 2.28). These indeed turn up regularly as subproblems in the algorithms. In this chapter, we solve the problem

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} x^T Q x + b^T x + c\,, \tag{9.1}$$

where Q is a symmetric $n \times n$ matrix, positive definite, $b \in \mathbb{R}^n$ and $c \in \mathbb{R}$. According to Theorem 5.10, if Q is not positive definite or semidefinite, the problem has no solution. The case where Q is positive semidefinite and singular is discussed in Theorem 5.10, but is not dealt with here. One should immediately note that the value of c has no impact on the solution to the problem (9.1). Therefore, we focus on the problem

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} x^T Q x + b^T x\,. \tag{9.2}$$

The value of c is added to the optimal value of the objective function of (9.2) to obtain the optimal value of the objective function of (9.1).

By employing Theorem 5.10, the unique global minimum of (9.2) can be easily obtained by solving the system of linear equations

$$Qx = -b\,. \tag{9.3}$$

9.1 Direct solution

Classical linear algebra algorithms can be used to solve (9.3). The solution details can be found in the literature for linear algebra (see in particular Golub and Van Loan, 1996). Typically, the solution algorithm has the following structure.

Algorithm 9.1: Quadratic problems: direct solution

Objective
To find the global minimum of (9.2).
Input
The symmetric and positive definite matrix $Q \in \mathbb{R}^{n \times n}$.
The vector $b \in \mathbb{R}^n$.
Output
The solution $x^* \in \mathbb{R}^n$.
Calculate the Cholesky factorization $Q = LL^T$.
Calculate y^*, the solution to the lower triangular system $Ly = -b$.
Calculate x^*, the solution to the upper triangular system $L^T x = y^*$.

We refer the reader to Higham (1996) for a discussion about the numerical issues associated with the direct method. Note that the above algorithm does not preserve the sparsity of the matrix. Indeed, if n is large, and the number of non zero entries of the matrix Q is significantly less than n^2, it is convenient to adopt data structures that store only those elements (see, for instance, Dongarra, 2000 and Montagne and Ekambaram, 2004). Unfortunately, even if Q is sparse[1], the matrix L resulting from the factorization is not, and these data structures cannot be used. The algorithm presented in the next section is able to exploit the sparsity of the matrix.

9.2 Conjugate gradient method

The conjugate gradient method is an iterative method used to solve (9.2). It was independently discovered by Stiefel (1952) and Hestenes (1951), who completed and published it together (Hestenes and Stiefel, 1952). Quite unpopular during the 1950s and 1960s, the method generated interest during the 1970s, when the size of problems to solve increased significantly (see Golub and O'Leary, 1989, for more historical details).

We describe this method in two steps, first presenting the *conjugate directions* method in a general manner.

Definition 9.1 (Conjugate directions)**.** Let $Q \in \mathbb{R}^{n \times n}$ be a positive definite matrix. The non zero vectors of $\mathbb{R}^n$ $d_1, \ldots, d_k$ are said to be Q-conjugate if

$$d_i^T Q d_j = 0, \quad \forall i, j \text{ such that } i \neq j. \tag{9.4}$$

[1] A matrix is said to be sparse if most of its elements are zero.

Note that if Q is the identity matrix I, the conjugate directions are orthogonal. If not, we may define the inner product:

$$\langle d_i, d_j \rangle_Q = d_i Q d_j, \tag{9.5}$$

so that d_i is Q-conjugate with d_j if and only if d_i is orthogonal to d_j with respect to the inner product $\langle \cdot \rangle_Q$. We can derive the following result directly from the definition.

Theorem 9.2 (Independence of conjugate directions)**.** *Let $Q \in \mathbb{R}^{n \times n}$ be a positive definite matrix and $d_1, \ldots, d_k$ be a set of non zero and Q-conjugate directions. Then, the vectors $d_1, \ldots, d_k$ are linearly independent.*

Proof. We assume by contradiction that there exist $\lambda_1, \ldots, \lambda_{k-1}$, not all zero, such that

$$d_k = \lambda_1 d_1 + \cdots + \lambda_{k-1} d_{k-1} .$$

Therefore,

$$d_k^T Q d_k = \lambda_1 d_k^T Q d_1 + \cdots + \lambda_{k-1} d_k^T Q d_{k-1} = 0 ,$$

because the directions are Q-conjugate. This is impossible because d_k is non zero and Q is positive definite. □

An immediate corollary is that, in $\mathbb{R}^n$, the maximum number of Q-conjugate directions is n.

The idea behind the conjugate directions method is to define an iterative algorithm using n conjugate directions $d_1, \ldots, d_n$, with the following structure:

$$x_{k+1} = x_k + \alpha_k d_k , \qquad k = 1, \ldots, n ,$$

where α_k is chosen to minimize the function in the direction d_k, that is

$$\alpha_k = \operatorname{argmin}_\alpha f(x_k + \alpha d_k) .$$

We can identify some of the properties of this type of method.

Lemma 9.3. *Let $Q \in \mathbb{R}^{n \times n}$ be a positive definite matrix, $f(x) = \frac{1}{2} x^T Q x + b^T x$ and $d_1, \ldots, d_n$ be a set of Q-conjugate directions in $\mathbb{R}^n$. Let $x_1, \ldots, x_{n+1}$ be the iterates generated by a conjugate directions method. Then,*

1. for all $k = 1, \ldots, n$, the step α_k is defined by

$$\alpha_k = -\frac{d_k^T (Q x_k + b)}{d_k^T Q d_k} = -\frac{d_k^T \nabla f(x_k)}{d_k^T Q d_k} ; \tag{9.6}$$

2. for all $k = 1, \ldots, n$, $\nabla f(x_k)$ is orthogonal to $d_1, \ldots, d_{k-1}$, i.e.,

$$\nabla f(x_k)^T d_i = 0 , \qquad i = 1, \ldots, k-1 ; \tag{9.7}$$

3. $\nabla f(x_{n+1}) = 0$*;*

4. let us take k *such that* $\nabla f(x_k) = 0$ *; then,*

$$\nabla f(x_i) = 0\,, \qquad i = k, \dots, n+1\,. \tag{9.8}$$

Proof. 1. Since α_k is the minimum of the function in the direction d_k, its value corresponds to a zero directional derivative of f in the direction d_k (Definition 2.7), i.e.,

$$d_k^T \nabla f(x_k + \alpha_k d_k) = d_k^T \nabla f(x_{k+1}) = 0 \tag{9.9}$$

and, applying the formula (2.42) of the gradient of a quadratic function, we obtain

$$\begin{aligned} 0 &= d_k^T \nabla f(x_k + \alpha_k d_k) \\ &= d_k^T \big(Q(x_k + \alpha_k d_k) + b\big) \\ &= d_k^T Q x_k + \alpha_k d_k^T Q d_k + d_k^T b \end{aligned}$$

to obtain (9.6).

2. Since $x_{k+1} = x_k + \alpha_k d_k$, we have for any $i = 1, \dots, k-1$,

$$\begin{aligned} x_k &= x_{k-1} + \alpha_{k-1} d_{k-1} \\ &= x_{k-2} + \alpha_{k-2} d_{k-2} + \alpha_{k-1} d_{k-1} \\ &\vdots \\ &= x_{i+1} + \sum_{j=i+1}^{k-1} \alpha_j d_j\,. \end{aligned} \tag{9.10}$$

Therefore, for $i = 1, \dots, k-1$,

$$\begin{aligned} d_i^T \nabla f(x_k) &= d_i^T (Q x_k + b) && \text{according to (2.42)} \\ &= d_i^T \left(Q(x_{i+1} + \sum_{j=i+1}^{k-1} \alpha_j d_j) + b \right) && \text{according to (9.10)} \\ &= d_i^T Q x_{i+1} + d_i^T b + \sum_{j=i+1}^{k} \alpha_j d_i^T Q d_j \\ &= d_i^T (Q x_{i+1} + b) && \text{according to (9.4)} \\ &= \nabla f(x_{i+1})^T d_i && \text{according to (2.42)} \\ &= 0 && \text{according to (9.9)}\,. \end{aligned}$$

3. Let $d \neq 0$ be an arbitrary vector of $\mathbb{R}^n$. Since $d_1, \dots, d_n$ is a set of n linearly independent vectors in $\mathbb{R}^n$, this is a basis and d can be written as

$$d = \sum_{i=1}^{n} \lambda_i d_i\,.$$

Therefore

$$\nabla f(x_{n+1})^T d = \sum_{i=1}^{n} \lambda_i \nabla f(x_{n+1})^T d_i = 0$$

by the point 2. Since d is arbitrary, we obtain $\nabla f(x_{n+1}) = 0$.

4. If $\nabla f(x_k) = 0$, then $\alpha_k = 0$, according to (9.6). The result follows by simple induction on k.

□

The most important result related to the conjugate direction methods is that they identify the global minimum of a problem in, at most, n iterations. In fact, they are able to solve the problem of increasing dimension in subspaces.

Theorem 9.4 (Conjugate directions method). *Let $Q \in \mathbb{R}^{n\times n}$ be positive definite. Let $d_1, \ldots, d_\ell$, $\ell \leq n$, be a set of Q-conjugate directions, let us take $x_1 \in \mathbb{R}^n$ and let*

$$M_\ell = x_1 + \langle d_1, \ldots, d_\ell \rangle = \left\{ x \;\middle|\; x = x_1 + \sum_{k=1}^{\ell} \lambda_k d_k \,,\ \lambda \in \mathbb{R}^\ell \right\}$$

be the affine subspace spanned by the directions $d_1, \ldots, d_\ell$. Then, the global minimum of the problem

$$\min_{x\in M_\ell} f(x) = \frac{1}{2} x^T Q x + b^T x \tag{9.11}$$

is

$$x_{\ell+1} = x_1 + \sum_{k=1}^{\ell} \alpha_k d_k \tag{9.12}$$

with

$$\alpha_k = \operatorname{argmin}_\alpha f(x_k + \alpha d_k) = -\frac{d_k^T (Q x_k + b)}{d_k^T Q d_k}. \tag{9.13}$$

Proof. We consider the function

$$g : \mathbb{R}^\ell \longrightarrow \mathbb{R} : \lambda \rightsquigarrow g(\lambda) = f\left(x_1 + \sum_{i=1}^{\ell} \lambda_i d_i \right)$$

that enables us to transform problem (9.11) into an unconstrained problem

$$\min_{\lambda\in\mathbb{R}^\ell} g(\lambda)$$

such that

$$\frac{\partial g}{\partial \lambda_i}(\lambda) = d_i^T \nabla f\left(x_1 + \sum_{j=1}^{\ell} \lambda_j d_j \right).$$

According to Lemma 9.3, when the coefficients λ are replaced by the steps α_k defined by (9.13) (that is, (9.6)), we have

$$\frac{\partial g}{\partial \lambda_i}(\alpha_1, \dots, \alpha_\ell) = d_i^T \nabla f(x_{\ell+1}) = 0, \quad \forall i .$$

Then, $\nabla g(\alpha_1, \dots, \alpha_\ell) = 0$. Moreover,

$$\frac{\partial^2 g}{\partial \lambda_i \lambda_j}(\alpha_1, \dots, \alpha_\ell) = d_i^T Q d_j .$$

As the directions d_i are Q-conjugate, the second derivatives matrix of g is a diagonal matrix with positive eigenvalues. It is therefore positive definite. We now need only use the sufficient optimality conditions (Theorems 5.7 and 5.9) to demonstrate that $\alpha_1, \dots, \alpha_\ell$ is the global minimum of g and that $x_{\ell+1}$ defined by (9.12) is the global minimum of (9.11). □

The specific case $\ell = n$ is particularly important.

Corollary 9.5 (Convergence of the conjugate directions method). *Let $Q \in \mathbb{R}^{n \times n}$ be positive definite. Let $d_1, \dots, d_n$, be a set of Q-conjugate directions. Let $x_1 \in \mathbb{R}^n$ be arbitrary. The algorithm based on the recurrence*

$$x_{k+1} = x_k + \alpha_k d_k$$

with

$$\alpha_k = -\frac{d_k^T (Q x_k + b)}{d_k^T Q d_k}$$

identifies the global minimum of the problem

$$\min_{x \in \mathbb{R}^n} f(x) = \frac{1}{2} x^T Q x + b^T x$$

in at most n iterations.

Proof. We apply Theorem 9.4 with $\ell = n$ to demonstrate that $x_{\ell+1}$ is the global minimum. As the conjugate directions are linearly independent (Theorem 9.2), n directions span the entire space $\mathbb{R}^n$, that is $M_\ell = M_n = \mathbb{R}^n$. □

This result makes the conjugate directions methods particularly attractive. It remains to show how to obtain Q-conjugate directions. We proceed in two steps. First, we start from an arbitrary set of linearly independent vectors and apply the Gram-Schmidt orthogonalization procedure to obtain Q-conjugate directions. Indeed, as discussed above, two directions are Q-conjugate if they are orthogonal with respect to the inner product $\langle \cdot \rangle_Q$ defined by (9.5). Second, we identify a specific set of linearly independent vectors, which simplifies considerably the formulation.

Consider the set of ℓ vectors $\xi_1, \dots \xi_\ell$, that are linearly independent. The Q-conjugate vectors are defined by induction in such a way that, at each step i of the

induction, $i = 1, \ldots, \ell$, the vector subspace spanned by $\xi_1, \ldots \xi_i$ is the same as the subspace spanned by $d_1, \ldots, d_i$, i.e.,

$$\langle \xi_1, \ldots \xi_i \rangle = \langle d_1, \ldots, d_i \rangle . \tag{9.14}$$

We initiate the induction with $d_1 = \xi_1$. Then, for any given $i \geq 2$, we assume that we have Q-conjugate vectors $d_1, \ldots, d_{i-1}$, such that

$$\langle \xi_1, \ldots \xi_{i-1} \rangle = \langle d_1, \ldots, d_{i-1} \rangle .$$

We thus choose d_i of the form

$$d_i = \xi_i + \sum_{k=1}^{i-1} \alpha_k^i d_k . \tag{9.15}$$

We calculate the coefficients α_k^i in order for d_i to be Q-conjugate with $d_1, \ldots, d_{i-1}$. Let $1 \leq j \leq i-1$ be any arbitrary index.

$$\begin{aligned} 0 &= d_j^T Q d_i \\ &= d_j^T Q \xi_i + \sum_{k=1}^{i-1} \alpha_k^i d_j^T Q d_k \\ &= d_j^T Q \xi_i + \alpha_j^i d_j^T Q d_j , \end{aligned}$$

because all the other terms of the sum are zero by Q-conjugation. Then,

$$\alpha_j^i = -\frac{d_j^T Q \xi_i}{d_j^T Q d_j}$$

and (9.15) is written as

$$d_i = \xi_i - \sum_{k=1}^{i-1} \frac{d_k^T Q \xi_i}{d_k^T Q d_k} d_k . \tag{9.16}$$

The calculation of d_i is well-defined. Indeed, the denominator $d_k^T Q d_k$ is non zero because Q is positive definite. Since the vectors $\xi_1, \ldots, \xi_i$ are linearly independent, ξ_i is linearly independent from any direction in the subspace $\langle \xi_1, \ldots \xi_{i-1} \rangle$. From (9.14), it is also independent from any direction in the subspace $\langle d_1, \ldots, d_{i-1} \rangle$. Consequently,

$$\xi_i \neq \sum_{k=1}^{i-1} \frac{d_k^T Q \xi_i}{d_k^T Q d_k} d_k ,$$

and d_i is not zero.

The Gram-Schmidt procedure described above can be applied to any set of linearly independent vectors. We see now that a judicious choice of the vector ξ allows us to greatly simplify (9.16). The method called the *conjugate gradient* method utilizes

$$\xi_i = -\nabla f(x_i) = -Q x_i - b .$$

In order to apply the Gram-Schmidt procedure, we must verify that the vectors $\nabla f(x_i)$, $i = 1, \ldots, n$ are linearly independent. Actually, Theorem 9.6 proposes a stronger result: they are orthogonal.

Theorem 9.6 (Orthogonal gradients). *We consider the conjugate directions method where each direction d_i is generated by the Gram-Schmidt method applied to the directions $-\nabla f(x_1), \ldots, -\nabla f(x_i)$, i.e.,*

$$d_i = -\nabla f(x_i) + \sum_{k=1}^{i-1} \frac{d_k^T Q \nabla f(x_i)}{d_k^T Q d_k} d_k \,. \tag{9.17}$$

Then,

$$\langle \nabla f(x_1), \ldots, \nabla f(x_i) \rangle = \langle d_1, \ldots, d_i \rangle \tag{9.18}$$

and

$$\nabla f(x_i)^T \nabla f(x_k) = 0 \,, \qquad k = 1, \ldots, i-1 \,. \tag{9.19}$$

Proof. $i = 1$: (9.18) is trivially satisfied because $d_1 = -\nabla f(x_1)$ and (9.19) does not apply.

$i = 2$: We have

$$d_2 = -\nabla f(x_2) + \frac{d_1^T Q \nabla f(x_2)}{d_1^T Q d_1} d_1$$

and (9.18) is satisfied. Moreover, according to Lemma 9.3,

$$0 = \nabla f(x_2)^T d_1 = -\nabla f(x_2)^T \nabla f(x_1)$$

and (9.19) is satisfied.

$i > 2$: we now assume that the result is satisfied for $i-1$. Since the vectors $\nabla f(x_1), \ldots, \nabla f(x_{i-1})$ are orthogonal, they are linearly independent. Therefore, (9.17) directly implies that (9.18) is satisfied for i. According to Lemma 9.3, we have that

$$\nabla f(x_i)^T d_k = 0 \,, \qquad k = 1, \ldots, i-1 \,, \tag{9.20}$$

and $\nabla f(x_i)$ is orthogonal to the subspace $\langle d_1, \ldots, d_{i-1} \rangle$. Since (9.18) is satisfied for $i-1$, $\nabla f(x_i)$ is orthogonal to the subspace $\langle \nabla f(x_1), \ldots, \nabla f(x_{i-1}) \rangle$, and (9.19) is satisfied for i. □

We now demonstrate a proposition that enables us to simplify the conjugate gradient method.

Theorem 9.7 (Conjugate gradients). *We consider the conjugate directions method where each direction d_i is generated by the Gram-Schmidt method applied to the directions $-\nabla f(x_1), \ldots, -\nabla f(x_i)$, i.e., according to (9.17). If $\nabla f(x_i) \neq 0$, then*

$$d_i = -\nabla f(x_i) + \beta_i d_{i-1} \tag{9.21}$$

with

$$\beta_i = \frac{\nabla f(x_i)^T \nabla f(x_i)}{\nabla f(x_{i-1})^T \nabla f(x_{i-1})} \,. \tag{9.22}$$

Proof. For all $k = 1, \dots, i-1$, we have

$$\nabla f(x_{k+1}) - \nabla f(x_k) = Qx_{k+1} + b - Qx_k - b = Q(x_k + \alpha_k d_k - x_k) = \alpha_k Q d_k \,.$$

Since $\nabla f(x_i) \neq 0$ by assumption, then $\nabla f(x_k) \neq 0$, $k = 1, \dots, i-1$ (item 4 of Lemma 9.3), and $\alpha_k \neq 0$, so that

$$Q d_k = \frac{1}{\alpha_k} \left(\nabla f(x_{k+1}) - \nabla f(x_k) \right).$$

Then, from the orthogonality of the gradients (Theorem 9.6), we have

$$\begin{aligned} \nabla f(x_i)^T Q d_k &= \frac{1}{\alpha_k} \nabla f(x_i)^T \left(\nabla f(x_{k+1}) - \nabla f(x_k) \right) \\ &= \begin{cases} \dfrac{1}{\alpha_i} \nabla f(x_i)^T \nabla f(x_i) & \text{if } k = i-1 \\ 0 & \text{if } k = 1, \dots, i-2 \,. \end{cases} \end{aligned}$$

Similarly, we have

$$d_k^T Q d_k = \frac{1}{\alpha_k} d_k^T \left(\nabla f(x_{k+1}) - \nabla f(x_k) \right).$$

Therefore, (9.17) simplifies into

$$d_i = -\nabla f(x_i) + \frac{d_{i-1}^T Q \nabla f(x_i)}{d_{i-1}^T Q d_{i-1}} d_{i-1} = -\nabla f(x_i) + \beta_i d_{i-1} \tag{9.23}$$

with

$$\beta_i = \frac{d_{i-1}^T Q \nabla f(x_i)}{d_{i-1}^T Q d_{i-1}} = \frac{\nabla f(x_i)^T \nabla f(x_i)}{d_{i-1}^T \left(\nabla f(x_i) - \nabla f(x_{i-1}) \right)} \,. \tag{9.24}$$

Since

$$d_{i-1} = -\nabla f(x_{i-1}) + \beta_{i-1} d_{i-2} \,,$$

the denominator is written as

$$\begin{aligned} d_{i-1}^T \left(\nabla f(x_i) - \nabla f(x_{i-1}) \right) &= -\nabla f(x_{i-1})^T \left(\nabla f(x_i) - \nabla f(x_{i-1}) \right) \\ &\quad + \beta_{i-1} d_{i-2}^T \left(\nabla f(x_i) - \nabla f(x_{i-1}) \right) \\ &= -\nabla f(x_{i-1})^T \nabla f(x_i) \qquad (= 0) \\ &\quad + \nabla f(x_{i-1})^T \nabla f(x_{i-1}) \\ &\quad + \beta_{i-1} d_{i-2}^T \nabla f(x_i) \qquad (= 0) \\ &\quad - \beta_{i-1} d_{i-2}^T \nabla f(x_{i-1}) \qquad (= 0) \,, \end{aligned}$$

where the three indicated terms are zero according to (9.7) and (9.19). And we obtain (9.22) from (9.24). □

All these results are combined to obtain Algorithm 9.2, the conjugate gradient method. An important characteristic of the conjugate gradient algorithm is that the matrix Q defining the problem is never needed as such. There is not even the need to store it. It is used exclusively to calculate the matrix-vector products Qx_k or Qd_k. This is particularly interesting for problems of large size, for which the matrix Q is generally sparse. In this case, the matrix-vector products can be efficiently implemented without ever explicitly forming the matrix Q.

Algorithm 9.2: Conjugate gradient method

Objective

To find the global minimum of (9.2), i.e., $\min_{x\in\mathbb{R}^n} \frac{1}{2}x^TQx + b^Tx$.

Input

A first approximation x_1 of the solution.
The symmetric positive definite matrix $Q \in \mathbb{R}^{n\times n}$.
The vector $b \in \mathbb{R}^n$.

Output

The solution $x^* \in \mathbb{R}^n$.

Initialization

$k := 1$,
$d_1 := -Qx_1 - b$.

Repeat

$$\alpha_k := -\frac{d_k^T(Qx_k + b)}{d_k^TQd_k}.$$

$x_{k+1} := x_k + \alpha_k d_k$.

$$\beta_{k+1} := \frac{\nabla f(x_{k+1})^T\nabla f(x_{k+1})}{\nabla f(x_k)^T\nabla f(x_k)} = \frac{(Qx_{k+1} + b)^T(Qx_{k+1} + b)}{(Qx_k + b)^T(Qx_k + b)}.$$

$d_{k+1} := -Qx_{k+1} - b + \beta_{k+1}d_k$.
$k := k + 1$.

Until $\|\nabla f(x_k)\| = 0$ *or* $k = n + 1$

$x^* = x_k$.

Example 9.8 (Conjugate gradient method)**.** We apply Algorithm 9.2 to the quadratic problem (9.2) defined by

$$Q = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} -4 \\ -7 \\ -9 \\ -10 \end{pmatrix}.$$

The iterations are detailed in Table 9.1. The algorithm converges after having generated 4 directions, as predicted by Corollary 9.5. It is easy to verify that the directions generated by the algorithm are well conjugated and that the gradients are orthogonal to each other, as stated in Theorem 9.6.

Table 9.1: Iterations for the conjugate gradient method for Example 9.8

k	x_k	$\nabla f(x_k)$	d_k	α_k	β_k
1	+5.00000e+00	+1.60000e+01	-1.60000e+01	+1.20766e-01	
	+5.00000e+00	+2.80000e+01	-2.80000e+01		
	+5.00000e+00	+3.60000e+01	-3.60000e+01		
	+5.00000e+00	+4.00000e+01	-4.00000e+01		
2	+3.06775e+00	+1.50810e+00	-1.52579e+00	+1.02953e+00	+1.10547e-03
	+1.61856e+00	+9.48454e-01	-9.79407e-01		
	+6.52430e-01	-2.29750e-01	+1.89953e-01		
	+1.69367e-01	-1.06038e+00	+1.01616e+00		
3	+1.49690e+00	+1.70656e-01	-1.97676e-01	+2.37172e+00	+1.77089e-02
	+6.10224e-01	-1.55585e-01	+1.38241e-01		
	+8.47993e-01	-9.20500e-02	+9.54138e-02		
	+1.21554e+00	+1.23492e-01	-1.05497e-01		
4	+1.02806e+00	+5.77796e-03	-8.27569e-03	+3.39118e+00	+1.26355e-02
	+9.38093e-01	-1.65085e-02	+1.82552e-02		
	+1.07429e+00	+2.31118e-02	-2.19062e-02		
	+9.65332e-01	-1.15559e-02	+1.02229e-02		
5	+1.00000e+00	-1.66356e-12			
	+1.00000e+00	-3.12639e-12			
	+1.00000e+00	-4.21174e-12			
	+1.00000e+00	-4.78906e-12			

9.3 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of the present project is to solve several quadratic problems and compare the direct solution with the conjugate gradient method for ill-conditioned problems.

Approach

Perform the following experiments.

1. Generate a problem of dimension 10 for which the eigenvalues are randomly distributed between 1 and 3, and solve it with Algorithms 9.1 and 9.2. Compare the solutions. After how many iterations does the conjugate gradient algorithm identify an iterate such that the norm of the gradient is below 10^{-6}? Is this consistent with theory?
2. Carry out the same approach for a problem of dimension 100.
3. Generate a vector with 100 eigenvalues randomly distributed between 0 and 1. Subsequently, multiply the last 50 ones by 10,000 and generate a quadratic problem by using the procedure described below. After how many iterations does the conjugate gradient algorithm identify an iterate with the norm of the gradient below 10^{-6}? Is this consistent with theory?
4. Generate a quadratic problem defined by a Hilbert matrix of dimension 10 and another of dimension 100 (see Exercise 9.2 for the definition of a Hilbert matrix). Apply Algorithms 9.1 and 9.2. Compare the solutions. After how many iterations does the conjugate gradient algorithm identify an iterate with the norm of the gradient below 10^{-6}? Is this consistent with theory?

Algorithms

Algorithms 9.1 and 9.2.

Problems

Exercise 9.1. Use the following procedure to generate quadratic problems for which the solution and conditioning are known.

a) Consider any randomly defined matrix $A \in \mathbb{R}^{n\times n}$, for instance

$$A = \begin{pmatrix} 0.071744 & 0.039717 & 0.868964 & 0.880528 & 0.969800 \\ 0.085895 & 0.145339 & 0.832277 & 0.691063 & 0.621372 \\ 0.857871 & 0.357765 & 0.151824 & 0.396765 & 0.258813 \\ 0.412037 & 0.521116 & 0.348378 & 0.632816 & 0.416459 \\ 0.806180 & 0.110585 & 0.332506 & 0.986633 & 0.476912 \end{pmatrix}.$$

b) Carry out a QR factorization of A to obtain an orthogonal matrix B

$$B = \begin{pmatrix} -0.057291 & -0.025777 & 0.733559 & -0.072330 & -0.672839 \\ -0.068592 & -0.258524 & 0.619722 & 0.339972 & 0.654846 \\ -0.685055 & -0.035490 & -0.204420 & 0.657980 & -0.233910 \\ -0.329033 & -0.830212 & -0.119684 & -0.433105 & -0.024104 \\ -0.643777 & 0.491924 & 0.147388 & -0.508596 & 0.251334 \end{pmatrix}.$$

c) Choose non negative eigenvalues $\lambda_1, \ldots, \lambda_n$ and define D as a diagonal matrix containing these values in the diagonal. For instance,

$$D = \begin{pmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 1\,000 \end{pmatrix}.$$

d) Define the matrix $Q = BDB^T$.

$$Q = \begin{pmatrix} 458.618 & -438.512 & 151.130 & 18.496 & -164.356 \\ -438.512 & 444.290 & -132.059 & -31.033 & 148.085 \\ 151.130 & -132.059 & 98.474 & -22.563 & -92.529 \\ 18.496 & -31.033 & -22.563 & 20.182 & 15.406 \\ -164.356 & 148.085 & -92.529 & 15.406 & 89.536 \end{pmatrix}.$$

e) Choose a vector x^*, for instance $x^* = \begin{pmatrix} 1 & \ldots & 1 \end{pmatrix}^T$, and define $b = -Qx^*$.

$$b = \begin{pmatrix} -25.37482 \\ 9.22945 \\ -2.45361 \\ -0.48793 \\ 3.85804 \end{pmatrix}.$$

Then, x^* is the solution to $\min \frac{1}{2}x^TQx + b^Tx$ and the eigenvalues of Q are the same as those of D. The same goes for the conditioning.

Exercise 9.2. We also consider the matrix $H_n \in \mathbb{R}^n$ for which the elements $H_n(i,j)$ are defined by

$$H_n(i,j) = \frac{1}{i+j-1}.$$

This matrix is called the *Hilbert matrix* of dimension n. It is symmetric, positive definite, but extremely ill-conditioned (calculate its eigenvalues).

Chapter 10

Newton's local method

We now apply Newton's method in the context of optimization. In this chapter, we do it blindly. And Algorithm 10.1 does not work in general! Its only utility is to give inspiration in order to define the algorithms that develop the same rate of convergence as Newton's method.

Contents

10.1 Solving the necessary optimality conditions

The idea behind Newton's local method is simply to use Algorithm 7.3 to solve the system of equations (5.1),

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

which defines the necessary optimality conditions. The algorithm, applied to $F(x) = \nabla f(x)$ and $J(x) = \nabla^2 f(x)$, is described as Algorithm 10.1.

It inherits all the properties of Algorithm 7.3. In particular,

1. the method converges q-quadratically under favorable conditions (Theorem 7.13),
2. the method can diverge if the starting point is too far from the solution,
3. the method is not defined if the matrix $\nabla^2 f(x_k)$ is singular.

When employed in the context of optimization, Newton's local method presents a further disadvantage. Indeed, solving the necessary optimality conditions of the first degree does not guarantee that the identified solution is a minimum. Newton's method has no mechanism enabling it to discern minima from maxima and saddle

Algorithm 10.1: Newton's local method

Objective

To find (an approximation of) a solution to the system

$$\nabla f(x) = 0\,. \tag{10.1}$$

Input

The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.

The Hessian of the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.

A first approximation of the solution $x_0 \in \mathbb{R}^n$.

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the solution $x^* \in \mathbb{R}^n$.

Initialization

$k := 0$.

Repeat

Calculate d_k solution of $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$.

$x_{k+1} := x_k + d_k$.

$k := k + 1$.

Until $\left\|\nabla f(x_k)\right\| \leq \varepsilon$

$x^* = x_k$.

points. For instance, by applying Algorithm 10.1 to minimize the function of Example 5.8, with $x_0 = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$, Newton's local method converges rapidly toward

$$x^* = \begin{pmatrix} 0 \\ \pi/2 \end{pmatrix}, \qquad \nabla f(x^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \nabla^2 f(x^*) = \begin{pmatrix} 1 & -1 \\ -1 & 0 \end{pmatrix},$$

which does not satisfy the second-order necessary optimality conditions (Theorem 5.1) and is consequently not a local minimum. It is actually a saddle point. The iterations of the method are illustrated in Figure 10.1 and in Table 10.1.

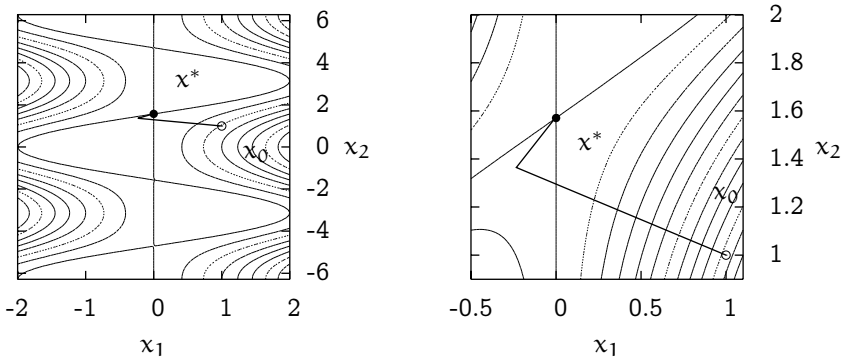
Since Newton's local method cannot be used as is, we develop alternative methods in the following chapters. However, the fast rate of convergence of Newton's method prompts us to use it when appropriate. We conclude this chapter with a geometric interpretation of Newton's local method in the context of optimization.

10.2 Geometric interpretation

The main idea of Newton's method when solving non linear equations is to replace a complicated non linear function by a simpler model. In the context of equations, this model is linear (Definition 7.9). Newton's local method, applied in the context of optimization, can be motivated in a similar manner. In this case, the model is no

Table 10.1: Newton's local method for the minimization of Example 5.8

k	x_k	$\nabla f(x_k)$	$\|\nabla f(x_k)\|$	$f(x_k)$
0	1.00000000e+00	1.54030230e+00	1.75516512e+00	1.04030231e+00
	1.00000000e+00	-8.41470984e-01		
1	-2.33845128e-01	-2.87077027e-02	2.30665381e-01	7.53121618e-02
	1.36419220e+00	2.28871986e-01		
2	1.08143752e-02	-3.22524807e-03	1.12840544e-02	-9.33543838e-05
	1.58483641e+00	-1.08133094e-02		
3	-2.13237666e-06	9.22828706e-07	2.32349801e-06	8.79175320e-12
	1.57079327e+00	2.13237666e-06		
4	1.99044272e-17	8.11347449e-17	8.35406072e-17	1.35248527e-25
	1.57079632e+00	-1.99044272e-17		



(a) Iterates (b) Zoom

Figure 10.1: Iterates of Newton's local method for Example 5.8

longer linear, but quadratic. It is obtained thanks to Taylor's second-order theorem (Theorem C.2).

Definition 10.1 (Quadratic model of a function)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function. The quadratic model of f in $\widehat{x}$ is a function $m_{\widehat{x}} : \mathbb{R}^n \to \mathbb{R}$ defined by

$$m_{\widehat{x}}(x) = f(\widehat{x}) + (x - \widehat{x})^T \nabla f(\widehat{x}) + \frac{1}{2}(x - \widehat{x})^T \nabla^2 f(\widehat{x})(x - \widehat{x}), \tag{10.2}$$

where $\nabla f(\widehat{x})$ is the gradient of f in $\widehat{x}$ (Definition 2.5) and $\nabla^2 f(\widehat{x})$ is the hessian matrix of f in $\widehat{x}$ (Definition 2.19). Defining $d = x - \widehat{x}$, we obtain the equivalent formulation:

$$m_{\widehat{x}}(\widehat{x} + d) = f(\widehat{x}) + d^T \nabla f(\widehat{x}) + \frac{1}{2} d^T \nabla^2 f(\widehat{x}) d. \tag{10.3}$$

Note that Definition 10.1 is consistent with Definition 2.28, with $Q = \nabla^2 f(\widehat{x})$, $g = \nabla f(\widehat{x})$ and $c = f(\widehat{x})$. If we minimize the model instead of the function, we get the problem

$$\min_{d \in \mathbb{R}^n} m_{\widehat{x}}(\widehat{x} + d) = f(\widehat{x}) + d^T \nabla f(\widehat{x}) + \frac{1}{2} d^T \nabla^2 f(\widehat{x}) d. \tag{10.4}$$

The sufficient first-order optimality condition (Theorem 5.7) for (10.4) is written as:

$$\nabla m_{\widehat{x}}(\widehat{x} + d) = \nabla f(\widehat{x}) + \nabla^2 f(\widehat{x}) d = 0, \tag{10.5}$$

i.e.,

$$d = -\nabla^2 f(\widehat{x})^{-1} \nabla f(\widehat{x}) \tag{10.6}$$

or

$$x = \widehat{x} - \nabla^2 f(\widehat{x})^{-1} \nabla f(\widehat{x}). \tag{10.7}$$

The sufficient second-order optimality condition requires that the matrix $\nabla^2 f(\widehat{x})$ be positive definite.

Note also that (10.7) is exactly the main formula of Newton's local method (Algorithm 10.1).

When the hessian matrix of the function is positive definite in x_k, an iteration of Newton's local method corresponds to minimizing the quadratic model of the function in x_k and thus defining

$$x_{k+1} = \operatorname{argmin}_{x \in \mathbb{R}^n} m_{x_k}(x)\,. \tag{10.8}$$

Algorithm 10.2: Newton's local method by quadratic modeling

Objective

To find (an approximation of) a solution to the system

$$\nabla f(x) = 0\,. \tag{10.9}$$

Input

The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
The hessian of the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.
A first approximation of the solution $x_0 \in \mathbb{R}^n$.
The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the solution $x^* \in \mathbb{R}^n$.

Initialization

$k := 0$.

Repeat

Create the quadratic model

$$m_{x_k}(x_k + d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2}\, d^T \nabla^2 f(x_k) d\,. \tag{10.10}$$

Calculate

$$d_k = \operatorname{argmin}_d m_{x_k}(x_k + d) \tag{10.11}$$

using the direct method (Algorithm 9.1) or the conjugate gradient algorithm (Algorithm 9.2).

$x_{k+1} := x_k + d_k$.
$k := k + 1$.

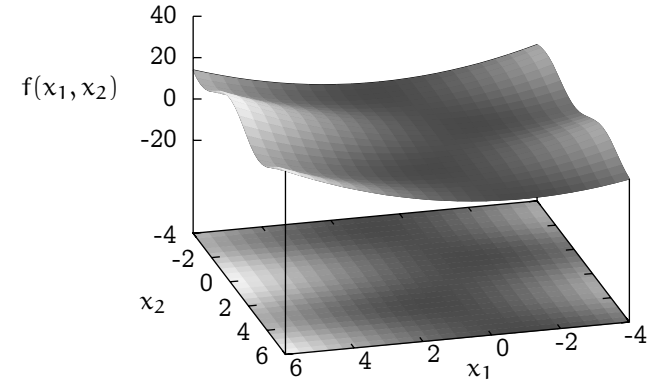
Until $\|\nabla f(x_k)\| \leq \varepsilon$

$x^* = x_k$.

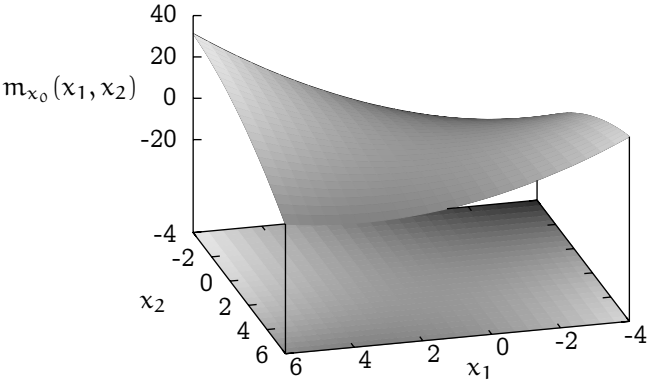
Algorithm 10.2 is the version of Algorithm 10.1 using the quadratic model. It is important to note that Algorithms 10.1 and 10.2 are equivalent only when the hessian matrix at the current iterate is positive definite, that is when the function is locally convex at the current iterate. When solving Example 5.8, illustrated in Table 10.1, this interpretation is not valid. Indeed, we have

$$\nabla^2 f(x_0) = \begin{pmatrix} \texttt{1.00000000e+00} & \texttt{-8.41470984e-01} \\ \texttt{-8.41470984e-01} & \texttt{-5.40302305e-01} \end{pmatrix},$$

for which the eigenvalues are `-9.10855416e-01` and `1.37055311e+00`. This matrix is not positive definite. Therefore, the necessary optimality condition for the quadratic problem is never satisfied, and there exists no solution to the minimization problem of the quadratic model in x_0. The model is shown in Figure 10.2(b). It is not bounded from below. Therefore, Algorithm 10.2 cannot be applied.



(a) Objective function



(b) Quadratic model $m_{x_0}(x)$

Figure 10.2: Quadratic model for Example 5.8

We illustrate with Example 10.2 the limitations of Newton's local method by using the geometric interpretation (Algorithm 10.2).

Example 10.2 (Quadratic model)**.** Consider the function

$$f(x) = -x^4 + 12x^3 - 47x^2 + 60x\,. \tag{10.12}$$

We consider three different points and apply Newton's local method.

1. $x_k = 3$. The quadratic model is

$$m_3(x) = 7x^2 - 48x + 81\,,$$

for which the minimum is $x_{k+1} = 24/7 \approx 3.4286$. Moreover, $f(x_k) = 0$ and $f(x_{k+1}) \approx -1.32$ and then $f(x_{k+1}) < f(x_k)$. This is a favorable case. The model is shown in Figure 10.3. It can be seen that there is a good adequacy between the model and the function in the neighborhood of the iterates x_k and x_{k+1}.

2. $x_k = 4$. The quadratic model is

$$m_4(x) = x^2 - 4x\,,$$

for which the minimum is $x_{k+1} = 2$. In this case, $f(x_k) = 0$ and $f(x_{k+1}) = 12$. The iterate generated by the method is worse (in terms of the value of the objective function) than the current iterate. The model is illustrated in Figure 10.4. It can be seen that the iterate x_{k+1} lies in a region where the model is a poor approximation of the function. Recall that Taylor's theorem guarantees a good adequacy only in a neighborhood of x_k, without mentioning the size of that neighborhood. Here, the iterate x_{k+1} clearly lies outside it.

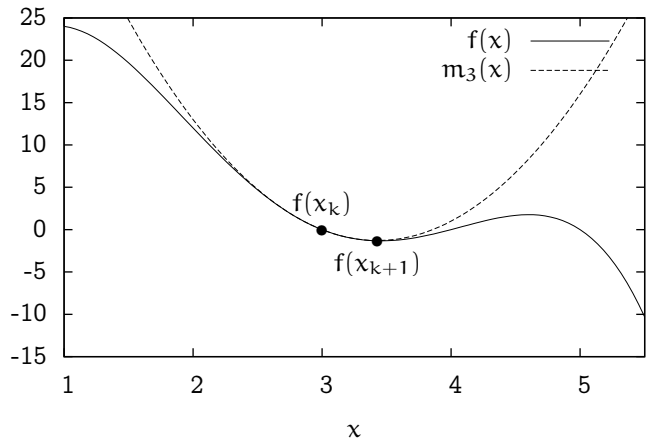


Figure 10.3: Illustration of Example 10.2 with $x_k = 3$

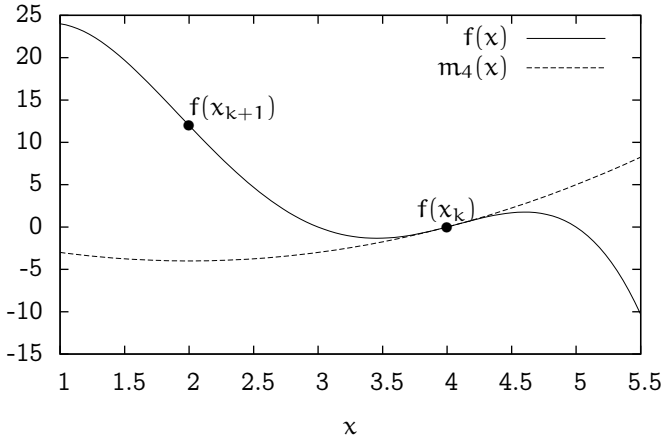


Figure 10.4: Illustration of Example 10.2 with $x_k = 4$

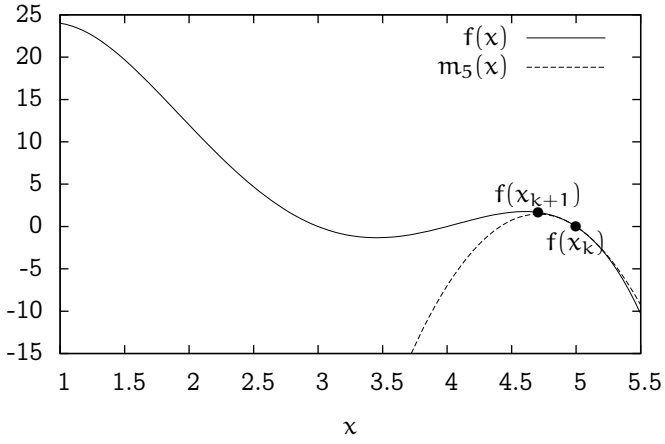


Figure 10.5: Illustration of Example 10.2 with $x_k = 5$

3. $x_k = 5$. The quadratic model is

$$m_5(x) = -17\,x^2 + 160\,x - 375\,.$$

This model is concave (its second derivative is negative) and it is not bounded from below. It is not possible to minimize it and Algorithm 10.2 does not work. Applying Newton's local method (Algorithm 10.1) in $x_k = 5$ corresponds to *maximizing* this quadratic model, which goes against the desired effect.

We conclude this chapter by defining two particular points that play a role later on. On the one hand, the point obtained during the iteration of Newton's local method is often called *Newton's point*.

Definition 10.3 (Newton's point)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function and let us take $x_k \in \mathbb{R}^n$ such that $\nabla^2 f(x_k)$ is positive definite. Newton's point of f in x_k is the point

$$x_N = x_k + d_N \,, \tag{10.13}$$

where d_N is the solution to the system of equations

$$\nabla^2 f(x_k) d_N = -\nabla f(x_k) \,. \tag{10.14}$$

The system (10.14) is often called Newton's equations.

Newton's point minimizes the quadratic model of the function in x_k. If $\nabla^2 f(x_k)$ is positive definite, we have a minimum of the quadratic model in x_k. On the other hand, the point minimizing the quadratic model in the direction with the steepest descent is called the *Cauchy point.*[1]

Augustin-Louis Cauchy was born in Paris on August 21, 1789. Cauchy was a pioneer in the study of analysis. In 1814, he published a thesis on definite integrals that became the basis of complex functions theory. One year later, he was appointed professor of analysis at Ecole Polytechnique. In his work, he tried to demonstrate the proposals that had been put forward so far as evident and for which there was no proof. Cauchy was the first to provide rigorous conditions for the convergence of infinite series and he also gave a precise definition of the integral. He was a prolific researcher (he wrote approximately 800 mathematical articles), and was unliked by most of his colleagues. He was a convinced royalist and legitimist, and spent some time in Italy after having refused to pledge allegiance. He resumed his chair at the Sorbonne in 1848 after the abdication of Louis-Philippe. He kept it until his death in Sceaux, on May 22, 1857.

Figure 10.6: Augustin-Louis Cauchy

Definition 10.4 (Cauchy's point)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function and let us take $x_k \in \mathbb{R}^n$. The Cauchy point of f in x_k is the point x_C that minimizes the quadratic model of f in the direction with the steepest descent, i.e.,

$$x_C = x_k - \alpha_C \nabla f(x_k) \,, \tag{10.15}$$

where

$$\alpha_C \in \operatorname{argmin}_{\alpha \in \mathbb{R}_0^+} m_{x_k}\left(x_k - \alpha \nabla f(x_k)\right) . \tag{10.16}$$

[1] We here refer to Dennis and Schnabel (1996, page 139). Other references (particularly Conn et al., 2000, page 124) define Cauchy's point as the minimum of the quadratic model along the arc obtained by projecting the steepest descent direction onto the trust region.

It is well defined if f is convex in the direction of the gradient. In this case, there is only one minimizer. Using (10.3), we obtain

$$\alpha_C = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_k)^T \nabla^2 f(x_k) \nabla f(x_k)}. \tag{10.17}$$

10.3 Exercises

For each of the following problems, determine Cauchy's point x_C and Newton's point x_N in $\bar{x}$. Each time, compare the value of the objective function at these three points.

Exercise 10.1. $\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} i x_i^2$, $\bar{x} = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T$.

Exercise 10.2. $\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} x_i^2$, any $\bar{x}$.

Exercise 10.3. $\min_{x \in \mathbb{R}^2} 2x_1 x_2 \, e^{-(4x_1^2 + x_2)/8}$, $\bar{x} = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$.

Exercise 10.4. $\min_{x \in \mathbb{R}^2} 2x_1 x_2 \, e^{-(4x_1^2 + x_2)/8}$, $\bar{x} = \begin{pmatrix} 4 & 4 \end{pmatrix}^T$.

Exercise 10.5. $\min_{x \in \mathbb{R}^2} 100 \left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$, $\bar{x} = \begin{pmatrix} -1 & -1 \end{pmatrix}^T$.

Chapter 11

Descent methods and line search

Newton's local method may be fast, but it fails regularly. We address the problem in a different manner. Intuitively, in order to identify iterates with a lower value of the objective function, we choose to follow the direction with the steepest descent given by the opposite of the gradient. This idea turns out to be functional, but disastrously slow. We demonstrate in this chapter how to correct this shortcoming, and how to combine the two approaches in order to obtain a method that is both fast and robust.

Contents

We leave Newton's method aside for now (we return to it in Section 11.5) and focus on specific methods for an optimization problem. The main idea is simple. Since we seek the minimum of a function, we attempt to descend, i.e., to generate a sequence of iterates $(x_k)_k$ such that

$$f(x_{k+1}) \leq f(x_k), \qquad k = 1, 2, \ldots$$

Theorem 2.11 ensures that such an iterate can be found in a direction d such that $\nabla f(x_k)^T d < 0$. The methods presented here, often called *descent methods*, consist of a process involving three main steps:

1. Find a direction d_k such that $\nabla f(x_k)^T d_k < 0$.

2. Find a step α_k such that $f(x_k + \alpha_k d_k) < f(x_k)$.
3. Calculate $x_{k+1} = x_k + \alpha_k d_k$ and verify a stopping criterion.

11.1 Preconditioned steepest descent

The first idea that comes to mind to define a concrete descent method is to invoke the theorem of the steepest descent (Theorem 2.13) and to choose $d_k = -\nabla f(x_k)$. Indeed, it is in this direction that the function has its steepest descent. We often refer to this method as the *steepest descent method*. An iteration of this method consists in

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)\,. \tag{11.1}$$

When it comes to the step α_k, we choose for the moment one that gives the largest reduction of the function in the direction d_k, i.e.,

$$\alpha_k \in \operatorname{argmin}_{\alpha \in \mathbb{R}_0^+} f(x_k + \alpha d_k)\,. \tag{11.2}$$

In the presence of multiple minima, it is common to use the first one, that is α_k is the small element of $\operatorname{argmin}_{\alpha \in \mathbb{R}_0^+} f(x_k + \alpha d_k)$. Example 11.1 illustrates this method for a simple case.

Example 11.1 (Steepest descent method)**.** We minimize the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f(x) = \frac{1}{2} x_1^2 + \frac{9}{2} x_2^2\,, \tag{11.3}$$

by using the steepest descent method. Let x_k be the current iterate. The direction of the steepest descent is

$$d_k = -\nabla f(x_k) = \begin{pmatrix} -(x_k)_1 \\ -9(x_k)_2 \end{pmatrix}.$$

To calculate the step α_k, we solve the problem in one dimension

$$\min_{\alpha} f\big(x_k - \alpha \nabla f(x_k)\big) = \min_{\alpha} \frac{1}{2}\left((x_k)_1 - \alpha (x_k)_1\right)^2 + \frac{9}{2}\left((x_k)_2 - 9\alpha (x_k)_2\right)^2,$$

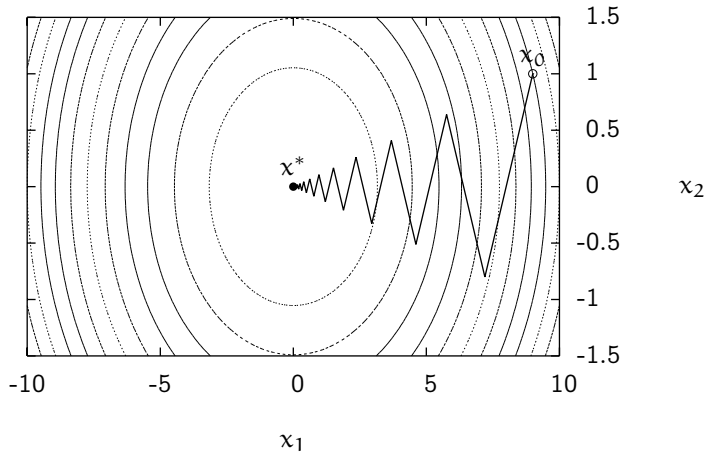
for which the optimal solution is

$$\alpha = \frac{(x_k)_1^2 + 81\,(x_k)_2^2}{(x_k)_1^2 + 729\,(x_k)_2^2}\,.$$

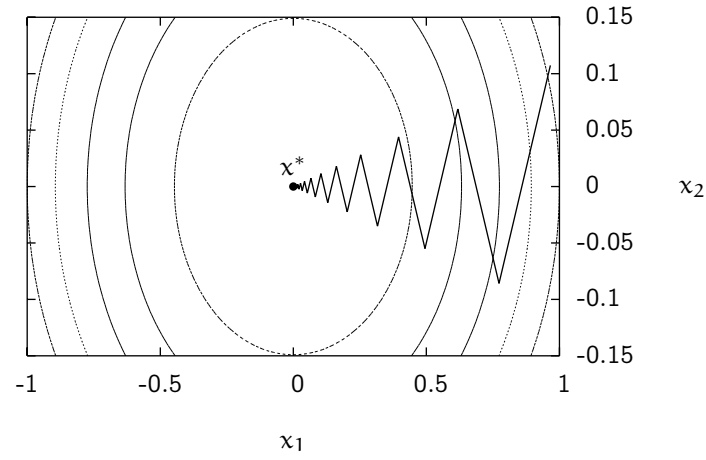
At each iteration, the steepest descent method generates the point

$$x_{k+1} = x_k + \frac{(x_k)_1^2 + 81\,(x_k)_2^2}{(x_k)_1^2 + 729\,(x_k)_2^2} \begin{pmatrix} -(x_k)_1 \\ -9(x_k)_2 \end{pmatrix}.$$

By applying this algorithm, starting from the point $x_0 = \begin{pmatrix} 9 & 1 \end{pmatrix}^T$, we obtain the iterations illustrated in Figure 11.1 and listed (in part) in Table 11.1.



(a) Iterations



(b) Zoom

Figure 11.1: Steepest descent method: illustration of Example 11.1.

In Example 11.1, it is remarkable how slow the steepest descent method is, even though the function to minimize is simple. The zigzag behavior illustrated in Figure 11.1 is characteristic. We show next that the performance can be improved by preconditioning the function (the concepts of *conditioning* and *preconditioning* are discussed in Section 2.5).

Table 11.1: Steepest descent method for Example 11.1

k	$(x_k)_1$	$(x_k)_2$	$\nabla f(x_k)_1$	$\nabla f(x_k)_2$	α_k	$f(x_k)$
0	+9.000000E+00	+1.000000E+00	+9.000000E+00	+9.000000E+00	0.2	+4.500000E+01
1	+7.200000E+00	-8.000000E-01	+7.200000E+00	-7.200000E+00	0.2	+2.880000E+01
2	+5.760000E+00	+6.400000E-01	+5.760000E+00	+5.760000E+00	0.2	+1.843200E+01
3	+4.608000E+00	-5.120000E-01	+4.608000E+00	-4.608000E+00	0.2	+1.179648E+01
4	+3.686400E+00	+4.096000E-01	+3.686400E+00	+3.686400E+00	0.2	+7.549747E+00
5	+2.949120E+00	-3.276800E-01	+2.949120E+00	-2.949120E+00	0.2	+4.831838E+00
⋮						
20	+1.037629E-01	+1.152922E-02	+1.037629E-01	+1.037629E-01	0.2	+5.981526E-03
21	+8.301035E-02	-9.223372E-03	+8.301035E-02	-8.301035E-02	0.2	+3.828177E-03
22	+6.640828E-02	+7.378698E-03	+6.640828E-02	+6.640828E-02	0.2	+2.450033E-03
23	+5.312662E-02	-5.902958E-03	+5.312662E-02	-5.312662E-02	0.2	+1.568021E-03
24	+4.250130E-02	+4.722366E-03	+4.250130E-02	+4.250130E-02	0.2	+1.003534E-03
25	+3.400104E-02	-3.777893E-03	+3.400104E-02	-3.400104E-02	0.2	+6.422615E-04
⋮						
50	+1.284523E-04	+1.427248E-05	+1.284523E-04	+1.284523E-04	0.2	+9.166662E-09
51	+1.027618E-04	-1.141798E-05	+1.027618E-04	-1.027618E-04	0.2	+5.866664E-09
52	+8.220947E-05	+9.134385E-06	+8.220947E-05	+8.220947E-05	0.2	+3.754665E-09
53	+6.576757E-05	-7.307508E-06	+6.576757E-05	-6.576757E-05	0.2	+2.402985E-09
54	+5.261406E-05	+5.846007E-06	+5.261406E-05	+5.261406E-05	0.2	+1.537911E-09
55	+4.209125E-05	-4.676805E-06	+4.209125E-05	-4.209125E-05	0.2	+9.842628E-10

Example 11.2 (Preconditioned steepest descent method)**.** We minimize the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f(x) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2, \tag{11.4}$$

by using the steepest descent method and the preconditioning technique from Section 2.5. We have

$$\nabla f(x) = \begin{pmatrix} x_1 \\ 9x_2 \end{pmatrix} \quad \text{and} \quad \nabla^2 f(x) = \begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}^T.$$

We use the equations (2.53) and (2.54) to define the change of variables

$$\begin{aligned} x_1' &= x_1 \\ x_2' &= 3x_2 \end{aligned}$$

and we obtain the function

$$\tilde{f}(x') = \frac{1}{2}x_1'^2 + \frac{9}{2}\left(\frac{1}{3}x_2'\right)^2 = \frac{1}{2}x_1'^2 + \frac{1}{2}x_2'^2.$$

Therefore, the direction of the steepest descent is

$$d = -\nabla\tilde{f}(x') = \begin{pmatrix} -x_1' \\ -x_2' \end{pmatrix}.$$

To calculate the step α, we solve the problem in one dimension

$$\min_\alpha \tilde{f}(x' - \alpha\nabla\tilde{f}(x')) = \min_\alpha \frac{1}{2}\left(x_1' - \alpha x_1'\right)^2 + \frac{1}{2}\left(x_2' - \alpha x_2'\right)^2,$$

for which the optimal solution is $\alpha = 1$. Then, regardless of the current iterate x', the steepest descent method always generates the point

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} + \begin{pmatrix} -x_1' \\ -x_2' \end{pmatrix} = 0,$$

which is the optimal solution to the problem. In this case, the method identifies the minimum of the function in a single iteration.

Clearly, the performance of the steepest descent method can be significantly improved when the function is preconditioned. We can generalize this idea. Let H_k be a symmetric positive definite matrix such that $H_k = L_k L_k^T$. We use L_k to define a change of variables, according to Definition 2.32, i.e.,

$$x' = L_k^T x. \tag{11.5}$$

The steepest descent method for the variables x' is written as

$$x_{k+1}' = x_k' - \alpha_k \nabla\tilde{f}(x_k'). \tag{11.6}$$

By using (2.52), (11.6) is expressed as

$$x'_{k+1} = x'_k - \alpha_k L_k^{-1} \nabla f(L_k^{-T} x'_k)\,. \tag{11.7}$$

In the original variables, we obtain, by using (11.5),

$$L_k^T x_{k+1} = L_k^T x_k - \alpha_k L_k^{-1} \nabla f(x_k) \tag{11.8}$$

or, by multiplying by L_k^{-T}

$$\begin{aligned} x_{k+1} &= x_k - \alpha_k L_k^{-T} L_k^{-1} \nabla f(x_k) \\ &= x_k - \alpha_k H_k^{-1} \nabla f(x_k)\,. \end{aligned} \tag{11.9}$$

Therefore, the preconditioned steepest descent method gives

$$x_{k+1} = x_k + \alpha_k d_k \tag{11.10}$$

with

$$d_k = -H_k^{-1} \nabla f(x_k)\,. \tag{11.11}$$

If we denote $D_k = H_k^{-1}$, we obtain in a similar manner

$$d_k = -D_k \nabla f(x_k)\,. \tag{11.12}$$

It is admittedly a descent method. Indeed, when $\nabla f(x_k) \neq 0$,

$$\nabla f(x_k)^T d_k = -\nabla f(x_k)^T D_k \nabla f(x_k) < 0\,,$$

because H_k is positive definite as is D_k. We note that the index k of D_k enables us to precondition the method differently for each iteration.

It is important to note that Algorithm 11.1 is not complete. Indeed, nothing is specified regarding the manner in which to generate the positive definite matrices D_k. Moreover, the suggested method to calculate α_k at step 15 is not trivial to implement. Finally, certain additional assumptions are necessary in order to ensure that the method converges.

Section 11.2 describes algorithms that are designed to identify an (approximation) of a local minimum of the function along the selected direction d_k, that is

$$\alpha_k \in \operatorname{argmin}_{\alpha \geq 0} f(x_k + \alpha d_k). \tag{11.13}$$

However, it is not necessary to select a step α_k that minimizes the function along d_k. In order to save computing time, we propose in Section 11.3 a characterization of steps that are "acceptable." Section 11.3 proposes an inexact line search algorithm based on this characterization. After including the line search approach into the steepest descent algorithm in Section 11.4, we propose in Section 11.5 a way to define the preconditioning matrices D_k, inspired by Newton's method.

Algorithm 11.1: Preconditioned steepest descent method

Objective

To find (an approximation of) a local minimum of the problem

$$\min_{x \in \mathbb{R}^n} f(x). \tag{11.14}$$

Input

The differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
A family of preconditioners $(D_k)_k$ such that D_k is positive definite for all k.
An initial solution $x_0 \in \mathbb{R}^n$.
The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}^n$.

Initialization

$k := 0$.

Repeat

$d_k := -D_k \nabla f(x_k)$.
Determine α_k, for instance $\alpha_k \in \operatorname{argmin}_{\alpha \geq 0} f(x_k + \alpha d_k)$.
$x_{k+1} := x_k + \alpha_k d_k$.
$k := k + 1$.

Until $\|\nabla f(x_k)\| \leq \varepsilon$

$x^* := x_k$.

11.2 Exact line search

As suggested in Algorithm 11.1, the step to perform along the direction d_k may be obtained from solving (11.13). We call this way of calculating the step size an "exact line search," referring to the fact that we are seeking the exact minimum.

The optimization problem (11.13) is a problem with one variable, α, and can be written as

$$\min_{\alpha \geq 0} h(\alpha) = f(x_k + \alpha d_k), \tag{11.15}$$

where x_k is the current iterate and d_k is a descent direction. From Theorem 2.11, we know that $\alpha = 0$ is not a local minimum of this function. Therefore, the constraint $\alpha \geq 0$ is inactive at the optimal solution and can be ignored (see Theorem 3.5).

Clearly, Newton's method can be used to solve the problem, if a good approximation of the local optimum is known. The derivatives of h are

$$h'(\alpha) = \frac{dh(\alpha)}{d\alpha} = \nabla f(x_k + \alpha d_k)^T d_k, \tag{11.16}$$

and

$$h''(\alpha) = \frac{d^2 h(\alpha)}{d\alpha^2} = d_k^T \nabla^2 f(x_k + \alpha d_k)^T d_k. \tag{11.17}$$

We describe two other techniques: the quadratic interpolation method and the golden section method.

11.2.1 Quadratic interpolation

The quadratic interpolation method requires that the function h is continuous and uses only the value of the function, not its derivatives.

Consider three distinct points $a < b < c$ such that $h(a) > h(b)$ and $h(c) > h(b)$, so that a local minimum of the function lies in the interval $[a, c]$ by continuity of h. Such points can be generated by Algorithm 11.2. Note that the condition $h(\delta) < h(0)$ guarantees that the algorithm does not stop at the first iteration, that is, when only two points have been generated.

Algorithm 11.2: Initialization of the exact line search

Objective
  Find a, b, and c such that $a < b < c$, $h(a) > h(b)$ and $h(c) > h(b)$.
Input
  A continuous function $h : \mathbb{R} \to \mathbb{R}$ such that the function decreases at 0.
  δ such that $h(\delta) < h(0)$.
Initialization
  $x_0 := 0$
  $x_1 := \delta$
  $k := 1$
Repeat
  $x_{k+1} := 2x_k$
  $k = k + 1$
Until $h(x_k) > h(x_{k-1})$
$a = x_{k-2}$
$b = x_{k-1}$
$c = x_k$

We interpolate a parabola q at the three points. To do so, we identify the parameters β_1, β_2, and β_3 of the quadratic function

$$q(x) = \beta_1 (x - a)(x - b) + \beta_2 (x - a) + \beta_3 (x - b), \tag{11.18}$$

such that $q(a) = h(a)$, $q(b) = h(b)$ and $q(c) = h(c)$. As $q(a) = h(a)$, we obtain immediately that

$$\beta_3 = \frac{h(a)}{a - b}. \tag{11.19}$$

Similarly, as $q(b) = h(b)$, we have

$$\beta_2 = \frac{h(b)}{b-a}. \tag{11.20}$$

From the last interpolation condition, $q(c) = h(c)$, we obtain after some straightforward derivation,

$$\beta_1 = \frac{(b-c)h(a) + (c-a)h(b) + (a-b)h(c)}{(a-b)(c-a)(c-b)}. \tag{11.21}$$

As $q(a) > q(b)$ and $q(c) > q(b)$, the quadratic q is convex, and its minimum x^* corresponds to the point where the derivative is 0. As

$$q'(x^*) = \beta_1(2x - a - b) + \beta_2 + \beta_3 = 0, \tag{11.22}$$

we have

$$x^* = \frac{\beta_1(a+b) - \beta_2 - \beta_3}{2\beta_1}. \tag{11.23}$$

The numerator $\beta_1(a+b) - \beta_2 - \beta_3$ is equal to

$$\frac{h(a)(b^2-c^2) + h(b)(c^2-a^2) + h(c)(a^2-b^2)}{(a-b)(c-a)(c-b)},$$

so that

$$x^* = \frac{1}{2}\frac{h(a)(b^2-c^2) + h(b)(c^2-a^2) + h(c)(a^2-b^2)}{h(a)(b-c) + h(b)(c-a) + h(c)(a-b)}. \tag{11.24}$$

Now, we need to generate a new set of 3 points a^+, b^+, c^+, with the same properties ($a^+ < b^+ < c^+$, $h(a^+) > h(b^+)$, $h(c^+) > h(b^+)$), and such that the interval $[a^+, c^+]$ is strictly smaller than $[a, c]$. We assume that $h(x^*) \neq h(b)$. If it happens not to be the case, perturb x^* by a small amount to enforce $h(x^*) \neq h(b)$. Note that assuming $h(x^*) \neq h(b)$ implies that $x^* \neq b$.

Suppose first that x^* lies between b and c, that is $a < b < x^* < c$.

- If $h(x^*) > h(b)$, we set $a^+ = a$, $b^+ = b$, and $c^+ = x^*$. The condition $a^+ < b^+ < c^+$ is trivially verified. The condition $h(a^+) > h(b^+)$ is $h(a) > h(b)$, which is verified by assumption, and the condition $h(c^+) > h(b^+)$ is $h(x^*) > h(b)$, which is the condition of the case that is treated.
- If $h(x^*) < h(b)$, we set $a^+ = b$, $b^+ = x^*$, and $c^+ = c$. The condition $a^+ < b^+ < c^+$ is trivially verified. The condition $h(a^+) > h(b^+)$ is $h(b) > h(x^*)$, which is the condition of the case being treated. The condition $h(c^+) > h(b^+)$ is $h(c) > h(x^*)$, which is verified because $h(c) > h(b) > h(x^*)$.

Suppose next that x^* lies between a and b, that is $a < x^* < b < c$.

- If $h(x^*) > h(b)$, we set $a^+ = x^*$, $b^+ = b$, and $c^+ = c$. The condition $a^+ < b^+ < c^+$ is trivially verified. The condition $h(a^+) > h(b^+)$ is $h(x^*) > h(b)$, which is the condition being treated. The condition $h(c^+) > h(b^+)$ is $h(c) > h(b)$, verified by assumption.

- If $h(x^*) < h(b)$, we set $a^+ = a$, $b^+ = x^*$, and $c^+ = b$. The condition $a^+ < b^+ < c^+$ is trivially verified. The condition $h(c^+) > h(b^+)$ is $h(b) > h(x^*)$, which is the condition being treated. The condition $h(a^+) > h(b^+)$ is $h(a) > h(x^*)$, which is verified because $h(a) > h(b) > h(x^*)$.

The complete procedure is described as Algorithm 11.3.

Algorithm 11.3: Exact line search: quadratic interpolation

Objective
  Find a local minimum of $\min_{\alpha \geq 0} h(\alpha)$
Input
  A continuous function $h : \mathbb{R} \to \mathbb{R}$ such that the function decreases at 0.
  A step δ such that $h(\delta) < h(0)$.
  The desired precision $\varepsilon > 0$.
Output
  α^* local minimum of $\min_{\alpha \geq 0} h(\alpha)$
Initialization
  Compute a, b, and c such that $a < b < c$, $h(a) > h(b)$, and $h(c) > h(b)$ using Algorithm 11.2.
Repeat

$$x^* := \frac{1}{2} \frac{h(a)(b^2 - c^2) + h(b)(c^2 - a^2) + h(c)(a^2 - b^2)}{h(a)(b - c) + h(b)(c - a) + h(c)(a - b)}.$$

  while $h(x^*) = h(b)$ **do** x^* is perturbed to avoid being stalled
    if *b-a < c-b* **then**
      $x^* := x^* + \varepsilon/2$
    else
      $x^* := x^* - \varepsilon/2$
  if $x^* > b$ **then**
    if $h(x^*) > h(b)$ **then** the new triplet is a, b, x^*
      $c := x^*$
    else the new triplet is b, x^*, c
      $a := b$
      $b := x^*$
  else if $x^* < b$ **then**
    if $h(x^*) > h(b)$ **then** the new triplet is x^*, b, c
      $a := x^*$
    else the new triplet is a, x^*, b
      $c := b$
      $b := x^*$
Until $\max(h(a), h(c)) - h(b) \leq \varepsilon$ *or* $c - a \leq \varepsilon$
$\alpha^* := b$

Example 11.3 (Exact line search with quadratic interpolation)**.** Consider the function

$$h(x) = (2 + x)\cos(2 + x). \tag{11.25}$$

In order to identify a local minimum of h, we apply Algorithm 11.3 with $\delta = 6$ and $\varepsilon = 10^{-3}$. Note that $-1.1640 = h(\delta) < h(0) = -0.83229$. This value of δ has been chosen to make the example illustrative. In practice, a smaller value is used (try with $\delta = 2$).

The first four iterations are illustrated in Figures 11.2–11.5, and all iterations are reported in Table 11.2.

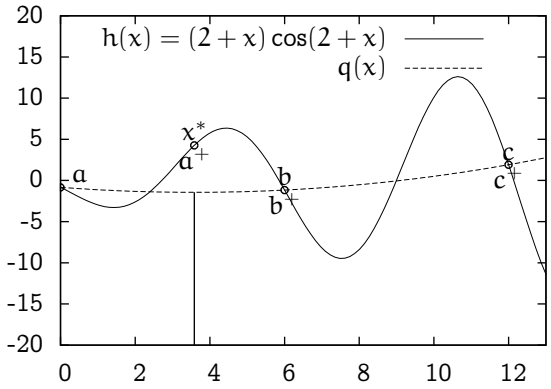


Figure 11.2: Quadratic interpolation – Iteration 1

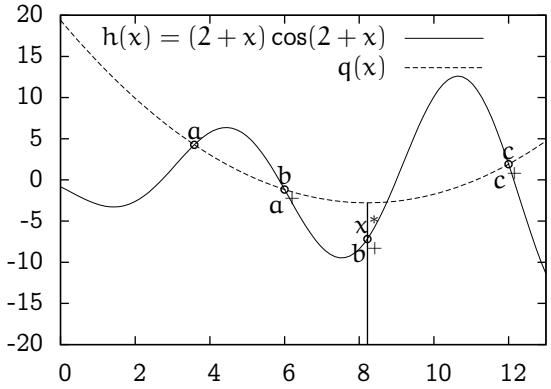


Figure 11.3: Quadratic interpolation – Iteration 2

Table 11.2: Iterates of Example 11.3

Iter.	a	b	c	x^*	h(a)	h(b)	h(c)	$h(x^*)$
1	0.0	6.0	12.0	3.58364	-0.832294	-1.164	1.91432	4.27225
2	3.58364	6.0	12.0	8.21855	4.27225	-1.164	1.91432	-7.16487
3	6.0	8.21855	12.0	8.69855	-1.164	-7.16487	1.91432	-3.13122
4	6.0	8.21855	8.69855	7.43782	-1.164	-7.16487	-3.13122	-9.43702
5	6.0	7.43782	8.21855	7.45558	-1.164	-9.43702	-7.16487	-9.45109
6	7.43782	7.45558	8.21855	7.52836	-9.43702	-9.45109	-7.16487	-9.47729
7	7.45558	7.52836	8.21855	7.52898	-9.45109	-9.47729	-7.16487	-9.47729
8	7.52836	7.52898	8.21855	7.52933	-9.47729	-9.47729	-7.16487	-9.47729
9	7.52898	7.52933	8.21855	7.52933	-9.47729	-9.47729	-7.16487	-9.47729
10	7.52933	7.52933	8.21855	7.52933	-9.47729	-9.47729	-7.16487	-9.47729
11	7.52933	7.52933	8.21855	7.52933	-9.47729	-9.47729	-7.16487	-9.47729
12	7.52933	7.52933	8.21855	7.52933	-9.47729	-9.47729	-7.16487	-9.47729
13	7.52933	7.52933	7.52933	7.52933	-9.47729	-9.47729	-9.47729	-9.47729

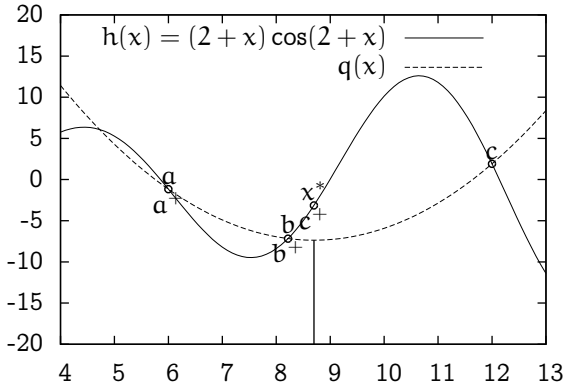


Figure 11.4: Quadratic interpolation – Iteration 3

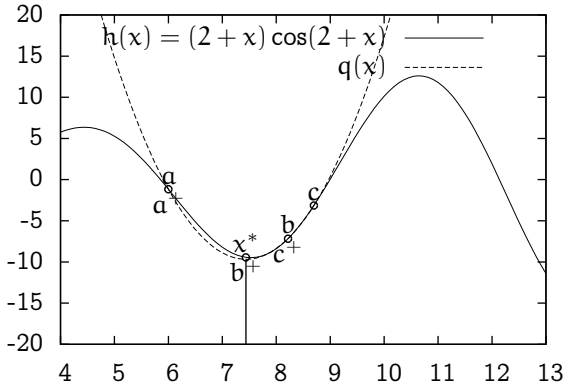


Figure 11.5: Quadratic interpolation – Iteration 4

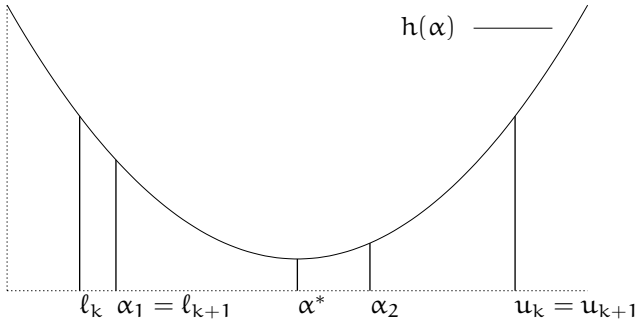
11.2.2 Golden section

The golden section method requires that the function h be strictly unimodal on an interval $[0, T]$ (see Definition B.6), and α^* the global minimum of h on $[0, T]$ (Definition 1.7). The method generates a sequence of intervals $[\ell_k, u_k]$ such that for each k,

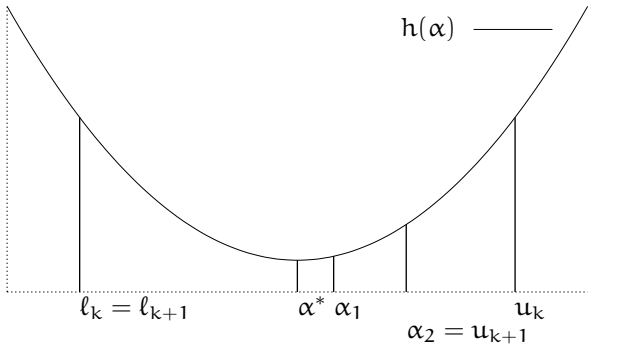
- $[\ell_{k+1}, u_{k+1}] \subset [\ell_k, u_k]$, and
- $\alpha^* \in [\ell_k, u_k]$.

Consider two points α_1^k and α_2^k such that $\ell_k < \alpha_1^k < \alpha_2^k < u_k$. If $h(\alpha_1^k) > h(\alpha_2^k)$, then h is decreasing from α_1^k and α_2^k. Therefore, the global minimum α^* cannot be smaller than α_1^k (due to the strict unimodality of h). Therefore, $\alpha^* \in [\alpha_1^k, u_k]$, which is the next interval of the sequence: $\ell_{k+1} = \alpha_1^k$ and $u_{k+1} = u_k$. If $h(\alpha_1^k) < h(\alpha_2^k)$, then h is increasing from α_1^k and α_2^k. Therefore, the global minimum α^* cannot be greater than α_2^k (due to the strict unimodality of h). Therefore, $\alpha^* \in [\ell_k, \alpha_2^k]$, which is the next interval of the sequence: $\ell_{k+1} = \ell_k$ and $u_{k+1} = \alpha_2^k$. These two cases are illustrated

in Figure 11.6. If it happens that $h(\alpha_1^k) = h(\alpha_2^k)$, then the strict unimodularity of h guarantees that $\alpha^* \in [\alpha_1^k, \alpha_2^k]$, so that it becomes the next interval, and $\ell_{k+1} = \alpha_1^k$ and $u_{k+1} = \alpha_2^k$.



(a) Case $h(\alpha_1^k) > h(\alpha_2^k)$



(b) Case $h(\alpha_1^k) < h(\alpha_2^k)$

Figure 11.6: Next interval of the golden section method

We now define specific rules to choose α_1^k and α_2^k. First, we impose a symmetric reduction of the intervals, that is

$$\alpha_1^k - \ell_k = u_k - \alpha_2^k = \rho(u_k - \ell_k), \tag{11.26}$$

where $\rho < 1/2$ is the shrinking factor of the interval, which is constant across iterations. Second, we choose ρ in order to save on function evaluations. At each iteration where only one of the two values becomes the bound of the next interval, we recycle the other value for the next reduction, as illustrated in Figure 11.7.

During iteration k, the next interval happens to be $[\ell_{k+1}, u_{k+1}] = [\ell_k, \alpha_2^k]$. We select α_2^{k+1} to be equal to α_1^k, so that there is no need to recalculate the value of the

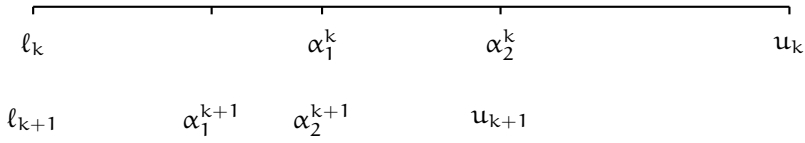


Figure 11.7: Golden section method: recycling function evaluations

function in α_2^{k+1}. Denote λ the length of the interval:

$$\lambda = u_k - \ell_k. \tag{11.27}$$

By symmetry (11.26), we have

$$\alpha_1^k - \ell_k = u_k - \alpha_2^k = \rho(u_k - \ell_k) = \rho\lambda, \tag{11.28}$$

and for the next iteration

$$\alpha_1^{k+1} - \ell_{k+1} = u_{k+1} - \alpha_2^{k+1} = \rho(u_{k+1} - \ell_{k+1}). \tag{11.29}$$

We now exploit the fact that $\ell_{k+1} = \ell_k$, $\alpha_2^{k+1} = \alpha_1^k$, and $u_{k+1} = \alpha_2^k$ (see Figure 11.7) to obtain

$$\alpha_1^{k+1} - \ell_k = \alpha_2^k - \alpha_1^k = \rho(\alpha_2^k - \ell_k). \tag{11.30}$$

We first derive

$$\begin{aligned}
\alpha_2^k - \alpha_1^k &= \alpha_2^k - \alpha_1^k + \ell_k - \ell_k + u_k - u_k \\
&= -(\alpha_1^k - \ell_k) - (u_k - \alpha_2^k) + u_k - \ell_k \\
&= -\rho\lambda - \rho\lambda + \lambda && \text{from (11.27) and (11.28)} \\
&= \lambda(1 - 2\rho). && (11.31)
\end{aligned}$$

Then, we derive

$$\begin{aligned}
\alpha_2^k - \ell_k &= \alpha_2^k - \ell_k + u_k - u_k \\
&= -(u_k - \alpha_2^k) + (u_k - \ell_k) \\
&= -\rho\lambda + \lambda && \text{from (11.27) and (11.28)} \\
&= \lambda(1 - \rho). && (11.32)
\end{aligned}$$

Using (11.31) and (11.32) into (11.30), we obtain

$$\lambda(1 - 2\rho) = \rho\lambda(1 - \rho), \tag{11.33}$$

or equivalently,

$$\rho^2 - 3\rho + 1 = 0. \tag{11.34}$$

This equation has two solutions:

$$\frac{3 + \sqrt{5}}{2} \text{ and } \frac{3 - \sqrt{5}}{2}. \tag{11.35}$$

As the shrinking factor ρ has to be less than $1/2$, we select

$$\rho = \frac{3 - \sqrt{5}}{2}. \tag{11.36}$$

Example 11.4 (Exact line search with golden section)**.** Consider the function

$$h(x) = (2 + x)\cos(2 + x). \tag{11.37}$$

The function is strictly unimodular in the interval $[5, 10]$. We apply Algorithm 11.4 with $\varepsilon = 10^{-3}$ to identify the global minimum of h in this interval.

Algorithm 11.4: Exact line search: golden section

Objective
  Find (an approximation of) the global minimum of $h(\alpha)$ on $[\ell, u]$.
Input
  An interval $[\ell, u] \subset \mathbb{R}$.
  A function $h : \mathbb{R} \to \mathbb{R}$ strictly unimodular on $[\ell, u]$.
  The desired precision $\varepsilon > 0$.
Output
  α^*, an approximation of the global minimum of $h(\alpha)$ on $[\ell, u]$.
Initialization
  $k := 1$
  $\ell_1 := \ell$
  $u_1 := u$
  $\rho := (3 - \sqrt{5})/2$
  $\alpha_1^1 := \ell_1 + \rho(u_1 - \ell_1)$
  $\alpha_2^1 := u_1 - \rho(u_1 - \ell_1)$.
Repeat
  if $h(\alpha_1^k) = h(\alpha_2^k)$ **then**
    $\ell_{k+1} := \alpha_1^k$
    $u_{k+1} := \alpha_2^k$
    $\alpha_1^{k+1} := \ell_{k+1} + \rho(u_{k+1} - \ell_{k+1})$
    $\alpha_2^{k+1} := u_{k+1} - \rho(u_{k+1} - \ell_{k+1})$.
  else if $h(\alpha_1^k) > h(\alpha_2^k)$ **then**
    $\ell_{k+1} := \alpha_1^k$
    $u_{k+1} := u_k$
    $\alpha_1^{k+1} := \alpha_2^k$
    $\alpha_2^{k+1} := u_{k+1} - \rho(u_{k+1} - \ell_{k+1})$.
  else
    $\ell_{k+1} := \ell_k$
    $u_{k+1} := \alpha_2^k$
    $\alpha_1^{k+1} := \ell_{k+1} + \rho(u_{k+1} - \ell_{k+1})$
    $\alpha_2^{k+1} := \alpha_1^k$.
  $k := k + 1$.
Until $u_k - \ell_k \leq \varepsilon$
$\alpha^* = (\ell_k + u_k)/2$

The intervals generated during the first iterations are represented in Figure 11.8. The details of each iteration is reported in Table 11.3.

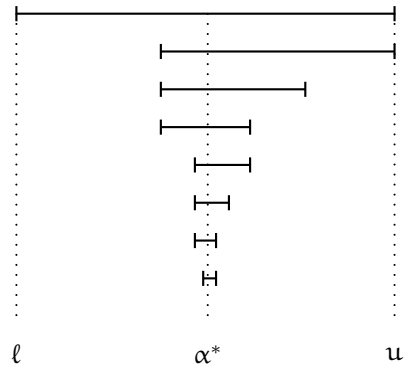


Figure 11.8: Intervals of the first iterations of the golden section method on Example 11.4

Table 11.3: Iterates of Example 11.4

k	ℓ_k	α_1^k	α_2^k	u_k	$h(\alpha_1^k)$	$h(\alpha_2^k)$
1	5.0	6.90983	8.09017	10.0	-7.75439	-7.93768
2	6.90983	8.09017	8.81966	10.0	-7.93768	-1.89353
3	6.90983	7.63932	8.09017	8.81966	-9.41833	-7.93768
4	6.90983	7.36068	7.63932	8.09017	-9.34146	-9.41833
5	7.36068	7.63932	7.81153	8.09017	-9.41833	-9.08684
6	7.36068	7.53289	7.63932	7.81153	-9.47723	-9.41833
7	7.36068	7.46711	7.53289	7.63932	-9.45863	-9.47723
8	7.46711	7.53289	7.57354	7.63932	-9.47723	-9.4678
9	7.46711	7.50776	7.53289	7.57354	-9.47504	-9.47723
10	7.50776	7.53289	7.54842	7.57354	-9.47723	-9.47553
11	7.50776	7.52329	7.53289	7.54842	-9.47712	-9.47723
12	7.52329	7.53289	7.53882	7.54842	-9.47723	-9.47686
13	7.52329	7.52922	7.53289	7.53882	-9.47729	-9.47723
14	7.52329	7.52696	7.52922	7.53289	-9.47727	-9.47729
15	7.52696	7.52922	7.53062	7.53289	-9.47729	-9.47729
16	7.52696	7.52836	7.52922	7.53062	-9.47729	-9.47729
17	7.52836	7.52922	7.52976	7.53062	-9.47729	-9.47729
18	7.52836	7.52889	7.52922	7.52976	-9.47729	-9.47729
19	7.52889	7.52922	7.52943	7.52976	-9.47729	-9.47729

The name of the method comes from the *golden ratio*. Two quantities a and b

are in the golden ratio if

$$\frac{a+b}{a} = \frac{a}{b} = \phi, \tag{11.38}$$

where

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618 \tag{11.39}$$

is the golden ratio. In geometry, a *golden rectangle* is a rectangle that can be cut into a square and a rectangle similar to the original one (see Figure 11.9). Its side lengths are in the golden ratio. The golden rectangle has been used in architecture, for its aesthetical properties.

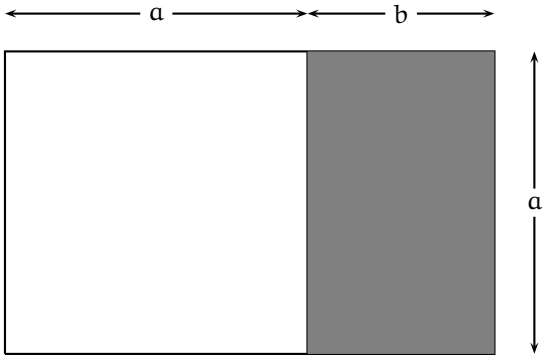


Figure 11.9: A golden rectangle

In Algorithm 11.4, the distance of the two points α_1 and α_2 to the lower bound of the interval, that is $\alpha_1 - \ell$ and $\alpha_2 - \ell$, are in the golden ratio. Indeed,

$$\begin{aligned}
\frac{\alpha_2 - \ell}{\alpha_1 - \ell} &= \frac{\alpha_2 - \alpha_1 + \alpha_1 - \ell}{\alpha_1 - \ell} \\
&= \frac{\lambda(1-2\rho) + \rho\lambda}{\rho\lambda} && \text{from (11.31)} \\
&= \frac{1-\rho}{\rho} \\
&= \frac{1+\sqrt{5}}{2} && \text{from (11.36)} \\
&= \phi && \text{from (11.39).}
\end{aligned}$$

11.3 Inexact line search

We want to spend as little effort as possible calculating the step. Instead of trying to solve a one-dimensional optimization problem as in the previous section, we consider here a trial-and-error approach, where various values are tested for the step α, and the first one that is suitable is accepted. It means that we need formal conditions that distinguish acceptable from unacceptable steps. In principle, to maintain consistency with theory, small steps should be used. Indeed, Taylor's theorem guarantees that performing a small step along a descent direction decreases the value of the function. However, we would also like our algorithm to progress rapidly, which would encourage to consider large steps. In order to reconcile these two contradictory objectives, we establish a kind of contract: large steps are acceptable provided that the reduction that is achieved is substantial. If not, they are rejected, and smaller steps should be considered. In order to formally characterize this "contract," we introduce the notion of *sufficient decrease* of the function.

Solving the problem

$$\alpha_k \in \operatorname{argmin}_{\alpha \geq 0} f(x_k + \alpha d_k)\,, \tag{11.40}$$

at each iteration using a technique like those described in Section 11.2 may be unnecessarily demanding in terms of computational efforts.

Instead of an exact line search method, we describe here an inexact line search, based on the characterization of what is *acceptable* and what is not. Once these characterization conditions are defined, "candidates" are generated for step lengths, thanks to simple and computationally cheap algorithms, until an acceptable step is produced.

We start by illustrating the fact that the condition $f(x_k + \alpha_k d_k) < f(x_k)$ is not sufficient for α_k to be considered an acceptable step.

Example 11.5 (Descent method: too large steps)**.** Consider the one-variable function

$$f(x) = x^2\,.$$

We apply Algorithm 11.1 with $x_0 = 2$ and

$$D_k = \frac{1}{2|x_k|} = \frac{\operatorname{sgn}(x_k)}{2x_k}$$
$$\alpha_k = 2 + 3(2^{-k-1})\,.$$

Note that D_k is positive (definite) for all k. Since $\nabla f(x_k) = 2x_k$, we have $d_k = -D_k \nabla f(x_k) = -\operatorname{sgn}(x_k)$. In this case, the method is written as

$$x_{k+1} = \begin{cases} x_k - 2 - 3(2^{-k-1}) & \text{if } x_k \geq 0 \\ x_k + 2 + 3(2^{-k-1}) & \text{if } x_k < 0, \end{cases} \tag{11.41}$$

which gives the sequence of iterates listed in Table 11.4 and illustrated in Figure 11.10. We show by induction that, in this case,

$$x_k = (-1)^k(1 + 2^{-k}) \tag{11.42}$$

and

$$|x_{k+1}| < |x_k|\,. \tag{11.43}$$

Table 11.4: Iterates of Example 11.5

k	x_k	d_k	α_k
0	+2.000000e+00	-1	+3.500000e+00
1	-1.500000e+00	1	+2.750000e+00
2	+1.250000e+00	-1	+2.375000e+00
3	-1.125000e+00	1	+2.187500e+00
4	+1.062500e+00	-1	+2.093750e+00
5	-1.031250e+00	1	+2.046875e+00
⋮			
46	+1.000000e+00	-1	+2.000000e+00
47	-1.000000e+00	1	+2.000000e+00
48	+1.000000e+00	-1	+2.000000e+00
49	-1.000000e+00	1	+2.000000e+00
50	+1.000000e+00	-1	+2.000000e+00

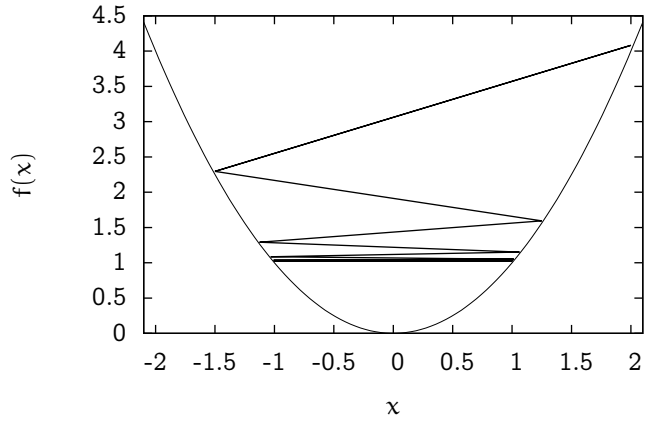


Figure 11.10: Iterates of Example 11.5

The cases $k = 0$ and $k = 1$ are verified numerically (Table 11.4). We now assume that k is even and that (11.42) and (11.43) are verified for k. We note that the parity

of k and (11.42) ensure that $x_k > 0$. Then,

$$\begin{aligned}
x_{k+1} &= x_k - 2 - 3(2^{-k-1}) && \text{from (11.41)} \\
&= (-1)^k(1+2^{-k}) - 2 - 3(2^{-k-1}) && \text{from (11.42)} \\
&= (1+2^{-k}) - 2 - 3(2^{-k-1}) && \text{because } k \text{ is even} \\
&= 1 + 2^{1-(k+1)} - 2 - 3(2^{-(k+1)}) \\
&= -1 - 2^{-(k+1)} \\
&= (-1)^{k+1}(1+2^{-(k+1)}) && \text{because } k \text{ is even}.
\end{aligned}$$

Since k is even, $x_k > 0$ and $x_{k+1} < 0$. Therefore,

$$\begin{aligned}
|x_k| - |x_{k+1}| &= x_k + x_{k+1} \\
&= 1 + 2^{-k} - 1 - 2^{-k-1} \\
&= \frac{1}{2}\,2^{-k} > 0.
\end{aligned}$$

The case where k is odd is demonstrated in a similar manner. We deduce directly from (11.43) that $x_{k+1}^2 < x_k^2$ and

$$f(x_{k+1}) < f(x_k),$$

demonstrating that it is indeed a descent method generating iterates, each of which is strictly better than the previous one, not only because the objective function strictly decreases, but also by the fact that each iterate is closer to the minimum than the previous one. However, the sequence $(x_k)_k$ does not converge and has two accumulation points in -1 and 1. Neither of these points is a local minimum of the function.

The reason that the presented algorithm fails in Example 11.5 is the disproportion between the length of the step and the resulting decrease of the objective function. Indeed, the notion of a descent direction (Definition 2.10) is based on Taylor's theorem, which is valid only in a neighborhood of the current iterate. As soon as we select a larger step than η of Theorem 2.11, the fact that the direction is a descent direction is no longer relevant, and the fact that the new iterate happens to be better is coincidental. This is the case with Example 11.5, where the next iterate actually lies in a region where the function is increasing along the followed direction.

To avoid this inconvenience, it is necessary to impose a condition characterizing the notion of *sufficient decrease* of the function. One idea is to consider a decrease of the function to be sufficient if the improvement of the objective function is proportional to the length of the step. Concretely, we select $\gamma > 0$, and consider a step α_k to be acceptable if

$$f(x_k) - f(x_k + \alpha_k d_k) \geq \alpha_k \gamma,$$

or

$$f(x_k + \alpha_k d_k) \leq f(x_k) - \alpha_k \gamma. \tag{11.44}$$

The factor γ cannot be chosen arbitrarily. In particular, it should vary from one direction to another. Returning to Example 11.2, Figure 11.11 illustrates the shape of the function $f(x_0+\alpha d)$ when going from $x_0 = \begin{pmatrix} 10 & 1 \end{pmatrix}^T$ in two different normalized directions, as well as the straight line $f(x_0) - \alpha\gamma$, with $\gamma = 6$.

According to Figure 11.11(a), a sufficient decrease of the function in the direction $d = \begin{pmatrix} -10/\sqrt{181} & -9/\sqrt{181} \end{pmatrix}^T$ is obtained for several values of α, especially between 0 and 3.25478. However, it can be seen in Figure 11.11(b) that no value of α allows a sufficient decrease in the direction $d = \begin{pmatrix} -2/\sqrt{5} & 1/\sqrt{5} \end{pmatrix}^T$ with regard to the condition (11.44). Indeed, the straight line is too steep, while the function is relatively flat in this direction. The requirement associated with this value of γ is too strong.

Instead of using an arbitrary fixed value for γ, it is more appropriate to define it proportional to the slope of the function in x_k in the direction d_k:

$$\gamma = -\beta \nabla f(x_k)^T d_k \,,$$

with $0 < \beta < 1$. Then, the closer the directional derivative $\nabla f(x_k)^T d_k$ is to zero, the smaller the slope of the line, and vice versa. Note that, if $\beta = 0$, (11.44) would collapse to $f(x_k + \alpha_k d_k) \leq f(x_k)$, that we have shown to be inappropriate. Geometrically, the line setting the threshold for the objective function value is horizontal in this case. So the value $\beta = 0$ is excluded. The value $\beta = 1$ corresponds to the tangent line. If the function happens to be convex at x_k (which is the case close to a local minimum), the tangent lies entirely below the function (see Theorem 2.16), and no value of α_k verifies (11.44). Again, this value of β is excluded, justifying the condition $0 < \beta < 1$.

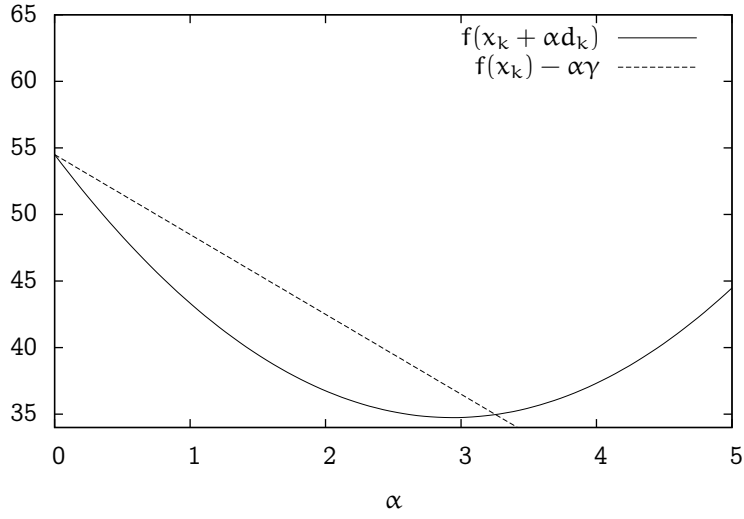
Definition 11.6 (Sufficient decrease: the first Wolfe condition)**.** Consider the differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, a point $x_k \in \mathbb{R}^n$, a (descent) direction $d_k \in \mathbb{R}^n$ such that $\nabla f(x_k)^T d_k < 0$ and a step $\alpha_k \in \mathbb{R}$, $\alpha_k > 0$. We say that the function f decreases sufficiently in $x_k + \alpha_k d_k$ compared with x_k if

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \alpha_k \beta_1 \nabla f(x_k)^T d_k \,, \tag{11.45}$$

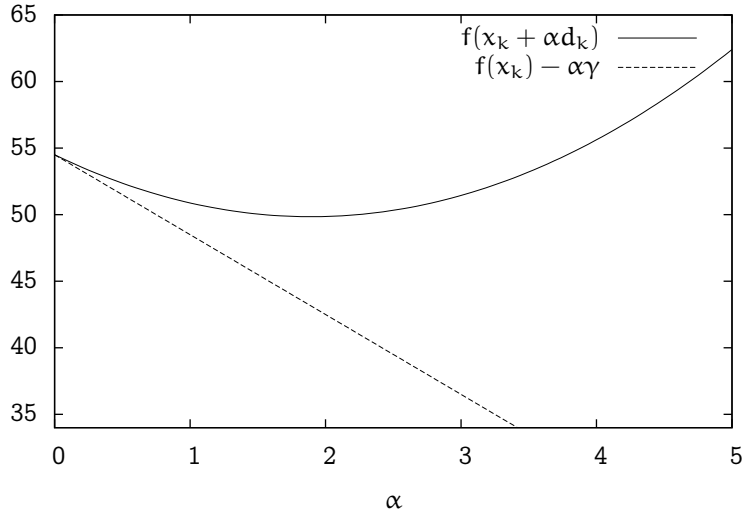
with $0 < \beta_1 < 1$. The condition (11.45) is called the first Wolfe condition after Wolfe (1969), or the Armijo condition, after Armijo (1966).

It is important to note that (2.14) in the theorem on descent directions (Theorem 2.11) guarantees that there always exist steps satisfying the condition (11.45). The condition (11.45) is illustrated in Figure 11.12 with $\beta_1 = 0.5$ and in Figure 11.13 with $\beta_1 = 0.1$. In each case, there exist steps ensuring a sufficient decrease.

The condition (11.45) enables us to reject steps that, due to being too large, do not provide a sufficient decrease of the function. Having solved the problem of too large steps, we now consider steps that are too small. These may cause a problem, as shown in Example 11.7.

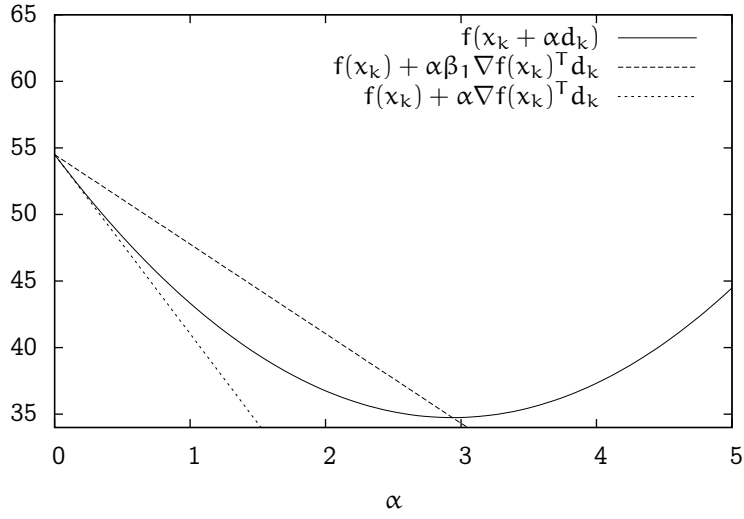


(a) $d_k = (-10/\sqrt{181} \quad -9/\sqrt{181})^T$

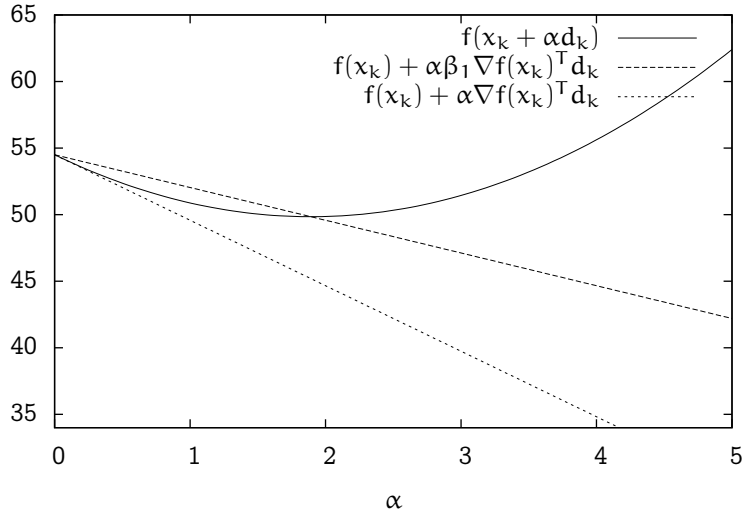


(b) $d_k = \left(-2/\sqrt{5} \quad 1/\sqrt{5}\right)^T$

Figure 11.11: Decrease of the function of Example 11.2

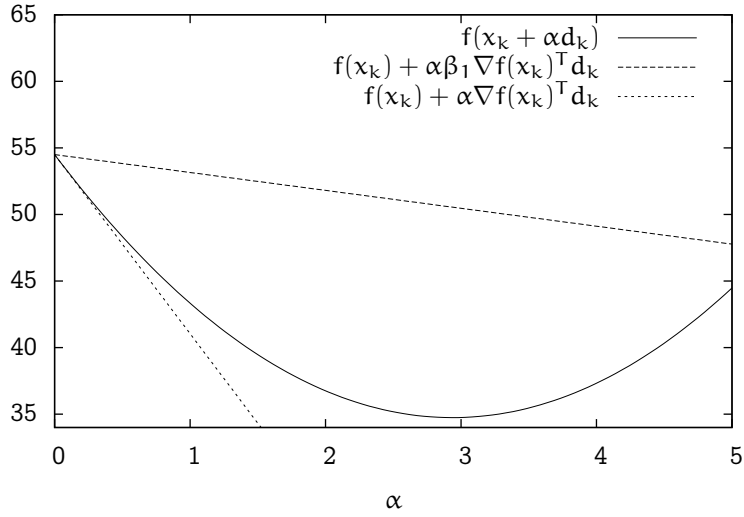


(a) $d_k = (-10/\sqrt{181} \quad -9/\sqrt{181})^T$

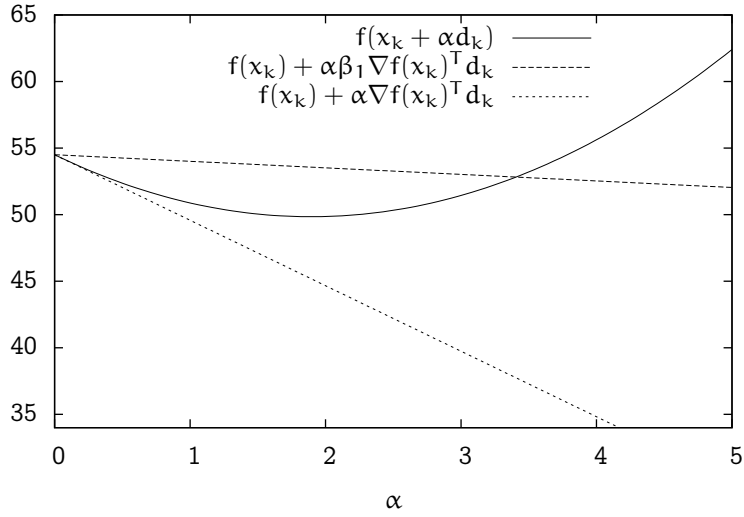


(b) $d_k = (-2/\sqrt{5} \quad 1/\sqrt{5})^T$

Figure 11.12: Condition (11.45) with $\beta_1 = 0.5$



(a) $d_k = (-10/\sqrt{181} \quad -9/\sqrt{181})^T$



(b) $d_k = (-2/\sqrt{5} \quad 1/\sqrt{5})^T$

Figure 11.13: Condition (11.45) with $\beta_1 = 0.1$

Example 11.7 (Descent method: too small steps)**.** Consider the one-variable function

$$f(x) = x^2 .$$

We apply Algorithm 11.1 with $x_0 = 2$ and

$$D_k = \frac{1}{2x_k}$$
$$\alpha_k = 2^{-k-1} .$$

Note that D_k is positive (definite) for all k if $x_k > 0$, which is the case in this example. Since $\nabla f(x_k) = 2x_k$, we have $d_k = -D_k \nabla f(x_k) = -1$. In this case, the method is written as

$$x_{k+1} = x_k - 2^{-k-1} .$$

The sequence of iterates $(x_k)_k$ is listed in Table 11.5. We show by induction that it is defined by

$$x_k = 1 + 2^{-k} . \tag{11.46}$$

Table 11.5: Iterates of Example 11.7

k	x_k	d_k	α_k
0	+2.000000e+00	-1	+5.000000e-01
1	+1.500000e+00	-1	+2.500000e-01
2	+1.250000e+00	-1	+1.250000e-01
3	+1.125000e+00	-1	+6.250000e-02
4	+1.062500e+00	-1	+3.125000e-02
5	+1.031250e+00	-1	+1.562500e-02
⋮			
46	+1.000000e+00	-1	+7.105427e-15
47	+1.000000e+00	-1	+3.552714e-15
48	+1.000000e+00	-1	+1.776357e-15
49	+1.000000e+00	-1	+8.881784e-16
50	+1.000000e+00	-1	+4.440892e-16

The cases $k = 0$ and $k = 1$ are numerically verified (Table 11.5). If we assume that (11.46) is verified for k, then

$$\begin{aligned} x_{k+1} &= x_k - 2^{-k-1} = 1 + 2^{-k} - 2^{-k-1} \\ &= 1 + 2^{-k-1}(2-1) = 1 + 2^{-(k+1)} \end{aligned}$$

and the recurrence is verified. From Equation (11.46), we immediately deduce $x_{k+1} < x_k$ and, consequently, $f(x_{k+1}) < f(x_k)$. We thus have a descent method. However, the sequence $(x_k)_k$ converges toward 1, which is not a local minimum of the function.

In this case, the reason the method fails is due to the degeneracy of the steps α_k. Although these steps are positive, they are closer and closer to 0 and at some point, the method can no longer progress. A technique aiming to prevent this again exploits the derivative of the function in the direction d_k. At the point x_k, the directional derivative $\nabla f(x_k)^T d_k$ is negative, because d_k is a descent direction. If we were performing an exact line search (see Section 11.2) where the step α^* corresponds to a local minimum of the function in the direction k, we would have $\nabla f(x_k + \alpha^* d_k)^T d_k = 0$. Then, between x_k and $x_k + \alpha^* d_k$, the derivative of the function increases compared to its initial negative value. To ensure sufficiently large steps, the idea is to obtain a step such that the directional derivative increases sufficiently.

Definition 11.8 (Sufficient progress: the second Wolfe condition)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, and let us take a point $x_k \in \mathbb{R}^n$, a (descent) direction $d_k \in \mathbb{R}^n$ such that $\nabla f(x_k)^T d_k < 0$ and a step $\alpha_k \in \mathbb{R}$, $\alpha_k > 0$. We say that the point $x_k + \alpha_k d_k$ enables *sufficient progress* compared with x_k if

$$\nabla f(x_k + \alpha_k d_k)^T d_k \geq \beta_2 \nabla f(x_k)^T d_k \tag{11.47}$$

with $0 < \beta_2 < 1$. The condition (11.47) is called the second Wolfe condition, after Wolfe (1969).

The condition (11.47) can also be written as

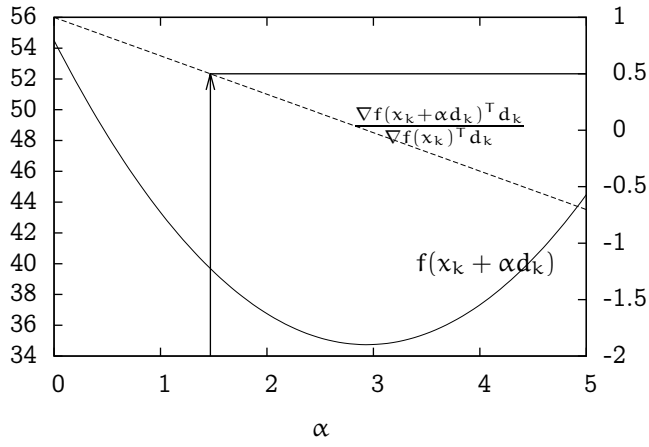
$$\frac{\nabla f(x_k + \alpha_k d_k)^T d_k}{\nabla f(x_k)^T d_k} \leq \beta_2 \,. \tag{11.48}$$

This is illustrated in Figure 11.14, where the straight dotted lines represent the ratio of the left term of (11.48). By choosing, for instance, $\beta_2 = 0.5$, the step α should be such that $\alpha \geq 1.4687$ in Figure 11.14(a) and $\alpha \geq 0.94603$ in Figure 11.14(b).

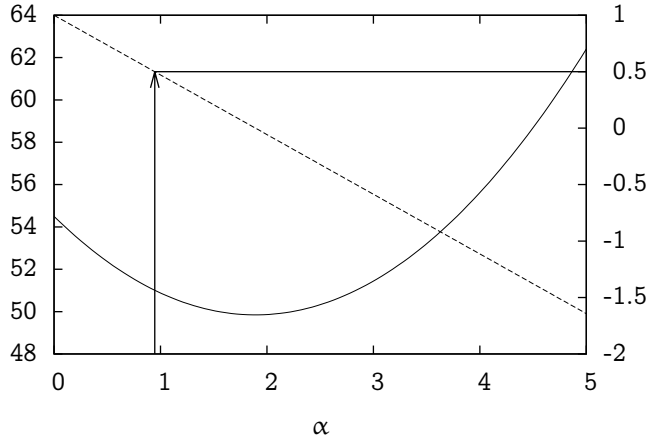
The conditions (11.45) and (11.47) are called the *Wolfe conditions*, after Wolfe (1969) and Wolfe (1971). They are sometimes called *Armijo-Goldstein conditions*, making reference to the work of Armijo (1966), and Goldstein and Price (1967).

As the two conditions have conflicting goals (one forbidding long steps, one forbidding short steps), it may happen that they are incompatible, and that no step verifies both. The next theorem shows that if the parameter β_1 of the first condition, and the parameter β_2 of the second are chosen such that $0 < \beta_1 < \beta_2 < 1$, the two conditions are compatible.

Theorem 11.9 (Validity of the Wolfe conditions)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, and let us take a point $x_k \in \mathbb{R}^n$ and a (descent) direction $d_k \in \mathbb{R}^n$ such that $\nabla f(x_k)^T d_k < 0$. We assume that f is bounded from below in the direction d_k, i.e., that there exists f_0 such that $f(x_k + \alpha d_k) \geq f_0$ for all $\alpha \geq 0$. If $0 < \beta_1 < 1$, there exists η such that the first Wolfe condition (11.45) is satisfied for all $\alpha_k \leq \eta$. Moreover, if $0 < \beta_1 < \beta_2 < 1$, there exists $\alpha_2 > 0$ such that the two Wolfe conditions (11.45) and (11.47) are both satisfied.*



(a) $d_k = (-10/\sqrt{181} \quad -9/\sqrt{181})^T$



(b) $d_k = \left(-2/\sqrt{5} \quad 1/\sqrt{5}\right)^T$

Figure 11.14: Sufficient progress of the function of Example 11.2 ($\beta_2 = 0.5$)

Proof. Since d_k is a descent direction, we invoke the theorem of descent directions (Theorem 2.11). Note that (2.14) is equivalent to (11.45), which proves the first part of the theorem. Note also that there exist steps such that the condition (11.45) is not satisfied. Indeed, as the condition is defined by a decreasing line, that is, an unbounded function, the fact that f is bounded from below guarantees that, for some large steps, the line lies below the function. In particular, when

$$\alpha > \frac{f_0 - f(x_k)}{\beta_1 \nabla f(x_k)^T d_k}, \tag{11.49}$$

we have (the direction of the inequality changes because $\nabla f(x_k)^T d_k < 0$)

$$f(x_k) + \alpha\beta_1 \nabla f(x_k)^T d_k < f_0 \leq f(x_k + \alpha d_k)\,.$$

By continuity of f, there exists α_1 such that

$$f(x_k + \alpha_1 d_k) = f(x_k) + \alpha_1 \beta_1 \nabla f(x_k)^T d_k\,, \tag{11.50}$$

i.e., such that the straight line $f(x_k) + \alpha\beta_1 \nabla f(x_k)^T d_k$ intersects the function.

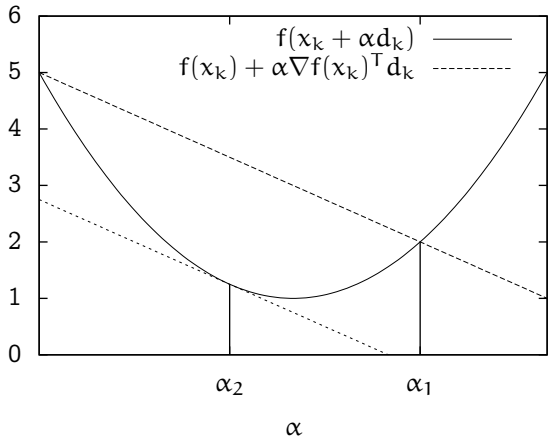


Figure 11.15: Validity of the Wolfe conditions: illustration of the proof

We invoke the mean value theorem (Theorem C.1) that says that there is a step α_2 between 0 and α_1 such that the function has the same slope in $x_k + \alpha_2 d_k$ as the line (see Figure 11.15). Formally, we use Equation (C.2) with $d = \alpha_1 d_k$. There exists $0 \leq \alpha' \leq 1$ such that

$$f(x_k + \alpha_1 d_k) = f(x_k) + \alpha_1 d_k^T \nabla f(x_k + \alpha' \alpha_1 d_k)\,. \tag{11.51}$$

If we take $\alpha_2 = \alpha' \alpha_1$, we combine (11.50) and (11.51) to obtain

$$f(x_k) + \alpha_1 \beta_1 \nabla f(x_k)^T d_k = f(x_k) + \alpha_1 d_k^T \nabla f(x_k + \alpha_2 d_k)$$

or

$$\beta_1 = \frac{d_k^T \nabla f(x_k + \alpha_2 d_k)}{\nabla f(x_k)^T d_k}\,.$$

Then, since $\beta_2 > \beta_1$, we have

$$\beta_2 > \frac{d_k^T \nabla f(x_k + \alpha_2 d_k)}{\nabla f(x_k)^T d_k}$$

and (11.47) is satisfied for α_2. □

An inexact line search method enables us to identify a step that satisfies the Wolfe conditions (11.45) and (11.47). Algorithm 11.5 is attributed to Fletcher (1980) and Lemaréchal (1981). The idea is simple: a trial step is tested. If it is too long, that is, if it violates the first Wolfe condition, it is shortened. If it is too short, that is, if it violates the second Wolfe condition, it is made longer. This process is repeated until a step verifying both conditions is found. Theorem 11.9 guarantees that such a step exists when $0 < \beta_1 < \beta_2 < 1$.

Algorithm 11.5: Line search

Objective
- To find a step α^* such that the Wolfe conditions (11.45) and (11.47) are satisfied.

Input
- The continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
- The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
- A vector $x \in \mathbb{R}^n$.
- A descent direction d such that $\nabla f(x)^T d < 0$.
- An initial solution $\alpha_0 > 0$ (e.g. $\alpha_0 = 1$).
- Parameters β_1 and β_2 such that $0 < \beta_1 < \beta_2 < 1$ (e.g., $\beta_1 = 10^{-4}$ and $\beta_2 = 0.99$).
- A parameter $\lambda > 1$ (e.g., $\lambda = 2$).

Output
- A step α^* such that the conditions (11.45) and (11.47) are satisfied.

Initialization
- $i := 0$.
- $\alpha_\ell := 0$.
- $\alpha_r := +\infty$.

Repeat
- **if** α_i *violates (11.45)* **then** the step is too long
  - $\alpha_r := \alpha_i$
  - $\alpha_{i+1} := \dfrac{\alpha_\ell + \alpha_r}{2}$.
- **if** α_i *does not violate (11.45) but violates (11.47)* **then** the step is too short
  - $\alpha_\ell := \alpha_i$
  - **if** $\alpha_r < +\infty$ **then**
    - $\alpha_{i+1} := \dfrac{\alpha_\ell + \alpha_r}{2}$
  - **else**
    - $\alpha_{i+1} := \lambda \alpha_i$
- $i := i + 1$.

Until α_i *satisfies the conditions (11.45) and (11.47)*

$\alpha^* := \alpha_i$

Table 11.6 lists the steps of the algorithm applied to the function of Example 11.2, with

$$x = \begin{pmatrix} 10 \\ 1 \end{pmatrix}, \qquad d = \begin{pmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{pmatrix}$$

$$\alpha_0 = 10^{-3}, \qquad \beta_1 = 0.3, \qquad \beta_2 = 0.7, \qquad \lambda = 20.$$

Note that, for reasons of implementation, the quantity $+\infty$ is represented by `9.99999000e+05`. The value of the parameters used in this example have been chosen to illustrate all the cases and are not appropriate in practice. The value of β_1 should be close to 0 (for instance, $\beta_1 = 10^{-4}$) and the value of β_2 should be close to 1 (for instance, $\beta_2 = 0.99$). A smaller value for λ (such as $\lambda = 2$) is also more appropriate in practice.

Table 11.6: Illustration of the line search for Example 11.2

α_i	α_ℓ	α_r	Violated cond.
`1.00000000e-03`	`0.00000000e+00`	`9.99999000e+05`	(11.47)
`2.00000000e-02`	`1.00000000e-03`	`9.99999000e+05`	(11.47)
`4.00000000e-01`	`2.00000000e-02`	`9.99999000e+05`	(11.47)
`8.00000000e+00`	`4.00000000e-01`	`9.99999000e+05`	(11.45)
`4.20000000e+00`	`4.00000000e-01`	`8.00000000e+00`	(11.45)
`2.30000000e+00`	`4.00000000e-01`	`4.20000000e+00`	—

Theorem 11.10 (Finiteness of the line search algorithm). *Following the same assumptions as those of Theorem 11.9, the line search (Algorithm 11.5) ends after a finite number of iterations.*

Proof. We first assume, by contradiction, that $\lim_{i\to\infty} \alpha_i = +\infty$. This signifies that α_r permanently keeps its initial value $\alpha_r = \infty$, and that condition on line 21 of the algorithm is always verified. Therefore, α_i never violates (11.45). This is impossible, as was discussed in the proof of Theorem 11.9. Indeed, the condition (11.45) is violated as soon as α_i is sufficiently large, i.e., as soon as (11.49) is satisfied. Then, after a finite number of iterations, $\alpha_r < \infty$, and the following iterations all consist in a reduction of the step

$$\alpha_{i+1} = \frac{\alpha_\ell + \alpha_r}{2},$$

either at line 20 or 24 of the algorithm.

We now assume (by contradiction) that the algorithm performs an infinite number of iterations. In this case,

$$\lim_{i\to\infty} \alpha_r^i - \alpha_\ell^i = 0,$$

where α_r^i and α_ℓ^i are values of α_r and α_ℓ, respectively, at iteration i. Indeed, regardless of the case that applies, the interval is divided by two at each iteration, and we always have

$$\alpha_r^{i+1} - \alpha_\ell^{i+1} = \frac{\alpha_r^i - \alpha_\ell^i}{2}.$$

Then, there exists α^* such that

$$\alpha^* = \lim_{i\to\infty} \alpha_r^i = \lim_{i\to\infty} \alpha_\ell^i = \lim_{i\to\infty} \alpha_i \,.$$

As α_ℓ is updated only when the condition on line 21 of the algorithm is verified, it means that the condition (11.45) is satisfied for all α_ℓ^i, that is,

$$f(x_k + \alpha_\ell^i d_k) \leq f(x_k) + \alpha_\ell^i \beta_1 \nabla f(x_k)^T d_k \quad \text{for each } i.$$

Taking the limit $i \to \infty$, we obtain

$$f(x_k + \alpha^* d_k) \leq f(x_k) + \alpha^* \beta_1 \nabla f(x_k)^T d_k \,. \tag{11.52}$$

Similarly, as α_r is updated only when the condition on line 18 of the algorithm is verified, the condition (11.45) is **not** satisfied for any α_r^i, that is,

$$f(x_k + \alpha_r^i d_k) > f(x_k) + \alpha_r^i \beta_1 \nabla f(x_k)^T d_k \quad \text{for each } i. \tag{11.53}$$

At the limit,

$$f(x_k + \alpha^* d_k) \geq f(x_k) + \alpha^* \beta_1 \nabla f(x_k)^T d_k \,. \tag{11.54}$$

Note that the equality is not satisfied for any α_r^i, but can be reached at the limit. Actually, by combining (11.52) and (11.54), we observe that it is reached at the limit, as we have

$$f(x_k + \alpha^* d_k) = f(x_k) + \alpha^* \beta_1 \nabla f(x_k)^T d_k \,. \tag{11.55}$$

Therefore, the limit value α^* does not violate the Wolfe condition (11.45), while every α_r^i does. Consequently, α^* has to be different than any α_r^i, that is, $\alpha_r^i > \alpha^*$ or $\alpha_r^i - \alpha^* > 0$. In (11.53), we replace $f(x_k)$ by its value derived from (11.55) to obtain

$$f(x_k + \alpha_r^i d_k) > f(x_k + \alpha^* d_k) + (\alpha_r^i - \alpha^*) \beta_1 \nabla f(x_k)^T d_k \text{ for each } i.$$

By dividing by $\alpha_r^i - \alpha^*$, which is positive, we obtain for each i

$$\frac{f(x_k + \alpha_r^i d_k) - f(x_k + \alpha^* d_k)}{\alpha_r^i - \alpha^*} > \beta_1 \nabla f(x_k)^T d_k \,.$$

We take the limit $i \to \infty$ to obtain to the left the directional derivative of f in $x_k + \alpha^* d_k$ in the direction d_k, and

$$\nabla f(x_k + \alpha^* d_k)^T d_k \geq \beta_1 \nabla f(x_k)^T d_k \,.$$

Since $\beta_2 > \beta_1$ and $\nabla f(x_k)^T d_k < 0$ (by assumption), we get

$$\nabla f(x_k + \alpha^* d_k)^T d_k > \beta_2 \nabla f(x_k)^T d_k \,. \tag{11.56}$$

As α_ℓ is updated only when the condition on line 21 of the algorithm is verified, it means that the condition (11.47) is violated for all α_ℓ^i, that is,

$$\nabla f(x_k + \alpha_\ell^i d_k)^T d_k < \beta_2 \nabla f(x_k)^T d_k,$$

and at the limit $i \to \infty$,

$$\nabla f(x_k + \alpha^* d_k)^T d_k \leq \beta_2 \nabla f(x_k)^T d_k \,. \tag{11.57}$$

Since the two conclusions (11.56) and (11.57) are contradictory, the assumption that the algorithm performs an infinite number of iterations is incorrect, which proves the result. □

11.4 Steepest descent method

The steepest descent method is certainly among the least efficient algorithms for unconstrained optimization, even though it is regularly reinvented. Here, it is solely presented for the sake of comparison with the others. In practice, it should not be used. It is a full working version combining the preconditioned steepest descent method (Algorithm 11.1) and line search (Algorithm 11.5). Here, the matrix D_k of Algorithm 11.1 is the identity matrix.

Algorithm 11.6: Steepest descent method

Objective

To find (an approximation of) a local minimum of the problem

$$\min_{x \in \mathbb{R}^n} f(x)\,. \tag{11.58}$$

Input

The differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
An initial solution $x_0 \in \mathbb{R}^n$.
The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}$.

Initialization

$k := 0$.

Repeat

$d_k := -\nabla f(x_k)$.
Determine α_k by applying the line search (Algorithm 11.5) with $\alpha_0 = 1$.
$x_{k+1} := x_k + \alpha_k d_k$.
$k := k + 1$.

Until $\|\nabla f(x_k)\| \leq \varepsilon$

$x^* = x_k$.

11.5 Newton method with line search

We now provide a complete working version of the preconditioned steepest descent method (Algorithm 11.1), by combining the local Newton method (Algorithm 10.1) and line search (Algorithm 11.5). An iteration of the local Newton method is

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k) \tag{11.59}$$

and an iteration of the preconditioned steepest descent method is

$$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k)\,, \tag{11.60}$$

where D_k is positive definite. Therefore, if $\nabla^2 f(x_k)$ is positive definite and the step $\alpha_k = 1$ is acceptable (according to the Wolfe conditions), an iteration of the local Newton method represents exactly an iteration of the preconditioned steepest descent method with $D_k = \nabla^2 f(x_k)^{-1}$.

If the step $\alpha_k = 1$ is not acceptable, it suffices to apply Algorithm 11.5 to obtain a step satisfying the Wolfe conditions. However, when the Hessian matrix $\nabla^2 f(x_k)$ is not positive definite, it is necessary to choose another preconditioner D_k. Several possibilities exist.

One of them involves choosing D_k diagonal, with entries

$$D_k(i,i) = \max\left(\varepsilon, \frac{\partial^2 f}{\partial (x)_i^2}(x_k)\right)^{-1}, \tag{11.61}$$

with $\varepsilon > 0$. Then, each diagonal element (i.e., each eigenvalue) is greater than or equal to ε, which guarantees the positive definiteness of the matrix, all the while incorporating into it information related to the second derivative.

Algorithm 11.7: Modified Cholesky factorization

Objective
  To modify a matrix in order to make it positive definite.
Input
  A symmetric matrix $A \in \mathbb{R}^{n\times n}$.
Output
  A lower triangular matrix L and $\tau \geq 0$ such that $A + \tau I = LL^T$ is positive definite.
Initialization
  $k := 0$.
  if $\min_i a_{ii} > 0$ **then**
    $\tau_k := 0$
  else
    $\tau_k := \frac{1}{2}\|A\|_F$.
Repeat
  Calculate the Cholesky factorization LL^T of $A + \tau I$.
  if *the factorization is not successful* **then**
    $\tau_{k+1} := \max(2\tau_k, \frac{1}{2}\|A\|_F)$.
  $k := k + 1$.
Until *the factorization is successful*

In general, the most widely used technique consists in generating a matrix E such that

$$D_k = \left(\nabla^2 f(x_k) + E\right)^{-1}$$

is positive definite. In particular, this is always possible if E is a multiple of the identity.[1]

Algorithm 11.7 proposes a simple method to obtain E as well as a Cholesky factorization of $\nabla^2 f(x_k) + E$. Note that this algorithm is simplistic and computationally demanding. Several Cholesky factorizations may be required before a positive definite matrix is found. More sophisticated and effective methods have been proposed, among others by Gill and Murray (1974), Gill et al. (1981), and Schnabel and Eskow (1999). Putting everything together, Algorithm 11.8 describes the Newton algorithm with line search.

Algorithm 11.8: Newton algorithm with line search

Objective

To find (an approximation of) a local minimum of the problem

$$\min_{x \in \mathbb{R}^n} f(x)\,. \tag{11.62}$$

Input

The twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
The Hessian of the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$.
An initial solution $x_0 \in \mathbb{R}^n$.
The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}$.

Initialization

$k := 0$.

Repeat

Calculate a lower triangular matrix L_k and τ such that

$$L_k L_k^T = \nabla^2 f(x_k) + \tau I\,,$$

by using for instance the modified Cholesky factorization (Algorithm 11.7).
Find z_k by solving the triangular system $L_k z_k = \nabla f(x_k)$.
Find d_k by solving the triangular system $L_k^T d_k = -z_k$.
Determine α_k by applying line search (Algorithm 11.5) with $\alpha_0 = 1$.
$x_{k+1} := x_k + \alpha_k d_k$.
$k := k + 1$.

Until $\left\|\nabla f(x_k)\right\| \leq \varepsilon$

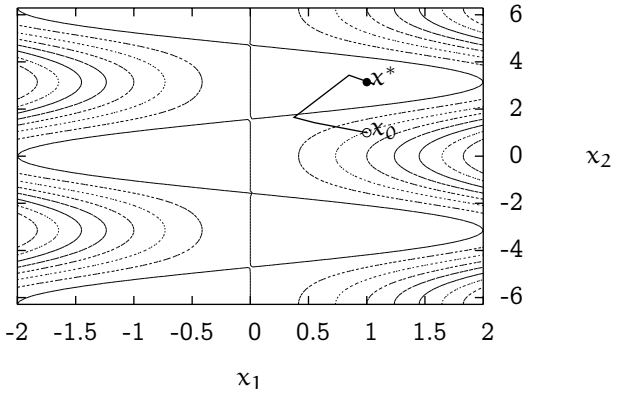
$x^* = x_k$.

[1] Apply Theorem C.18 with $A = \nabla^2 f(x_k)$, $B = I$.

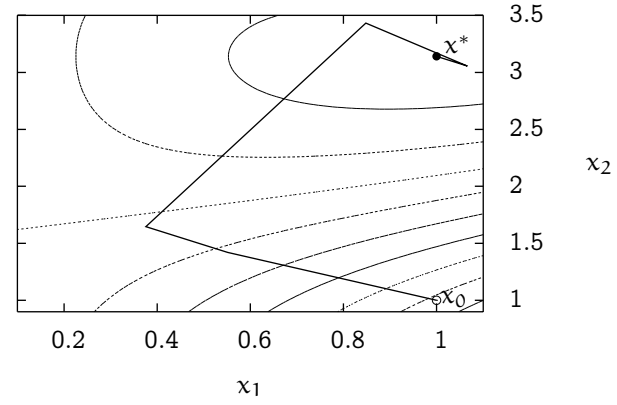
This algorithm is operational in the sense that all steps are well defined. To compare with the local Newton method from Chapter 10, we apply the Newton method with line search (Algorithm 11.8) to Example 5.8 starting from the same point $x_0 = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$. In this case, it converges to

$$x^* = \begin{pmatrix} 1 \\ \pi \end{pmatrix}, \qquad \nabla f(x^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \nabla^2 f(x^*) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

which is a local minimum since it satisfies the sufficient optimality conditions (see Theorem 5.7 and the discussions for Example 5.8). The iterations are illustrated in Figure 11.16, and it is interesting to compare with Figure 10.1.



(a) Iterates



(b) Zoom

Figure 11.16: Iterates of the Newton method with line search for Example 5.8

Table 11.7 lists the values of α_k and of τ employed in each iteration. Note that, starting from iteration 4, the algorithm performs the exact same steps as the local Newton method, as $\alpha_k = 1$ and $\tau = 0$, and it achieves quadratic convergence.

Table 11.7: Illustration of the Newton method with line search (Algorithm 11.8) for Example 5.8

k	$f(x_k)$	$\|\nabla f(x_k)\|$	α_k	τ
0	1.04030231e+00	1.75516512e+00		
1	2.34942031e-01	8.88574897e-01	1	1.64562250e+00
2	4.21849003e-02	4.80063696e-01	1	1.72091923e+00
3	-4.52738278e-01	2.67168927e-01	3	8.64490594e-01
4	-4.93913638e-01	1.14762780e-01	1	0.00000000e+00
5	-4.99982955e-01	5.85174623e-03	1	0.00000000e+00
6	-5.00000000e-01	1.94633135e-05	1	0.00000000e+00
7	-5.00000000e-01	2.18521663e-10	1	0.00000000e+00
8	-5.00000000e-01	1.22460635e-16	1	0.00000000e+00

11.6 The Rosenbrock problem

To illustrate the algorithms, we consider a problem proposed by Rosenbrock (1960) in order to illustrate the superiority of his algorithm (an improvement of the steepest descent method based on an orthogonalization procedure), compared with the steepest descent method. It is defined by

$$\min_{x \in \mathbb{R}^2} f(x_1, x_2) = 100 \left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 . \tag{11.63}$$

It is actually the function used in Example 5.3 to illustrate the necessary optimality conditions. It has a valley that follows the parabola $x_2 = x_1^2$, which forces all descent methods to follow a curved trajectory. Figure 11.17 plots the function for x_1 between -2 and 2, and x_2 between -4 and 4.

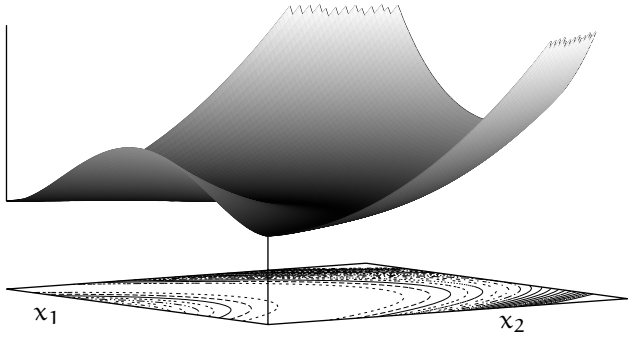


Figure 11.17: The Rosenbrock function

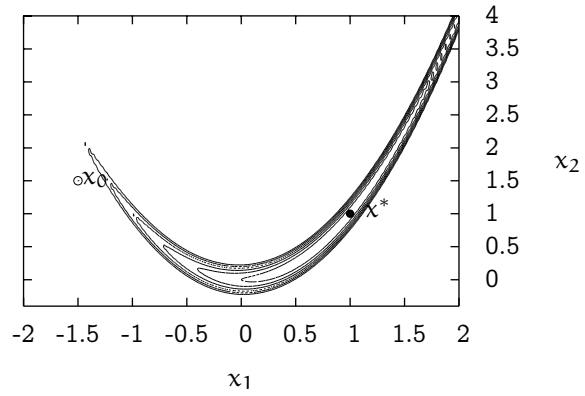
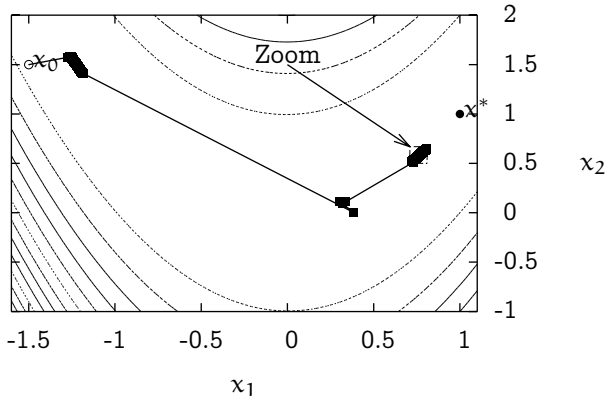
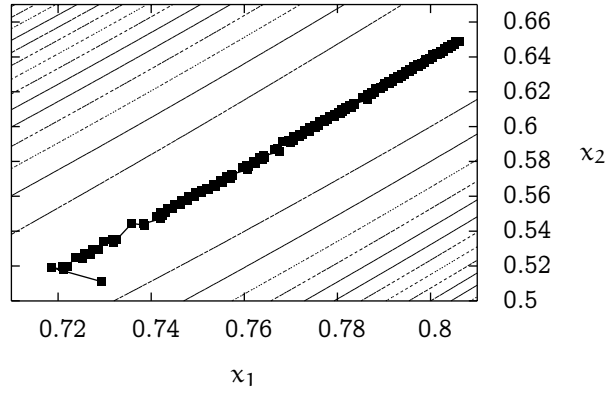


Figure 11.18: Level curves of the Rosenbrock function



(a) Stopped at 200 iterations



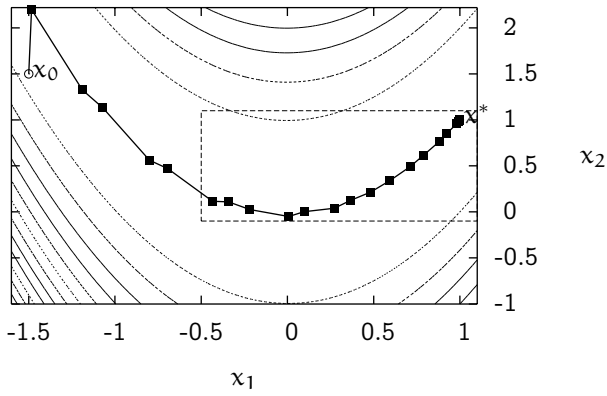
(b) Zoom

Figure 11.19: Steepest descent method

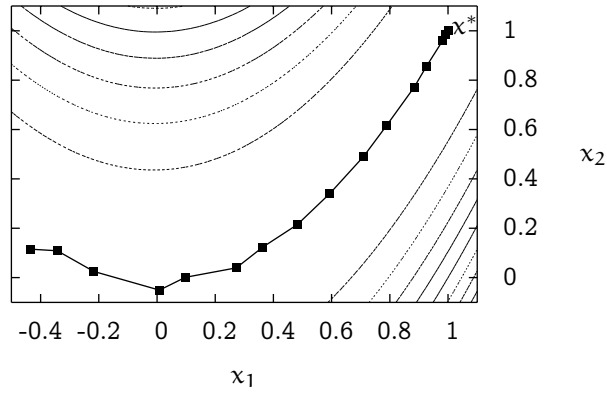
Figure 11.18 represents the level curves 0 to 6 of the function, as well as the location of the starting point $x_0 = \begin{pmatrix} -1.5 & 1.5 \end{pmatrix}^T$ and the location of the optimal solution $x^* = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$.

The steepest descent algorithm (Algorithm 11.6) has a hard time solving this problem. The zigzag trajectory of the iterates, already illustrated in Example 11.1, is unacceptable here (Figure 11.19). With two exceptions, the steps made by the method are small, which hinders the algorithm from progressing. This one was interrupted after 200 iterations, without having converged. A large step could be taken after two iterations thanks to the line search strategy (Algorithm 11.5), which starts by attempting large steps and sometimes succeeds.

The superiority of the Newton method with line search (Algorithm 11.8) is illustrated in Figure 11.20. The algorithm exploits pretty well the information about the curvature of the function provided by the second derivatives. The iterations follow the valley pretty smoothly, along the parabola that defines it.



(a) 23 iterations



(b) Zoom

Figure 11.20: Newton method with line search

11.7 Convergence

The local Newton method (Algorithm 10.1) has a quadratic convergence rate (Theorem 7.13). However, it only works when the starting point is sufficiently close to the optimal solution. In practice, it is clearly not possible to guarantee that this hypothesis is verified. Moreover, the more non linear and ill-conditioned the function is, the closer the starting point needs to be to the optimal solution (Eq. (7.21)). The main motivation for developing the Newton method with line search (Algorithm 11.8) is to obtain an algorithm that converges regardless of the starting point given by the user. We call such an algorithm *globally convergent*.

Definition 11.11 (Global convergence)**.** Consider an iterative algorithm that generates a sequence $(x_k)_k$ in $\mathbb{R}^n$, in order to solve the unconstrained minimization problem

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

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function. The algorithm is said to be *globally convergent* if

$$\lim_{k \to \infty} \|\nabla f(x_k)\| = 0\,, \tag{11.64}$$

regardless of $x_0 \in \mathbb{R}^n$.

Care should be taken not to confuse "global convergence" and "global minimum." An algorithm can be globally convergent and converge toward a local minimum.

As line search guarantees sufficient decrease and sufficient progress along a descent direction, the only way to stall a descent direction algorithm is for the directions to become asymptotically orthogonal to the gradient. Then, even if the algorithm guarantees that

$$\nabla f(x_k)^T d_k < 0\,,$$

it is necessary to also guarantee that

$$\liminf_k \nabla f(x_k)^T d_k < 0\,.$$

In other words, the cosine of the angle between the descent direction and the direction of the steepest slope cannot approach 0. We denote this angle θ_k, i.e.,

$$\cos\theta_k = \frac{-\nabla f(x_k)^T d_k}{\|\nabla f(x_k)\| \|d_k\|}\,. \tag{11.65}$$

In order to demonstrate this, we need the following theorem, attributed to Zoutendijk.

Theorem 11.12 (Zoutendijk's theorem)**.** *Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, that is bounded from below, differentiable, and its gradient is Lipschitz continuous (Definition B.16), i.e., there exists $M > 0$ such that*

$$\|\nabla f(x) - \nabla f(y)\| \leq M\|x - y\|\,, \qquad \forall x, y \in \mathbb{R}^n\,. \tag{11.66}$$

Consider an algorithm generating the sequence $(x_k)_k$*, defined by the iterations*

$$x_{k+1} = x_k + \alpha_k d_k\,, \qquad k = 0, 1, \dots\,, \tag{11.67}$$

with d_k *a descent direction (i.e.,* $\nabla f(x_k)^T d_k < 0$*) and* α_k *that satisfies the Wolfe conditions (11.45) and (11.47). In this case, the series*

$$\sum_{k=0}^{+\infty} \cos^2\theta_k \left\|\nabla f(x_k)\right\|^2, \tag{11.68}$$

where

$$\cos\theta_k = \frac{-\nabla f(x_k)^T d_k}{\left\|\nabla f(x_k)\right\| \|d_k\|} \tag{11.69}$$

is convergent.

Proof. Take an arbitrary k. We have

$$\begin{aligned}
\nabla f(x_k + \alpha_k d_k)^T d_k &\geq \beta_2 \nabla f(x_k)^T d_k && \text{from (11.47)} \\
\nabla f(x_{k+1})^T d_k &\geq \beta_2 \nabla f(x_k)^T d_k && \text{from (11.67)} \\
\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T d_k &\geq (\beta_2 - 1)\nabla f(x_k)^T d_k\,.
\end{aligned}$$

Moreover, we have

$$\begin{aligned}
\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T d_k &\leq \left\|\nabla f(x_{k+1}) - \nabla f(x_k)\right\| \|d_k\| \\
&\leq M\|x_{k+1} - x_k\|\,\|d_k\| && \text{from (11.66)} \\
&\leq M\alpha_k \|d_k\|^2 && \text{from (11.67)}\,.
\end{aligned}$$

By grouping these two results, we have

$$(\beta_2 - 1)\nabla f(x_k)^T d_k \leq M\alpha_k \|d_k\|^2$$

or

$$\alpha_k \geq \frac{\beta_2 - 1}{M} \frac{\nabla f(x_k)^T d_k}{\|d_k\|^2}\,. \tag{11.70}$$

The first Wolfe condition (11.45) ensures that

$$f(x_k + \alpha_k d_k) - f(x_k) \leq \alpha_k \beta_1 \nabla f(x_k)^T d_k\,.$$

Therefore, since $\beta_1 \nabla f(x_k)^T d_k < 0$, we get

$$f(x_k + \alpha_k d_k) - f(x_k) \leq \beta_1 \frac{\beta_2 - 1}{M} \frac{\left(\nabla f(x_k)^T d_k\right)^2}{\|d_k\|^2} = -\widehat{\beta} \cos^2\theta_k \left\|\nabla f(x_k)\right\|^2,$$

by using (11.69) and defining $\widehat{\beta} = \beta_1(1-\beta_2)/M > 0$. Consequently, for an arbitrary K, we have

$$\sum_{k=0}^{K} f(x_k + \alpha_k d_k) - f(x_k) \leq -\sum_{k=0}^{K} \widehat{\beta} \cos^2 \theta_k \left\|\nabla f(x_k)\right\|^2$$

$$f(x_{K+1}) - f(x_0) \leq -\widehat{\beta} \sum_{k=0}^{K} \cos^2 \theta_k \left\|\nabla f(x_k)\right\|^2 .$$

Multiplying this last inequality by -1, we obtain

$$\widehat{\beta} \sum_{k=0}^{K} \cos^2 \theta_k \left\|\nabla f(x_k)\right\|^2 \leq f(x_0) - f(x_{K+1}) .$$

Since f is bounded from below, there exists f_0 such that $f(x) \geq f_0$, for all x. Therefore,

$$f(x_0) - f(x_{K+1}) \leq f(x_0) - f_0 , \quad \forall K ,$$

and then

$$\widehat{\beta} \sum_{k=0}^{K} \cos^2 \theta_k \left\|\nabla f(x_k)\right\|^2 \leq f(x_0) - f_0 , \quad \forall K .$$

Taking the limit $K \to \infty$, we conclude that the sequence is convergent, i.e.,

$$\sum_{k=0}^{+\infty} \cos^2 \theta_k \left\|\nabla f(x_k)\right\|^2 \leq f(x_0) - f_0 . \tag{11.71}$$

□

A necessary condition for the sequence of Zoutendijk's theorem to be convergent is that

$$\lim_{k\to\infty} \cos^2 \theta_k \left\|\nabla f(x_k)\right\|^2 = 0 .$$

In the context of global convergence (Definition 11.11), we want

$$\lim_{k\to\infty} \left\|\nabla f(x_k)\right\|^2 = 0 .$$

This would be the case if the sequence $\cos^2 \theta_k$ did not approach zero, i.e., if there exists $\delta > 0$ such that

$$-\frac{\nabla f(x_k)^T d_k}{\left\|\nabla f(x_k)\right\| \|d_k\|} \geq \delta , \quad \forall k .$$

A sequence of directions $\left(d_k\right)_k$ satisfying this property is said to be *gradient related* with the sequence of iterates $\left(x_k\right)_k$.

Definition 11.13 (Gradient related directions). Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, that is bounded from below and differentiable. Consider also an iterative algorithm that generates a sequence $(x_k)_k$ in $\mathbb{R}^n$, defined by x_0 and the iterations

$$x_{k+1} = x_k + \alpha_k d_k\,, \qquad k = 0, 1, \dots$$

The sequence $(d_k)_k$ is said to be *gradient related* with the sequence $(x_k)_k$ if, for all subsequences $(x_k)_{k\in\mathcal{K}}$ converging toward a non stationary point, i.e., all subsequences such that

$$\nabla f\left(\lim_{k\in\mathcal{K}} x_k\right) \neq 0\,,$$

the corresponding subsequence $(d_k)_{k\in\mathcal{K}}$ is bounded and satisfies

$$\limsup_{k\in\mathcal{K}} \nabla f(x_k)^T d_k < 0\,. \tag{11.72}$$

Then, if the sequence $(d_k)_k$ is gradient related with the sequence $(x_k)_k$, the angle between d_k and $\nabla f(x_k)$ does not come too close to 90 degrees.

Corollary 11.14 (Global convergence). *Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, that is bounded from below, differentiable, and for which the gradient is Lipschitz continuous (Definition B.16), i.e., that there exists $M > 0$ such that*

$$\left\|\nabla f(x) - \nabla f(y)\right\| \leq M\|x - y\|\,, \qquad \forall x, y \in \mathbb{R}^n\,. \tag{11.73}$$

Consider an algorithm generating the sequence $(x_k)_k$, defined by x_0 and the iterations

$$x_{k+1} = x_k + \alpha_k d_k\,, \qquad k = 0, 1, \dots\,,$$

with d_k gradient related with x_k (according to Definition 11.13), and α_k satisfies the Wolfe conditions (11.45) and (11.47). Then, regardless of $x_0 \in \mathbb{R}^n$,

$$\lim_{k\to\infty} \left\|\nabla f(x_k)\right\| = 0\,. \tag{11.74}$$

Proof. It is an immediate consequence of Zoutendijk's theorem. □

In the context of the preconditioned steepest descent method, in order for the sequence of Newton directions to be gradient related with the iterates, it suffices that the conditioning of the matrices D_k is bounded, i.e., that there exists $C > 0$ such that

$$\|D_k\|_2 \left\|D_k^{-1}\right\|_2 \leq C\,, \quad \forall k\,.$$

According to the Rayleigh-Ritz theorem (Theorem C.4), and since $\left\|D_k^{-1}\right\|_2$ is opposite to the smallest eigenvalue of D_k, we get

$$-\nabla f(x_k)^T d_k = \nabla f(x_k)^T D_k \nabla f(x_k) \geq \frac{\left\|\nabla f(x_k)\right\|^2}{\left\|D_k^{-1}\right\|_2}.$$

Then, by using (11.69),

$$\cos\theta_k \geq \frac{\left\|\nabla f(x_k)\right\|^2}{\left\|D_k^{-1}\right\|_2 \left\|\nabla f(x_k)\right\| \|d_k\|}.$$

Since

$$\|d_k\| \leq \left\|D_k\right\|_2 \left\|\nabla f(x_k)\right\|,$$

we get

$$\cos\theta_k \geq \frac{1}{\left\|D_k\right\|_2 \left\|D_k^{-1}\right\|_2} \geq \frac{1}{C} > 0.$$

Then, the cosine of the angle is bounded by a positive constant, and the directions do not degenerate by becoming asymptotically orthogonal to the gradient.

The presentation of the proof of Theorem 11.12 was inspired by Nocedal and Wright (1999). Examples 11.5 and 11.7 were inspired by Dennis and Schnabel (1996).

11.8 Project

The general organization of the projects is described in Appendix D.

Objective

The objective of the present project is to analyze the behavior of the descent methods, and to understand the role of the preconditioner and the role of the line search parameters.

Approach

1. Implement the preconditioned steepest descent method (Algorithm 11.1) with line search (Algorithm 11.5) and the following preconditioners:
   (a) $D_k = I$, to obtain the steepest descent method.
   (b) D_k diagonal matrix, with $D_k(i,i) = \max\left(1, \frac{\partial^2 f(x_k)}{\partial x_i^2}\right)^{-1}$.
   (c) D_k diagonal matrix, with $D_k(i,i) = \max\left(1, \frac{\partial^2 f(x_0)}{\partial x_i^2}\right)^{-1}$.
   (d) D_k diagonal matrix, with $D_k(i,i) = \frac{1}{\max\left(1, \left|(x_k)_i\right|\right)}$.

(e) D_k diagonal matrix, with $D_k(i,i) = \dfrac{1}{\max\left(1, \left|(x_0)_i\right|\right)}$.

Utilize several starting points. Each time, verify that the optimality conditions are satisfied at the final solution and compare the number of iterations.

2. In the line search (Algorithm 11.5), vary the parameters, by using (for instance) the following values:
$\alpha_0 = 0.1\,;\ 1.0\,;\ 10.0.$
$\beta_1 = 0.1\,;\ 0.5\,;\ 0.9.$
$\beta_2 = 0.1(\beta_1 - 1) + 1\,;\ 0.5(\beta_1 - 1) + 1\,;\ 0.9(\beta_1 - 1) + 1.$

Algorithms

Algorithms 11.1 and 11.5.

Problems

Exercise 11.1. The James Bond problem, described in Section 1.1.5.

Exercise 11.2. The problem

$$\min_{x\in\mathbb{R}^2} 2x_1x_2\, e^{-(4x_1^2+x_2)/8}\,.$$

Advice: draw the function and the level curves with a software such as Gnuplot, visually identify the stationary points, and then choose the starting points, either close to or far from the stationary points.

Exercise 11.3. The problem

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{n} i\alpha x_i^2\,,$$

$\bar{x} = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T$, with various values of n and α.

Exercise 11.4. The problem

$$\min_{x\in\mathbb{R}^2} 3x_1^2 + x_2^4\,.$$

Recommended starting point: $\begin{pmatrix} 1 & -2 \end{pmatrix}^T$.

Exercise 11.5. The Rosenbrock problem

$$\min_{x\in\mathbb{R}^2} 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 \qquad \text{(Section 11.6)}\,.$$

Recommended starting points: $\begin{pmatrix} 1.2 & 1.2 \end{pmatrix}^T$ and $\begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$.

Exercise 11.6. The problem

$$\min_{x\in\mathbb{R}^6} \sum_{i=1}^{m} \big(-e^{-0.1\,i} + 5\,e^{-i} - 3\,e^{-0.4i} + x_3\,e^{-0.1\,ix_1} - x_4\,e^{-0.1\,ix_2} + x_6\,e^{-0.1\,ix_5}\big)^2,$$

with various values of m.
Recommended starting point: $\begin{pmatrix} 1 & 2 & 1 & 1 & 4 & 3 \end{pmatrix}^T$.

Chapter 12

Trust region

Contents

In Chapter 11, we addressed a class of optimization methods enabling us to get around the shortcomings of Newton's local method, all the while maintaining its essential qualities when possible. In particular, the line search approach allows for global convergence, that is, the guarantee that the algorithm would converge to a local minimum, whatever the starting point, while reaching a quadratic convergence when the iterates would come close to a local minimum. The so-called *trust region* methods target the same objective, through a different approach.

Newton's local method (Algorithm 10.2) consists in minimizing a quadratic model of the function at each iteration. Taylor's theorem (Theorem C.2) shows us that the quadratic model of a function is a good approximation of the latter close to the point where it is defined. It is legitimate to define a region around the current iterate x_k within which we can have trust in the quadratic model. This region is called the *trust region*. It is defined by its radius Δ_k and a point x belongs to this region if

$$\|x_k - x\| \leq \Delta_k \,, \tag{12.1}$$

where $\|\cdot\|$ is a norm on $\mathbb{R}^n$.

Assuming that we knew the value of Δ_k, the wisest thing to do would be to minimize the quadratic model within this region, rather than over all of $\mathbb{R}^n$. The minimization problem (10.11) in Algorithm 10.2 can be replaced by the following problem, called the *trust region subproblem*.

Definition 12.1 (Trust region subproblem)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function, $\widehat{x} \in \mathbb{R}^n$, $m_{\widehat{x}}$ the quadratic model of f in $\widehat{x}$ (Definition 10.1) and $\Delta_k > 0$. The *trust region subproblem* is the following minimization problem:

$$\min_d m_{\widehat{x}}(\widehat{x} + d) = f(\widehat{x}) + d^T \nabla f(\widehat{x}) + \frac{1}{2} d^T \nabla^2 f(\widehat{x}) d \tag{12.2}$$

subject to

$$\|d\| \leq \Delta_k \,. \tag{12.3}$$

It is interesting to analyze the optimality conditions of the trust region subproblem for the Euclidean norm. To simplify this analysis, we rewrite (12.3)

$$\frac{1}{2}\left(\|d\|_2^2 - \Delta_k^2\right) \leq 0 \,. \tag{12.4}$$

The Lagrangian (Definition 4.3) of this problem is

$$L(d, \mu) = f(\widehat{x}) + d^T \nabla f(\widehat{x}) + \frac{1}{2} d^T \nabla^2 f(\widehat{x}) d + \frac{\mu}{2}\left(\|d\|_2^2 - \Delta_k^2\right) . \tag{12.5}$$

If d^* is the optimal solution to the trust region subproblem, the necessary optimality conditions guarantee that there exists $\mu^* \in \mathbb{R}$ such that

$$\nabla_d L(d^*, \mu^*) = \nabla f(\widehat{x}) + \nabla^2 f(\widehat{x}) d^* + \mu^* d^* = 0 \tag{12.6}$$

$$\mu^* \geq 0 \tag{12.7}$$

$$\mu^*\left(\|d^*\|_2^2 - \Delta_k^2\right) = 0 \,. \tag{12.8}$$

If d^* is strictly within the trust region, i.e., if $\|d^*\| < \Delta_k$, (12.8) guarantees that $\mu^* = 0$. Therefore, (12.6) can be simplified and corresponds to the necessary optimality conditions of the unconstrained problem. The constraint of the trust region, inactive in d^*, can be ignored. We obtain an iteration of Newton's local method.

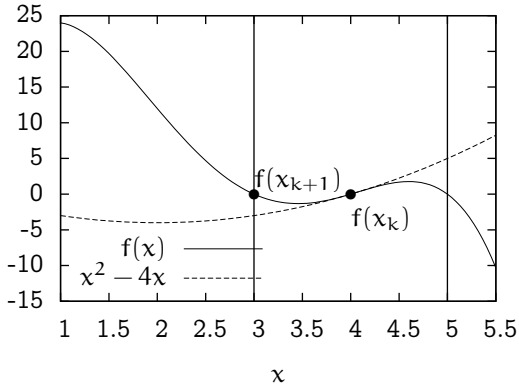
If d^* is at the border of the trust region, i.e., $\|d^*\|_2 = \Delta_k$, then (12.6) is written as

$$\left(\nabla^2 f(\widehat{x}) + \mu^* I\right) d^* = -\nabla f(\widehat{x}) \,.$$

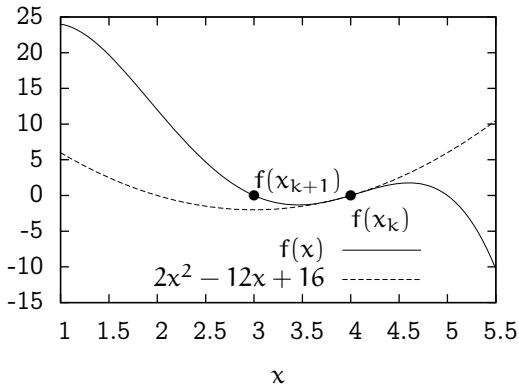
It is possible to demonstrate (see Conn et al., 2000, Theorem 7.2.1, page 172) that the matrix $\nabla^2 f(\widehat{x}) + \mu^* I$ is positive semi-definite. The analogy with the technique used in Newton's method with line search (Algorithm 11.8) is interesting. Indeed, in both cases, a multiple of the identity is added to the Hessian matrix to correct potential problems of Newton's local method. The following example illustrates the relationship between the trust region problem and the unconstrained minimization of a perturbed quadratic model.

Example 12.2 (Trust region subproblem)**.** Consider Example 10.2 and create the trust region subproblem in $x_k = 4$ with $\Delta_k = 1$.

$$\min_x x^2 - 4x \text{ subject to } \frac{1}{2}(x-4)^2 \leq \frac{1}{2} \,. \tag{12.9}$$



(a) Trust region



(b) Perturbed quadratic model

Figure 12.1: Illustration of Example 12.2

The optimal solution to this problem is $x_{k+1} = x^* = 3$, as illustrated in Figure 12.1(a). The trust region constraint is active at the solution. The Lagrangian of the problem is

$$L(x, \mu) = x^2 - 4x + \frac{\mu}{2}\big((x-4)^2 - 1\big)$$

and, as $x^* = 3$,

$$\begin{aligned}\nabla_x L(x^*, \mu^*) &= 2x^* - 4 + \mu^*(x^* - 4)\\ &= 2 - \mu^* = 0.\end{aligned}$$

Then, $\mu^* = 2$. The same result can be obtained by minimizing, without constraint $L(x, 2)$, i.e.,

$$2x^2 - 12\,x + 15\,.$$

Equivalently, we can minimize any model of the form

$$2x^2 - 12\,x + c\,,$$

where c is a constant and obtain the same result. By choosing $c = 16$, the quadratic function interpolates f in x_k. This is illustrated in Figure 12.1(b). It shows that imposing a trust region constraint can be equivalent to modifying the quadratic model, and optimize it without constraint.

In order to render the trust region method operational, we need only clarify two things:

1. How to solve the trust region subproblem (12.2)–(12.3).
2. How to determine the value of Δ_k.

12.1 Solving the trust region subproblem

In Chapter 11, we justified the inexact line search algorithm (Algorithm 11.5) by noticing that it was useless and computationally too demanding to solve exactly the minimization subproblem at each iteration. The same argument applies here. The trust region subproblem (12.2)–(12.3) is solved approximately.

There are many ways to solve this problem. Here we present two methods. The first is called the *dogleg* method. It is valid when the Hessian matrix at the current point $\nabla^2 f(\widehat{x})$ is positive definite. The second is based on the conjugate gradient method (Algorithm 9.2), and is therefore appropriate for large scale problems.

12.1.1 The dogleg method

The main idea of the dogleg method is the following:

- If the trust region is small, the first-order Taylor approximation of the function is probably already good, and the quadratic term plays only a minor role. It is therefore wise to follow the steepest descent direction toward the Cauchy point (Definition 10.4).
- If the trust region is larger, the second-order term becomes significant, and the Newton point (Definition 10.3) becomes the preferred target.
- In order to combine these two directions, the *dogleg* method consists in following a path that leads first to the Cauchy point, and then takes the Newton direction. This path is continued to the Newton point, or to the border of the trust region. If the Newton point is reached without leaving the trust region, this means that a local Newton iteration can be carried out.

We first assume that $\nabla^2 f(\widehat{x})$ is positive definite. Formally, we define a trajectory from the current point $\widehat{x}$, defined by $\widehat{x} + p(\alpha)$, with

$$p(\alpha) = \begin{cases} \alpha d_C & 0 \leq \alpha \leq 1 \\ d_C + (\alpha - 1)(x_d - x_C) & 1 \leq \alpha \leq 2 \\ \big(\eta(3-\alpha) + \alpha - 2\big) d_N & 2 \leq \alpha \leq 3\,, \end{cases} \tag{12.10}$$

where

- the Cauchy (i.e., steepest descent) direction (Definition 10.4) is defined by

$$d_C = -\frac{\nabla f(\widehat{x})^T \nabla f(\widehat{x})}{\nabla f(\widehat{x})^T \nabla^2 f(\widehat{x}) \nabla f(\widehat{x})} \nabla f(\widehat{x}),$$

- $x_C = \widehat{x} + d_C$ is the Cauchy point,
- the Newton direction (Definition 10.3) is defined by

$$d_N = -\nabla^2 f(\widehat{x})^{-1} \nabla f(\widehat{x}),$$

  if $\nabla^2 f(\widehat{x})$ is positive definite,
- the dogleg point is defined by

$$x_d = \widehat{x} + \eta d_N,$$

  where $\eta \leq 1$ defines the position of the dogleg point in the Newton direction; the recommended value in the literature is $\eta = 0.8\|d_C\|/\|d_N\| + 0.2$.

As illustrated in Figure 12.2(a), this trajectory connects $\widehat{x}$ ($\alpha = 0$), x_C ($\alpha = 1$), x_d ($\alpha = 2$), and x_N ($\alpha = 3$). Note that $p(1) = d_C$, $p(2) = x_d - \widehat{x} = \eta d_N$, and $p(3) = d_N$. This trajectory is followed either all the way (the Newton point), or to the border of the trust region.

Example 12.3 (Dogleg path)**.** Consider the function

$$\frac{1}{2} x_1^2 + \frac{9}{2} x_2^2$$

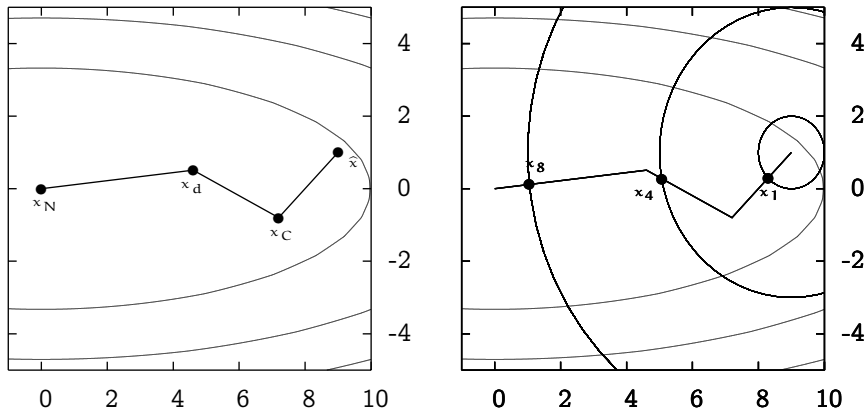
from Example 11.1 as well as the point $\widehat{x} = \begin{pmatrix} 9 & 1 \end{pmatrix}^T$. The Cauchy point ($x_C$), the Newton point ($x_N$), and the *dogleg* point (x_d), where the path joins the Newton direction, are

$$x_C = \begin{pmatrix} 7.2 \\ -0.8 \end{pmatrix}, \qquad x_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad x_d = \begin{pmatrix} 4.608 \\ 0.512 \end{pmatrix},$$

as illustrated in Figure 12.2(a). We consider three trust regions of radii $\Delta = 1, 4, 8$. The approximate solution to the trust region subproblem for each radius is

$$x_1 = \begin{pmatrix} 8.29289 \\ 0.29289 \end{pmatrix}, \qquad x_4 = \begin{pmatrix} 5.06523 \\ 0.28056 \end{pmatrix}, \qquad x_8 = \begin{pmatrix} 1.04893 \\ 0.11655 \end{pmatrix},$$

as illustrated in Figure 12.2(b).



(a) Definition of the path (b) Intersections with the trust regions

Figure 12.2: Illustration of the *dogleg* method

We need to calculate the intersection of the trajectory with the radius of the trust region. In the steepest descent direction, we have

$$\left\| -\frac{\Delta_k}{\|\nabla f(x_k)\|} \nabla f(x_k) \right\| = \Delta_k \,,$$

and the point

$$\widehat{x} - \frac{\Delta_k}{\|\nabla f(x_k)\|} \nabla f(x_k)$$

is located at the border of the trust region. The technique is identical in the Newton direction, where the point

$$\widehat{x} + \frac{\Delta_k}{\|d_N\|} d_N$$

is located at the border of the trust region. To find where the segment $x_d - x_C$ intersects with the trust region, we need to find the value of λ that solves the equation

$$\left\| x_C + \lambda(x_d - x_C) - \widehat{x} \right\|_2 = \Delta_k \,.$$

Lemma 12.4. *Consider $\Delta > 0$, x_C located in the trust region centered in $\widehat{x}$, i.e., such that $\|d_C\| = \|x_C - \widehat{x}\| \leq \Delta$, and x_d outside the trust region, i.e., such that $\|d_d\| = \|x_d - \widehat{x}\| > \Delta$. The step λ such that*

$$\left\| d_C + \lambda(d_d - d_C) \right\|_2 = \left\| x_C - \widehat{x} + \lambda(x_d - x_C) \right\|_2 = \Delta \tag{12.11}$$

is given by

$$\lambda = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{12.12}$$

with

$$a = \|d_d - d_C\|_2^2$$
$$b = 2d_C^T(d_d - d_C)$$
$$c = \|d_C\|_2^2 - \Delta^2 .$$

Proof. The result is obtained by denoting $d = x_d - x_C = d_d - d_C$ and by calculating the roots to the equation

$$\|d_C + \lambda d\|_2^2 = \Delta^2$$

or

$$d^T d\lambda^2 + 2d_C^T d\lambda + d_C^T d_C - \Delta^2 = 0 .$$

The coefficient of λ^2 is $a = d^T d$, that of λ is $b = 2d_C^T d$ and the independent term $c = d_C^T d_C - \Delta^2$. The discriminant of this equation is

$$b^2 - 4ac = 4(d_C^T d^2 - d^T d d_C^T d_C + d^T d\Delta^2) .$$

Since $d_C^T d_C \leq \Delta^2$, then $d^T d(\Delta^2 - d_C^T d_C) \geq 0$ and the discriminant is non negative. The equation has two solutions:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad \frac{-b - \sqrt{b^2 - 4ac}}{2a} .$$

We demonstrate that the first is always non negative and the second non positive. It corresponds to the intersection with the trust region in the direction d from x_c. The second root corresponds to the direction $-d$. We discuss the sign of b.

$b = 0$: in this case, it is trivial to show that the first root is positive and the second negative.

$b > 0$: since $b = 2d_C^T d > 0$, the second solution is negative because $a > 0$. For the first, we have

$$\begin{aligned}
\sqrt{b^2 - 4ac} &= 2\sqrt{(d_C^T d)^2 - d^T d(d_C^T d_C - \Delta^2)} \\
&\geq 2\sqrt{(d_C^T d)^2} && \text{because } \Delta^2 - d_C^T d_C \geq 0 \\
&\geq 2d_C^T d && \text{because } d_C^T d > 0 .
\end{aligned}$$

Therefore,

$$-b + \sqrt{b^2 - 4ac} \geq -2d_C^T d + 2d_C^T d = 0 ,$$

and the first root is positive as $a > 0$.

$b < 0$: since $b = 2d_C^T d < 0$, the first solution is positive because $a > 0$. For the second, we have

$$\begin{aligned}
\sqrt{b^2 - 4ac} &= 2\sqrt{(d_C^T d)^2 - d^T d(d_C^T d_C - \Delta^2)} \\
&\geq 2\sqrt{(d_C^T d)^2} && \text{because } \Delta^2 - d_C^T d_C \geq 0 \\
&\geq -2d_C^T d && \text{because } d_C^T d < 0 .
\end{aligned}$$

Therefore,

$$-b - \sqrt{b^2 - 4ac} \leq -2d_C^T d + 2d_C^T d = 0,$$

and the second root is negative. □

Algorithm 12.1: Intersection with the trust region

Objective

To find the intersection between a direction and the border of the trust region.

Input

$d_C \in \mathbb{R}^n$ such that $\|d_C\| \leq \Delta$

$d = d_d - d_C \in \mathbb{R}^n$ such that $d \neq 0$

$\Delta \in \mathbb{R}$ such that $\Delta > 0$

Output

The step λ such that $\|d_C + \lambda d\| = \Delta$.

$a := d^T d$

$b := 2d_C^T d$

$c := d_C^T d_C - \Delta^2$

$$\lambda := \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

For the method to work, we have to define the safeguards when the matrix $\nabla^2 f(\hat{x})$ is not positive definite. This is simple. If the function is concave in the steepest descent direction, i.e., if

$$\nabla f(\hat{x})^T \nabla^2 f(\hat{x}) \nabla f(\hat{x}) \leq 0,$$

then the quadratic function is unbounded from below in this direction, and decreases towards $-\infty$. It can therefore be followed until the border of the trust region. If the function is concave in the Newton direction, then it is ignored and the Cauchy point is selected. Algorithm 12.2 describes the dogleg method to solve approximately the trust region subproblem.

12.1.2 Steihaug-Toint method

The conjugate gradient method, presented in Section 9.2, is designed to minimize strictly convex quadratic problems. Steihaug (1983) and Toint (1981) proposed an adaptation of this method to solve the trust region subproblem.

The basic idea is the following. At each iteration of the conjugate gradient method, we first test whether the quadratic model is convex in the direction d_k. If this is not the case, we follow this direction until the border of the trust region and stop the iterations. Furthermore, as soon as an iterate is outside the trust region, we follow the last calculated direction until the border of the trust region and stop the algorithm. In all other cases, the method is applied in its original version. We thus obtain Algorithm 12.3, which should be used with $Q = \nabla^2 f(x_k)$ and $b = \nabla f(x_k)$.

Algorithm 12.2: Dogleg method

Objective

To find an approximate solution to the trust region subproblem $\min_{d\in\mathbb{R}^n} d^T\nabla f(\widehat{x}) + \frac{1}{2} d^T\nabla^2 f(\widehat{x})d$ subject to $\|d\|_2 \leq \Delta$.

Input

The gradient at the current point: $\nabla f(\widehat{x}) \in \mathbb{R}^n \neq 0$.

The Hessian at the current point: $\nabla^2 f(\widehat{x}) \in \mathbb{R}^{n\times n}$.

The radius of the trust region: $\Delta > 0$.

Output

Approximate solution d^*.

Cauchy point

$\beta := \nabla f(\widehat{x})^T\nabla^2 f(\widehat{x})\nabla f(\widehat{x})$. `Curvature in the steepest descent direction`

if $\beta \leq 0$ **then** the model is not convex

STOP with $d^* = -\dfrac{\Delta}{\|\nabla f(\widehat{x})\|}\nabla f(\widehat{x})$.

$\alpha := \nabla f(\widehat{x})^T\nabla f(\widehat{x})$.

$d_C := -\dfrac{\alpha}{\beta}\nabla f(\widehat{x})$ `using (10.17)`

if $\|d_C\| \geq \Delta$ **then** Cauchy point outside the trust region

STOP with $d^* := \dfrac{\Delta}{\|d_C\|}d_C$.

Newton point

Calculate d_N by solving $\nabla^2 f(\widehat{x})d_N = -\nabla f(\widehat{x})$.

if $d_N^T\nabla^2 f(\widehat{x})d_N \leq 0$ **then** the model is not convex

STOP with the Cauchy point, $d^* = d_C$.

if $\|d_N\| \leq \Delta$ **then** Newton point within the trust region

STOP with $d^* = d_N$.

Dogleg point

Calculate $d_d := \left(0.2 + \dfrac{0.8\alpha^2}{\beta|\nabla f(\widehat{x})^T d_N|}\right) d_n$.

if $\|d_d\| \leq \Delta$ **then** dogleg point within the trust region

STOP with $d^* = \dfrac{\Delta}{\|d_N\|}d_N$.

Between Cauchy and dogleg

Use Algorithm 12.1 to calculate λ^* such that $d_C + \lambda^*(d_d - d_C)$ is the intersection point between the segment connecting the Cauchy point and the dogleg point, with the border of the trust region.

STOP with $d^* = d_C + \lambda^*(d_d - d_C)$.

Algorithm 12.3: Steihaug-Toint truncated conjugate gradient method

Objective
  To find an approximate solution to the trust region subproblem $\min_x \frac{1}{2} x^T Q x + x^T b$ subject to $\|x\|_2 \leq \Delta$.
Input
  $Q \in \mathbb{R}^{n \times n}$
  $b \in \mathbb{R}^n$
  Radius of the trust region Δ
Output
  The approximate solution $x^* \in \mathbb{R}^n$
Initialization
  $k := 1$
  $x_1 := 0$
  $d_1 := -b$
Repeat
  if $d_k^T Q d_k \leq 0$ **then** the function is not convex along d_k
    $x^* = x_k + \lambda d_k$ where λ is obtained by Algorithm 12.1
  Calculate the step $\alpha_k := -\dfrac{d_k^T (Q x_k + b)}{d_k^T Q d_k}$
  Calculate the next iterate: $x_{k+1} := x_k + \alpha_k d_k$.
  if $\|x_{k+1}\| > \Delta$ **then**
    $x^* = x_k + \lambda d_k$ where λ is obtained by Algorithm 12.1
  Calculate $\beta_{k+1} := \dfrac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)} = \dfrac{(Q x_{k+1} + b)^T (Q x_{k+1} + b)}{(Q x_k + b)^T (Q x_k + b)}$.
  Calculate the new direction $d_{k+1} := -Q x_{k+1} - b + \beta_{k+1} d_k$. $k := k + 1$.
Until $\left\|\nabla f(x_k)\right\| = 0$ *or* $k = n + 1$
$x^* := x_k$

12.2 Calculation of the radius of the trust region

The radius of the trust region is determined by trial and error. At the first iteration, an arbitrary value is chosen ($\Delta = 10$, for instance). Subsequently, we evaluate the quality of the approximate solution to the trust region subproblem and the radius of the trust region is adjusted according to the evaluation.

We assume that the optimal solution (possibly approximate) to the trust region subproblem is d^*. In this case, we can compare the reduction of the model

$$m_{\widehat{x}}(\widehat{x}) - m_{\widehat{x}}(\widehat{x} + d^*)$$

with the reduction of the function

$$f(\widehat{x}) - f(\widehat{x} + d^*)\,.$$

If the model is reliable, these two quantities should be close. We calculate the ratio

$$\rho = \frac{f(\widehat{x}) - f(\widehat{x} + d^*)}{m_{\widehat{x}}(\widehat{x}) - m_{\widehat{x}}(\widehat{x} + d^*)}\,. \tag{12.13}$$

We consider three cases:

1. ρ is close to 1, or larger, and the model is very good,
2. ρ is close to 0, or smaller, and the model is poor,
3. ρ is in between, and the model is just good.

These cases are characterized by the constants η_1 and η_2 such that $0 < \eta_1 \leq \eta_2 < 1$. Typically, we take $\eta_1 = 0.01$ and $\eta_2 = 0.9$.

$\rho \geq \eta_2$: the fit between the model and the function seems to be *very good*, in the sense that the reduction predicted by the model has practically been reached or even exceeded.

$\eta_1 \leq \rho < \eta_2$: the fit between the model and the function is not perfect, but this model has nevertheless enabled to reduce the value of the function. We refer to it as *good*.

$\rho < \eta_1$: the fit between the model and the function is *poor*, in the sense that either the reduction of the function is negligible compared to the prediction made based on the model, or the value of the function has increased.

Several strategies for updating the trust region by using ρ have been proposed in the literature. Here is one of the most simple ones:

- If the fit is very good, the radius of the trust region is doubled.
- If the fit is good, the radius remains unchanged.
- If the fit is poor, the radius is reduced to $\frac{1}{2}\|d^*\|$.

Putting everything together, we obtain Newton's method with trust region (Algorithm 12.4).

Algorithm 12.4: Newton's method with trust region

Objective
  To find (an approximation of) a local minimum of the problem $\min_{x\in\mathbb{R}^n} f(x)$.
Input
  The twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
  The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
  The Hessian of the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.
  An initial solution $x_0 \in \mathbb{R}^n$.
  The radius of the first trust region Δ_0 (by default, $\Delta_0 = 10$).
  The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.
  The parameters $0 < \eta_1 \leq \eta_2 < 1$ (by default, $\eta_1 = 0.01$ and $\eta_2 = 0.9$).
Output
  An approximation of the optimal solution $x^* \in \mathbb{R}$.
Initialization
  $k := 0$.
Repeat
  Calculate d_k by solving (approximately) the trust region subproblem (12.2)–(12.3), with the *dogleg* method (Algorithm 12.2) or the Steihaug-Toint truncated conjugate gradient method (Algorithm 12.3).
  Calculate $\rho = \dfrac{f(x_k) - f(x_k + d_k)}{m_{x_k}(x_k) - m_{x_k}(x_k + d_k)}$.
  if $\rho < \eta_1$ **then** failure
    $x_{k+1} := x_k$
    $\Delta_{k+1} := \frac{1}{2}\|d_k\|$
  else success
    $x_{k+1} = x_k + d_k$
    if $\rho \geq \eta_2$ **then** very good
      $\Delta_{k+1} = 2\Delta_k$
    else just good
      $\Delta_{k+1} = \Delta_k$
  $k := k + 1$.
Until $\|\nabla f(x_k)\| \leq \varepsilon$
$x^* := x_k$.

As an illustration, we apply this method to Example 5.8, i.e.,

$$\min_{x\in\mathbb{R}^2} \frac{1}{2}x_1^2 + x_1 \cos x_2 \,,$$

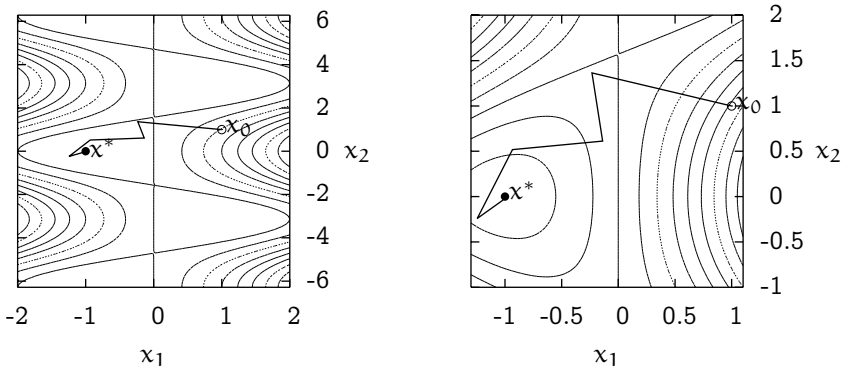
from the same starting point $x_0 = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$. In this case, the algorithm converges to

$$x^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \qquad \nabla f(x^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \nabla^2 f(x^*) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

which is a local minimum because it satisfies the sufficient optimality conditions (Theorem 5.7 and the discussions of Example 5.8). The iterations are illustrated in Figure 12.3, which is interesting to compare with Figure 10.1 and Figure 11.16. Table 12.1 shows, for each iteration,

- the iterate x_k,
- the value of the function,
- the gradient norm,
- the radius of the trust region Δ_k,
- the ratio ρ defined by (12.13),
- the manner in which the *dogleg* method (Algorithm 12.2) is ended (1: partial Cauchy step, 2: pure Newton step, 3: partial Newton step, 4: dogleg between Cauchy and Newton, -2: Cauchy point due to the negative curvature of the Newton direction),
- the state of the iteration: poor ($-$), good ($+$), very good ($++$).

We notice that after two iterations where a negative curvature has been detected, the iterations are the same as Newton's local method starting from iteration 4.



(a) Iterates (b) Zoom

Figure 12.3: Iterates of Newton's method with trust region and dogleg for Example 5.8 ($\Delta_0 = 10$)

To further illustrate the trust region method, it is worth trying the same algorithm with a smaller initial radius ($\Delta_0 = 1$). The iterations are illustrated in Figure 12.4. Table 12.2 is similar to Table 12.1. We note that the iterations 3 to 11 are actually equivalent to the steepest descent method. Indeed, since the curvature of the function is negative in the Newton direction, the dogleg method could not be applied. At iteration 13, the dogleg method generated a point located on the arc between the Cauchy point and the dogleg point. As of iteration 14, the iterations are those of Newton's local method, and a rapid convergence is achieved.

Table 12.1: Newton's method with trust region for the minimization of Example 5.8 ($\Delta_0 = 10$)

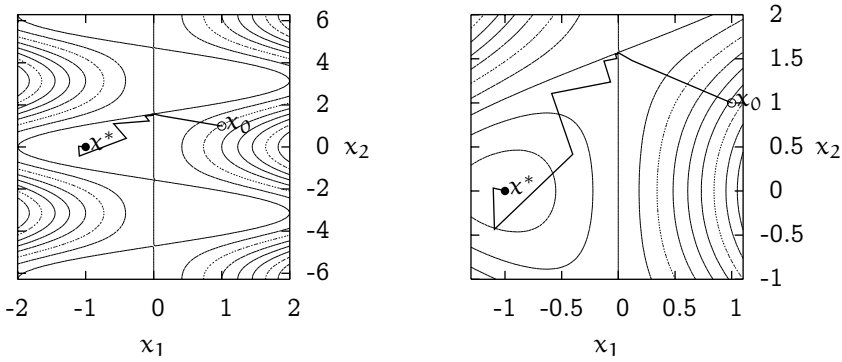
k	x_k		$f(x_k)$	$\|\nabla f(x_k)\|$	Δ_k	ρ		
0	+1.00000e+00	+1.00000e+00	+1.04030e+00	+1.75517e+00	+1.00000e+01			
1	-2.33845e-01	+1.36419e+00	-2.06286e-02	+2.30665e-01	+2.00000e+01	+9.61445e-01	2	++
2	-1.39549e-01	+6.12415e-01	-1.04451e-01	+6.83438e-01	+4.00000e+01	+9.59237e-01	-2	++
3	-9.34497e-01	+5.18458e-01	-3.75047e-01	+4.67749e-01	+8.00000e+01	+9.89241e-01	-2	++
4	-1.24534e+00	-2.41828e-01	-4.33668e-01	+4.05285e-01	+8.00000e+01	+3.53577e-01	2	+
5	-1.01925e+00	-3.99531e-02	-4.99001e-01	+4.53782e-02	+1.60000e+02	+1.06883e+00	2	++
6	-1.00077e+00	-7.03374e-04	-4.99999e-01	+1.04323e-03	+3.20000e+02	+1.01414e+00	2	++
7	-1.00000e+00	-5.40691e-07	-5.00000e-01	+5.94432e-07	+6.40000e+02	+1.00035e+00	2	++

Table 12.2: Newton's method with trust region for the minimization of Example 5.8 ($\Delta_0 = 1$)

k	x_k		$f(x_k)$	$\|\nabla f(x_k)\|$	Δ_k	ρ		
0	+1.00000e+00	+1.00000e+00	+1.04030e+00	+1.75517e+00	+1.00000e+00			
1	+1.22417e-01	+1.47943e+00	+1.86628e-02	+2.45993e-01	+2.00000e+00	+9.47588e-01	1	++
2	-1.01629e-03	+1.57003e+00	-2.61464e-07	+1.04679e-03	+4.00000e+00	+9.97536e-01	2	++
3	-5.36408e-04	+1.56809e+00	-1.30949e-06	+2.23824e-03	+8.00000e+00	+1.00000e+00	-2	++
4	-5.08985e-03	+1.56696e+00	-6.55830e-06	+5.24259e-03	+1.60000e+01	+9.99998e-01	-2	++
5	-2.68657e-03	+1.55723e+00	-3.28448e-05	+1.12089e-02	+3.20000e+01	+1.00000e+00	-2	++
6	-2.54882e-02	+1.55160e+00	-1.64466e-04	+2.62487e-02	+6.40000e+01	+9.99957e-01	-2	++
7	-1.34638e-02	+1.50289e+00	-8.22887e-04	+5.60207e-02	+1.28000e+02	+1.00002e+00	-2	++
8	-1.27230e-01	+1.47480e+00	-4.10176e-03	+1.30473e-01	+2.56000e+02	+9.98929e-01	-2	++
9	-6.84750e-02	+1.23764e+00	-2.00488e-02	+2.66527e-01	+5.12000e+02	+1.00051e+00	-2	++
10	-5.88466e-01	+1.10750e+00	-8.98399e-02	+5.45134e-01	+1.02400e+03	+9.77015e-01	-2	++
11	-4.02533e-01	+4.16075e-01	-2.87173e-01	+5.37370e-01	+2.04800e+03	+1.01116e+00	-2	++
12	-4.02533e-01	+4.16075e-01	-2.87173e-01	+5.37370e-01	+1.09534e+00	-2.88565e+00	2	-
13	-1.09350e+00	-4.33824e-01	-3.94333e-01	+4.95902e-01	+1.09534e+00	+2.99489e-01	4	+
14	-1.10395e+00	+3.38629e-02	-4.93964e-01	+1.11009e-01	+2.19067e+00	+9.35399e-01	2	++
15	-1.00047e+00	+3.16268e-03	-4.99995e-01	+3.19902e-03	+4.38135e+00	+1.00813e+00	2	++
16	-1.00000e+00	+1.44712e-06	-5.00000e-01	+5.20201e-06	+8.76269e+00	+1.00045e+00	2	++
17	-1.00000e+00	+7.23075e-12	-5.00000e-01	+7.30618e-12	+1.75254e+01	+1.00001e+00	2	++

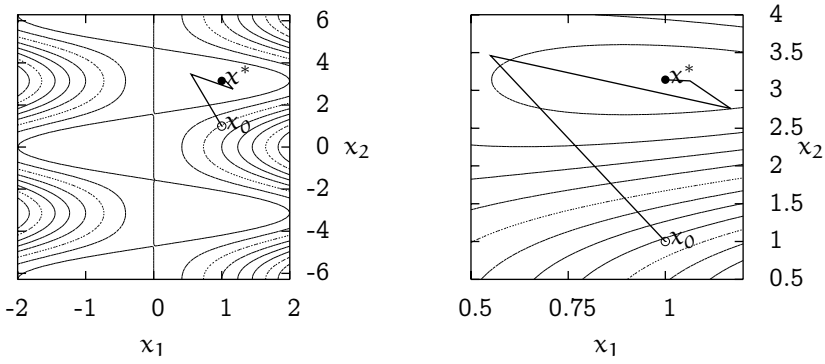
Table 12.3: Newton's method with trust region and Algorithm 12.3 for the minimization of Example 5.8 ($\Delta_0 = 10$)

k	x_k		$f(x_k)$	$\|\nabla f(x_k)\|$	Δ_k	ρ		
0	+1.00000e+00	+1.00000e+00	+1.04030e+00	+1.75517e+00	+1.00000e+01			
1	+1.00000e+00	+1.00000e+00	+1.04030e+00	+1.75517e+00	+5.00000e+00	-7.51975e-02	3	-
2	+1.00000e+00	+1.00000e+00	+1.04030e+00	+1.75517e+00	+2.50000e+00	-1.23991e-01	3	-
3	+5.50230e-01	+3.45921e+00	-3.71332e-01	+4.35121e-01	+2.50000e+00	+4.19624e-01	3	+
4	+1.16790e+00	+2.76142e+00	-4.02518e-01	+4.95063e-01	+2.50000e+00	+1.70028e-01	1	+
5	+1.06365e+00	+3.12536e+00	-4.97834e-01	+6.60783e-02	+5.00000e+00	+1.04357e+00	1	++
6	+1.00012e+00	+3.14062e+00	-5.00000e-01	+9.75276e-04	+1.00000e+01	+1.00343e+00	1	++
7	+1.00000e+00	+3.14159e+00	-5.00000e-01	+4.81675e-07	+2.00000e+01	+1.00011e+00	1	++



(a) Iterates (b) Zoom

Figure 12.4: Iterates of Newton's method with trust region and dogleg for Example 5.8 ($\Delta_0 = 1$)



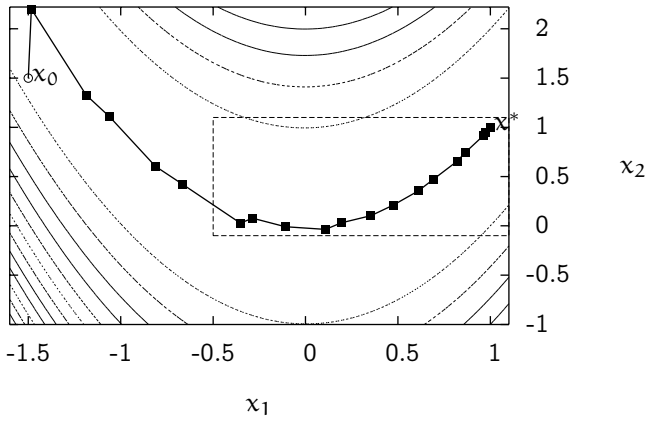
(a) Iterates (b) Zoom

Figure 12.5: Iterates of Newton's method with trust region and Steihaug-Toint for Example 5.8 ($\Delta_0 = 10$)

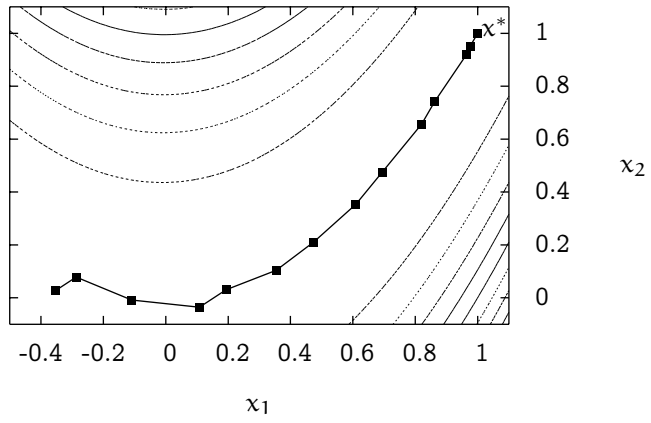
Finally, we apply the trust region method by using the truncated conjugate gradient method (Algorithm 12.3) to solve the trust region subproblem. Table 12.3 lists the iterations. The penultimate column gives the reasons why Algorithm 12.3 is stopped. Either it converges toward the unconstrained minimum of the quadratic model (1), or it generates an iterate outside the trust region (2), or it detects a direction in which the model has a negative curvature (3). We notice that, as of iteration 4, the constraint of the trust region subproblem no longer plays a role. In this case, the iterations are those of Newton's local method, and a rapid convergence is achieved.

12.3 The Rosenbrock problem

Following on the analysis performed in Section 11.6, we apply the algorithms presented in this chapter to the Rosenbrock problem. Qualitatively, we reach the same conclusions: the exploitation of the second derivatives by Newton's method (here with trust region) makes it significantly superior to the steepest descent algorithm, as illustrated in Figure 12.6. Compared to the line search approach, the number of iterations is roughly the same (23 for line search, 29 for trust region).



(a) 29 iterations



(b) Zoom

Figure 12.6: Newton's method with trust region

12.4 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of the present project is to analyze the behavior of trust region methods when solving the following problems and to compare it with that of descent methods.

Approach

Implement Algorithm 12.4, once with the dogleg method (Algorithm 12.2) and once with the Steihaug-Toint truncated conjugate gradient method (Algorithm 12.3), in order to solve the trust region subproblem. Test several variations by varying the following parameters:

- $\Delta_0 = 0.1\,;\ 1.0\,;\ 10.0\,;\ 100.0\ ;$
- $\eta_1 = 0.1\,;\ 0.5\,;\ 0.9\ ;$
- $\eta_2 = 0.1\,(\eta_1 - 1) + 1\,;\ 0.5\,(\eta_1 - 1) + 1\,;\ 0.9\,(\eta_1 - 1) + 1.$

Compare these algorithms with the descent method in Chapter 11. Analyze the results by using the method described in Section D.2.

Algorithms

Algorithms 12.2, 12.3, and 12.4.

Problems

Exercise 12.1. The James Bond problem, described in Section 1.1.5.

Exercise 12.2. The problem

$$\min_{x \in \mathbb{R}^2} 2x_1 x_2\, e^{-(4x_1^2 + x_2)/8}\,.$$

Advice: draw the function and the level curves with a software such as Gnuplot, visually identify the stationary points, and then choose the starting points, either close to or far from the stationary points.

Exercise 12.3. The problem

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} i\alpha x_i^2\,, \qquad \bar{x} = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T,$$

with various values for n and for α.

Exercise 12.4. The problem

$$\min_{x \in \mathbb{R}^2} 3x_1^2 + x_2^4\,.$$

Recommended starting point: $\begin{pmatrix} 1 & -2 \end{pmatrix}^T$.

Exercise 12.5. The Rosenbrock problem

$$\min_{x\in\mathbb{R}^2} 100\left(x_2 - x_1^2\right)^2 + (1-x_1)^2 \qquad \text{(Section 12.3)}.$$

Recommended starting point: $\begin{pmatrix} 1.2 & 1.2 \end{pmatrix}^T$ and $\begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$.

Exercise 12.6. The problem

$$\min_{x\in\mathbb{R}^6} \sum_{i=1}^{m} \big(-e^{-0.1\,i} + 5\,e^{-i} - 3\,e^{-0.4\,i} + x_3\,e^{-0.1\,i x_1} - x_4\,e^{-0.1\,i x_2} + x_6\,e^{-0.1\,i x_5}\big)^2,$$

Recommended starting point $\begin{pmatrix} 1 & 2 & 1 & 1 & 4 & 3 \end{pmatrix}^T$.

Chapter 13

Quasi-Newton methods

Contents

Newton's method, either with line search (Algorithm 11.8) or with trust region (Algorithm 12.4), requires the use of the Hessian matrix of the function at each iteration. This matrix poses some practical problems. First, the analytical calculation of the second derivatives, as well as their implementation, is often tedious and error-prone. Moreover, once the work has been done, the calculation of this matrix at each iteration of the algorithm is time-consuming and can be detrimental to the effectiveness of the algorithms. We therefore adapt the quasi-Newton methods of Chapter 8 to optimization problems in order to maintain the structure of the algorithm without using the Hessian matrix.

13.1 BFGS

Keeping in mind that Newton's method aims at solving the system of equations $\nabla f(x) = 0$, it is natural to be directly inspired by the secant methods for systems of equations presented in Chapter 8 and to propose to approximate the matrix $\nabla^2 f(\hat{x})$ by using the Broyden update (8.15), i.e., with a matrix H_k defined by

$$H_k = H_{k-1} + \frac{(y_{k-1} - H_{k-1} d_{k-1}) d_{k-1}^T}{d_{k-1}^T d_{k-1}}, \tag{13.1}$$

with

$$\begin{aligned} d_{k-1} &= x_k - x_{k-1} \\ y_{k-1} &= \nabla f(x_k) - \nabla f(x_{k-1}), \end{aligned} \tag{13.2}$$

which is (8.11) where F has been replaced by ∇f.

This matrix satisfies the secant equation (8.10), i.e., the quadratic model formed from H_k has the same gradient as the function in x_{k-1} and in x_k. The main problem with this method is that the matrix H_k is generally not symmetric nor positive definite, as can be seen for iteration 18 in Table 8.7.

William Cooper Davidon was born in Fort Lauderdale in Florida, on March 18, 1927. A physicist by education, he is currently professor emeritus of mathematics at Haverford College, in Pennsylvania. Nocedal and Wright (1999) tell the following story. In the middle of the 1950s, Davidon attempted to solve an optimization problem at the Argonne National Laboratory. The computers at the time were not too stable. They always stopped before the algorithm was done. Stimulated by this frustration, Davidon developed a faster method in order to solve his problem. This first quasi-Newton algorithm was one of the most revolutionary ideas in non linear optimization. The irony of the story is that the original article by Davidon (1959) was not accepted for publication at the time. It wasn't published until 1991, in the first issue of *SIAM Journal on Optimization* (Davidon, 1991).

Figure 13.1: William C. Davidon

In the context of optimization, as the matrix is approximating the second derivatives matrix, it makes sense to force it to be symmetric. Moreover, as Newton's method has to be modified when the second derivative matrix is non positive definite, it makes sense to also enforce positive definiteness, in order to avoid these modifications.

The process can be initialized with a symmetric positive definite matrix (the identity matrix, for instance), but these properties must be maintained by the update formula that generates H_k from H_{k-1}. The following procedure is proposed:

1. Consider H_{k-1} symmetric positive definite.
2. Calculate the Cholesky factorization (Definition B.18) of $H_{k-1} = L_{k-1}L_{k-1}^T$.
3. Perform an update of L_{k-1}, in order to obtain a matrix A_k.
4. Take $H_k = A_k A_k^T$, in order to obtain a symmetric and positive definite matrix.

The secant equation (8.10) is written as

$$A_k A_k^T d_{k-1} = y_{k-1} \tag{13.3}$$

and can be decomposed into two equations:

$$A_k x = y_{k-1} \tag{13.4}$$

$$A_k^T d_{k-1} = x\,. \tag{13.5}$$

Based on the principles of the secant method, we calculate A_k so that it satisfies (13.4) by being as close as possible to L_{k-1}. We use the Broyden update formula

(8.15). In order to simplify the notations, we temporarily abandon the indices k and $k-1$, to obtain

$$A = L + \frac{(y - Lx)x^T}{x^T x}. \tag{13.6}$$

We must now establish the value of x in order for Equation (13.5) to also be satisfied. By combining (13.5) and (13.6), we get

$$x = A^T d = L^T d + \frac{(y - Lx)^T d}{x^T x} x. \tag{13.7}$$

The latter equation can only have a solution if $L^T d$ is a multiple of x, i.e., if there exists $\alpha \in \mathbb{R}$ such that

$$x = \alpha L^T d. \tag{13.8}$$

We immediately note that

$$x^T x = \alpha^2 d^T L L^T d = \alpha^2 d^T H d. \tag{13.9}$$

By combining (13.7), (13.8), and (13.9), we obtain

$$\begin{aligned}
\alpha L^T d &= L^T d + \frac{\alpha}{\alpha^2 d^T H d} (y - \alpha H d)^T d \, L^T d \\
&= L^T d + \frac{1}{\alpha d^T H d} (y^T d - \alpha d^T H d) \, L^T d \\
&= L^T d + \frac{y^T d}{\alpha d^T H d} L^T d - L^T d \\
&= \frac{y^T d}{\alpha d^T H d} L^T d.
\end{aligned}$$

Then,

$$\alpha^2 L^T d = \frac{y^T d}{d^T H d} L^T d$$

or

$$\alpha^2 = \frac{y^T d}{d^T H d}. \tag{13.10}$$

It is important to note that (13.10) only makes sense if $y^T d > 0$.

At this point, the equations (13.6), (13.8), and (13.10) define the matrix A:

$$A = L + \frac{1}{y^T d} \left(\alpha y d^T L - \frac{y^T d}{d^T H d} H d d^T L \right).$$

The calculation of AA^T is tedious, but direct.

$$\begin{aligned}
AA^T = H &+ \frac{\alpha}{y^T d} H d y^T - \frac{\alpha^2}{y^T d} H d d^T H \\
&+ \frac{\alpha}{y^T d} y d^T H + \frac{\alpha^2}{(y^T d)^2} y d^T H d y^T - \frac{\alpha^3}{(y^T d)^2} y d^T H d d^T H \\
&- \frac{\alpha^2}{y^T d} H d d^T H - \frac{\alpha^3}{(y^T d)^2} H d d^T H d y^T + \frac{\alpha^4}{(y^T d)^2} H d d^T H d d^T H.
\end{aligned}$$

By simplifying, and reintegrating the indices, we obtain an update formula for H_k

$$A_k A_k^T = H_k = H_{k-1} + \frac{y_{k-1} y_{k-1}^T}{y_{k-1}^T d_{k-1}} - \frac{H_{k-1} d_{k-1} d_{k-1}^T H_{k-1}}{d_{k-1}^T H_{k-1} d_{k-1}}. \tag{13.11}$$

This update formula was discovered independently in the late 1960s by the mathematicians C. G. Broyden, R. Fletcher, D. Goldfarb, and D. F. Shanno (Figure 13.2), and is now called the "BFGS" update, which is the acronym of their names.

Figure 13.2: C. G. Broyden, R. Fletcher, D. Goldfarb, and D. F. Shanno

Definition 13.1 (BFGS update)**.** Consider the differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ and two iterates x_{k-1} and x_k such that $d_{k-1}^T y_{k-1} > 0$, with $d_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$. Consider a symmetric positive definite matrix $H_{k-1} \in \mathbb{R}^{n \times n}$. The BFGS update is defined by

$$H_k = H_{k-1} + \frac{y_{k-1} y_{k-1}^T}{y_{k-1}^T d_{k-1}} - \frac{H_{k-1} d_{k-1} d_{k-1}^T H_{k-1}}{d_{k-1}^T H_{k-1} d_{k-1}}. \tag{13.12}$$

It is important to emphasize that the symmetric and positive definite secant equation (13.3) does not always have a solution.

Lemma 13.2. *Consider $d, y \in \mathbb{R}^n$, $d \neq 0$. There then exists a non singular matrix $A \in \mathbb{R}^{n \times n}$ such that*

$$AA^T d = y$$

if and only if

$$d^T y > 0.$$

Proof. Necessary condition. The above development, and in particular the equations (13.6), (13.8), and (13.10), ensure that if $d^T y > 0$, the following matrix is a solution to the secant equation:

$$A = L + \frac{1}{y^T d}\left(\alpha y d^T L - \frac{y^T d}{d^T H d} H d d^T L\right)$$

with $H = LL^T$ and α such that (13.10) is satisfied.

Sufficient condition. If $AA^T d = y$, then $d^T AA^T d = d^T y$. Since AA^T is positive definite, then $d^T y > 0$. □

The condition $d^T y > 0$ is always satisfied if the second Wolfe condition (Definition 11.8) is used. Indeed,

$$\begin{aligned} \nabla f(x_{k-1} + \alpha_{k-1} d_{k-1})^T d_{k-1} &\geq \beta_2 \nabla f(x_{k-1})^T d_{k-1} && \text{from (11.47)} \\ \nabla f(x_k)^T d_{k-1} - \nabla f(x_{k-1})^T d_{k-1} &\geq (\beta_2 - 1)\nabla f(x_{k-1})^T d_{k-1} \\ y_{k-1}^T d_{k-1} &\geq (\beta_2 - 1)\nabla f(x_{k-1})^T d_{k-1} && \text{from (13.2)}. \end{aligned}$$

If d_{k-1} is a descent direction, then $\nabla f(x_{k-1})^T d_{k-1} < 0$ (Definition 2.10). Since $\beta_2 < 1$ (Definition 11.8), we have

$$y_{k-1}^T d_{k-1} \geq (\beta_2 - 1)\nabla f(x_{k-1})^T d_{k-1} > 0\,.$$

We can thus adapt Newton's method with line search (Algorithm 11.8), by replacing the Hessian of f by the BFGS approximation. Note that, unlike the Hessian of f, we are certain that the matrix H_k is positive definite, which significantly simplifies the algorithm. The direction d_k of the algorithm is calculated by solving the system of equations

$$H_k d_k = -\nabla f(x_k)\,.$$

In order to avoid solving this system at each iteration, it may be appropriate to analytically calculate H_k^{-1} and obtain d_k by a simple matrix-vector product

$$d_k = -H_k^{-1} \nabla f(x_k)\,.$$

We need only[1] apply to (13.12) the Sherman-Morrison-Woodbury formula (Theorem C.17) to obtain

$$H_k^{-1} = \left(I - \frac{d_{k-1} y_{k-1}^T}{d_{k-1}^T y_{k-1}}\right) H_{k-1}^{-1} \left(I - \frac{y_{k-1} d_{k-1}^T}{d_{k-1}^T y_{k-1}}\right) + \frac{d_{k-1} d_{k-1}^T}{d_{k-1}^T y_{k-1}}\,. \tag{13.13}$$

The method is described in Algorithm 13.1.

The iterations of the BFGS method applied to Example 5.8 are listed in Table 13.1 and shown in Figure 13.3. The values of H_k^{-1} and $d_{k-1}^T y_{k-1}$ at each iteration are given in Table 13.2.

[1] Again, this is tedious, but straightforward.

Algorithm 13.1: Quasi-Newton BFGS method

Objective

To find (an approximation of) a local minimum of the problem $\min_{x \in \mathbb{R}^n} f(x)$

Input

The continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.

An initial solution $x_0 \in \mathbb{R}^n$.

A first approximation of the inverse of the Hessian $H_0^{-1} \in \mathbb{R}^{n \times n}$ which is symmetric positive definite. By default, $H_0^{-1} = I$.

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}$

Initialization

$k := 0$

Repeat

$d_k := -H_k^{-1} \nabla f(x_k)$

Determine α_k by applying a line search (Algorithm 11.5) with $\alpha_0 = 1$

$x_{k+1} := x_k + \alpha_k d_k$

$k := k + 1$

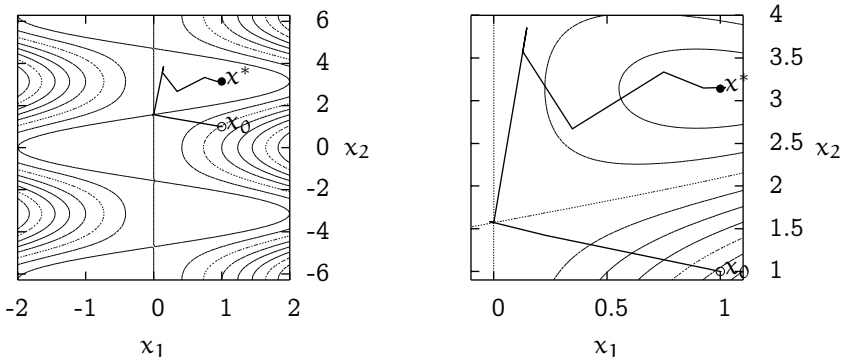
Update H_k^{-1}

$$H_k^{-1} := \left(I - \frac{\bar{d}_{k-1} y_{k-1}^T}{\bar{d}_{k-1}^T y_{k-1}} \right) H_{k-1}^{-1} \left(I - \frac{\bar{y}_{k-1} d_{k-1}^T}{\bar{d}_{k-1}^T y_{k-1}} \right) + \frac{\bar{d}_{k-1} \bar{d}_{k-1}^T}{\bar{d}_{k-1}^T y_{k-1}}$$

with $\bar{d}_{k-1} = \alpha_{k-1} d_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$.

Until $\|\nabla f(x_k)\| \leq \varepsilon$

$x^* := x_k$



(a) Iterates (b) Zoom

Figure 13.3: Iterates of the quasi-Newton BFGS method for Example 5.8

Table 13.1: Iterates of the BFGS method (Algorithm 13.1) of Example 5.8

k	x_k		$f(x_k)$	$\|\nabla f(x_k)\|_2$
0	1.00000000e+00	1.00000000e+00	1.04030231e+00	1.75516512e+00
1	2.29848847e-01	1.42073549e+00	6.07772543e-02	4.42214869e-01
2	-1.82864218e-02	1.58305828e+00	3.91418158e-04	3.56023470e-02
3	2.08564945e-04	1.57097473e+00	-1.54584949e-08	2.10734922e-04
4	1.46166453e-01	3.85753135e+00	-9.95969816e-02	6.15829050e-01
5	1.28297837e-01	3.57470602e+00	-1.08221093e-01	7.81223552e-01
6	3.46702460e-01	2.67154972e+00	-2.49000880e-01	5.67023829e-01
7	7.50084904e-01	3.33664226e+00	-4.54548154e-01	2.72899395e-01
8	9.24914367e-01	3.14934086e+00	-4.97153311e-01	7.53969637e-02
9	1.02337559e+00	3.15574887e+00	-4.99624251e-01	2.75857811e-02
10	1.00059290e+00	3.13396355e+00	-4.99970706e-01	7.65884832e-03
11	9.98441459e-01	3.14306370e+00	-4.99997705e-01	2.14077500e-03
12	1.00009954e+00	3.14162418e+00	-4.99999995e-01	1.04416861e-04
13	9.99996251e-01	3.14158835e+00	-5.00000000e-01	5.70573905e-06
14	9.99999984e-01	3.14159270e+00	-5.00000000e-01	4.86624055e-08

Table 13.2: Secant approximation of the BFGS method (Algorithm 13.1) for Example 5.8

k	$(H_k^{-1})_{1,1}$	$(H_k^{-1})_{2,2}$	$(H_k^{-1})_{1,2}$	$d_{k-1}^T y_{k-1}$
1	7.33361383e-01	9.35044922e-01	1.32282407e-01	1.15252889e+00
2	6.98321680e-01	9.20088043e-01	1.55175319e-01	1.41567951e-01
3	7.00564006e-01	9.15084679e-01	1.58271872e-01	7.89012355e-04
4	2.08781169e+00	1.16981556e+02	1.47289281e+01	1.31040555e-01
5	4.60491698e-01	1.26014080e+01	-1.44703795e+00	1.49596079e-02
6	5.06374127e-01	3.75733536e+00	-4.72552140e-01	2.41674525e-01
7	1.42353095e+00	2.34847423e+00	-1.43694279e-01	3.27748268e-01
8	1.33482497e+00	1.62667185e+00	2.40692428e-01	5.31426899e-02
9	1.00323127e+00	1.58983802e+00	-5.30801141e-02	9.74843071e-03
10	1.17686642e+00	1.17697397e+00	-1.85946123e-01	1.00256605e-03
11	1.08720063e+00	1.00549956e+00	2.39576346e-02	8.75212743e-05
12	1.02776154e+00	1.03763150e+00	3.12543311e-02	4.81635307e-06
13	1.00185313e+00	1.01535053e+00	-5.35546070e-03	1.19529543e-08
14	1.00741129e+00	1.00546900e+00	-6.36327477e-03	3.28322670e-11

13.2 Symmetric update of rank 1 (SR1)

The BFGS update is an update of rank 2, i.e., the matrix $H_k - H_{k-1}$ is a matrix of rank 2. It is also possible to define a symmetric update of rank 1, i.e., such that

$$H_k = H_{k-1} + \beta v v^T, \tag{13.14}$$

where $v \in \mathbb{R}^n$ and $\beta = 1$ or -1. The secant equation is then written as

$$y_{k-1} = H_k d_{k-1} = H_{k-1} d_{k-1} + \beta v v^T d_{k-1},$$

i.e., as $v^T d_{k-1}$ is a scalar,

$$y_{k-1} - H_{k-1} d_{k-1} = \beta v^T d_{k-1} v = \frac{1}{\gamma} v, \tag{13.15}$$

or

$$v = \gamma(y_{k-1} - H_{k-1} d_{k-1}) \tag{13.16}$$

with

$$\frac{1}{\gamma} = \beta v^T d_{k-1}. \tag{13.17}$$

By replacing (13.16) in (13.17), we obtain

$$d_{k-1}^T (y_{k-1} - H_{k-1} d_{k-1}) = \frac{1}{\beta \gamma^2}. \tag{13.18}$$

According to (13.16), we have

$$\beta v v^T = \beta \gamma^2 (y_{k-1} - H_{k-1} d_{k-1})(y_{k-1} - H_{k-1} d_{k-1})^T. \tag{13.19}$$

We need only combine (13.14), (13.18), and (13.19) to get

$$H_k = H_{k-1} + \frac{(y_{k-1} - H_{k-1} d_{k-1})(y_{k-1} - H_{k-1} d_{k-1})^T}{d_{k-1}^T (y_{k-1} - H_{k-1} d_{k-1})}. \tag{13.20}$$

Definition 13.3 (SR1 update)**.** Consider the differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ and the two iterates x_{k-1} and x_k. Let $H_{k-1} \in \mathbb{R}^{n \times n}$ be a symmetric matrix. The *symmetric rank one* (SR1) update is defined by

$$H_k = H_{k-1} + \frac{(y_{k-1} - H_{k-1} d_{k-1})(y_{k-1} - H_{k-1} d_{k-1})^T}{d_{k-1}^T (y_{k-1} - H_{k-1} d_{k-1})}, \tag{13.21}$$

with $d_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$.

Note that this update is well defined only if $d_{k-1}^T (y_{k-1} - H_{k-1} d_{k-1}) \neq 0$. Also, it does not necessarily generate a positive definite matrix, even if H_{k-1} is one. For this reason, it is preferable to use BFGS when dealing with algorithms based on line search. However, in the context of trust region methods, the SR1 update has proven effective. We use the SR1 update in Newton's method with trust region to obtain Algorithm 13.2.

The iterations of Algorithm 13.2 applied to Example 5.8 are listed in Table 13.4 and shown in Figure 13.4. The values of H_k at each iteration, as well as its two eigenvalues, are given in Table 13.3.

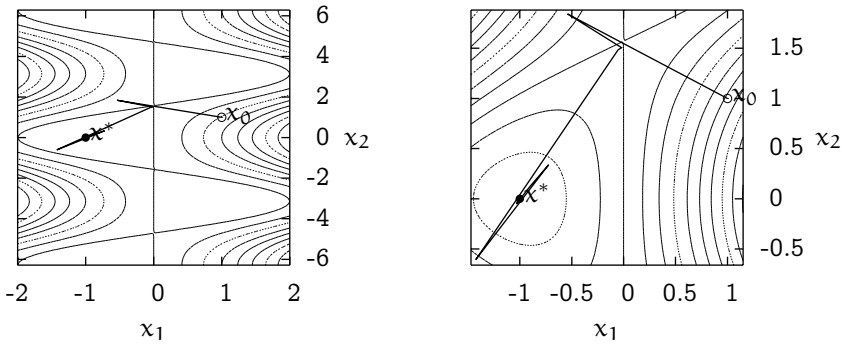
We notice that the matrix is not positive definite during some iterations (3, 5, 7, etc.).

Algorithm 13.2: Quasi-Newton SR1 method

Objective
  To find (an approximation of) a local minimum of the problem $\min_{x\in\mathbb{R}^n} f(x)$
Input
  The continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
  The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$
  An initial solution $x_0 \in \mathbb{R}^n$
  A first approximation of the symmetric Hessian $H_0 \in \mathbb{R}^{n\times n}$ (by default, $H_0 = I$)
  The radius of the first trust region Δ_0 (by default, $\Delta_0 = 10$)
  The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$
  The parameters $0 < \eta_1 \leq \eta_2 < 1$ (by default $\eta_1 = 0.01$ and $\eta_2 = 0.9$)
Output
  An approximation of the optimal solution $x^* \in \mathbb{R}$
Initialization
  $k := 0$
Repeat
  Calculate d_k by solving (approximately) the trust region subproblem by using the Steihaug-Toint truncated conjugate gradient method (Algorithm 12.3)
  $\rho := \dfrac{f(x_k) - f(x_k + d_k)}{m_{x_k}(x_k) - m_{x_k}(x_k + d_k)}$.
  if $\rho < \eta_1$ **then** failure
    $x_{k+1} := x_k$
    $\Delta_{k+1} := \frac{1}{2}\|d_k\|$
  else success
    $x_{k+1} := x_k + d_k$.
    if $\rho \geq \eta_2$ **then** very good
      $\Delta_{k+1} := 2\Delta_k$
    else just good
      $\Delta_{k+1} := \Delta_k$.
  $k := k + 1$
  Define $\bar{d}_{k-1} := x_k - x_{k-1}$ and $y_{k-1} := \nabla f(x_k) - \nabla f(x_{k-1})$.
  if $|\bar{d}_{k-1}(y_{k-1} - H_{k-1}d_{k-1})| \geq 10^{-8}\|\bar{d}_{k-1}\|\|y_{k-1} - H_{k-1}d_{k-1}\|$ **then** the denominator is non zero

$$H_k = H_{k-1} + \frac{(y_{k-1} - H_{k-1}\bar{d}_{k-1})(y_{k-1} - H_{k-1}\bar{d}_{k-1})^T}{\bar{d}_{k-1}^T(y_{k-1} - H_{k-1}\bar{d}_{k-1})},$$

  else
    $H_k = H_{k-1}$.
Until $\|\nabla f(x_k)\| \leq \varepsilon$
$x^* := x_k$



(a) Iterates (b) Zoom

Figure 13.4: Iterates from the quasi-Newton SR1 method for Example 5.8

13.3 The Rosenbrock problem

We come back to the Rosenbrock problem introduced in Sections 11.6 and 12.3. The quasi-Newton BFGS method (Figure 13.5) presents a behavior that is similar to Newton's method, with a few more iterations, but without the calculation of the second derivative. The SR1 method (Figure 13.6), on the other hand, encountered some more difficulties. The trust region is relatively small, which explains the large number of iterations (Figure 13.8). In this case, the method is often close to the steepest descent method. It is important to note that the SR1 method used with line search (Figure 13.7) is much more effective.

Clearly, these observations cannot be generalized. In particular, the trust region algorithm with SR1 can be effective for other problems. We encourage the reader to carry out the project of Section 13.5 for a more systematic analysis of the performances related to the algorithms.

13.4 Comments

We conclude this chapter with a few comments.

- The BFGS update can also be combined with a trust region algorithm. In this case, we cannot guarantee that the condition of Theorem 13.2 is satisfied. The technique that consists in not performing an update in this case can be ineffective. We refer the reader to Powell (1977), and Nocedal and Wright (1999, page 540) for a description of an alternative update.

Table 13.3: Secant approximation of the SR1 method (Algorithm 13.2) for Example 5.8

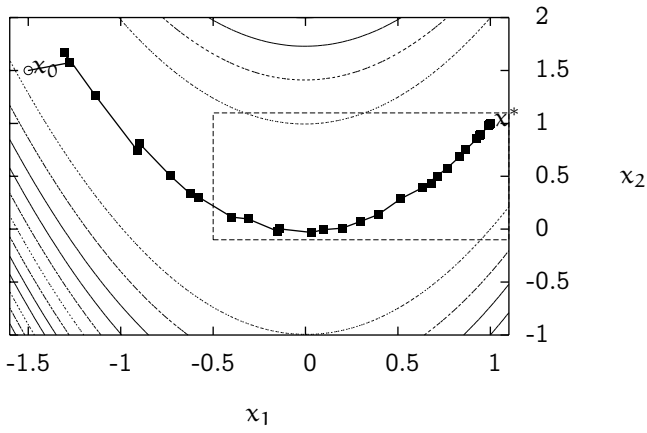
k	$(H_k)_{1,1}$	$(H_k)_{2,2}$	$(H_k)_{1,2}$	λ_1	λ_2
1	+1.38780e+00	+1.16113e+00	-2.49977e-01	+1.00000e+00	+1.54894e+00
2	+1.50852e+00	+1.18087e+00	-2.01166e-01	+1.08526e+00	+1.60413e+00
3	+9.92963e-01	-1.42553e-02	-9.86122e-01	-6.17922e-01	+1.59663e+00
4	+1.56592e+00	+5.95949e-01	-3.94837e-01	+4.55548e-01	+1.70632e+00
5	+9.81728e-01	+8.85544e-03	-9.80476e-01	-5.99219e-01	+1.58980e+00
6	+1.33046e+00	+1.28506e+00	-3.13357e-01	+9.93580e-01	+1.62194e+00
7	+9.95038e-01	+3.68905e-02	-9.60395e-01	-5.57287e-01	+1.58922e+00
8	+1.87015e+00	+3.76238e-01	-4.15446e-01	+2.68479e-01	+1.97791e+00
9	+1.07848e+00	+2.20504e-01	-7.66576e-01	-2.28956e-01	+1.52794e+00
10	+1.86165e+00	+2.21270e-01	-7.42082e-01	-6.46135e-02	+2.14753e+00
11	+5.29518e-01	-4.83122e-01	+2.26599e-01	-5.31516e-01	+5.77911e-01
12	+5.29604e-01	-2.67251e-03	+2.20180e-01	-8.19450e-02	+6.08876e-01
13	+5.56284e-01	+1.50541e-01	+1.56244e-01	+9.73485e-02	+6.09477e-01
14	+5.85590e-01	+3.94032e-01	+2.40717e-01	+2.30739e-01	+7.48883e-01
15	+8.04073e-01	+1.08584e+00	-1.48061e-01	+7.40579e-01	+1.14934e+00
16	+1.30324e+00	+1.09341e+00	-8.66119e-02	+1.06228e+00	+1.33437e+00
17	+1.00670e+00	+1.08398e+00	-3.37434e-02	+9.94037e-01	+1.09664e+00
18	+9.99334e-01	+9.54417e-01	-2.85711e-03	+9.54236e-01	+9.99515e-01
19	+9.99555e-01	+9.99873e-01	+3.11648e-04	+9.99364e-01	+1.00006e+00
20	+1.00066e+00	+1.00000e+00	-6.64095e-05	+9.99995e-01	+1.00067e+00

Table 13.4: Quasi-Newton SR1 method for the minimization of Example 5.8 ($\Delta_0 = 10$)

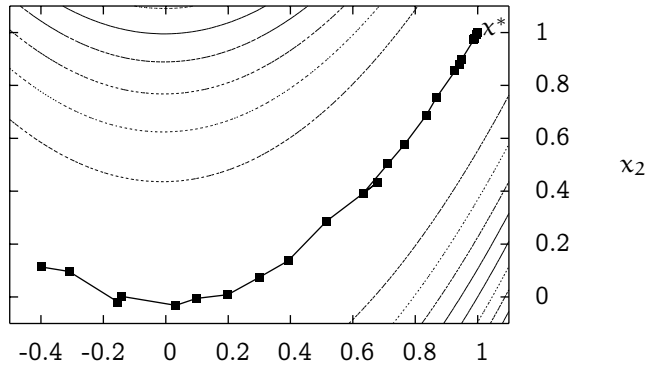
k	x_k		$f(x_k)$	$\|\nabla f(x_k)\|$	Δ_k	
0	+1.00000e+00	+1.00000e+00	+1.04030e+00	+1.75517e+00	+1.00000e+01	
1	-5.40302e-01	+1.84147e+00	+2.90430e-01	+9.60942e-01	+1.00000e+01	+
2	-1.88588e-02	+1.50535e+00	-1.05553e-03	+5.02012e-02	+2.00000e+01	++
3	-5.26023e-02	+1.48367e+00	-3.19396e-03	+6.26948e-02	+4.00000e+01	++
4	-5.26023e-02	+1.48367e+00	-3.19396e-03	+6.26948e-02	+2.00000e+01	-
5	-1.05608e-01	+1.36062e+00	-1.64571e-02	+1.45885e-01	+4.00000e+01	++
6	-1.05608e-01	+1.36062e+00	-1.64571e-02	+1.45885e-01	+2.00000e+01	-
7	-2.07848e-01	+1.25531e+00	-4.28899e-02	+2.22561e-01	+4.00000e+01	++
8	-2.07848e-01	+1.25531e+00	-4.28899e-02	+2.22561e-01	+2.00000e+01	-
9	-4.35002e-01	+4.79315e-01	-2.91369e-01	+4.94801e-01	+4.00000e+01	++
10	-4.35002e-01	+4.79315e-01	-2.91369e-01	+4.94801e-01	+2.00000e+01	-
11	-4.35002e-01	+4.79315e-01	-2.91369e-01	+4.94801e-01	+1.00000e+01	-
12	-4.35002e-01	+4.79315e-01	-2.91369e-01	+4.94801e-01	+5.00000e+00	-
13	-4.35002e-01	+4.79315e-01	-2.91369e-01	+4.94801e-01	+2.50000e+00	-
14	-1.05435e+00	-2.10470e-01	-4.75256e-01	+2.33154e-01	+2.50000e+00	+
15	-1.05435e+00	-2.10470e-01	-4.75256e-01	+2.33154e-01	+3.26811e-01	-
16	-9.18547e-01	+1.09084e-02	-4.96628e-01	+8.20075e-02	+3.26811e-01	+
17	-9.81944e-01	-3.27718e-03	-4.99832e-01	+1.83348e-02	+6.53623e-01	++
18	-9.99794e-01	-8.64122e-04	-5.00000e-01	+8.88152e-04	+1.30725e+00	++
19	-9.99997e-01	+4.04748e-05	-5.00000e-01	+4.05730e-05	+2.61449e+00	++
20	-1.00000e+00	-3.34769e-09	-5.00000e-01	+3.39719e-09	+5.22898e+00	++

- The SR1 update can also be combined with a line search algorithm. In this case, the matrix is not necessarily positive definite, and a modified Cholesky factorization is necessary, just as for Newton's method with line search (Algorithm 11.8).

- Other update formulas have been proposed, including that of Davidon, studied by Fletcher and Powell. It is called DFP, after the initials of the names of the three researchers. The subject is vast and, in this book, we have chosen to include only the two formulas that seem the most effective in practice.

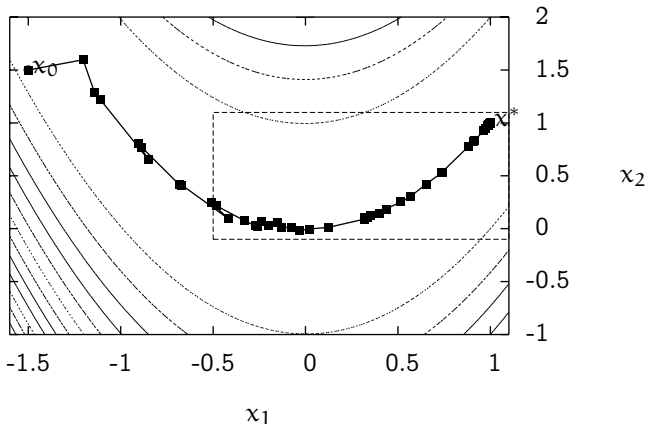


(a) 34 iterations

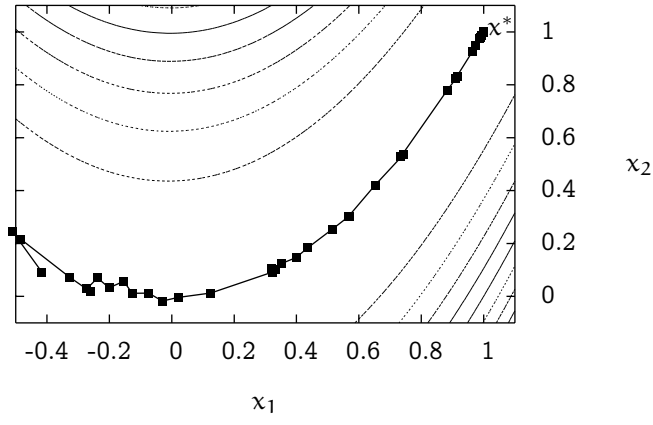


(b) Zoom

Figure 13.5: BFGS method with line search

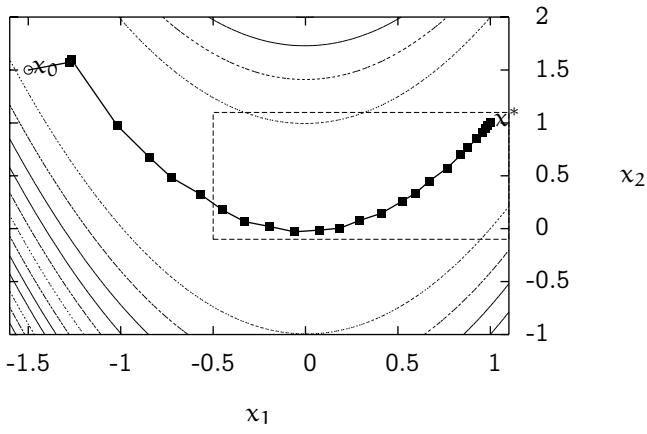


(a) 97 iterations

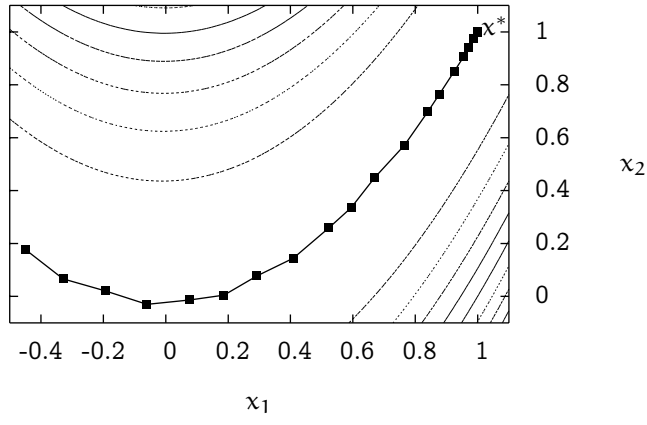


(b) Zoom

Figure 13.6: SR1 method with trust region



(a) 28 iterations



(b) Zoom

Figure 13.7: SR1 method with line search

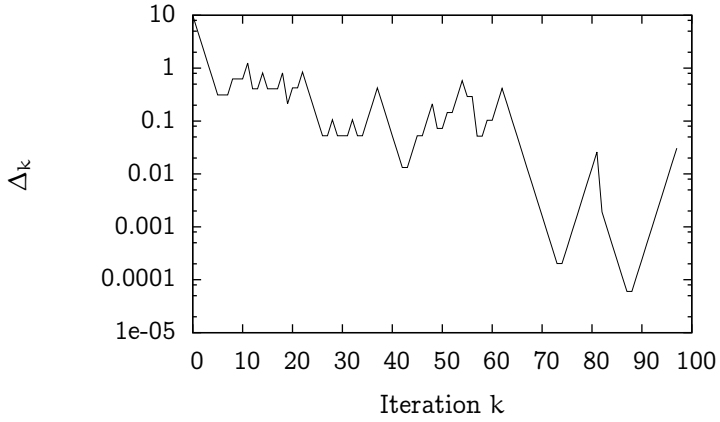


Figure 13.8: Evolution of the radius of the trust region for SR1

13.5 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of the present project is to analyze the behavior of quasi-Newton methods and compare them to descent methods and trust region methods.

Approach

Implement the BFGS method (Algorithm 13.1) and the SR1 method (Algorithm 13.2). Also implement a version of BFGS with trust region and a version of SR1 with line search, by taking into account the comments in Section 13.4.

Compare these algorithms with the descent methods and the trust region methods. Analyze the results by means of the method described in Section D.2. Choose as performance index on the one hand the resolution time and on the other hand the number $\#f + n\#g$, i.e., the number of evaluations of the function plus n times the number of gradient evaluations.

Algorithms

Algorithms 13.1 and 13.2.

Problems

Exercise 13.1. The James Bond problem, described in Section 1.1.5.

Exercise 13.2. The problem

$$\min_{x\in\mathbb{R}^2} 2x_1x_2\, e^{-(4x_1^2+x_2)/8}\,.$$

Advice: draw the function and the level curves with a software such as Gnuplot, visually identify the stationary points, and then choose the starting points, either close to or far from the stationary points.

Exercise 13.3. The problem

$$\min_{x\in\mathbb{R}^n} \sum_{i=1}^{n} i\alpha x_i^2\,, \qquad \bar{x} = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T,$$

with different values of n and of α.

Exercise 13.4. The problem

$$\min_{x\in\mathbb{R}^2} 3x_1^2 + x_2^4\,.$$

Recommended starting point: $\begin{pmatrix} 1 & -2 \end{pmatrix}^T$.

Exercise 13.5. The Rosenbrock problem

$$\min_{x\in\mathbb{R}^2} 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 \qquad \text{(Section 11.6)}\,.$$

Recommended starting points: $\begin{pmatrix} 1.2 & 1.2 \end{pmatrix}^T$ and $\begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$.

Exercise 13.6. The problem

$$\min_{x\in\mathbb{R}^6} \sum_{i=1}^{m} \big(-e^{-0.1\,i} + 5\,e^{-i} - 3\,e^{-0.4\,i} + x_3\,e^{-0.1\,i x_1} - x_4\,e^{-0.1\,i x_2} + x_6\,e^{-0.1\,i x_5}\big)^2\,,$$

Recommended starting point: $\begin{pmatrix} 1 & 2 & 1 & 1 & 4 & 3 \end{pmatrix}^T$.

Chapter 14

Least squares problem

Contents

Least squares problems are optimization problems expressed in the form

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \left\| g(x) \right\|^2 = \frac{1}{2} g(x)^T g(x) = \frac{1}{2} \sum_{i=1}^{m} g_i(x)^2 , \tag{14.1}$$

where $g : \mathbb{R}^n \to \mathbb{R}^m$ is a differentiable function. In particular, these problems arise when one wants to calibrate the parameters of a mathematical model by using data.

Example 14.1 (Unemployement in Switzerland)**.** We are interested in analyzing the relationship between the number of men and the number of women unemployed in Switzerland. Table 14.1 reports the number of unemployed people from January 2012 to December 2013 (source: Swiss State Secretariat for Economic Affairs, www.amstat.ch). It is postulated that, at any point in time, the number β of unemployed women is related to the number α of unemployed men in the following way:

$$\beta = m(\alpha; x_1, x_2) = x_1 \alpha + x_2, \tag{14.2}$$

where x_1 and x_2 are unknown parameters to be determined.[1] Data in Table 14.1 reports for each month t the pair (α_t, β_t) of observed number of men and women who were unemployed at time t. Note that it is impossible to find parameters x_1

[1] In statistics, the quantity α is called an *explanatory* or *independent* variable, and the quantity β is the *dependent variable*. Note that in the statistics literature, it is common to denote by y the dependent variable, by x the independent variable, and by Greek letters the parameters. However, in this textbook, we focus on the optimization problem that identifies the values of the parameters such that the model fits the data well. To be consistent with the rest of the book, we use x to denote the variables of this optimization problem, which are the unknown parameters of the model.

and x_2 such that (14.2) is verified for each data point. The model (14.2) is only an approximation of the real relationship between the two quantities. Moreover, errors are always presents in any measurement. Therefore, it is assumed that the model is

$$\beta = m(\alpha; x_1, x_2) = x_1\alpha + x_2 + \varepsilon, \tag{14.3}$$

where ε is a random variable.

Table 14.1: Unemployment data in Switzerland

	Men	Women
January 2012	77,005	57,312
February 2012	76,315	56,839
March 2012	70,891	55,501
April 2012	67,667	55,491
May 2012	64,643	54,217
June 2012	61,770	53,098
July 2012	61,593	54,701
August 2012	63,227	56,596
September 2012	63,684	56,663
October 2012	66,914	58,622
November 2012	72,407	59,660
December 2012	82,413	59,896
January 2013	86,515	61,643
February 2013	84,896	61,105
March 2013	79,660	59,333
April 2013	75,827	60,024
May 2013	72,606	58,684
June 2013	69,423	57,075
July 2013	69,690	58,826
August 2013	69,744	60,212
September 2013	70,418	60,654
October 2013	71,998	61,445
November 2013	77,268	61,805
December 2013	87,299	62,138

The least squares estimation of the unknown parameters is performed by solving the following optimization problem:

$$\min_{x_1, x_2} \sum_{t=1}^{24} (m(\alpha_t; x_1, x_2) - \beta_t)^2 = \sum_{t=1}^{24} (x_1\alpha_t + x_2 - \beta_t)^2. \tag{14.4}$$

The optimal solution is $x_1^* = 0.253$ and $x_2^* = 39{,}982$. The fitted model and the data are presented in Figure 14.1.

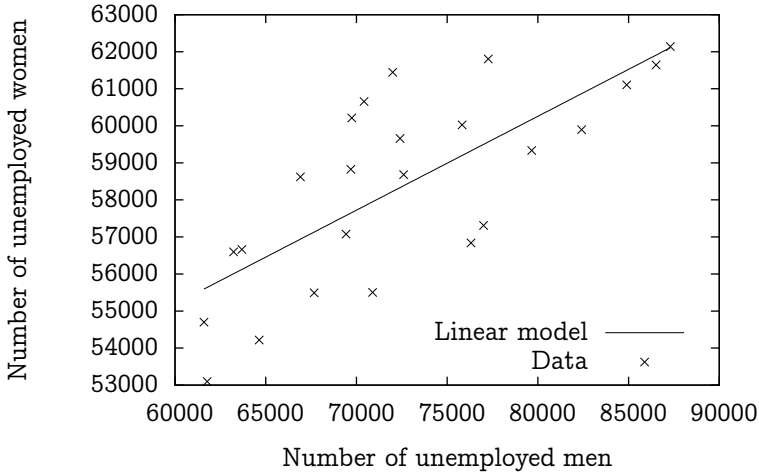


Figure 14.1: Unemployment in Switzerland: data and fitted linear model

To generalize the formulation of Example 14.1, let us consider a system where each configuration i defined by the input values α_i produces the output values β_i. We have at our disposal a mathematical model enabling to predict the output values based on the input values, i.e.,

$$\beta_i + \varepsilon_i = m(\alpha_i; x)\,, \tag{14.5}$$

where x represents the parameters of the model and ε_i is a random variable with mean zero and variance σ^2 corresponding to the modeling and measurement errors. Formally, the least squares problem can be derived from the theory of maximum likelihood estimation, assuming that the error terms ε_i are normally distributed. Here, we prefer to interpret it using an optimization argument, that is as the identification of the parameters x that minimize the error, i.e.,

$$\min_{x,\varepsilon} \sum_i \varepsilon_i^2$$

with constraint

$$\beta_i + \varepsilon_i = m(\alpha_i; x)\,, \qquad i = 1, \dots,$$

or, by using the constraint to eliminate ε,

$$\min_x \sum_i \big(m(\alpha_i; x) - \beta_i\big)^2\,, \tag{14.6}$$

which is a least squares problem, with $g_i(x) = m(\alpha_i; x) - \beta_i$. In the context of Example 14.1, the mathematical model is linear. Example 14.2 presents a more complex model.

Example 14.2 (Neural networks)**.** The training of neural networks can also be viewed as a least squares problem. We here provide a brief introduction to the concept of neural networks. The reader is referred to the abundant literature on the subject for more details such as Haykin (2008) or Iovine (2012). A neural network, in the mathematical sense of the term, is inspired by its biological analog. The idea is to develop a complex entity by connecting a large number of single units in a network, each performing a limited task, using information provided by the other units.

Consider a network organized in N layers of neurons. A neuron j in layer k uses the information provided by the neurons in layer $k-1$, process it and produces a result $v_{j,k}$:

$$v_{j,k} = \phi\left(\left(x_{jk}\right)_0 + \sum_{i=1}^{n_{k-1}} \left(x_{jk}\right)_i v_{i,k-1}\right), \tag{14.7}$$

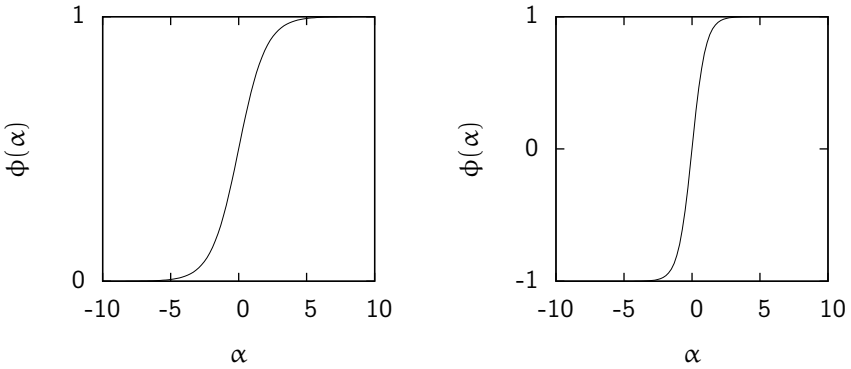
where n_{k-1} is the number of neurons in layer $k-1$, x_{jk} is a vector of $1+n_{k-1}$ parameters weighting the importance of the information received by the different neurons from the previous layer, and $\phi : \mathbb{R} \to \mathbb{R}$ is a continuously differentiable function, called an *activation function*. Two typical examples of activation functions are the sigmoidal function

$$\phi(\alpha) = \frac{1}{1+e^{-\alpha}} \tag{14.8}$$

and the hyperbolic tangent function

$$\phi(\alpha) = \frac{e^{\alpha}-e^{-\alpha}}{e^{\alpha}+e^{-\alpha}}\,. \tag{14.9}$$

Note that these functions are continuous approximations of "switches," that is, of a function with discrete output: 0 and 1 for (14.8) (see Figure 14.2(a)) and -1 and 1 for (14.9) (see Figure 14.2(b)).



(a) Sigmoid (b) Hyperbolic tangent

Figure 14.2: Examples of activation functions

The first layer uses information from the exterior of the system, denoted by $v_{j,0}$, $j = 1, \ldots, n_0$. The neural network can be seen as a model (14.5), where the input parameters α_i correspond to the information $v_{j,0}$, $j = 1, \ldots, n_0$, the output parameters β_i correspond to the information produced by the last layer, i.e., $v_{j,N}$, $j = 1, \ldots, n_N$, and the unknown parameters of the model are all the weightings x_{jk}, $j = 1, \ldots, n_{k-1}$, $k = 1, \ldots, N$. The calibration of the weights x_{jk} is sometimes called the *training phase* of the neural network.

It is important to note that this least squares problem is significantly more complex to solve than that of Example 14.1. Bertsekas (1999) uses the following example to illustrate that complexity:

$$\min_{x_0, x_1} \frac{1}{2} \sum_{i=1}^{5} \left(\beta_i - \phi(x_1 \alpha_i + x_2)\right)^2,$$

where ϕ is the hyperbolic tangent function (14.9). The five pairs (α_i, β_i) are listed in Table 14.2. The resulting objective function is shown in Figure 14.3.

Table 14.2: Data for the example on a neural network

α_i	β_i
1.165	1
0.626	−1
0.075	−1
0.351	1
-0.696	1

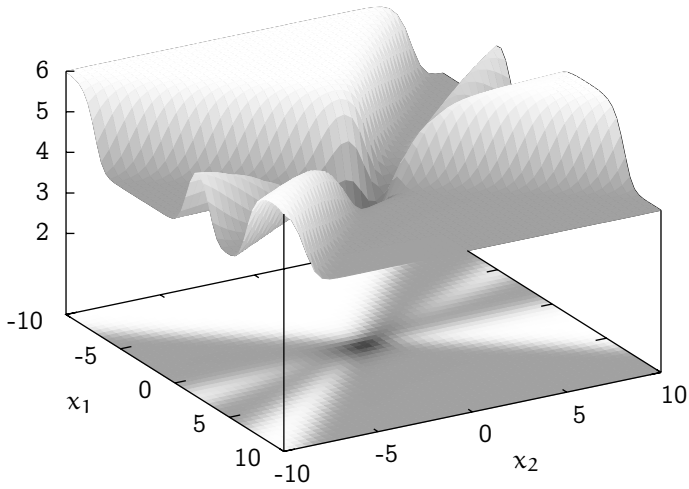


Figure 14.3: Example of an objective function for the training of a neural network

Now that we have defined and illustrated the problem, we present an algorithm that exploits its specific structure.

14.1 The Gauss-Newton method

We consider applying Newton's method to the least squares problem (14.1). And we focus on the local version first (Chapter 10). To do so, one must calculate the gradient and the Hessian matrix of the problem.

We derive

$$f(x) = \frac{1}{2} g(x)^T g(x)$$

to obtain

$$\nabla f(x) = \nabla g(x) g(x) = \sum_{i=1}^{m} \nabla g_i(x) g_i(x),$$

where $\nabla g(x) \in \mathbb{R}^{n \times m}$ is the gradient matrix of g (Definition 2.17) and $\nabla g_i(x) \in \mathbb{R}^n$ is the gradient of g_i. By differentiating again, we get

$$\begin{aligned} \nabla^2 f(x) &= \sum_{i=1}^{m} \left(\nabla g_i(x) \nabla g_i(x)^T + \nabla^2 g_i(x) g_i(x) \right) \\ &= \nabla g(x) \nabla g(x)^T + \sum_{i=1}^{m} \nabla^2 g_i(x) g_i(x). \end{aligned}$$

The second term used for calculating the Hessian matrix is generally computationally demanding in practice, since it involves the second derivatives of the functions g_i. It is therefore wise to ignore it and approximate the Hessian matrix using only the first term, i.e., $\nabla g(x) \nabla g(x)^T$. Note that this matrix is always positive semidefinite. We obtain the *Gauss-Newton* method presented as Algorithm 14.1, that uses only the first derivatives of g, not the second.

Note that, if $\nabla g(x_k) \nabla g(x_k)^T$ is non singular, it is positive definite and we have a descent method. If this is not the case, the technique presented in Section 11.5, based on a modified Cholesky factorization (Algorithm 11.7) is appropriate. The name *Levenberg-Marquardt* is often used in this context, in reference to the work of Levenberg (1944) and Marquardt (1963). Moreover, the method is locally convergent, i.e., it converges if x_0 is not too far from x^*. The adaptations that enable the global convergence of Newton's local method, namely linear search (Algorithm 11.8) and trust region (Algorithm 12.4), can of course be used to render the Gauss-Newton method globally convergent.

In the case where the function g is linear, i.e.,

$$g(x) = Ax - b$$

with $A \in \mathbb{R}^{m \times n}$, we have $\nabla g(x) = A^T$ and the Gauss-Newton equation (14.10) is written as

$$A^T A d_{k+1} = -A^T (A x_k - b).$$

Since $d_{k+1} = x_{k+1} - x_k$, we obtain

$$A^T A x_{k+1} = A^T b,$$

and this, regardless of x_k. This system of equations is called the system of *normal equations* of the linear least squares problem.

Algorithm 14.1: Gauss-Newton method

Objective

To find (an approximation of) a local optimum of the least squares problem $\min_{x\in\mathbb{R}^n} f(x) = \frac{1}{2}\, g(x)^T g(x)$.

Input

The differentiable function $g : \mathbb{R}^n \to \mathbb{R}^m$.

The gradient matrix of g: $\nabla g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$.

An initial solution $x_0 \in \mathbb{R}^n$.

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}^n$.

Initialization

$k := 0$.

Repeat

Calculate d_{k+1} that solves

$$\nabla g(x_k)\nabla g(x_k)^T d_{k+1} = -\nabla g(x_k) g(x_k). \tag{14.10}$$

$x_{k+1} := x_k + d_{k+1}$

$k = k + 1$

Until $\left\|\nabla g(x_k) g(x_k)\right\| \leq \varepsilon$

$x^* := x_k$

Definition 14.3 (Normal equations)**.** Consider the linear least squares problem

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\, \|Ax - b\|_2^2\,, \tag{14.11}$$

with $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. The equations of the system

$$A^T A x = A^T b \tag{14.12}$$

are called the *normal equations* of (14.11).

Theorem 14.4 (Normal equations)**.** *Consider $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. x^* solves the normal equations (14.12) if and only if x^* is an optimal solution to the linear least square problem (14.11). Moreover, if A is of full rank, x^* is the unique optimal solution to (14.11).*

Proof. The objective function of (14.11) is

$$f(x) = \frac{1}{2}(Ax - b)^T(Ax - b)$$

and

$$\nabla f(x) = A^T Ax - A^T b\,,$$

and

$$\nabla^2 f(x) = A^T A\,.$$

The normal equations are therefore equivalent to the first order optimality conditions $\nabla f(x^*) = 0$, and are necessary (Theorem 5.1). The eigenvalues of $A^T A$ are the singular values of A squared. Therefore, $\nabla^2 f(x)$ is positive semidefinite for all x and f is convex. Moreover, if A is of full rank, $\nabla^2 f(x)$ is positive definite and f is strictly convex. The sufficient global optimality conditions are therefore verified, and Theorem 5.9 applies to prove the result. □

From this result, it appears that the Gauss-Newton method identifies the optimal solution in a single iteration when the function g is linear. We now give another interpretation of the method that supports this observation.

To do this, we use a simple model of the function at each iteration. We already used this idea when presenting Newton's local method, where a quadratic model was used (Algorithm 10.2). Here, we model g and not f, and we utilize a linear model (Definition 7.9).

Consider the least squares problem (14.1) and an iterate x_k. Replace $g(x)$ by the linear model in x_k

$$m_{x_k}(x) = g(x_k) + \nabla g(x_k)^T(x - x_k)\,.$$

The problem then becomes

$$\min_x M(x) = \min_x \frac{1}{2}\left\|m_{x_k}(x)\right\|_2^2 = \min_x \frac{1}{2}\, m_{x_k}(x)^T m_{x_k}(x)\,.$$

We have

$$\begin{aligned} M(x) = \frac{1}{2}\, m_{x_k}(x)^T m_{x_k}(x) = \frac{1}{2}\, g(x_k)^T g(x_k) \\ + (x - x_k)^T \nabla g(x_k) g(x_k) \\ + \frac{1}{2}\,(x - x_k)^T \nabla g(x_k) \nabla g(x_k)^T (x - x_k)\,. \end{aligned}$$

The gradient of this expression is

$$\nabla M(x) = \nabla g(x_k) \nabla g(x_k)^T (x - x_k) + \nabla g(x_k) g(x_k)\,.$$

It is zero for

$$x_{k+1} = x_k - \left(\nabla g(x_k) \nabla g(x_k)^T\right)^{-1} \nabla g(x_k) g(x_k)\,,$$

if $\nabla g(x_k) \nabla g(x_k)^T$ is invertible. This equation is equivalent to (14.10). Then, an iteration of the Gauss-Newton method is a matter of linearizing the function g around x_k and solving the least squares problem thus obtained. It is therefore evident that, if g is already linear, the method converges in one iteration (if $\nabla g(x_k) \nabla g(x_k)^T = A^T A$ is invertible).

14.2 Kalman filter

We now consider the linear least squares problem when the data is organized into blocks. There are several motivations for this.

Nowadays, more and more data is available. In the era of "big data," the size of problem (14.11) can be huge. More precisely, m may be much larger than the number of parameters n. As a consequence, it may happen that the matrix A cannot be stored as such in memory, and has to be split into blocks just to be handled.

Another context is when data is collected by different sources. In this case, it is convenient not to wait until data is available from all sources to start solving the problem. Suppose that a couple of sources have not reported yet, and that the problem is solved with the available data. When the last pieces of data are finally made available, it is desirable to update the previous solution, rather than solving the whole problem from the beginning. In this context, each block of data would correspond to one of the sources.

The situation is similar when data is available over time, such as in real-time applications. In this case, each time period corresponds to a block of data. And, again, it is desirable to calculate a solution with the available data, and update it when more data is made available.

The matrices involved in the least squares problem (14.11) are represented in Figure 14.4. The matrix A and the vector b are sliced into J blocks, possibly of different sizes $m_1, \dots, m_J$.

We want to obtain a first approximation of the calibrated parameters with the first block of data and subsequently update it as other blocks become available. We write the least squares problem

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$$

as

$$\min_{x \in \mathbb{R}^n} \sum_{j=1}^{J} \left\|A_j x - b_j\right\|_2^2,$$

where $A_j \in \mathbb{R}^{m_j \times n}$ and $b_j \in \mathbb{R}^{m_j}$. The optimal solution to this problem is generated incrementally, starting with the optimal solution to the subproblem corresponding to the first block of data:

$$\min_{x \in \mathbb{R}^n} \left\|A_1 x - b_1\right\|_2^2.$$

We call $x_1 \in \mathbb{R}^n$ the solution of this problem. Assume that the matrix A_1 is of full rank. In this case, according to Theorem 14.4, x_1 is given by the normal equations

$$A_1^T A_1 x_1 = A_1^T b_1. \tag{14.13}$$

Since A_1 is of full rank, $A_1^T A_1$ is invertible and

$$x_1 = \left(A_1^T A_1\right)^{-1} A_1^T b_1.$$

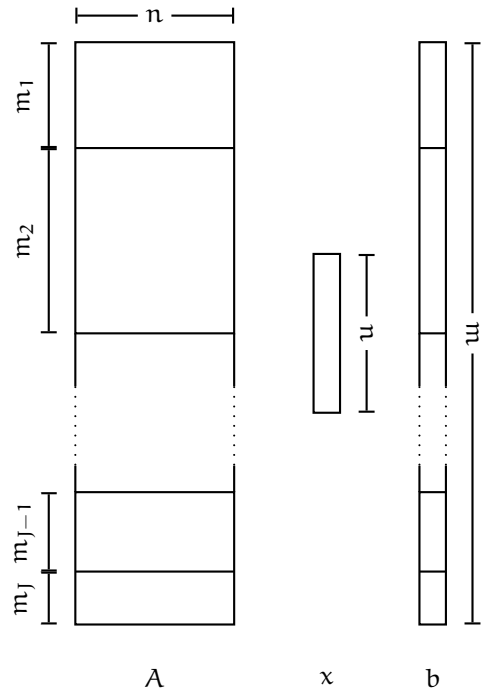


Figure 14.4: Structures of the matrix and vectors involved in the least squares problem organized by blocks

Consider now the problem composed of the first two blocks:

$$\min_{x\in\mathbb{R}^n} \|A_1x - b_1\|_2^2 + \|A_2x - b_2\|_2^2,$$

that we can also write

$$\min_{x\in\mathbb{R}^n} \left\| \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} x - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|_2^2.$$

We denote $x_2 \in \mathbb{R}^n$ the optimal solution of this problem. It verifies the normal equations

$$(A_1^TA_1 + A_2^TA_2)x_2 = A_1^Tb_1 + A_2^Tb_2\,.$$

From (14.13), we obtain

$$\begin{aligned}(A_1^TA_1 + A_2^TA_2)x_2 &= A_1^Tb_1 + A_2^Tb_2\\ &= A_1^TA_1x_1 + A_2^Tb_2\\ &= A_1^TA_1x_1 + A_2^TA_2x_1 - A_2^TA_2x_1 + A_2^Tb_2\\ &= (A_1^TA_1 + A_2^TA_2)x_1 + A_2^T(b_2 - A_2x_1)\,.\end{aligned}$$

Since $A_1^TA_1$ is invertible, $A_1^TA_1 + A_2^TA_2$ is too and we get

$$x_2 = x_1 + (A_1^TA_1 + A_2^TA_2)^{-1}A_2^T(b_2 - A_2x_1)\,.$$

The same technique is used for the following blocks:

$$x_3 = x_2 + (A_1^T A_1 + A_2^T A_2 + A_3^T A_3)^{-1} A_3^T (b_3 - A_3 x_2),$$
$$x_4 = x_3 + (A_1^T A_1 + A_2^T A_2 + A_3^T A_3 + A_4^T A_4)^{-1} A_4^T (b_4 - A_4 x_3),$$

and so on. For the sake of completeness, we can also obtain an incremental form for the first block. Indeed, for any $x_0 \in \mathbb{R}^n$,

$$\begin{aligned} x_1 &= x_0 - x_0 + \left(A_1^T A_1\right)^{-1} A_1^T b_1 \\ &= x_0 - \left(A_1^T A_1\right)^{-1} A_1^T A_1 x_0 + \left(A_1^T A_1\right)^{-1} A_1^T b_1 \\ &= x_0 + \left(A_1^T A_1\right)^{-1} A_1^T (b_1 - A_1 x_0)\,. \end{aligned}$$

If we write $H_j = \sum_{k=1}^{j} A_k^T A_k$, we obtain the formula:

$$x_j = x_{j-1} + H_j^{-1} A_j^T (b_j - A_j x_{j-1}), \tag{14.14}$$

where

$$H_j = H_{j-1} + A_j^T A_j. \tag{14.15}$$

Note that only data from block j, that is A_j and b_j, is used here, irrespectively of the number of blocks already treated. This is the Kalman filter algorithm, presented as Algorithm 14.2.

Algorithm 14.2: The Kalman filter method

Objective

To find, in an incremental manner, the optimal solution x^* to a linear least squares problem

$$\min_{x \in \mathbb{R}^n} \sum_{j=1}^{J} \left\| A_j x - b_j \right\|_2^2. \tag{14.16}$$

Input

The matrices $A_j \in \mathbb{R}^{m_j \times n}$, $j = 1, \ldots, J$.

The vectors $b_j \in \mathbb{R}^{m_j}$, $j = 1, \ldots, J$.

An initial solution $x_0 \in \mathbb{R}^n$ (default: $x_0 = 0$).

An initial matrix $H_0 \in \mathbb{R}^{n \times n}$ (default: $H_0 = 0$).

Output

The optimal solution $x^* \in \mathbb{R}^n$.

Initialization

$j := 0$.

Repeat

$j := j + 1$

$H_j := H_{j-1} + A_j^T A_j$

Calculate d_j solving $H_j d_j = A_j^T (b_j - A_j x_{j-1})$

$x_j := x_{j-1} + d_j$

Until $j = J$

$x^* = x_J$

Example 14.5 (Kalman filter)**.** We consider again the unemployment data of Example 14.1 (Table 14.1). At the end of 2012, a model is calibrated using the 2012 data. Then, as a new piece of data arrives every month, the estimated value of the parameters is updated with the new information. The estimated parameters at each stage of this process are reported in Table 14.3.

Block	x_1	x_2
1	0.210758	41998.1
2	0.23963	40074.4
3	0.249038	39451.5
4	0.249355	39431.6
5	0.255092	39122.6
6	0.255453	39157.6
7	0.254998	39200.6
8	0.251087	39579.7
9	0.24501	40172.3
10	0.24095	40617.4
11	0.241631	40726.7
12	0.253012	40011.7
13	0.253436	39982.5

Table 14.3: Kalman filter: estimated parameters of Example 14.5

Figure 14.5 compares the model estimated on data from 2012 only with the model estimated on the entire data set, 2012 and 2013.

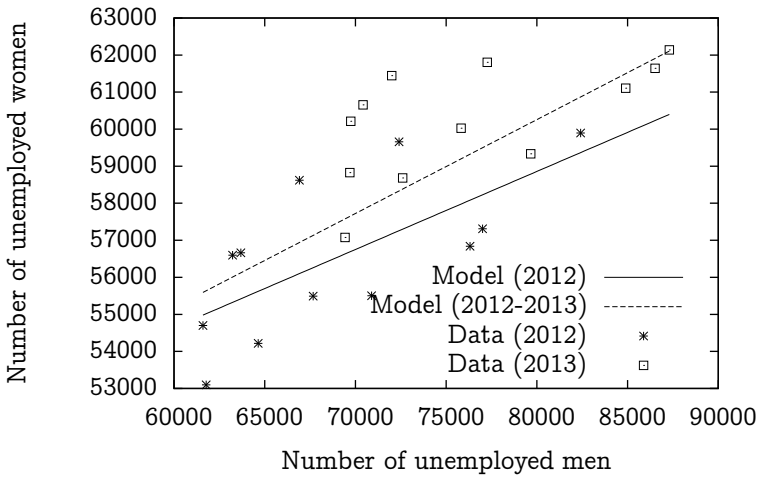


Figure 14.5: Unemployment in Switzerland: data and fitted linear models

The Kalman filter method only works if the matrix defining the first block is of full rank. To achieve this, it usually suffices to put enough data in the first block, in order for a first model to be estimated. Indeed, if n parameters have to be estimated, at least n pieces of data are necessary, and even more if there is colinearity. Where this is not possible, one must define H_0 such that the matrix $H_1 = H_0 + A_1^T A_1$ is invertible. Typically, we choose H_0 a multiple of the identity. Using this technique naturally affects the result. The thus-perturbed Kalman filter only gives an approximation of the optimal solution. However, if the number of blocks J is large, the impact of the arbitrary matrix H_0 should be sufficiently small for all practical purposes.

Kalman filter in real time

The Kalman filter is particularly well adapted for real-time calibration models, i.e., when the value of the estimated parameters must be updated as new data becomes available. Suppose, for example, that we are interested in calibrating the parameters of a model for vehicular traffic on a highway, using measurements of the flow of cars on each lane of the highway. Data arrives every minute, say, and the Kalman filter method is used to update the estimation of the parameters. During the afternoon, the data collected during the morning peak hour is less relevant to reflect the current state of the traffic, compared to the data collected 5 minutes ago. In this case, it is necessary to give greater weight to recent data as opposed to old data. Assume that the data becomes available in regular intervals and that the pair (A_j, b_j) corresponds to data from the time interval j. We introduce a discount parameter λ, with $0 < \lambda \leq 1$, that represents the relative importance of the data from one time interval j compared to the previous interval $j-1$. When time moves forward, we multiply the weight of all past data sets by λ. Consequently, the weight associated with the data from two intervals ago is λ^2, and the weight associated with data from interval $i \leq j$ is λ^{j-i}. The least squares problem that we have to solve is therefore

$$\min_{x \in \mathbb{R}^n} \sum_{j=1}^{J} \lambda^{J-j} \left\| A_j x - b_j \right\|_2^2$$

where J is the current time interval. We apply the Kalman filter update in this context, and obtain Algorithm 14.3. Note that the whole memory of the process is accumulated in the matrix H_{J-1} and the vector x_{J-1}, irrespectively of the number of past iterations.

14.3 Orthogonal regression

When estimating the parameters of a mathematical model (14.5) by using the least squares method, we assume that the observations β_i (in statistics called *dependent variables*) are subject to random errors, while the observations α_i (the *independent variables*) are accurately known.

Algorithm 14.3: The Kalman filter method in real time

Objective

To update the parameters of a linear model as the new data becomes available. At each time interval J, the previous solution of the time interval $J-1$ is updated to obtain the solution of the problem

$$\min_{x\in\mathbb{R}^n} \sum_{j=1}^{J} \lambda^{J-j} \left\| A_j x - b_j \right\|_2^2 . \tag{14.17}$$

Input

The matrix $A_J \in \mathbb{R}^{m_J \times n}$.
The vector $b_J \in \mathbb{R}^{m_J}$.
The previous solution $x_{J-1} \in \mathbb{R}^n$.
The previous matrix $H_{J-1} \in \mathbb{R}^{n\times n}$.
A discount factor λ such that $0 < \lambda \leq 1$.

Output

x_J and H_J

Update

$H_J := \lambda H_{J-1} + A_J^T A_J$
Calculate d_J solving $H_J d_J = A_J^T (b_J - A_J x_{J-1})$
$x_J := x_{J-1} + d_J$

The hypothesis concerning the accuracy of α_i is often too strong. In Example 14.1, there is no reason to assume that there are errors in the measurement of female unemployment and not in the measurement of male unemployment.

Assume now that both α_i and β_i are subject to measurement errors. The model is now written as

$$\beta_i + \varepsilon_i = m(\alpha_i + \xi_i; x) , \tag{14.18}$$

where ε_i and ξ_i are random variables assumed to be independent and with mean zero. In this case, the least squares problem amounts to finding the value of the parameters that make the errors as small as possible, that is

$$\min_{x,\varepsilon,\xi} \sum_i (\varepsilon_i^2 + \xi_i^2)$$

with the constraint

$$\beta_i + \varepsilon_i = m(\alpha_i + \xi_i; x) , \qquad i = 1, \dots ,$$

or, by using the constraint to eliminate ε,

$$\min_{x,\xi} \sum_i \left(\left(m(\alpha_i + \xi_i; x) - \beta_i \right)^2 + \xi_i^2 \right) . \tag{14.19}$$

It is important to note that if m is linear, we no longer have a standard linear least squares problem that can be solved with normal equations. Moreover, the number of unknowns is in this case $n + m$. Then, when the number m of pieces of data is large, the type of approach can considerably increase the size of the problem.

Geometrically, the problem (14.19) amounts to minimizing the distance between the observations (α_i, β_i) and the point predicted by the model $\big(\alpha_i + \xi_i, m(\alpha_i + \xi_i; x)\big)$, as illustrated in Figure 14.6.

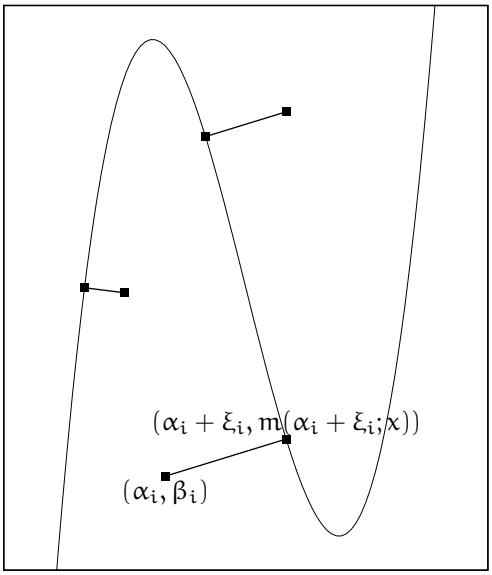


Figure 14.6: Orthogonal regression

14.4 Project

The general organization of the projects is described in Appendix D.

Objectives

To compare the Gauss-Newton method with the quasi-Newton methods, analyze the Kalman filter method, and understand the role of orthogonal regression.

Approach

1. Compare the performances of the quasi-Newton methods (BFGS and SR1) and the Gauss-Newton method for several non linear problems. Utilize the technique described in Chapter D to analyze the results.

2. Generate a large sample with a linear model and apply the Kalman filter method. After how many iterations does the approximation of the optimal solution become stabilized? If synthetic data is used, compare with the "real" value of the parameters. Perform this analysis with samples combining data of varying quality.
3. Generate a sample by introducing perturbations for α and β. Solve the classic least squares problem and compare with the optimal solution obtained by orthogonal regression and with the "real" value.

Algorithms

Algorithms 13.1, 13.2, 14.1 and 14.2.

Problems

As part of this project, we recommend working with real data to test the algorithms. If such data is unavailable, here is how to generate synthetic data.
Apply a model: choose a mathematical model $m(\alpha, x)$, with $\alpha \in \mathbb{R}^p$ and $x \in \mathbb{R}^n$. For instance, a simple linear model

$$m(\alpha, x) = \alpha x\,, \qquad p = 1,\ n = 1\,,$$

or a non linear model

$$m(\alpha, x) = \frac{1}{1 + e^{-\alpha^T x}}\,, \qquad p = n\,.$$

Generate data: generate a sample by randomly choosing values for α for each "observation." For instance, with $p = 2$,

Obs.	α_1	α_2
1	0.007021	6.760832
2	0.017676	4.377047
3	0.866839	0.883049
4	3.235088	0.329827
5	4.699759	0.186086
6	1.080447	0.275139
7	2.482403	0.492628
8	3.544106	1.038803

Choose the parameters: choose the values for the elements of x. In our example, let us take $x_1 = 1$ and $x_2 = -2$.
Generate dependent values: we use two methods.

1. Choose a value of variance σ^2. For each "observation" in the sample, pick a random number r according to the normal law $N(0, \sigma^2)$ and generate

$$\beta = m(\alpha, x) + r\,.$$

In our example, by taking $m(\alpha, x) = 1/(1 + e^{-\alpha^T x})$ and $\sigma = 0.05$, we get

Obs.	α_1	α_2	$m(\alpha, x)$	r	β
1	0.007021	6.760832	1.351E-06	0.072012	0.072013
2	0.017676	4.377047	0.0001606	-0.06527	-0.06511
3	0.866839	0.883049	0.28920265	0.032485	0.321687
4	3.235088	0.329827	0.92926373	0.022983	0.952247
5	4.699759	0.186086	0.9869726	-0.02112	0.965848
6	1.080447	0.275139	0.62952263	0.051388	0.680911
7	2.482403	0.492628	0.8171486	0.012699	0.829847
8	3.544106	1.038803	0.81252489	-0.03009	0.782435

2. Choose a diagonal matrix $\Sigma_\alpha \in \mathbb{R}^p$ and a value of variance σ_β^2. For each "observation," pick a random vector q according to the multivariate normal law $N(0, \Sigma_\alpha)$ and a random number r according to the normal law $N(0, \sigma_\beta^2)$ and generate

$$\beta = m(\alpha + q, x) + r\,.$$

In our example, let us take

$$\Sigma = \begin{pmatrix} 0.02 & 0 \\ 0 & 0.01 \end{pmatrix},$$

to obtain

Obs.	α_1	α_2	q_1	q_2
1	0.007021	6.760832	0.020367	0.003594
2	0.017676	4.377047	0.015533	0.004155
3	0.866839	0.883049	-0.01972	0.000366
4	3.235088	0.329827	-0.03204	0.007443
5	4.699759	0.186086	-0.0363	-0.01052
6	1.080447	0.275139	0.038858	-0.01315
7	2.482403	0.492628	0.026068	0.006862
8	3.544106	1.038803	0.008163	0.002044

and

Obs.	$m(\alpha + q, x)$	r	β
1	1.36895E-06	0.072012	0.072013
2	0.000161766	-0.06527	-0.06511
3	0.285016732	0.032485	0.317501
4	0.926116236	0.022983	0.949099
5	0.986775082	-0.02112	0.96565
6	0.644587833	0.051388	0.695976
7	0.818985987	0.012699	0.831685
8	0.813144895	-0.03009	0.783055

Variants: it is interesting to combine, in a single sample, several samples, generated with different variances. This simulates a process of data collection for which one subsample is more accurate than another.

The problem to solve is

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \sum_j \big(\beta(j) - m\big(\alpha(j), x\big)\big)^2\,.$$

Chapter 15

Direct search methods

Contents

In many applications, it is problematic to calculate the derivatives, either because they require long calculation times, or because their analytical formula is not available. It is conceivable to use finite-difference approximations of first derivatives (Algorithm 8.3) and then utilize quasi-Newton methods (Chapter 13) to approximate the second derivatives. Alternatively, it is possible to use *automatic differentiation methods* (Griewank, 2000), that automatically provide the derivatives of a function specified by a computer program, so that only the objective function needs to be formulated and coded by the user. These techniques may require a significant amount of calculation, though.

Sometimes, the value of the function is obtained by experiments or simulation tools, and the above methods cannot be used anymore. Indeed, when the value of the function is obtained by concrete experiments, it is often difficult to exactly reproduce the same experiment, with one parameter slightly altered, in order to calculate the finite-difference approximation.

Several methodologies have been proposed that require only the value of the objective function calculated at the requested iterates, and do not attempt to approximate the derivatives. *Derivative-free methods* utilize models of the objective function generated from interpolation techniques. We refer the reader to Conn et al. (2009) for an introduction. In this chapter, we introduce *direct search methods* that use geometrical objects to investigate the solution space.

15.1 Nelder-Mead

The best known of these approaches is the simplex method[1] by Nelder and Mead (1965).

Definition 15.1 (Simplex). A k-dimensional simplex is the convex hull of $k+1$ affinely independent vectors $x_1,\ldots,x_{k+1}$ of $\mathbb{R}^n$, $k \leq n$, i.e., the k vectors $x_1 - x_{k+1}$, $x_2 - x_{k+1}$, $\ldots$, $x_k - x_{k+1}$ are linearly independent. For example, three non aligned points in $\mathbb{R}^2$, or four non coplanar points in $\mathbb{R}^3$ are affinely independent and define 2- and 3-dimensional simplices, respectively.

The idea behind the Nelder-Mead method is to define an n-dimensional simplex in $\mathbb{R}^n$ from $n+1$ affinely independent vectors. We assume that these points are sorted such that

$$f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1}).$$

Then, the worst of these points, i.e., x_{n+1} is replaced by a better one.

In order to determine this better point, we calculate the center of gravity of the simplex formed by the other points, i.e.,

$$x_c = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and consider the direction pointing from x_{n+1} towards x_c, that is,

$$d = x_c - x_{n+1}\,.$$

The method then tries several iterates in this direction,

$$x(\alpha) = x_{n+1} + \alpha d = (1-\alpha)x_{n+1} + \alpha x_c\,.$$

Note that $x(1) = x_c$. If $0 < \alpha < 1$, $x(\lambda)$ lies between x_{n+1} and x_c, and if $\alpha > 1$, $x(\alpha)$ is beyond x_c. The values of α tested by the algorithm are $\frac{1}{2}$, 1, $\frac{3}{2}$, 2 and 3, as illustrated in Figure 15.1.

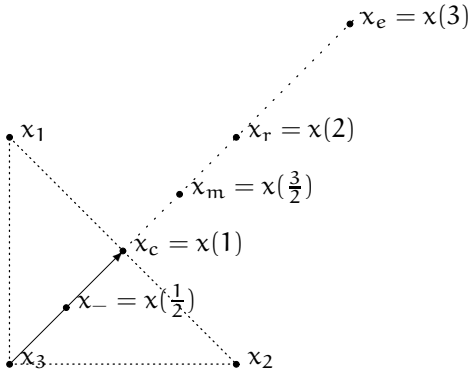


Figure 15.1: Nelder-Mead method

[1] Not to be confused with the simplex algorithm in linear optimization, described in Chapter 16.

The best of these five vectors is then chosen to replace x_{n+1}, which thus forms a new simplex. The method is described as Algorithm 15.1.

Algorithm 15.1: Nelder-Mead

Objective

To find (an approximation of) a local minimum of the optimization problem

$$\min_{x \in \mathbb{R}^n} f(x)\,. \tag{15.1}$$

Input

The continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

The affinely independent vectors $x_1^0, \ldots, x_{n+1}^0$ of $\mathbb{R}^n$, such that $f(x_i^0) \leq f(x_{i+1}^0)$, $i = 1, \ldots, n$.

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}$.

Initialization

$k := 0$

Repeat

$x_c := \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} x_i^k$

Define $d_k := x_c - x_{n+1}^k$

$x_r := x_{n+1}^k + 2d_k = 2x_c - x_{n+1}^k$

if $f(x_r) < f(x_1^k)$ **then** search beyond x_r

  $x_e := x_{n+1}^k + 3d_k = 2x_r - x_c$

  if $f(x_e) < f(x_r)$ **then**

    $\widehat{x} := x_e$

  else

    $\widehat{x} := x_r$

if $f(x_n^k) > f(x_r) \geq f(x_1^k)$ **then**

  $\widehat{x} := x_r$

if $f(x_r) \geq f(x_n^k)$ **then** search before x_r

  if $f(x_r) \geq f(x_{n+1}^k)$ **then**

    $\widehat{x} := x_{n+1}^k + \frac{1}{2}d_k = \frac{1}{2}(x_{n+1}^k + x_c)$

  else

    $\widehat{x} := x_{n+1}^k + \frac{3}{2}d_k = \frac{1}{2}(x_r + x_c)$

$x_{n+1}^{k+1} := \widehat{x}$

$x_i^{k+1} = x_i^k$, $i = 1, \ldots, n$.

$k := k + 1$.

Renumber to get $f(x_i^k) \leq f(x_{i+1}^k)$, $i = 1, \ldots, n$.

Until $\|d_k\| \leq \varepsilon$

$x^* = x_1^k$

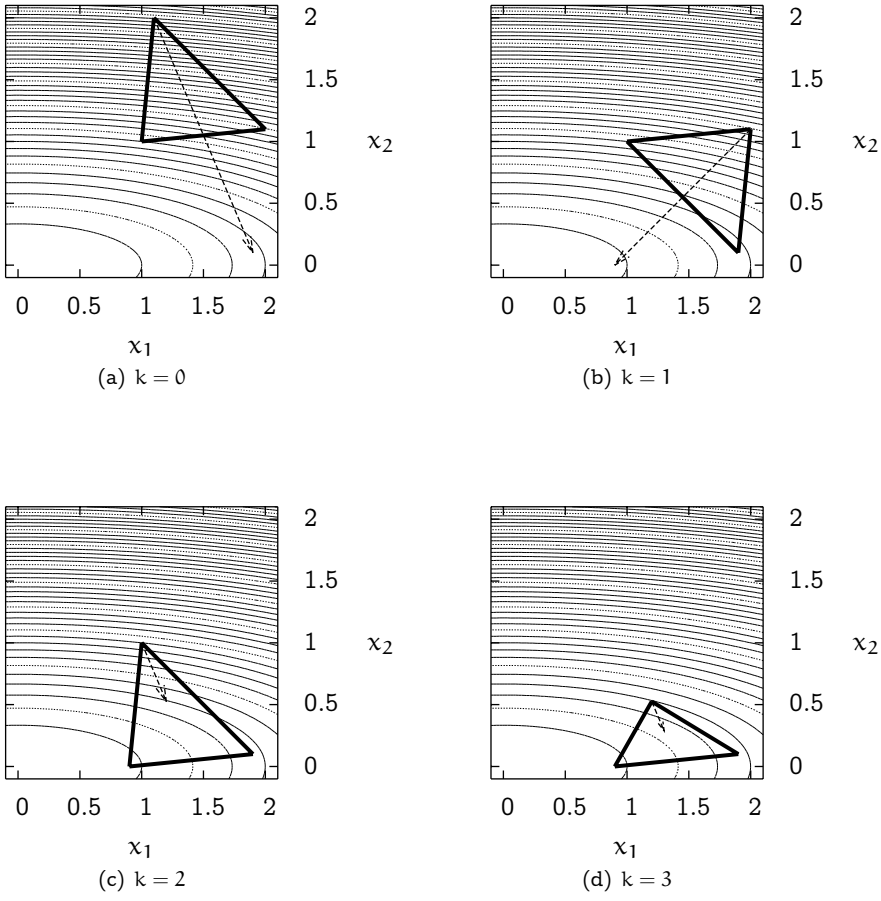
Example 15.2 (Nelder-Mead method)**.** Apply the Nelder-Mead method to the problem

$$\min_{x\in\mathbb{R}^2} f(x) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2,$$

by using as a starting simplex the one generated by the points

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \begin{pmatrix} 2 \\ 1.1 \end{pmatrix}, \qquad \begin{pmatrix} 1.1 \\ 2 \end{pmatrix}.$$

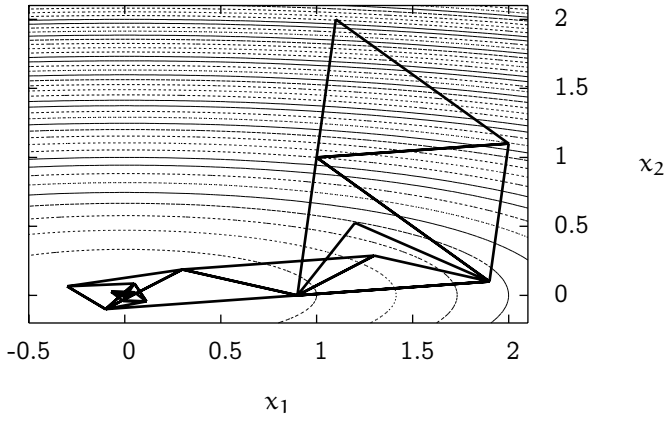
The first four iterations are illustrated in Figure 15.2 and all of the generated simplices are shown in Figure 15.3(a). The details of the first four and the last two iterations are listed in Table 15.1, where the 3 points of the simplex, the point x_r and the new point $\widehat{x}$ are given, in addition to the value of the associated function.



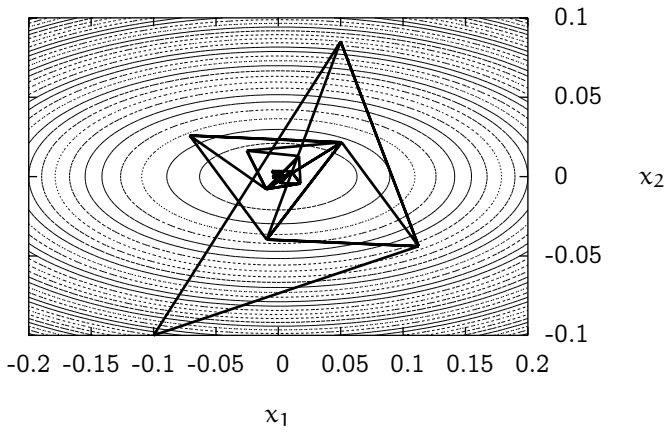
(a) $k = 0$ (b) $k = 1$

(c) $k = 2$ (d) $k = 3$

Figure 15.2: First four iterations of the Nelder-Mead method for Example 15.2



(a) All



(b) Zoom

Figure 15.3: Iterations of the Nelder-Mead method for Example 15.2

It is important to note that there is no guarantee that the Nelder-Mead method converges, as shown in the following example, proposed by McKinnon (1998). It is therefore a heuristic algorithm (see Definition 27.1).

Table 15.1: Iterations of the Nelder-Mead method for Example 15.2

	Simplex			x_r	$\widehat{x}$
1	1.0000e+00	2.0000e+00	1.1000e+00	1.9000e+00	1.9000e+00
	1.0000e+00	1.1000e+00	2.0000e+00	1.0000e-01	1.0000e-01
f	5.0000e+00	7.4450e+00	1.8605e+01	1.8500e+00	1.8500e+00
2	1.0000e+00	2.0000e+00	1.9000e+00	9.0000e-01	9.0000e-01
	1.0000e+00	1.1000e+00	1.0000e-01	-4.4409e-16	-4.4409e-16
f	5.0000e+00	7.4450e+00	1.8500e+00	4.0500e-01	4.0500e-01
3	1.0000e+00	9.0000e-01	1.9000e+00	1.8000e+00	1.2000e+00
	1.0000e+00	-4.4409e-16	1.0000e-01	-9.0000e-01	5.2500e-01
f	5.0000e+00	4.0500e-01	1.8500e+00	5.2650e+00	1.9603e+00
4	1.2000e+00	9.0000e-01	1.9000e+00	1.6000e+00	1.3000e+00
	5.2500e-01	-4.4409e-16	1.0000e-01	-4.2500e-01	2.8750e-01
f	1.9603e+00	4.0500e-01	1.8500e+00	2.0928e+00	1.2170e+00
		⋮			
42	-2.7372e-06	-1.3400e-05	6.6939e-06	1.7357e-05	-5.7107e-06
	-4.7526e-06	2.8202e-06	-6.4183e-07	-8.2146e-06	6.1485e-08
f	1.0539e-10	1.2557e-10	2.4258e-11	4.5428e-10	1.6323e-11
43	-2.7372e-06	-5.7107e-06	6.6939e-06	3.7204e-06	2.1060e-06
	-4.7526e-06	6.1485e-08	-6.4183e-07	4.1723e-06	1.9410e-06
f	1.0539e-10	1.6323e-11	2.4258e-11	8.5255e-11	1.9172e-11

Example 15.3 (The McKinnon example)**.** Consider the function

$$f(x) = \begin{cases} 360\,x_1^2 + x_2 + x_2^2 & \text{if } x_1 \le 0 \\ 6x_1^2 + x_2 + x_2^2 & \text{if } x_1 \ge 0\,, \end{cases}$$

for which the minimum is $\begin{pmatrix} 0 & \frac{1}{2} \end{pmatrix}^T$. Apply the Nelder-Mead algorithm with the initial simplex formed with the points

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \begin{pmatrix} \dfrac{1+\sqrt{33}}{8} \\ \dfrac{1-\sqrt{33}}{8} \end{pmatrix}, \qquad \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

In this case, the algorithm converges toward the point $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ (which is not stationary), as shown in Figure 15.4.

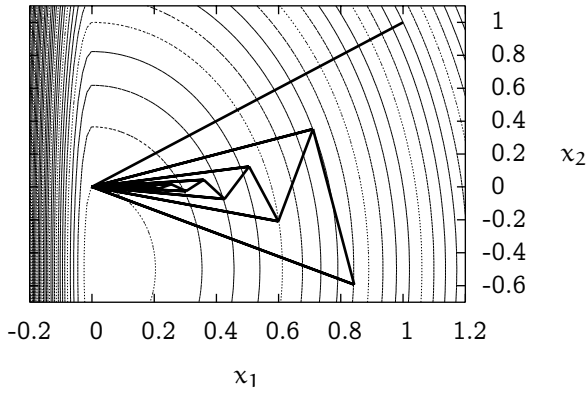
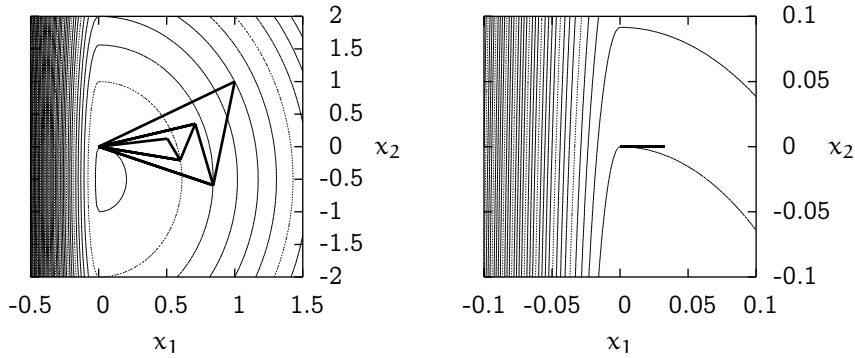


Figure 15.4: Iterations of the Nelder-Mead method for the McKinnon example

Table 15.2: Iterations of the Nelder-Mead method for the McKinnon example

	Simplex			x_r	$\widehat{x}$
1	1.0000e+00	8.4307e-01	0.0000e+00	-1.5693e-01	7.1077e-01
	1.0000e+00	-5.9307e-01	0.0000e+00	-1.5931e+00	3.5173e-01
f	8.0000e+00	4.0233e+00	0.0000e+00	9.8105e+00	3.5066e+00
2	7.1077e-01	8.4307e-01	0.0000e+00	-1.3230e-01	5.9923e-01
	3.5173e-01	-5.9307e-01	0.0000e+00	9.4480e-01	-2.0860e-01
f	3.5066e+00	4.0233e+00	0.0000e+00	8.1389e+00	1.9894e+00
3	7.1077e-01	5.9923e-01	0.0000e+00	-1.1154e-01	5.0519e-01
	3.5173e-01	-2.0860e-01	0.0000e+00	-5.6033e-01	1.2372e-01
f	3.5066e+00	1.9894e+00	0.0000e+00	4.2325e+00	1.6703e+00
4	5.0519e-01	5.9923e-01	0.0000e+00	-9.4037e-02	4.2591e-01
	1.2372e-01	-2.0860e-01	0.0000e+00	3.3232e-01	-7.3372e-02
f	1.6703e+00	1.9894e+00	0.0000e+00	3.6262e+00	1.0204e+00
		⋮			
63	2.5325e-05	2.1351e-05	0.0000e+00	-3.9743e-06	1.8000e-05
	8.5622e-15	-5.0780e-15	0.0000e+00	-1.3640e-14	3.0116e-15
f	3.8483e-09	2.7352e-09	0.0000e+00	5.6862e-09	1.9441e-09
64	1.8000e-05	2.1351e-05	0.0000e+00	-3.3506e-06	1.5176e-05
	3.0116e-15	-5.0780e-15	0.0000e+00	8.0896e-15	-1.7861e-15
f	1.9441e-09	2.7352e-09	0.0000e+00	4.0416e-09	1.3818e-09
65	1.8000e-05	1.5176e-05	0.0000e+00	-2.8248e-06	1.2794e-05
	3.0116e-15	-1.7861e-15	0.0000e+00	-4.7977e-15	1.0593e-15
f	1.9441e-09	1.3818e-09	0.0000e+00	2.8726e-09	9.8214e-10
66	1.2794e-05	1.5176e-05	0.0000e+00	-2.3815e-06	1.0786e-05
	1.0593e-15	-1.7861e-15	0.0000e+00	2.8454e-15	-6.2823e-16
f	9.8214e-10	1.3818e-09	0.0000e+00	2.0418e-09	6.9807e-10



(a) First four iterations (b) Iteration 20

Figure 15.5: Iterations of the Nelder-Mead method for the McKinnon example

15.2 Torczon's multi-directional search

The main reason for the failure of the Nelder-Mead method when it comes to the McKinnon example is that the simplex degenerates with the iterations. Indeed, the three vertices of the simplex become almost colinear. To remedy this, Torczon (1989) proposed a multi-directional search method that maintains the geometry of the simplex throughout the iterations and guarantees that no degeneration appears.

Contrary to the Nelder-Mead method, the Torczon method revolves around the best point of the simplex. Consider the simplex generated by the set of points $\mathcal{S} = \{x_1, \ldots, x_{n+1}\}$. We note

$$f(\mathcal{S}) = \min_{i=1,\ldots,n+1} f(x_i)\,.$$

We assume, without loss of generality, that the minimum is achieved with the first point, that is $f(\mathcal{S}) = f(x_1)$. The main idea is to consider the simplex obtained by reflection around the best point x_1, i.e., the simplex generated by $\mathcal{S}_r = \{x_1^r, \ldots, x_{n+1}^r\}$, with $x_1^r = x_1$, and $x_i^r = 2x_1 - x_i$, $i = 2, \ldots, n+1$ (see the illustration in Figure 15.6).

- If the best point of $\mathcal{S}_r$ is not x_1, i.e., if $f(\mathcal{S}_r) < f(\mathcal{S})$, then $\mathcal{S}_r$ is better than $\mathcal{S}$. In this case, we try to go further and expand the search even more by considering the simplex generated by $\mathcal{S}_e = \{x_1^e, \ldots, x_{n+1}^e\}$, with $x_1^e = x_1$ and $x_i^e = 3x_1 - 2x_i$, $i = 2, \ldots, n+1$ (e for "expansion"). If the expansion provides a better result, that is, if $f(\mathcal{S}_e) < f(\mathcal{S}_r)$, then we choose $\mathcal{S}_e$ for the next iteration. Otherwise, we choose $\mathcal{S}_r$.
- In the case where $\mathcal{S}_r$ is not better than $\mathcal{S}$, i.e., if $f(\mathcal{S}_r) \geq f(\mathcal{S})$, then we have to contract the simplex and use in the next iteration $\mathcal{S}_c = \{x_1^c, \ldots, x_{n+1}^c\}$, with $x_1^c = x_1$ and $x_i^c = \frac{1}{2}(x_1 + x_i)$, $i = 2, \ldots, n+1$.

These different simplices are illustrated in Figure 15.6, and Algorithm 15.2 describes the method.

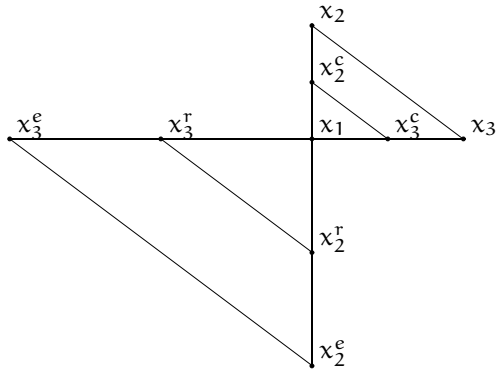


Figure 15.6: Simplices from Torczon's method

Algorithm 15.2: Torczon's multi-directional method

Objective

To find (an approximation of) a local minimum of the optimization problem

$$\min_{x \in \mathbb{R}^n} f(x)\,. \tag{15.2}$$

Input

The function $f : \mathbb{R}^n \to \mathbb{R}$.

The set of affinely independent vectors of $\mathbb{R}^n$ $\mathcal{S}_0 = \{x_1, \ldots, x_{n+1}\}$ such that $f(x_1) \leq f(x_{i+1})$ and $f(x_{n+1}) \geq f(x_i)$, $i = 1, \ldots, n$.

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}$.

Initialization

$k := 0$.

Repeat

Calculate $\mathcal{S}_r$ composed of x_1 and $x_i^r = 2x_1 - x_i$, $i = 2, \ldots, n+1$

if $f(\mathcal{S}_r) < f(x_1)$ **then**

Calculate $\mathcal{S}_e$ composed of x_1 and $x_i^e = 3x_1 - 2x_i = 2x_i^r - x_1$, $i = 2, \ldots, n+1$.

if $f(\mathcal{S}_e) < f(\mathcal{S}_r)$ **then**

$\mathcal{S}_{k+1} := \mathcal{S}_e$

else

$\mathcal{S}_{k+1} := \mathcal{S}_r$

else

Calculate $\mathcal{S}_{k+1}$ composed of x_1 and $x_i^c = (x_1 + x_i)/2$, $i = 2, \ldots, n+1$.

$k := k + 1$

Renumber such that x_1 is the best point and x_{n+1} is the least good in the new simplex

Until $\|x_{n+1} - x_1\| \leq \varepsilon$

$x^* = x_1$

The iterations of Torczon's multi-directional method applied to the McKinnon example are presented in Figure 15.7 and detailed in Table 15.3, which indicates the best point of the current simplex as well as the type of transformation ([C]ontraction, [R]eflection, [E]xpansion). We find that, contrary to the Nelder-Mead method, the Torczon method converges toward the optimal solution of the problem. It is actually the first direct search method that is associated with a proof of convergence (Torczon, 1991).

Table 15.3: Iterations for Torczon's method for the McKinnon example

$(x_1)_1$	$(x_1)_2$	$f(x_1)$	
0.0000e+00	0.0000e+00	0.0000e+00	C
0.0000e+00	0.0000e+00	0.0000e+00	C
0.0000e+00	0.0000e+00	0.0000e+00	C
1.0538e-01	-7.4134e-02	-2.0035e-03	E
6.6151e-02	-4.7240e-01	-2.2298e-01	C
6.6151e-02	-4.7240e-01	-2.2298e-01	C
6.6151e-02	-4.7240e-01	-2.2298e-01	R
3.6514e-03	-5.3490e-01	-2.4870e-01	C
3.6514e-03	-5.3490e-01	-2.4870e-01	C
3.6514e-03	-5.3490e-01	-2.4870e-01	C
3.6514e-03	-5.3490e-01	-2.4870e-01	C
3.6514e-03	-5.3490e-01	-2.4870e-01	R
3.5813e-04	-5.3258e-01	-2.4894e-01	E
1.5841e-03	-5.2014e-01	-2.4958e-01	R
2.8102e-03	-5.0769e-01	-2.4989e-01	C
2.8102e-03	-5.0769e-01	-2.4989e-01	R
3.4232e-03	-5.0147e-01	-2.4993e-01	C
1.4700e-03	-5.0342e-01	-2.4998e-01	R
-1.7658e-04	-5.0226e-01	-2.4998e-01	R
1.2992e-04	-4.9915e-01	-2.5000e-01	C
-2.3332e-05	-5.0071e-01	-2.5000e-01	C
5.3294e-05	-4.9993e-01	-2.5000e-01	C
5.3294e-05	-4.9993e-01	-2.5000e-01	C
5.3294e-05	-4.9993e-01	-2.5000e-01	C
4.3716e-05	-5.0003e-01	-2.5000e-01	C
1.7987e-05	-5.0001e-01	-2.5000e-01	C
1.7987e-05	-5.0001e-01	-2.5000e-01	R
5.1230e-06	-5.0000e-01	-2.5000e-01	C
5.1230e-06	-5.0000e-01	-2.5000e-01	C

Since the doctoral work of V. Torczon in 1989, numerous researchers have shown interest in direct search methods. The interested reader is in particular referred to Wright (1996) or Lewis et al. (2000).

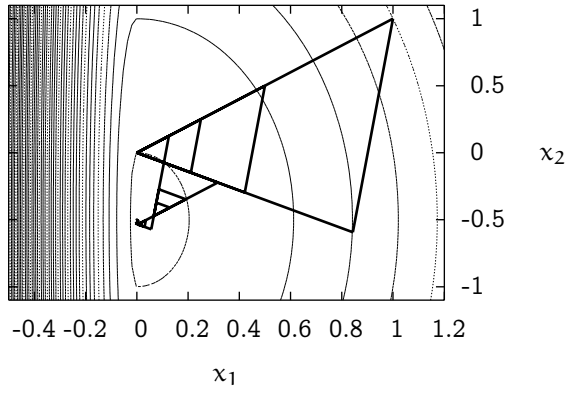


Figure 15.7: Torczon's algorithm for McKinnon's example

15.3 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of this project is to compare the quasi-Newton algorithms with nonderivative algorithms.

Approach

Each problem is to be solved with each algorithm. The gradient used by the quasi-Newton methods should be calculated by finite differences, in order for the contexts to be comparable. The performance indices are the execution time on the one hand, and the number of evaluations of the function on the other hand. The latter index is particularly important when the function to be optimized requires much computational effort to be evaluated. The method described in Section D.2 is to be used to analyze the results.

Algorithms

Algorithms 13.1, 13.2, 15.1, and 15.2.

Problems

Exercise 15.1. The problem

$$\min_{x \in \mathbb{R}^2} 2x_1 x_2 \, e^{-(4x_1^2 + x_2)/8}.$$

Advice: draw the function and the level curves with a software such as Gnuplot, visually identify the stationary points, and then choose the starting points, either close to or far from the stationary points.

Exercise 15.2. The problem

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{n} i\alpha x_i^2, \qquad \bar{x} = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T,$$

with various values of n and of α.

Exercise 15.3. The problem

$$\min_{x \in \mathbb{R}^2} 3x_1^2 + x_2^4.$$

Recommended starting point: $\begin{pmatrix} 1 & -2 \end{pmatrix}^T$.

Exercise 15.4. The Rosenbrock problem

$$\min_{x \in \mathbb{R}^2} 100 \left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

(Section 11.6). Recommended starting points: $\begin{pmatrix} 1.2 & 1.2 \end{pmatrix}^T$ and $\begin{pmatrix} -1.2 & 1 \end{pmatrix}^T$.

Exercise 15.5. The problem

$$\min_{x \in \mathbb{R}^6} \sum_{i=1}^{m} \left(-e^{-0.1\,i} + 5\,e^{-i} - 3\,e^{-0.4\,i} + x_3\,e^{-0.1\,i x_1} - x_4\,e^{-0.1\,i x_2} + x_6\,e^{-0.1\,i x_5}\right)^2.$$

Recommended starting point: $\begin{pmatrix} 1 & 2 & 1 & 1 & 4 & 3 \end{pmatrix}^T$.

Part V

Constrained optimization

The more constraints one imposes, the more one frees one's self. And the arbitrariness of the constraint serves only to obtain precision of execution.

Igor Stravinsky

We now address the description of algorithms for solving constrained optimization problems. Chapter 16 describes one of the most famous algorithms, the *simplex method* for linear optimization. A version of Newton's method for constrained problems, based on projection operators of these constraints, is described in Chapter 17. Dikin's method in Section 17.3 for linear optimization problems constitutes a transition towards a presentation of interior point methods (Chapter 18). Finally, augmented Lagrangian algorithms and sequential quadratic programming are described in Chapters 19 and 20, respectively.

Chapter 16

The simplex method

Contents

16.1 The simplex algorithm

The simplex method (Wood and Dantzig, 1949, Dantzig, 1949, Dantzig, 1963) is probably the most famous optimization algorithm, designed to solve linear optimization problems (6.159)–(6.160), i.e.,

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to

$$Ax = b$$
$$x \geq 0,$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. A central idea of the method is based on the fact that if an optimal solution exists, then there is one that is a vertex (Definition 3.34) of the constraint polyhedron.[1] This property is trivial when $n = 1$. For instance, the linear optimization problem in one variable $\min_{x \in \mathbb{R}} cx$ subject to $x \geq 0$ has an optimal solution if and only if $c \geq 0$, because the problem is unbounded if $c < 0$. When $c > 0$, $x^* = 0$ is the only optimal solution to the problem. If $c = 0$, then any $x \in \mathbb{R}$ is optimal, in particular $x^* = 0$. Note that we have used this property in Chapter 4 when deriving the conditions for the dual function to be bounded. We formalize this result for general bound constraints on x.

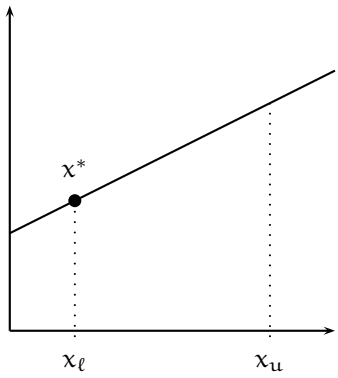
[1] We suggest that the reader carefully reads Sections 3.5 and 6.5 before proceeding with this chapter.

Lemma 16.1. *The optimal solution to*

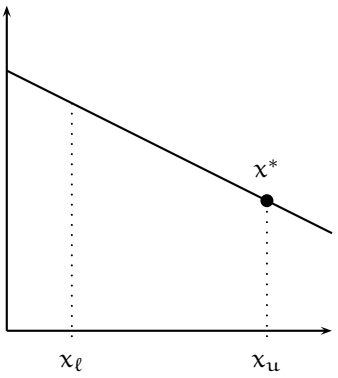
$$\min_{x\in\mathbb{R},\, x_\ell \le x \le x_u} ax + b$$

is $x^* = x_\ell$ *if* $a > 0$ *(Figure 16.1(a)) and* $x^* = x_u$ *if* $a < 0$ *(Figure 16.1(b)). If* $a = 0$*, every* $x_\ell \le x \le x_u$ *is optimal, in particular* x_ℓ *and* x_u *(Figure 16.1(c)). As a corollary, if there exists* x^* *different from* x_ℓ *and* x_u*, which is an optimal solution, then* $a = 0$ *and any feasible point is optimal.*

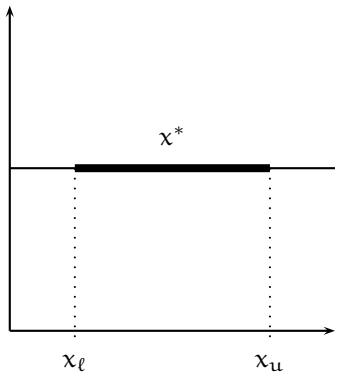
Proof. If $a > 0$ (respectively $a < 0$), the function is strictly increasing (respectively decreasing), and the minimum is reached for the smallest (respectively largest) feasible value of x. □



(a) $a > 0$



(b) $a < 0$



(c) $a = 0$

Figure 16.1: Three possible cases for Lemma 16.1

Theorem 16.2 (Vertex solution). *If the linear optimization problem (6.159)–(6.160) has an optimal solution, there exists an optimal vertex of the constraint polyhedron.*

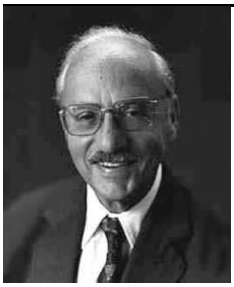
Proof. Let $\mathcal{P} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}$ be the constraint polyhedron and let $x^* \in \mathcal{P}$ be an optimal solution with $f^* = c^T x^*$. Consider $\mathcal{Q} = \left\{x \in \mathcal{P} \mid c^T x = f^*\right\} = \left\{x \in \mathbb{R}^n \mid Ax = b,\, c^T x = f^*,\, x \geq 0\right\}$, i.e., the set of optimal solutions to the problem. The set $\mathcal{Q}$ is also a polyhedron represented in standard form. It is non empty ($x^* \in \mathcal{Q}$) and contains at least one vertex y^* (Theorem 3.37). We assume by contradiction that y^* is not a vertex of $\mathcal{P}$. Therefore, from the definition of a vertex (Definition 3.34), there exist y and z, $y \neq z$, in $\mathcal{P}$ and $0 < \lambda^* < 1$ such that

$$y^* = \lambda^* y + (1 - \lambda^*) z\,. \tag{16.1}$$

According to Lemma 16.1, the one-dimensional linear optimization problem

$$\min_{0 \leq \lambda \leq 1} \lambda c^T y + (1 - \lambda) c^T z$$

has an optimal solution with a value $0 < \lambda^* < 1$ (corresponding to y^*). Therefore, any feasible value of λ is also optimal. In particular $\lambda = 0$ and $\lambda = 1$, so that $f^* = c^T y^* = c^T y = c^T z$ and y and z belong to $\mathcal{Q}$, contradicting the fact that y^* is a vertex of $\mathcal{Q}$. This shows that the optimal solution y^* is a vertex of $\mathcal{P}$. □



George Bernard Dantzig was born on November 8, 1914, in Portland, Oregon (USA). He was a student of Neyman at Berkeley. Arriving late to one of Neyman's classes, Dantzig took as exercises some unresolved statistical problems that Neyman had written on the blackboard and solved them. Even though he finished his dissertation in 1941, he didn't receive his PhD until 1946 because of World War II. George Dantzig's fundamental contribution is the simplex method in optimization. He is often referred to as the "father of linear programming," and his work is considered as the foundation of operations research as a science. Developed within the scope of a task for the US Air Force, the simplex method was first used to solve a diet problem. Linear programming and the simplex method are nowadays embedded in every decision support system. They represent the most important contribution of mathematics to the daily operations of businesses and industries around the world. The strength of the method always impressed Georges Dantzig himself. Dantzig was since 1966 professor of operational research and computer science at Stanford University. He died on Friday May 13, 2005.

Figure 16.2: George B. Dantzig

Before proposing algorithms, let us see how to solve the problem graphically for a simple example.

Example 16.3 (Graphical method)**.** Consider the optimization problem

$$\min_{x \in \mathbb{R}^2} -x_1 - 2x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_1 - x_2 &\leq 1 \\ x_1 &\geq 0 \\ x_2 &\geq 0\,. \end{aligned}$$

Put in standard form, the constraint polyhedron of this problem is identical to the polyhedron in Example 3.39. Figure 16.3 represents the feasible domain in the space of variables (x_1, x_2).

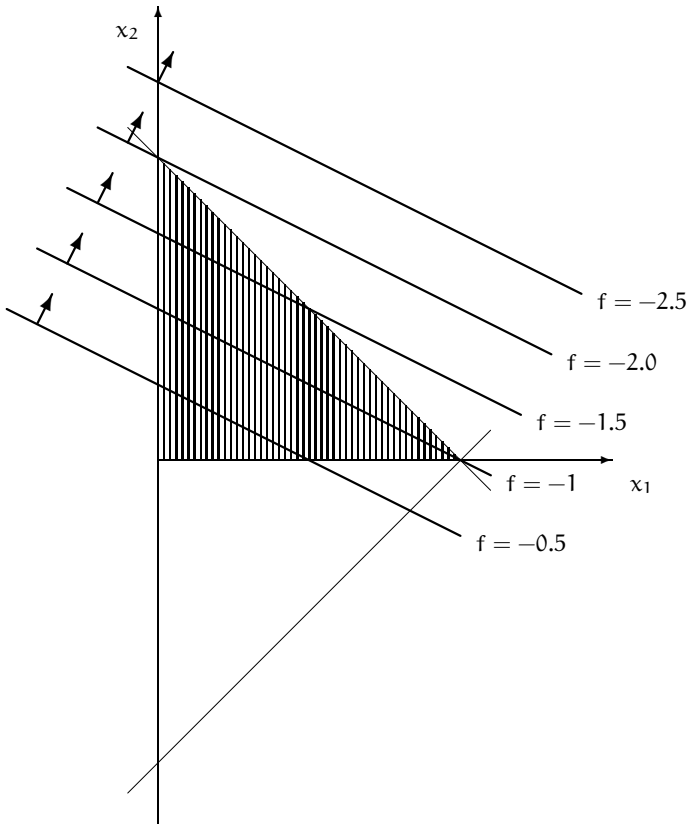


Figure 16.3: Graphical method to find the optimal solution to a linear optimization problem

The level lines corresponding to the different values of the objective function are also shown. The steepest descent direction, i.e.,

$$-\nabla f(x) = -c = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

is displayed with an arrow on each level line. In order to graphically identify the optimal solution to the problem, one must

1. draw an arbitrary level line intersecting the feasible domain,
2. move this line parallel to itself as far as possible in the direction of the vector $-c$, as long as it intersects the feasible domain.

All the points at the intersection of this line and the feasible domain are optimal.

In Example 16.3, the optimal solution is $(0, 1)$, with an optimal value of -2. The result of Theorem 16.2 enables us to propose a quite simple algorithm. To find the optimal solution to the problem, we need only go through all the vertices of the constraint polyhedron and choose the best one. This algorithm can be easily implemented thanks to Theorem 3.40, that says that the choice of a vertex of the constraint polyhedron amounts to the choice of the $n - m$ inequality constraints that are active at this vertex, that is to choose $n - m$ variables that are set to 0 (indeed, as the polyhedron is represented in standard form, the inequality constraints are non negativity constraints). These variables are said to be *non basic*. The m other variables are said to be *basic*. If the $m \times m$ matrix B gathering the columns of A corresponding to basic variables is non singular, $x_B = B^{-1}b$ provides the value of the basic variables at the considered vertex, if $x_B \geq 0$ (see the discussions in Section 3.5 for more details). The vertex enumeration method detailed in Algorithm 16.1 exploits this characterization.

Example 16.4 (Vertex enumeration)**.** We apply Algorithm 16.1 with

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad c = \begin{pmatrix} -1 \\ -2 \\ 0 \\ 0 \end{pmatrix}.$$

The vertex enumeration for this problem is carried out in the description of Example 3.39. We now need only calculate the values of the objective function to find the optimal solution.

Algorithm 16.1: Vertex enumeration

Objective
  To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160)
Input
  The matrix $A \in \mathbb{R}^{m \times n}$.
  The vector $b \in \mathbb{R}^m$.
  The vector $c \in \mathbb{R}^n$.
Output
  The set J^* of basic variables indices of the optimal solution
Initialization
  $\mathcal{C} := C_m(1, \ldots, n)$, set of all combinations of m indices among n.
  $k := 1$
  $f^* := +\infty$
Repeat
  Choose a potential basis, that is a set of m indices $J_k = (j_1^k, \ldots, j_m^k) \in \mathcal{C}$
  Let $B = (A_{j_1^k}, \ldots, A_{j_m^k})$ be the matrix formed by the columns of A corresponding to the indices of J_k
  if B *is invertible and* $B^{-1}b \geq 0$ **then**
    $f_k := c_B^T B^{-1} b$ where c_B contains the basic components of c
  else
    $f_k = +\infty$
  if $f_k < f^*$ **then**
    $J^* = J_k$
    $f^* = f_k$
  $\mathcal{C} := \mathcal{C} \setminus J_k$
  $k := k + 1$
Until $\mathcal{C} = \emptyset$

k	x_k	f_k
1	$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$	-1
2	$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$	-1
3	$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$	-1
4	$\begin{pmatrix} 0 & -1 & 2 & 0 \end{pmatrix}^T$	$B^{-1}b \not\geq 0$
5	$\begin{pmatrix} 0 & 1 & 0 & 2 \end{pmatrix}^T$	-2
6	$\begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix}^T$	0

The optimal solution is $x^* = \begin{pmatrix} 0 & 1 & 0 & 2 \end{pmatrix}^T$.

Algorithm 16.1 identifies the optimal solution to a linear optimization problem in a finite number of iterations, which is the total number of possible ways to choose m variables among n, that is

$$\frac{n!}{(n-m)!m!}.$$

This number becomes prohibitively large when n and m are large, and the algorithm is then inapplicable. We are facing a *combinatorial optimization* problem[2] (see Definition 25.5). We refer the reader to Avis and Fukuda (1992) for an algorithm enumerating the vertices of a polyhedron in arbitrary dimension.

The simplex method, which we now describe, also goes through the vertices of the constraint polyhedron, but does it intelligently, to avoid having to enumerate them all. It uses a strategy similar to the descent methods presented in Chapter 11: at each iteration, a descent direction is identified, and a step is calculated. As discussed below, the step may happen to be zero, so that several iterations may not produce any progress, and special care needs to be taken to avoid the algorithm being stalled. The algorithm exploits the equivalence between vertices of the constraint polyhedron and feasible basic solutions (Theorem 3.40).

The geometric interpretation of the simplex algorithm can be summarized as follows:

- The algorithm starts from a vertex of the constraint polyhedron.
- An edge of the polyhedron along which the objective function decreases is identified. If no such edge exists, the current vertex is an optimal solution.
- The edge is followed until the next vertex is reached.

A simple illustration is provided in Figure 16.4.

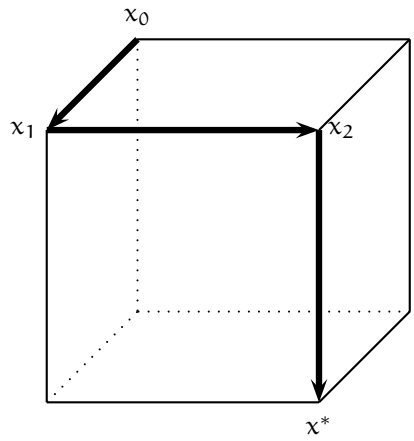


Figure 16.4: Geometric illustration of the simplex algorithm

[2] Linear optimization combines the features of continuous optimization and combinatorial optimization. In addition to its important role in many concrete applications, these features are exploited a lot in theoretical developments.

The analysis provided in Sections 3.5 and 6.5 allows the geometrical concepts to be translated into algebraic concepts that are combined to Algorithm 16.2.

- The vertices of the polyhedron are basic feasible solutions, characterized by the set of indices corresponding to the basic variables (Theorem 3.40). Note that one vertex may correspond to several basic feasible solutions.
- The edges of the polyhedron are characterized by the basic directions (Definition 3.42), and are used to go from one vertex to another.
- The reduced costs (Definition 6.28) represent the directional derivative of the function in the basic directions. They are utilized to identify the descent directions and verify the optimality of the current iterate.

Comments

- The algorithm must be initialized with an arbitrary feasible basic solution. For problems in standard form, such a feasible solution always exists if the polyhedron is non empty (Theorems 3.37 and 3.40). However, it is not necessarily simple to find such a feasible solution. We address this problem in Section 16.3, where such a basic feasible solution, or proof there is none, is furnished by the simplex algorithm applied to an auxiliary problem.
- According to Theorem 6.29, the reduced costs for the basic indices are zero. For this reason, in step 15, only the non basic indices are considered.
- In practice, the d_N part of the direction is never formed.
- Step 22 calculates the maximum step α_q that can be taken along the basic direction d_p, while remaining feasible. Geometrically, it is the step that corresponds to the first constraint activated along the basic direction. Therefore, the corresponding variable becomes 0 and becomes non basic (see Lemma 16.5).
- The reduced cost represents the directional derivative of the (linear) function in the basic direction. When it is negative, the basic direction is a descent direction and the new value of the objective function is

$$c^T x_{k+1} = c^T x_k + \alpha_q \bar{c}_p \,. \tag{16.2}$$

- Step 18 identifies the index of the non basic variable that is entering the basis, and step 22 identifies the index of the basic variable that is leaving the basis. In the presence of a degenerate basic feasible solution, several candidates may be possible. The algorithm then suggests selection of the smallest index among those that verify the condition. It allows for a systematic enumeration of the basic feasible solutions corresponding to the same vertex of the constraint polyhedron, and avoids the algorithm being stalled in an endless cycle. This guarantees that the algorithm terminates after a finite number of iterations. This choice is called Bland's rule, from the work of Bland (1977). Other strategies are suggested in the projects of Section 16.6.

Algorithm 16.2: Simplex method

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160).

Input

The matrix $A \in \mathbb{R}^{m \times n}$.

The vector $b \in \mathbb{R}^m$.

The vector $c \in \mathbb{R}^n$.

$J^0 = (j_1^0, \ldots, j_m^0)$ the set of indices of the basic variables corresponding to a basic feasible solution.

Output

A Boolean indicator U detecting an unbounded problem.

If U is false, $J^* = (j_1^*, \ldots, j_m^*)$ is the set of indices of an optimal basic feasible solution.

Initialization

$k := 0$.

Repeat

Let $B = \left(A_{j_1^k}, \ldots, A_{j_m^k}\right)$ be the matrix formed by the columns of A corresponding to the indices of J_k.

if $\bar{c}_j = c_j - c_B^T B^{-1} A_j \geq 0 \; \forall j \notin J^k$ **then** optimal solution

  $J^* = J^k$, U=FALSE, STOP.

else

  $p :=$ smallest index such that $\bar{c}_p < 0$.

Calculate the basic variables $x_B = B^{-1} b$.

Calculate the basic components of the p^{th} basic direction $d_B = -B^{-1} A_p$.

For each $i = 1, \ldots, m$, calculate the distance to the non negativity constraint, i.e.,

$$\alpha_i := \begin{cases} -\dfrac{(x_B)_i}{(d_B)_i} & \text{if } (d_B)_i < 0 \\ +\infty & \text{otherwise.} \end{cases} \tag{16.3}$$

Let q be the smallest index such that $\alpha_q = \min_i \alpha_i$.

if $\alpha_q = +\infty$ **then** the problem is unbounded and has no optimal solution

  U=TRUE. STOP.

$J^{k+1} := J^k \cup \{p\} \setminus j_q^k$.

$k = k + 1$.

Until *STOP*

- At each iteration, if the basic feasible solution is not degenerate, the basic direction is feasible (Theorem 3.44). Then, no α_i in (16.3) is zero. Therefore, the step α_q is positive and (16.2) guarantees that

$$c^T x_{k+1} < c^T x_k \,.$$

- For the algorithm to be valid, we need to demonstrate that x_{k+1} is again a basic feasible solution and that the matrix $\bar{B}$, obtained by replacing column j_q by A_p, is non singular.

Lemma 16.5. *After one iteration of the simplex method (Algorithm 16.2), the new set of indices defines a basic feasible solution.*

Proof. We assume without loss of generality that the numbering is such that the basic variables come first, that is, $j_i^k = i$ for all i. We consider the matrix $\bar{B}$ corresponding to the new set of indices and let us first demonstrate that it is non singular. We assume, by contradiction, that this is not the case. There exist coefficients $\lambda_1, \ldots, \lambda_m$, not all zero, such that

$$\sum_{i=1}^{m} \lambda_i \bar{B}_i = \sum_{\substack{i=1 \\ i \neq q}}^{m} \lambda_i \bar{B}_i + \lambda_q \bar{B}_q = \sum_{\substack{i=1 \\ i \neq q}}^{m} \lambda_i A_i + \lambda_q A_p = 0 \,,$$

as $\bar{B}$ has been obtained from B by removing column q and replacing it by A_p, all other columns being the same. Multiplying by B^{-1}, we obtain

$$\sum_{\substack{i=1 \\ i \neq q}}^{m} \lambda_i B^{-1} A_i + \lambda_q B^{-1} A_p = 0,$$

and these vectors are linearly dependent. However, for all $i \neq q$, $B_i = A_i$ and

$$B^{-1} A_i = B^{-1} B_i = e_i \,,$$

where e_i is the i^{th} column of the identity matrix. These vectors are linearly independent and their q^{th} component is zero. The vector $B^{-1} A_p$ is exactly $-d_B$ (see step 20 or (3.88)). The q^{th} component of $-d_B$ is not zero, as the index q is chosen among the indices for which $(d_p)_i < 0$ (step 21). Therefore, $B^{-1} A_p$ is linearly independent from all $B^{-1} A_i$, creating the contradiction.

So, the first part of Definition 3.38 is satisfied. We now demonstrate that all the non basic variables are zero. As a basic direction is followed during the iteration, only the non basic variable p is modified by the iteration (see Definition 3.42). It enters the basis and may become positive. All the others remain at zero, out of the basis. We analyze the q^{th} variable, that exits the basis during the iteration. Its value at the end of the iteration is

$$(x_{k+1})_q = (x_k)_q + \alpha_q (d_p)_q = (x_k)_q - \frac{(x_k)_q}{(d_p)_q} (d_p)_q = 0 \,.$$

The new iterate is therefore indeed a basic solution. We demonstrate that it is feasible.

- All the non basic variables are non negative, because they are zero.
- Let i be a basic index. We have

$$(x_{k+1})_i = (x_k)_i + \alpha_q (d_p)_i .$$

Since x_k is feasible and $(x_k)_i \geq 0$, only the indices i such that $(d_p)_i < 0$ may cause problems. However, according to step 22 of the algorithm we have $\alpha_q \leq \alpha_i$ for such indices. Therefore,

$$\begin{aligned}(x_{k+1})_i &= (x_k)_i + \alpha_q (d_p)_i \\ &\geq (x_k)_i + \alpha_i (d_p)_i \\ &= (x_k)_i - \frac{(x_k)_i}{(d_p)_i} (d_p)_i = 0\end{aligned}$$

and x_{k+1} is feasible.

□

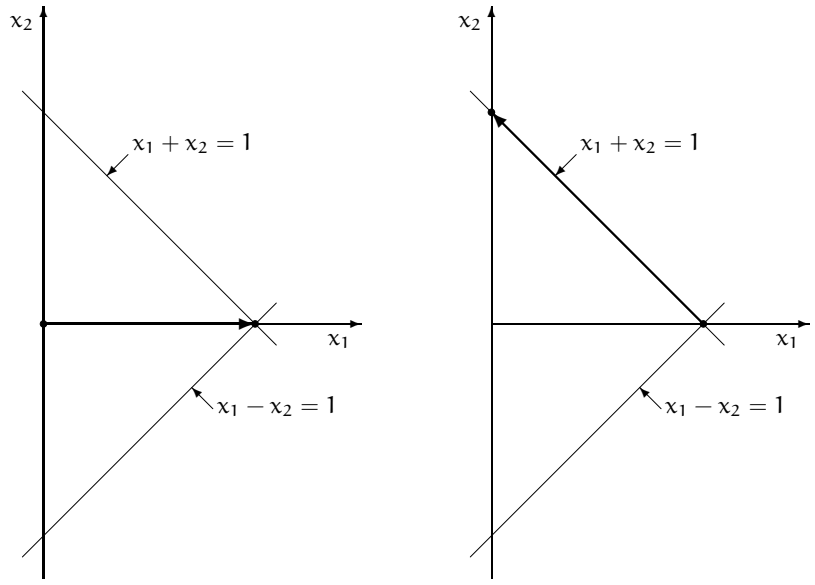
Example 16.6 (The simplex method – I)**.** We apply the simplex method to the same problem as in Example 16.4, with

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad c = \begin{pmatrix} -1 \\ -2 \\ 0 \\ 0 \end{pmatrix}.$$

Iteration 1

1. Consider $J^0 = \{3, 4\}$ and $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
2. Current iterate:
$$x_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \qquad c^T x_0 = 0 .$$
3. Reduced costs:
$$\begin{aligned}\bar{c}_1 &= -1 \\ \bar{c}_2 &= -2 \\ \bar{c}_3 &= 0 \\ \bar{c}_4 &= 0 .\end{aligned}$$
The index $p = 1$ is chosen to enter in the basis. Indeed, it is the smallest index corresponding to a negative reduced cost.
4. Basic direction:
$$d_{B_1} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \qquad d_1 = \begin{pmatrix} 1 \\ 0 \\ -1 \\ -1 \end{pmatrix}, \qquad p = 1 .$$

5. Distances to the constraint:
$$\alpha_3 = 1$$
$$\alpha_4 = 1\,.$$
The index $q = 3$ is chosen to leave the basis. In fact, since the two values of α_i are equal, the smallest index is chosen.
6. Index 1 replaces index 3 in the basis, and $J^1 = \{1, 4\}$.



(a) Iteration 1 (b) Iteration 2

Figure 16.5: Iterations for Example 16.6

Iteration 2

1. $J^1 = \{1, 4\}$ and $B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$.
2. Current iterate :
$$x_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad c^T x_1 = -1\,.$$
3. Reduced costs:
$$\bar{c}_1 = 0$$
$$\bar{c}_2 = -1$$
$$\bar{c}_3 = 1$$
$$\bar{c}_4 = 0\,.$$
The index $p = 2$ is chosen to enter in the basis.

4. Basic direction:

$$d_{B_2} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}, \qquad d_2 = \begin{pmatrix} -1 \\ 1 \\ 0 \\ 2 \end{pmatrix}, \qquad p = 2\,.$$

5. Distances to the constraint:

$$\alpha_1 = 1$$
$$\alpha_4 = +\infty\,.$$

The index $q = 1$ is chosen to leave the basis.

6. Index 2 replaces index 1 in the basis, and $J^2 = \{2, 4\}$.

Iteration 3

1. $J^2 = \{2, 4\}$ and $B = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$.

2. Reduced costs:

$$\bar{c}_1 = 1$$
$$\bar{c}_2 = 0$$
$$\bar{c}_3 = 2$$
$$\bar{c}_4 = 0\,.$$

The point is optimal.

3. Optimal solution:

$$x^* = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 2 \end{pmatrix}, \qquad c^T x^* = -2\,.$$

Example 16.7 (The simplex method – II)**.** We apply the simplex method to the following problem:

$$\min -10\,x_1 - 12\,x_2 - 12\,x_3$$

subject to

$$\begin{aligned} x_1 + 2x_2 + 2x_3 &\le 20 \\ 2x_1 + \ x_2 + 2x_3 &\le 20 \\ 2x_1 + 2x_2 + \ x_3 &\le 20\,, \end{aligned}$$

and

$$x_1,\, x_2,\, x_3 \ge 0.$$

By adding slack variables in order to obtain a problem in standard form, we get a problem with $n = 6$ variables and $m = 3$ constraints, defined by

$$A = \begin{pmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 20 \\ 20 \\ 20 \end{pmatrix}, \quad c = \begin{pmatrix} -10 \\ -12 \\ -12 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The choice of $J^0 = \{4, 5, 6\}$ produces an initial feasible basic solution, from which we can apply the simplex method. The details of the iterations are given in Table 16.1.

Table 16.1: Iterations with the simplex method for Example 16.7

k	J^k	$\bar{c}$	x_1	x_2	x_3	x_4	x_5	x_6	d_B	α_q	p	q	$c^T x$
0	4 5 6	−10 −12 −12	0	0	0	20	20	20	−1 −2 −2	10	1	5	0
1	1 4 6	−7 −2 5	10	0	0	10	0	0	−1.5 −0.5 −1	0	2	6	−100
2	1 2 4	−9 −2 7	10	0	0	10	0	0	−2.5 −1.5 1	4	3	4	−100
3	1 2 3	3.6 1.6 1.6	4	4	4	0	0	0					−136

Note that the basic feasible solution for iteration 1 is degenerate. Indeed, x_6 is zero even though it is in the basis. We also note that at this iteration, the method cannot progress. The step α_q is zero. However, the algorithm still performs a change of basis (the variable 6 is replaced by the variable 2). During iteration 2, the feasible basic solution is also degenerate (x_2 is zero in the basis), but the algorithm can now progress ($\alpha_q = 4$).

16.2 The simplex tableau

Algorithm 16.2 requires a significant computational effort for the linear algebra, mainly due to the need for the matrix B^{-1}:

- Step 15: calculating reduced costs $c^T - c_B^T B^{-1} A$.
- Step 19: calculating the current iterate $B^{-1} b$.
- Step 20: calculating the direction $-B^{-1} A_p$.

To improve this, we regroup all the important quantities used by the algorithm in a table, called the *simplex tableau*.

Definition 16.8 (Simplex tableau)**.** Consider a linear optimization problem in standard form $\min c^T x$ subject to $Ax = b$, $x \geq 0$, and let us take a basic matrix B corresponding to a basic feasible solution $\tilde{x}$. The table

$B^{-1}A$	$B^{-1}b$
$c^T - c_B^T B^{-1} A$	$-c_B^T B^{-1} b$

(16.4)

is called the *simplex tableau* corresponding to this basic feasible solution. In more detail, we have

$B^{-1}A_1$	$\cdots$	$B^{-1}A_n$	$\widetilde{x}_{j_1}$ $\vdots$ $\widetilde{x}_{j_m}$
$\bar{c}_1$	$\cdots$	$\bar{c}_n$	$-c^T\widetilde{x}$

(16.5)

where $\bar{c}_i$ is the reduced cost of the variable i.

To illustrate the concept, let us consider the table corresponding to the optimal solution of Example 16.6:

x_1	x_2	x_3	x_4		
1	1	1	0	1	x_2
2	0	1	1	2	x_4
1	0	2	0	2	$-c^Tx$

Basic variables

- Each column to the left side of the tableau corresponds to a variable of the problem.
- The columns with the basic variables contain the columns of the identity matrix.
- Each row corresponds to a basic variable. The correspondence is defined by the specific structure of the column corresponding to this basic variable: it is a column of the identity matrix, that is, all its elements are 0, except one which is 1. The row where element 1 is situated is the row corresponding to the basic variable. In the example, the only 1 in the column corresponding to x_2 is in the first row. Therefore, the first row is associated with variable x_2. Following the same reasoning, we see that the second row is associated with variable x_4.
- The m first rows of the last column contain the values of the basic variables. The other variables, that is, the non basic variables, are always zero.
- The last element of the last column contains the value of the objective function, *with opposite sign*. Indeed,

$$-c^Tx = -c_B^Tx_B - c_N^Tx_N = -c_B^TB^{-1}b - 0\,.$$

If a tableau is available, an iteration of Algorithm 16.2 greatly simplifies, as the quantities required for steps 15, 19, and 20 can now be read directly in the tableau instead of being calculated. The questions are now: how do we generate the first tableau, and how do we update the tableau from one iteration to the next, in an efficient way? We address now the second question. We come back to the first one in Section 16.3.

Clearly, in order for the algorithm to be effective, one must avoid recalculating the tableau at each iteration. Given that only one variable is replaced in the basis, only one column of B is modified from one iteration to the next. Therefore, we do

not expect the tableau in the next iteration to be too different from the tableau in the current iteration.

Let B be the basic matrix at the start of the iteration and let $\bar{B}$ be this matrix at the end of the iteration, obtained by replacing one column of B by another column from A. In Example 16.7, we have

$$A = \begin{pmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

The first basic matrix, corresponding to indices 4, 5, and 6, is

$$B = (A_4 \quad A_5 \quad A_6) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

After the first iteration, the variable 5 is replaced by variable 1 in the basis, so that

$$\bar{B} = (A_4 \quad A_1 \quad A_6) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 1 \end{pmatrix}.$$

We would like to find a simple transformation of B^{-1} to obtain $\bar{B}^{-1}$, i.e., a matrix Q such that

$$QB^{-1} = \bar{B}^{-1}$$

or, equivalently,

$$QB^{-1}\bar{B} = I.$$

It means that the matrix Q transforms the matrix $B^{-1}\bar{B}$ into the identity matrix. Since $B^{-1}B = I$, and B and $\bar{B}$ have the same columns except one, the matrix $B^{-1}\bar{B}$ is already "almost" the identity matrix, i.e.,

$$B^{-1}\bar{B} = \begin{pmatrix} 1 & 0 & & u_1 & & 0 \\ 0 & 1 & & u_2 & & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ \vdots & \vdots & & u_q & & \vdots \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & & u_m & & 1 \end{pmatrix}, \tag{16.6}$$

where the vector u is defined by $u = -d_p = B^{-1}A_p$. We must transform (16.6) into an identity matrix. The application Q that takes care of this is the composition of elementary row operations.

Definition 16.9 (Elementary row operations)**.** Consider a matrix A. An *elementary row operation* on A consists in multiplying by a constant β a row j of A and adding the result to the row i

$$a_i := a_i + \beta a_j \,,$$

where a_i denotes the ith row of A. This operation consists in multiplying A by the matrix Q_{ij}, which is the identity matrix (row dimension of A), of which element (i, j) is replaced by β.

Example 16.10 (Elementary row operations)**.** Consider the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}.$$

We choose $i = 2$, $j = 1$, $\beta = 4$. We multiply the first row by 4 and add the result to the second row to obtain

$$\bar{A} = \begin{pmatrix} 1 & 2 \\ 7 & 12 \\ 5 & 6 \end{pmatrix}.$$

We have $\bar{A} = Q_{ij}A$, with

$$Q_{ij} = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + I = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In order to transform the matrix (16.6) into an identity matrix, we must apply to it the following elementary row operations:

- For each row $i \neq q$, we add the row q multiplied by $-u_i/u_q$ to the i^{th} row. Note that $u_q = -(d_p)_q$ is not zero (see the proof of Theorem 16.5).
- The row q is divided by u_q.

Example 16.11 (Basis change)**.** Consider again Example 16.7. At the first iteration, we have

$$B^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The second row of B (corresponding to the variable 5) is replaced by A_1. Since $B^{-1}A_1 = \begin{pmatrix} 1 & 2 & 2 \end{pmatrix}^T$, we have

$$B^{-1}\bar{B} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 1 \end{pmatrix}.$$

In order for this matrix to be an identity matrix, we must apply elementary row operations to the following rows:

1. $a_1 := a_1 - (u_1/u_2)a_2$, that is $a_1 := a_1 - 0.5\,a_2$.
2. $a_3 := a_3 - (u_3/u_2)a_2$, that is $a_3 := a_3 - a_2$.
3. $a_2 := a_2/u_2$, that is $a_2 := a_2/2$.

Note that the modification of row 2 must be performed last, as this row is involved in the update of rows 1 and 3. By applying these operations to B^{-1}, we get

$$\bar{B}^{-1} = \begin{pmatrix} 1 & -0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & -1 & 1 \end{pmatrix}.$$

Indeed,

$$\bar{B}^{-1} = QB^{-1} = Q_{22}Q_{12}Q_{32}B^{-1} = Q_{22}Q_{12}Q_{32}$$

where

$$Q_{22} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad Q_{12} = \begin{pmatrix} 1 & -0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad Q_{32} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}.$$

Note that the matrix Q_{22} is applied last.

We now see that the same operations can be applied to the simplex tableau. Indeed, for the first part of the tableau, since the elementary operations are represented by the matrix $Q = \bar{B}^{-1}B$, we have

$$\boxed{QB^{-1}A \mid QB^{-1}b} = \boxed{\bar{B}^{-1}A \mid \bar{B}^{-1}b}$$

We show later on that the same is true for the last row. We use the following update procedure, called *pivoting*.

Example 16.12 (Pivoting)**.** Consider the following simplex tableau. We want to extract the variable x_6 from the basis (line 3 of the tableau) and enter the variable x_2 (column 2 of the tableau).

	x_1	x_2	x_3	x_4	x_5	x_6		
	0	1.5	1	1	−0.5	0	10	x_4
	1	0.5	1	0	0.5	0	10	x_1
T =	0	(1)	−1	0	−1	1	0	x_6
	0	−7	−2	0	5	0	100	

Basic variables

Algorithm 16.3: Tableau pivoting

Objective

To update the simplex tableau during an iteration of the simplex method.

Input

The simplex tableau T.

The index p of the pivot column, i.e., the column corresponding to the non basic variable that enters the basis.

The index q of the pivot row, i.e., the row corresponding to the basic variable that leaves the basis.

Output

The simplex tableau $\bar{T}$ corresponding to the new basis.

Initialization

if $T(q,p) = 0$ **then** Impossible to carry out the pivoting

STOP

for $i = 1, \ldots, m+1$, $i \neq q$ **do**

$T(i,k) := T(i,k) - T(i,p)T(q,k)/T(q,p) \quad k = 1, \ldots, n+1$

$T(q,k) := \dfrac{T(q,k)}{T(q,p)} \qquad k = 1, \ldots, n+1$

By applying the pivoting (Algorithm 16.3), we get the following tableau:

$\bar{T} =$

x_1	x_2	x_3	x_4	x_5	x_6		
0	0	2.5	1	1	−1.5	10	x_4
1	0	1.5	0	1	−0.5	10	x_1
0	1	−1	0	−1	1	0	x_2
0	0	−9	0	−2	7	100	

Basic variables

We now need only demonstrate that the last row of the new tableau corresponds to Definition 16.8. In the tableau T, this last row is of the type

$$T_{m+1} = (c^T \mid 0) - d^T(A \mid b) \qquad \text{with } d^T = c_B^T B^{-1}.$$

The pivot row is of the type

$$T_q = g^T(A \mid b),$$

where g^T is the q^{th} row of B^{-1}. During the elementary operation on the last row, we have

$$\bar{T}_{m+1} = T_{m+1} + \beta T_q$$

and this row takes the form

$$\bar{T}_{m+1} = (c^T \mid 0) - d^T(A \mid b) + \beta g^T(A \mid b) = (c^T \mid 0) + h^T(A \mid b)\,, \tag{16.7}$$

with $h^T = -d^T + \beta g^T$. We consider a column k of the tableau $\bar{T}$. We have

$$\bar{T}(m+1,k) = T(m+1,k) - \frac{T(m+1,p)}{T(q,p)}\,T(q,k)\,. \tag{16.8}$$

By definition of the tableau, $T(m+1,k)$ is the reduced cost associated with the variable k. Assume first that k was in the basis before the pivoting, so that $T(m+1,k) = 0$. The column k of T contains zero values, except in the row corresponding to the basic variable k. As k remains in the basis, it does not correspond to the pivot row q and $T(q,k) = 0$. Therefore, $\bar{T}(m+1,k) = T(m+1,k) = 0$.

Assume now that k is the column of the pivot, that is $k = p$. It therefore corresponds to a basic variable in the tableau $\bar{T}$. By replacing k by p in (16.8), we obtain that the reduced cost is

$$\bar{T}(m+1,p) = T(m+1,p) - \frac{T(m+1,p)}{T(q,p)}T(q,p) = 0\,.$$

Then, all the elements of the last row of $\bar{T}$ corresponding to the basic variables $\bar{B}$ are zero. Consequently, by taking only these columns in (16.7),

$$c_{\bar{B}}^T + h^T\bar{B} = 0\,,$$

which gives

$$h^T = -c_{\bar{B}}^T\bar{B}^{-1}.$$

Including it into (16.7), we obtain

$$\bar{T}_{m+1} = (c^T \mid 0) + h^T(A \mid b) = (c^T \mid 0) - c_{\bar{B}}\bar{B}^{-1}(A \mid b)\,,$$

which corresponds exactly to Definition 16.8.

We are now able to redefine Algorithm 16.2 by using the tableau, and we obtain Algorithm 16.4.

Example 16.13 (Simplex algorithm)**.** We apply the simplex algorithm to the following tableau:

x_1	x_2	x_3	x_4	x_5	x_6		
1	2	2	1	0	0	20	$\alpha_4 = 20$
(2)	1	2	0	1	0	20	$\alpha_5 = 10$
2	2	1	0	0	1	20	$\alpha_6 = 10$
−10	−12	−12	0	0	0	0	

We start examining the last row from left to right until we identify a negative reduced cost. There is one in the first column, so that it is selected as the pivot column, and x_1 is scheduled to enter the basis. The value of α is calculated for each row of the upper part of the tableau such that the entry in the pivot column is positive.

Algorithm 16.4: Simplex algorithm

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160).

Input

T_0, the simplex tableau corresponding to a basic feasible solution.

Output

Boolean indicator U identifying the unbounded problem.

If U is false, T^*, the simplex tableau corresponding to an optimal basic feasible solution.

Initialization

$k := 0$.

Repeat

Examine the reduced costs in the last row of T_k. If they are all non negative, then the tableau is optimal. $T^* = T_k$, U=FALSE. STOP.

Let p be the index of the column corresponding to the negative reduced cost that is the furthest to the left in the tableau.

For each i, calculate the distance to the constraint $x_i \geq 0$, i.e.,

$$\alpha_i = \begin{cases} \dfrac{T(i, n+1)}{T(i, p)} & \text{if } T(i, p) > 0 \\ +\infty & \text{otherwise.} \end{cases}$$

Let q be the smallest index such that $\alpha_q = \min_i \alpha_i$.

If $\alpha_q = +\infty$, the problem is unbounded and no optimal solution exists. U=TRUE. STOP.

The index p is integrated in the basis and the index q is removed. Apply the pivoting (Algorithm 16.3) to the tableau T_k to obtain T_{k+1}.

$k := k + 1$.

Until *STOP*

In this case, the three entries are positive. The smallest α is 10, so that both rows 2 and 3 are candidates to be the pivot row. Applying Bland's rule, we select row 2, corresponding to x_5. The pivot (circled) is therefore in column 1 and row 2. After pivoting, we obtain the following tableau.

x_1	x_2	x_3	x_4	x_5	x_6		
0	1.5	1	1	−0.5	0	10	$\alpha_4 = 20/3$
1	0.5	1	0	0.5	0	10	$\alpha_1 = 20$
0	(1)	−1	0	−1	1	0	$\alpha_6 = 0$
0	−7	−2	0	5	0	100	

The first column of the tableau is now a column of the identity matrix. As the 1 is at the second row, the second row of the tableau corresponds to the variable x_1. Note that the reduced cost of the variable x_1 was -10. Remember that it is the slope of the objective function in the corresponding basic direction. As a step of length $\alpha = 10$ is performed, the value of the objective function decreases by 100 units during this first iteration. This is reflected in the rightmost cell of the last row, that contains the value of the objective function with the opposite sign. We indeed observe that the objective function moved from 0 to -100.

Applying the same procedure to the new tableau, we obtain a pivot in column 2 and row 3 (circled), so that x_2 enters the basis and x_6 leaves it. Note that we have here a degenerate basic feasible solution, and the basic direction happens not to be feasible. The maximum step that can be performed is the smallest value of α, that is $\alpha_6 = 0$. After pivoting, we obtain the following tableau.

x_1	x_2	x_3	x_4	x_5	x_6		
0	0	(2.5)	1	1	−1.5	10	$\alpha_4 = 4$
1	0	1.5	0	1	−0.5	10	$\alpha_1 = 20/3$
0	1	−1	0	−1	1	0	$\alpha_2 = +\infty$
0	0	−9	0	−2	7	100	

Note that the value of the objective function is still -100. Indeed, the algorithm has performed a step of length 0. Even if the tableau is different, it corresponds to the same vertex of the constraint polyhedron.

The leftmost negative reduced cost is in column 3, so that x_3 enters the basis. Note that there are only two positive entries in the column, so that only two α's are calculated. The smallest one is in row 1, corresponding to x_4, that leaves the basis. After pivoting, we obtain the following tableau.

x_1	x_2	x_3	x_4	x_5	x_6		
0	0	1	0.4	0.4	−0.6	4	x_3
1	0	0	−0.6	0.4	0.4	4	x_1
0	1	0	0.4	−0.6	0.4	4	x_2
0	0	0	3.6	1.6	1.6	136	

A step of length 4 in a direction of slope -9 has been performed, so that the value of the objective function has decreased by 36 units. All reduced costs in the last tableau are non negative. We have an optimal solution. The value of the basic variables can be read in the last column of the tableau: $x^* = \begin{pmatrix} 4 & 4 & 4 & 0 & 0 & 0 \end{pmatrix}^T$, with $c^T x^* = -136$.

16.3 The initial tableau

Algorithm 16.4 enables us to solve a linear optimization problem provided that an initial basic feasible solution is known and the associated tableau is calculated. In some cases, this task is simple. In others, it can be quite difficult.

For instance, the identification of a first tableau is simple when the linear optimization problem is given in the form

$$\min c^T x$$

subject to

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

and when *the vector* b *only contains non negative elements*. In this case, we transform the problem into standard form by adding slack variables.

The problem then becomes

$$\min c^T x + 0^T x^s$$

subject to

$$\begin{aligned} Ax + Ix^s &= b \\ x &\geq 0 \\ x^s &\geq 0, \end{aligned}$$

where $x^s \in \mathbb{R}^m$ is the vector of slack variables. The point $x = 0$, $x^s = b$ is a basic feasible solution (because $b \geq 0$), where the slack variables x^s are in the basis and the associated basic matrix is the identity matrix. As $B = I$ and $c_B = 0$, the tableau (16.4) simplifies into

A	b
c^T	0

(16.9)

It is illustrated with the following example.

Example 16.14 (Initial tableau)**.** Consider the problem

$$\min -2x_1 - x_2$$

subject to

$$\begin{aligned} x_1 - x_2 &\leq 2 \\ x_1 + x_2 &\leq 6 \\ x_1, x_2 &\geq 0. \end{aligned}$$

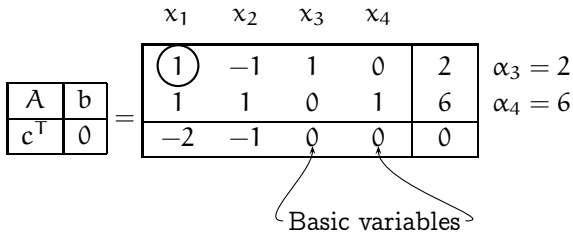
The problem in standard form is obtained after including the slack variables $x_1^s = x_3$ and $x_2^s = x_4$.

$$\min -2x_1 - x_2$$

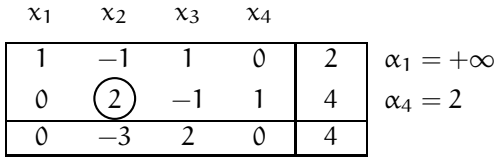
subject to

$$x_1 - x_2 + x_3 = 2$$
$$x_1 + x_2 + x_4 = 6$$
$$x_1, x_2, x_3, x_4 \geq 0.$$

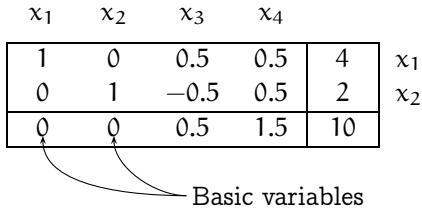
The slack variables are selected to be in the basis, and the first tableau is



We can now apply the simplex algorithm. During the first iteration, x_1 enters the basis, replacing x_3.



During the second iteration, x_2 enters the basis instead of x_4.



The final tableau is optimal as all reduced costs (in the last row) are non negative. The optimal value of the basic variables are available in the last column, so that $x^* = \begin{pmatrix} 4 & 2 & 0 & 0 \end{pmatrix}^T$ and $c^T x^* = -10$.

It is important to note that the condition $b \geq 0$ is restrictive. If b happens to contain a negative element, the corresponding constraint should be multiplied by -1 to obtain a positive element. As the constraints are inequality constraints, the sign of the inequality would therefore change, and the modified constraints are not consistent with the requested format. For instance, the following set of constraints cannot be transformed into the requested form (try it):

$$x_1 - x_2 \leq 2$$
$$x_1 + x_2 \leq -6$$
$$x_1, x_2 \geq 0.$$

In order to identify an initial tableau for any problem, that is, a feasible basic solution, we consider an auxiliary optimization problem. We design it in a way that its initial tableau is easy to construct. We consider the problem in standard form

$$\min c^T x$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0, \end{aligned}$$

and call it problem $(\mathcal{P})$. Here, as we have equality constraints, we can assume without loss of generality that $b \geq 0$. If one of the components of b happens to be negative, we need only multiply the corresponding constraint by -1.

We now create the auxiliary problem. In order to easily obtain a first tableau, we need to select basic variables corresponding to the column of the identity matrix, so that $B = B^{-1} = I$. As such columns may not be present in problem $(\mathcal{P})$, we enforce it and introduce an auxiliary variable for each constraint. We obtain the following constraints for the auxiliary problem, where the identity matrix appears explicitly:

$$\begin{aligned} Ax + Ix^a &= b \\ x, x^a &\geq 0, \end{aligned} \tag{16.10}$$

where $x^a \in \mathbb{R}^m$ is the vector of auxiliary variables. Recall that the objective of the auxiliary problem is to identify a first valid tableau. Therefore, the objective function of problem $(\mathcal{P})$ can be ignored here. Instead, our objective is to get rid of these auxiliary variables, that were added to artificially enforce feasibility. If we denote $e \in \mathbb{R}^m$ the vector of dimension m consisting of only 1, we obtain the following objective function

$$\min x_1^a + x_2^a + \cdots + x_m^a = 0^T x + e^T x^a, \tag{16.11}$$

where the variables of problem $(\mathcal{P})$ play no role. The objective is to give the smallest possible value (that is, 0) to all auxiliary variables. The auxiliary problem with objective function (16.11) and constraints (16.10) is called problem $(\mathcal{A})$. Note that, by construction, the value of the objective function is the sum of the auxiliary variables and is thus always non negative.

We consider x_0, a feasible point of problem $(\mathcal{P})$. As $Ax_0 = b$, it is easy to check that the point $x = x_0$ and $x^a = 0$ is a feasible point of the auxiliary problem $(\mathcal{A})$. The value of the objective function of this point in $(\mathcal{A})$ is the sum of the variables x^a, that is 0. Since zero is also the smallest possible value of the objective function of $(\mathcal{A})$, we are dealing with an optimal solution of problem $(\mathcal{A})$.

If x_0 is a feasible point of $(\mathcal{P})$, then $x = x_0$, $x^a = 0$ is an optimal solution to $(\mathcal{A})$ with a zero value of the objective function. The contrapositive statement is as follows. *If the optimal solution to $(\mathcal{A})$ corresponds to a nonzero, positive, value of the objective function, then $(\mathcal{P})$ has no feasible solution.* As discussed later, this provides us with a convenient way to detect an infeasible problem.

In order to solve $(\mathcal{A})$, we use the simplex algorithm. The point $x = 0$, $x^a = b$ is a basic feasible solution of $(\mathcal{A})$ (because $b \geq 0$) with auxiliary variables in the basis to that $B = I$. The initial tableau is

$$\begin{array}{|c|c|} \hline A & b \\ \hline -\tilde{c}_B^T A | 0 & -\tilde{c}_B^T b \\ \hline \end{array} = \begin{array}{|c|c|} \hline A & b \\ \hline -e^T A | 0 & -e^T b \\ \hline \end{array}$$

where $\tilde{c}_B = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^T = e$ is the vector of the coefficients of the basic variables (here, the auxiliary variables x^a) in the objective function of $(\mathcal{A})$. The last row of the tableau is simply the sum of the elements of the corresponding column, with the sign changed.

Problem $(\mathcal{A})$ can be solved by the simplex algorithm (Algorithm 16.4). The algorithm cannot detect an unbounded problem at step 15, as the objective function of $(\mathcal{A})$ is bounded below by 0. It always produces an optimal solution x^*, x^{a*}. Consequently, one of the following two possibilities occurs:

1. The optimal value of $(\mathcal{A})$ is zero. As it is the sum of the auxiliary variables, which are non negative, each of them have to be zero: $x^{a*} = 0$. As $Ax^* + x^{a*} = Ax^* = b$, x^* is a feasible solution of $(\mathcal{P})$.
2. The optimal value of $(\mathcal{A})$ is positive. This signifies that there is no feasible solution for $(\mathcal{P})$.

Note that the auxiliary problem $(\mathcal{A})$ is always feasible and bounded, with 0 as a lower bound.

In summary, solving the auxiliary problem $(\mathcal{A})$ either provides a feasible solution of $(\mathcal{P})$, or provides a certificate of infeasibility. We illustrate this in an example.

Example 16.15 (Initial point)**.** Consider the problem

$$\min x_1 + x_2 + x_3$$

subject to

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 3 \\ -x_1 + 2x_2 + 6x_3 &= 2 \\ 4x_2 + 9x_3 &= 5 \\ 3x_3 + x_4 &= 1 \\ x_1, x_2, x_3, x_4 &\geq 0\,. \end{aligned}$$

The auxiliary problem is written as

$$\min x_1^a + x_2^a + x_3^a + x_4^a$$

subject to

$$\begin{aligned} x_1 + 2x_2 + 3x_3 + x_1^a &= 3 \\ -x_1 + 2x_2 + 6x_3 + x_2^a &= 2 \\ 4x_2 + 9x_3 + x_3^a &= 5 \\ 3x_3 + x_4 + x_4^a &= 1 \\ x_1, x_2, x_3, x_4, x_1^a, x_2^a, x_3^a, x_4^a &\geq 0\,. \end{aligned}$$

The initial tableau for the auxiliary problem and the iterations of the simplex algorithm (Algorithm 16.4) are listed below.

x_1	x_2	x_3	x_4	x_1^a	x_2^a	x_3^a	x_4^a		
1	2	3	0	1	0	0	0	3	3/2
−1	(2)	6	0	0	1	0	0	2	1
0	4	9	0	0	0	1	0	5	5/4
0	0	3	1	0	0	0	1	1	$+\infty$
0	−8	−21	−1	0	0	0	0	−11	

x_1	x_2	x_3	x_4	x_1^a	x_2^a	x_3^a	x_4^a		
(2)	0	−3	0	1	−1	0	0	1	1/2
−1/2	1	3	0	0	1/2	0	0	1	$+\infty$
2	0	−3	0	0	−2	1	0	1	1/2
0	0	3	1	0	0	0	1	1	$+\infty$
−4	0	3	−1	0	4	0	0	−3	

x_1	x_2	x_3	x_4	x_1^a	x_2^a	x_3^a	x_4^a		
1	0	−3/2	0	1/2	−1/2	0	0	1/2	$+\infty$
0	1	9/4	0	1/4	1/4	0	0	5/4	5/9
0	0	0	0	−1	−1	1	0	0	$+\infty$
0	0	(3)	1	0	0	0	1	1	1/3
0	0	−3	−1	2	2	0	0	−1	

x_1	x_2	x_3	x_4	x_1^a	x_2^a	x_3^a	x_4^a		
1	0	0	1/2	1/2	−1/2	0	1/2	1	x_1
0	1	0	−3/4	1/4	1/4	0	−3/4	1/2	x_2
0	0	0	0	−1	−1	1	0	0	x_3^a
0	0	1	1/3	0	0	0	1/3	1/3	x_3
0	0	0	0	2	2	0	1	0	

Basic variables

The last tableau is optimal since all the reduced costs are non negative. The optimal solution is $x_1 = 1$, $x_2 = 1/2$, $x_3 = 1/3$, $x_4 = 0$, $x_1^a = x_2^a = x_3^a = x_4^a = 0$. The optimal value of the objective function and all the auxiliary variables are zero. It is easy to verify that (x_1, x_2, x_3, x_4) is a feasible point of the initial problem.

The optimal solution to the auxiliary problem renders it possible to identify a feasible point of the initial problem, if such a point exists. However, in order to apply the simplex algorithm (Algorithm 16.4), it is necessary to use the associated simplex tableau. In the case where all the auxiliary variables are non basic, we need only remove the corresponding columns and calculate the reduced costs in order to obtain an initial tableau for the initial problem. It may happen (like in Example 16.15) that the basis corresponding to the optimal tableau of the auxiliary problem contains auxiliary variables. Note that the feasible basic solution is necessarily degenerate

because this (auxiliary) basic variable is zero. Therefore, it can be exchanged with a variable of the original problem that is non basic and, therefore, equal to zero as well. To do so, we choose a column corresponding to a variable from the original problem such that the pivot is non zero and carry out the pivoting which exchanges the two variables in the basis. Since the auxiliary basic variable is zero, the pivoting does not affect the values of the last column of the tableau. Then, the new tableau corresponds exactly to the same feasible solution as the former. Only the basis has changed.

In the case where the matrix A of the initial problem is of full rank, such pivoting is always possible. In Example 16.15, this hypothesis is not satisfied. The third constraint is the sum of the first two and is redundant. If we want to remove the variable x_3^a from the basis, the only possible candidate to take its place in the basis is x_4, the only variable of the original problem that is out of the basis. But the corresponding pivot is zero and the procedure cannot be applied.

x_1	x_2	x_3	x_4	x_1^a	x_2^a	x_3^a	x_4^a		
1	0	0	1/2	1/2	−1/2	0	1/2	1	x_1
0	1	0	−3/4	1/4	1/4	0	−3/4	1/2	x_2
0	0	0	(0)	−1	−1	1	0	0	x_3^a
0	0	1	1/3	0	0	0	1/3	1/3	x_3
0	0	0	0	2	2	0	1	0	

The impossibility of removing an auxiliary variable from the basis is therefore a convenient way to identify a redundant constraint. Since such a constraint can be ignored without modifying the problem, the corresponding row of the matrix can simply be removed, as well as the column of the corresponding auxiliary variable:

x_1	x_2	x_3	x_4	x_1^a	x_2^a	x_4^a		
1	0	0	1/2	1/2	−1/2	1/2	1	x_1
0	1	0	−3/4	1/4	1/4	−3/4	1/2	x_2
0	0	1	1/3	0	0	1/3	1/3	x_3
0	0	0	0	2	2	1	0	

When there are no longer any auxiliary variables in the basis, the corresponding columns can be removed to obtain a tableau.

x_1	x_2	x_3	x_4		
1	0	0	1/2	1	x_1
0	1	0	−3/4	1/2	x_2
0	0	1	1/3	1/3	x_3
—	—	—	—	—	

To obtain a feasible tableau from the initial problem, we need to calculate the elements of the last row. These are reduced costs, defined by (6.166) and the value of the objective function with the opposite sign. We thus obtain an algorithm in two phases (Algorithm 16.5).

Algorithm 16.5: Simplex algorithm in two phases

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160).

Input

The matrix $A \in \mathbb{R}^{m \times n}$.

The vector $b \in \mathbb{R}^m$.

The vector $c \in \mathbb{R}^n$.

Output

Boolean indicator U identifying an unbounded problem.

Boolean indicator F identifying an infeasible problem.

If U and F are false, T^*, the simplex tableau corresponding to an optimal basic feasible solution.

Phase I

By multiplying the relevant constraints by -1, modify the problem such that $b \geq 0$.

Introduce the auxiliary variables $x_1^a, \ldots, x_m^a$ and define

$$T_0 = \begin{array}{c|ccc|c} & x_1 \ \ldots \ x_n & x_1^a \ \ldots & x_m^a & \\ \hline & A & I & & b \\ \hline & -e^T A & 0 & & -e^T b \end{array}$$

where e is the vector of $\mathbb{R}^m$ for which all components are 1.

Solve the auxiliary problem by using the simplex algorithm (Algorithm 16.4) to obtain T_0^*.

If the optimal value of the auxiliary problem is non zero, then F=TRUE. STOP. Otherwise, F=FALSE.

for *each basic auxiliary variable* **do**

Pivot the tableau with Algorithm 16.3 to exchange it with an original variable.

If all the potential pivots are zero, remove the row corresponding to this basic variable. The associated constraint is redundant.

When there are no more basic auxiliary variables, remove the corresponding columns of the tableau to obtain the tableau $\bar{T}_0^*$.

Phase II

Calculate the last row of $\bar{T}_0^*$: for $j = 1, \ldots, n+1$,

$$\bar{T}_0^*(m+1, j) := \begin{cases} c_j - c_B^T B^{-1} A_j & \text{if j is non basic} \\ 0 & \text{if j is basic} \\ -c_B^T B^{-1} b & \text{if j=n+1.} \end{cases}$$

Solve the problem by using the simplex algorithm (Algorithm 16.4) to obtain U and T^*.

Example 16.16 (Simplex algorithm in two phases). Consider the problem

$$\min_{x\in\mathbb{R}^5} 2x_1 + 3x_2 + 3x_3 + x_4 - 2x_5$$

subject to

$$\begin{array}{rcrcrcrcl} x_1 & + & 3x_2 & & & + & 4x_4 & + & x_5 & = & 2 \\ x_1 & + & 2x_2 & & & - & 3x_4 & + & x_5 & = & 2 \\ -x_1 & - & 4x_2 & + & 3x_3 & & & & & = & 1 \\ & & & & & & x_1, x_2, x_3, x_4, x_5 & & & \geq & 0. \end{array}$$

We apply the simplex algorithm in two phases (Algorithm 16.5) with

$$A = \begin{pmatrix} 1 & 3 & 0 & 4 & 1 \\ 1 & 2 & 0 & -3 & 1 \\ -1 & -4 & 3 & 0 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \quad c = \begin{pmatrix} 2 \\ 3 \\ 3 \\ 1 \\ -2 \end{pmatrix}.$$

Phase I

1. Initial tableau of the auxiliary problem:

x_1	$x_{1.5}$	x_3	x_4	x_5	x_1^a	x_2^a	x_3^a		
(1)	3	0	4	1	1	0	0	2	2
1	2	0	−3	1	0	1	0	2	2
−1	−4	3	0	0	0	0	1	1	
−1	−1	−3	−1	−2	0	0	0	−5	

2. The simplex algorithm is applied to solve the auxiliary problem:

x_1	x_2	x_3	x_4	x_5	x_1^a	x_2^a	x_3^a		
1	3	0	4	1	1	0	0	2	
0	−1	0	−7	0	−1	1	0	0	
0	−1	(3)	4	1	1	0	1	3	1
0	2	−3	3	−1	1	0	0	−3	

x_1	x_2	x_3	x_4	x_5	x_1^a	x_2^a	x_3^a		
1	3	0	4	1	1	0	0	2	x_1
0	−1	0	−7	0	−1	1	0	0	x_2^a
0	−1/3	1	4/3	1/3	1/3	0	1/3	1	x_3
0	1	0	7	0	2	0	1	0	

   The last tableau is optimal, and the optimal value is zero. We have identified a basic feasible solution: $x_1 = 2$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$, $x_5 = 0$.

3. The auxiliary variable x_2^a is basic. It is exchanged with x_2.

x_1	x_2	x_3	x_4	x_5	x_1^a	x_2^a	x_3^a	
1	3	0	4	1	1	0	0	2
0	(−1)	0	−7	0	−1	1	0	0
0	−1/3	1	4/3	1/3	1/3	0	1/3	1
0	1	0	7	0	2	0	1	0

x_1	x_2	x_3	x_4	x_5	x_1^a	x_2^a	x_3^a	
1	0	0	−17	1	−2	3	0	2
0	1	0	7	0	1	−1	0	0
0	0	1	3.67	1/3	2/3	−1/3	1/3	1
0	0	0	0	0	1	1	1	0

No auxiliary variable is left in the basis. The tableau is ready to be cleaned.

4. Remove the columns corresponding to the auxiliary variables.

x_1	x_2	x_3	x_4	x_5	
1	0	0	−17	1	2
0	1	0	7	0	0
0	0	1	3.67	1/3	1
0	0	0	0	0	0

Phase II

1. Calculate the last row of the tableau. The vector c is reported above the tableau to facilitate the calculations:

   - $\bar{c}_4 = c_4 - c_B^T(B^{-1}A_4) = 1 - (2 \cdot (-17) + 3 \cdot 7 + 3 \cdot 3.67) = 3$,
   - $\bar{c}_5 = c_5 - c_B^T(B^{-1}A_5) = -2 - (2 \cdot 1 + 3 \cdot 0 + 3 \cdot 1/3) = -5$.

	x_1	x_2	x_3	x_4	x_5	
c =	2	3	3	1	−2	
	1	0	0	−17	1	2
	0	1	0	7	0	0
	0	0	1	3.67	1/3	1
	0	0	0	3	−5	−7

2. Iterations for phase II :

x_1	x_2	x_3	x_4	x_5		
1	0	0	−17	(1)	2	2
0	1	0	7	0	0	
0	0	1	3.67	1/3	1	3
0	0	0	3	−5	−7	

x_1	x_2	x_3	x_4	x_5		
1	0	0	−17	1	2	
0	1	0	(7)	0	0	0
−1/3	0	1	9.33	0	1/3	0.04
5	0	0	−82	0	3	

x_1	x_2	x_3	x_4	x_5		
1	2.43	0	0	1	2	x_5
0	0.14	0	1	0	0	x_4
−1/3	−1.33	1	0	0	1/3	x_3
5	11.71	0	0	0	3	

The last tableau is optimal, because all the reduced costs are non negative. The optimal solution is

$$x_1 = 0, \qquad x_2 = 0, \qquad x_3 = \frac{1}{3}, \qquad x_4 = 0, \qquad x_5 = 2$$

and the optimal value is -3. Note that the optimal solution was already reached at the previous iteration, illustrating that the non negative reduced costs are sufficient but not necessary for optimality (see Theorems 6.30 and 6.31).

The presentation of the proof of Theorems 16.1 and 16.2 is inspired by Bertsimas and Tsitsiklis (1997).

16.4 The revised simplex algorithm

The motivation for developing the tableau in Section 16.2 was to deal with the computational burden of the simplex algorithm, associated with the involvement of the matrix B^{-1} at several stages of the algorithm. Another way to deal with this complexity is to exploit an LU factorization of the matrix B, that is, $PB = LU$ where $P \in \mathbb{R}^{m\times m}$ is a permutation matrix, $L \in \mathbb{R}^{m\times m}$ is a lower triangular matrix, and $U \in \mathbb{R}^{m\times m}$ an upper triangular matrix. Consider each step of Algorithm 16.2 where B^{-1} is involved.

Step 15 Calculation of the reduced costs $c - A^T B^{-T} c_B$. We know from Section 6.5 that the reduced costs of basic variables are zero (Theorem 6.29). The reduced costs for the non basic variables are given by (6.169):

$$\bar{c}_N = c_N - N^T B^{-T} c_B.$$

They can be computed in the following way using the LU factorization of B.

1. Define $z \in \mathbb{R}^m$ as the solution of the triangular system
$$U^T z = c_B.$$
2. Define $y^P \in \mathbb{R}^m$ as the solution of the triangular system
$$L^T y^P = z.$$
3. Permute the vector y^P to obtain $y \in \mathbb{R}^m$:
$$y = P^T y^P.$$
4. Then,
$$\bar{c}_N = c_N - N^T y.$$

Step 19 Calculation of the current iterate $x_B = B^{-1} b$. Using the LU factorization of B, the procedure is as follows.

1. Define $y \in \mathbb{R}^m$ as the solution of the triangular system

$$Ly = Pb.$$

2. Define $x_B \in \mathbb{R}^m$ as the solution of the triangular system

$$Ux_B = y.$$

Step 20 Calculation of the basic component of the p^{th} basic direction $d_B = -B^{-1}A_p$. Using the LU factorization of B, the procedure is as follows.

1. Define $y \in \mathbb{R}^m$ as the solution of the triangular system

$$Ly = -PA_p.$$

2. Define $d_B \in \mathbb{R}^m$ as the solution of the triangular system

$$Ud_B = y.$$

In order for the method to be efficient, the LU factorization of B must not be performed at each iteration. Instead, the factors L and U must be efficiently updated from one iteration to the next. Like in Section 16.2, this also exploits the fact that only one column of the matrix B is modified at each iteration. Various methods in linear algebra may be used to do that. We refer the interested reader to Bartels and Golub (1969), Forrest and Tomlin (1972), Suhl and Suhl (1993), Nocedal and Wright (1999, Chapter 13) for the technical details.

16.5 Exercises

Exercise 16.1. Consider the following optimization problem

$$\min_{x \in \mathbb{R}^2} -3x_1 - 2x_2$$

subject to

$$\begin{aligned} x_1 - x_2 &\geq -2 \\ 2x_1 + x_2 &\leq 8 \\ x_1 + x_2 &\leq 5 \\ x_1 + 2x_2 &\leq 10 \\ x_1 &\geq 0 \\ x_2 &\geq 0. \end{aligned}$$

1. Provide a graphic representation of the feasible set (see Exercise 3.5).
2. Solve the problem graphically.
3. Solve the problem using Algorithm 16.5.
4. Reformulate the same problem with a minimum number of constraints (see Exercise 3.5).
5. Solve the new formulation using Algorithm 16.5.

Exercise 16.2. Solve the following optimization problem using Algorithm 16.5.

$$\begin{aligned} \min_{x \in \mathbb{R}^2} \quad & -9x_1 - 4x_2 \\ & 5x_1 + 2x_2 \leq 31 \\ & -3x_1 + 2x_2 \leq 5 \\ & -2x_1 - 3x_2 \leq -1 \\ & x_1 \geq 0 \\ & x_2 \geq 0. \end{aligned}$$

Exercise 16.3. Consider the following simplex tableau:

x_1	x_2	x_3	x_4	x_5	
$-2/3$	0	δ	0	$1/6$	46
$-1/8$	0	0	1	$5/2$	γ
α	1	ε	η	$-1/6$	4
β	γ	ζ	θ	$1/2$	π

where α, β, γ, δ, ε, ζ, η, θ, γ and π are parameters. The objective function of the optimization problem is $c^T x$, where

$$c^T = (\mu \quad \rho \quad 0 \quad 0 \quad 0),$$

and μ and ρ are also parameters.

1. What are the basic variables in this tableau?
2. Give the values of γ, δ, ε, ζ, η, θ, π and ρ that makes it a valid tableau, and explain why.
3. What are the conditions on the remaining parameters for the optimization problem to be bounded?
4. What are the conditions on the remaining parameters if the tableau corresponds to a degenerate basic solution?
5. What are the conditions on the remaining parameters if the tableau corresponds to a unique optimal solution of the problem?
6. What are the conditions on the remaining parameters if the tableau corresponds to an optimal solution of the problem, and that there exist an infinite number of optimal solutions?

16.6 Project

The general organization of the projects is described in Appendix D.

Objective

The objective of this project is to implement the simplex algorithm in two phases and test the various pivoting rules.

Approach

Apply the algorithms to the problem described below, for different values of n, and compare the number of iterations. For the algorithms comprising random decisions, run the same problem several times to obtain an average performance index.

Algorithms

Algorithms 16.3, 16.4, and 16.5. The following versions of phase 2 of Algorithm 16.4 are tested.

1. Choose an index of the column corresponding to the negative reduced cost that is the furthest to the left in the tableau (rule already described in Algorithm 16.4).
2. Choose the index of the column corresponding to the most negative reduced cost.
3. Carry out a pivoting for each variable corresponding to a negative reduced cost and select the one that generates the most significant reduction in the objective function.
4. Randomly select an index corresponding to a negative reduced cost, by attributing the same probability to each of these indexes.
5. Randomly select an index corresponding to a negative reduced cost, for which the probability of selecting index j is

$$\begin{cases} \dfrac{-\bar{c}_j}{\displaystyle\sum_{\{k|\bar{c}_k<0\}} -\bar{c}_k} & \text{if } \bar{c}_j < 0 \\ 0 & \text{otherwise}. \end{cases}$$

Problems

The following problems are inspired by the ideas of Klee and Minty (1972) to demonstrate that the complexity of the simplex algorithm cannot be polynomial.

Exercise 16.4. The problem

$$\min -\sum_{i=1}^{n} 2^{n-i} x_i$$

subject to

$$\begin{aligned} x_1 &\leq 5 \\ 4x_1 + x_2 &\leq 25 \\ 8x_1 + 4x_2 + x_3 &\leq 125 \\ &\vdots \\ 2^n x_1 + 2^{n-1} x_2 + \cdots + 4x_{n-1} + x_n &\leq 5^n \\ x_1, x_2 &\geq 0. \end{aligned}$$

The optimal solution to this problem is $\begin{pmatrix} 0 & 0 & \ldots 0 & 5^n \end{pmatrix}^T$.

Exercise 16.5. The problem

$$\min -x_n$$

subject to

$$\begin{aligned} \varepsilon \le x_1 &\le 1 \\ \varepsilon x_{i-1} \le x_i &\le 1 - \varepsilon x_{i-1}, \qquad i = 2, \ldots, n, \end{aligned}$$

where $0 < \varepsilon < 1/2$.

Chapter 17

Newton's method for constrained optimization

Contents

In this chapter, Newton's method for unconstrained optimization is adapted to problems with convex constraints:

$$\min_x f(x)$$

subject to

$$x \in X \subseteq \mathbb{R}^n ,$$

where X is a closed convex subset of $\mathbb{R}^n$.

As demonstrated in Section 11.5, Newton's method can be seen as a preconditioned version of the steepest descent method. We proceed in a similar manner for constrained problems: we first present the method by using the steepest descent method and then precondition it in an appropriate manner.

17.1 Projected gradient method

The basic idea is to follow the steepest descent direction, as in the unconstrained case. When we obtain an infeasible point, we project it on the set X. Denote $[\,\cdot\,]^P$ the projection operator. At each iteration, we generate a feasible point y_k from a feasible point x_k

$$y_k = \left[x_k - \gamma_k \nabla f(x_k)\right]^P ,$$

with $\gamma_k > 0$ as the step length. The direction $d_k = y_k - x_k$ is feasible, as both x_k and y_k are feasible, and the feasible set is convex. We show that, if it is non zero, it is a descent direction for any value of $\gamma_k > 0$.

Lemma 17.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function and $X \subseteq \mathbb{R}^n$ a closed convex set. Take $x_k \in X$ and $x(\gamma) = x_k - \gamma \nabla f(x_k)$, with $\gamma > 0$. If the direction*

$$d(\gamma) = \left[x(\gamma)\right]^P - x_k \tag{17.1}$$

is non zero, it is a descent direction, for all $\gamma > 0$.

Proof. Projecting on a convex set consists in solving an optimization problem. In Example 6.6, we have derived the optimality conditions of this problem:

$$\left(\left[x(\gamma)\right]^P - x(\gamma)\right)^T \left(x - \left[x(\gamma)\right]^P\right) \geq 0, \qquad \forall x \in X.$$

Since $x_k \in X$, we have

$$\begin{aligned}\left(\left[x(\gamma)\right]^P - x(\gamma)\right)^T \left(x_k - \left[x(\gamma)\right]^P\right) &= -\left(\left[x(\gamma)\right]^P - x_k + \gamma \nabla f(x_k)\right)^T d(\gamma) \\ &= -\left(d(\gamma) + \gamma \nabla f(x_k)\right)^T d(\gamma) \\ &= -d(\gamma)^T d(\gamma) - \gamma d(\gamma)^T \nabla f(x_k) \\ &\geq 0.\end{aligned}$$

Then, as $\gamma > 0$,

$$d(\gamma)^T \nabla f(x_k) \leq -\frac{d(\gamma)^T d(\gamma)}{\gamma} \leq 0.$$

Since $d(\gamma) \neq 0$ by assumption, we have

$$d(\gamma)^T \nabla f(x_k) < 0$$

and $d(\gamma) = [x(\gamma)]^P - x_k$ is a descent direction. □

The direction $d(\gamma) = \left[x(\gamma)\right]^P - x_k$ has the following properties:

- If $d(\gamma) = 0$, then x_k is a stationary point because the necessary optimality condition (6.8) is satisfied.
- If $d(\gamma) \neq 0$, then $d(\gamma)$ is a descent direction according to Theorem 17.1.
- Since x_k and $\left[x(\gamma)\right]^P$ are feasible, the convexity of X ensures that $x_k + \alpha d(\gamma) \in X$ for any $0 \leq \alpha \leq 1$ (Theorem 3.11).

It is thus easy to generalize the steepest descent algorithm (Algorithm 11.6) to obtain Algorithm 17.1.

Algorithm 17.1: Projected gradient method

1 **Objective**

2 To find (an approximation of) a local minimum of the problem

$$\min_{x \in X \subseteq \mathbb{R}^n} f(x) \tag{17.2}$$

where X is closed, convex and non empty.

3 **Input**

4 The differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

5 The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.

6 The projection operator on X: $[\cdot]^P$.

7 An initial solution $x_0 \in \mathbb{R}^n$.

8 A parameter $\gamma > 0$ (for instance $\gamma = 1$).

9 The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

10 **Output**

11 An approximation of the optimal solution $x^* \in \mathbb{R}$.

12 **Initialization**

13 $k := 0$.

14 **Repeat**

15 $y_k := \left[x_k - \gamma \nabla f(x_k)\right]^P$.

16 $d_k := y_k - x_k$.

17 Determine α_k by applying a line search (Algorithm 11.5) with $\alpha_0 = 1$.

18 $x_{k+1} := x_k + \alpha_k d_k$.

19 $k := k + 1$.

20 **Until** $\|d_k\| \leq \varepsilon$

21 $x^* := x_k$.

It is important to note that step 15 of Algorithm 17.1 can sometimes be as difficult as the initial problem. In fact, y_k is obtained by solving the problem

$$\begin{aligned} y_k &= \operatorname{argmin}_{x \in X} \frac{1}{2} \left\| x_k - \gamma \nabla f(x_k) - x \right\|^2 \\ &= \operatorname{argmin}_{x \in X} \frac{1}{2} \|x - x_k\|^2 + \gamma \nabla f(x_k)^T (x - x_k) \,. \end{aligned} \tag{17.3}$$

This is an optimization problem on a convex set of constraints, with a convex objective function. In some cases, this problem is easy to solve.

In particular, the projection on bound constraints is trivial. Take $\ell, u \in \mathbb{R}^n$ and $X = \left\{x \mid \ell \leq x \leq u\right\}$. Then,

$$\left(\left[x_k - \gamma \nabla f(x_k)\right]^P\right)_i = \begin{cases} \ell_i & \text{if } \left(x_k - \gamma \nabla f(x_k)\right)_i \leq \ell_i \\ \left(x_k - \gamma \nabla f(x_k)\right)_i & \text{if } \ell_i \leq \left(x_k - \gamma \nabla f(x_k)\right)_i \leq u_i \\ u_i & \text{if } x_k - \gamma \nabla f(x_k) \geq u_i \,. \end{cases}$$

If the set X is defined by linear equality constraints $Ax = b$, we take $z = x - x_k$ and (17.3) is written as

$$\min_{z|Az=b-Ax_k} \frac{1}{2} z^T z + \gamma \nabla f(x_k)^T z .$$

The optimal solution is given by Theorem 6.37:

$$z^* = A^T \left(AA^T\right)^{-1} \left(b - A\left(x_k - \gamma \nabla f(x_k)\right)\right) - \gamma \nabla f(x_k)$$

and then $y_k = x_k + z^*$, i.e.,

$$y_k = \left(x_k - \gamma \nabla f(x_k)\right) + A^T \left(AA^T\right)^{-1} \left(b - A\left(x_k - \gamma \nabla f(x_k)\right)\right) . \tag{17.4}$$

We finally note that, in practice, it is also possible to carry out a line search along the projection arc $\left[x_k - \gamma \nabla f(x_k)\right]^P$, rather than in the direction d_k, as we presented in Algorithm 17.1 (see Bertsekas, 1976).

Example 17.2 (Projected gradient algorithm)**.** We apply Algorithm 17.1 to the problem

$$\min_{x \in \mathbb{R}^2} \frac{1}{2} x_1^2 + \frac{9}{2} x_2^2$$

subject to

$$-x_1 + x_2 = -1 .$$

Table 17.1 lists the iterates of the method. The first column contains the iteration number. The second contains the (infeasible) point obtained by following the steepest descent direction (except for iteration 0, for which the point $(5, 1)$ was arbitrarily chosen and where the starting point x_0 is obtained by projecting the former on the constraint). The third column contains the current iterate (always feasible). Finally, the last column lists the norm of d_k, which is used in the stopping criterion.

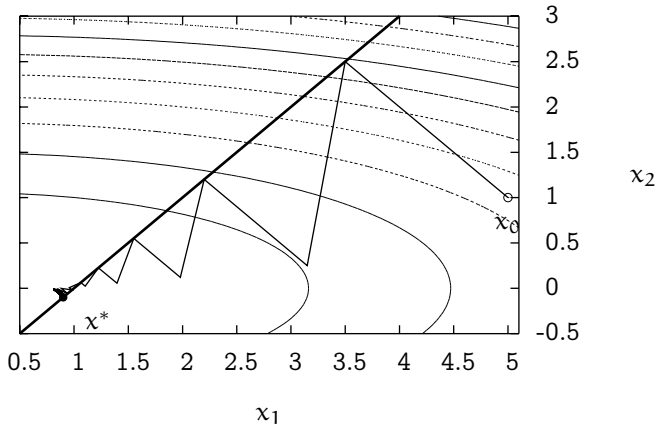
In order to be able to draw the iterates, the algorithm was also run with $\gamma = 0.1$. The iterates are shown in Table 17.2. Figure 17.1 illustrates the iterations. The typical zigzagging of the steepest descent method clearly appears on this example, justifying the need for preconditioning, as discussed in the next section.

Table 17.1: Projected gradient algorithm applied to Example 17.2 ($\gamma = 1$)

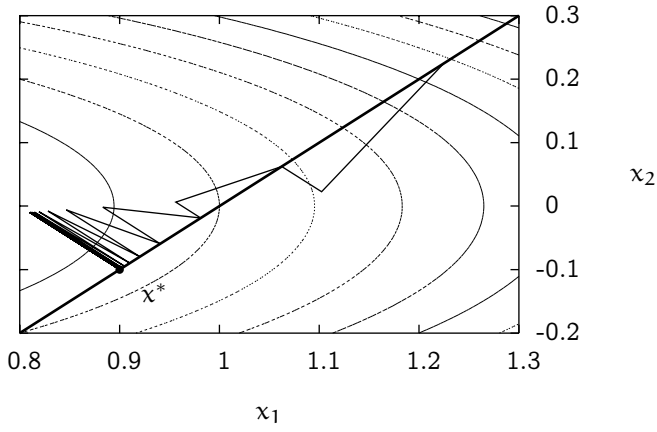
k	$x_{k-1} - \gamma \nabla f(x_{k-1})$		x_k		$\|d_k\|$
0	5.00000000e+00	1.00000000e+00	3.50000000e+00	2.50000000e+00	
1	0.00000000e+00	-2.00000000e+01	2.50000000e-01	-7.50000000e-01	1.83847763e+01
2	0.00000000e+00	6.00000000e+00	1.06250000e+00	6.25000000e-02	4.59619408e+00
3	0.00000000e+00	-5.00000000e-01	8.59375000e-01	-1.40625000e-01	1.14904852e+00
4	0.00000000e+00	1.12500000e+00	9.10156250e-01	-8.98437500e-02	2.87262130e-01
5	0.00000000e+00	7.18750000e-01	8.97460938e-01	-1.02539062e-01	7.18155325e-02
6	0.00000000e+00	8.20312500e-01	9.00634766e-01	-9.93652344e-02	1.79538831e-02
7	0.00000000e+00	7.94921875e-01	8.99841309e-01	-1.00158691e-01	4.48847078e-03
8	0.00000000e+00	8.01269531e-01	9.00039673e-01	-9.99603271e-02	1.12211769e-03
9	0.00000000e+00	7.99682617e-01	8.99990082e-01	-1.00009918e-01	2.80529424e-04
10	0.00000000e+00	8.00079346e-01	9.00002480e-01	-9.99975204e-02	7.01323559e-05
11	0.00000000e+00	7.99980164e-01	8.99999380e-01	-1.00000620e-01	1.75330890e-05
12	0.00000000e+00	8.00004959e-01	8.99999380e-01	-1.00000620e-01	4.38327225e-06

Table 17.2: Projected gradient algorithm applied to Example 17.2 ($\gamma = 0.1$)

k	$x_{k-1} - \gamma \nabla f(x_{k-1})$		x_k		$\|d_k\|$
0	5.00000000e+00	1.00000000e+00	3.50000000e+00	2.50000000e+00	
1	3.15000000e+00	2.50000000e-01	2.20000000e+00	1.20000000e+00	1.83847763e+00
2	1.98000000e+00	1.20000000e-01	1.55000000e+00	5.50000000e-01	9.19238816e-01
3	1.39500000e+00	5.50000000e-02	1.22500000e+00	2.25000000e-01	4.59619408e-01
4	1.10250000e+00	2.25000000e-02	1.06250000e+00	6.25000000e-02	2.29809704e-01
5	9.56250000e-01	6.25000000e-03	9.81250000e-01	-1.87500000e-02	1.14904852e-01
6	8.83125000e-01	-1.87500000e-03	9.40625000e-01	-5.93750000e-02	5.74524260e-02
7	8.46562500e-01	-5.93750000e-03	9.20312500e-01	-7.96875000e-02	2.87262130e-02
8	8.28281250e-01	-7.96875000e-03	9.10156250e-01	-8.98437500e-02	1.43631065e-02
9	8.19140625e-01	-8.98437500e-03	9.05078125e-01	-9.49218750e-02	7.18155325e-03
10	8.14570312e-01	-9.49218750e-03	9.02539063e-01	-9.74609375e-02	3.59077662e-03
11	8.12285156e-01	-9.74609375e-03	9.01269531e-01	-9.87304687e-02	1.79538831e-03
12	8.11142578e-01	-9.87304687e-03	9.00634766e-01	-9.93652344e-02	8.97694156e-04
13	8.10571289e-01	-9.93652344e-03	9.00317383e-01	-9.96826172e-02	4.48847078e-04
14	8.10285645e-01	-9.96826172e-03	9.00158691e-01	-9.98413086e-02	2.24423539e-04
15	8.10142822e-01	-9.98413086e-03	9.00079346e-01	-9.99206543e-02	1.12211769e-04
16	8.10071411e-01	-9.99206543e-03	9.00039673e-01	-9.99603271e-02	5.61058847e-05
17	8.10035706e-01	-9.99603271e-03	9.00019836e-01	-9.99801636e-02	2.80529424e-05
18	8.10017853e-01	-9.99801636e-03	9.00009918e-01	-9.99900818e-02	1.40264712e-05
19	8.10008926e-01	-9.99900818e-03	9.00009918e-01	-9.99900818e-02	7.01323559e-06



(a) Iterations



(b) Zoom

Figure 17.1: Projected gradient algorithm: illustration for Example 17.2 with $\gamma = 0.1$

17.2 Preconditioned projected gradient

We now apply the projected gradient method, but only after first carrying out a change of variables (Definition 2.32).

Take the original problem

$$\min_{x \in X} f(x)$$

and a positive definite matrix H and its Cholesky factorization is LL^T (Definition B.18). We define

$$x' = L^T x \iff x = L^{-T} x' .$$

With the new variables, the problem is written as

$$\min_{x' \in X'} g(x') = f(L^{-T} x')$$

with $X' = \{x' \mid L^{-T} x' \in X\}$. By using Equation (17.3), step 15 of Algorithm 17.1 is written as

$$y'_k = \operatorname{argmin}_{x' \in X'} \frac{1}{2} \left\| x' - x'_k \right\|^2 + \gamma \nabla g(x'_k)^T (x' - x'_k) .$$

In order to write this expression in the original variables, we note that

$$\nabla g(x'_k) = L^{-1} \nabla f(L^{-T} x'_k) = L^{-1} \nabla f(x_k) ,$$

which gives

$$y_k = \operatorname{argmin}_{x \in X} \frac{1}{2} \left(L^T x - L^T x_k \right)^T (L^T x - L^T x_k) + \gamma \nabla f(x_k)^T L^{-T} (L^T x - L^T x_k)$$

or

$$y_k = \operatorname{argmin}_{x \in X} \frac{1}{2} (x - x_k)^T H (x - x_k) + \gamma \nabla f(x_k)^T (x - x_k) . \tag{17.5}$$

Again, the calculation of y_k can be difficult. In a case where X is defined solely by linear equations, we have a quadratic problem for which the analytical solution is given by Theorem 6.38.

In the particular case where $H = \nabla f(x_k)^2 + \tau I$, with τ chosen so that H is positive definite, we obtain *Newton's method for constrained optimization* (Algorithm 17.2). By applying it to Example 17.2, we find convergence in 2 iterations (Table 17.3).

Table 17.3: Newton's method for constrained optimization applied to Example 17.2 ($\gamma = 1$)

k	y_k		x_k		$\|d_k\|$
0	5.0	1.0	3.5	2.5	
1	0.0	-20.0	0.9	-0.1	3.67695526e+00
2	0.0	0.8	0.9	-0.1	3.46944695e-16

Algorithm 17.2: Preconditioned projected gradient method

Objective

To find (an approximation of) a local minimum of the problem

$$\min_{x \in X \subseteq \mathbb{R}^n} f(x)\,, \tag{17.6}$$

where X is convex, closed and non empty.

Input

The differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.

The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.

A family of preconditioners $(H_k)_k$ such that H_k is positive definite for any k.

An initial solution $x_0 \in \mathbb{R}^n$.

A parameter $\gamma > 0$ (for instance $\gamma = 1$).

The required precision $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$.

Output

An approximation of the optimal solution $x^* \in \mathbb{R}$.

Initialization

$k := 0$.

Repeat

Calculate y_k by solving

$$y_k = \operatorname{argmin}_{x \in X} \frac{1}{2}(x - x_k)^T H_k (x - x_k) + \gamma \nabla f(x_k)^T (x - x_k)\,.$$

$d_k := y_k - x_k$.

Determine α_k by applying a line search (Algorithm 11.5) with $\alpha_0 = 1$.

$x_{k+1} := x_k + \alpha_k d_k$.

$k := k + 1$.

Until $\|d_k\| \leq \varepsilon$

$x^* := x_k$.

17.3 Dikin's method

Consider the linear optimization problem

$$\min_x c^T x$$

subject to

$$Ax = b$$
$$x \geq 0$$

and let us assume that the problem is bounded, and there there exists a feasible vector x such that $x > 0$. It is possible to apply the ideas of the preconditioned projected gradient method to this problem. It is important to note that the constraint $x \geq 0$ complicates the problem giving it a combinatorial dimension. Contrary to the simplex method that tries to identify which variables are zero at the optimal solution, we here work solely with strictly feasible iterates, i.e., such that $x > 0$.

Take x_k feasible and positive, and a positive definite matrix H. We apply an iteration of the preconditioned projected gradient method. We use (17.5) with $\nabla f(x_k) = c$, and obtain

$$y_k = \operatorname{argmin}_{x|Ax=b} \frac{1}{2}(x - x_k)^T H(x - x_k) + \gamma c^T(x - x_k),$$

for which the optimal solution is given by Theorem 6.38:

$$y_k = x_k - \gamma H^{-1}(c - A^T\lambda)$$

with

$$\lambda = \left(AH^{-1}A^T\right)^{-1} AH^{-1}c.$$

The step length γ is chosen sufficiently small so that $y_k > 0$ and y_k is then strictly feasible. An iteration of the preconditioned projected gradient proceeds in the direction $y_k - x_k$ with a step of length $\bar{\alpha}$, that is,

$$\begin{aligned} x_{k+1} &= x_k + \bar{\alpha}_k(y_k - x_k) = x_k - \bar{\alpha}_k\gamma H^{-1}(c - A^T\lambda) \\ &= x_k - \alpha_k H^{-1}(c - A^T\lambda), \end{aligned}$$

where $\alpha_k = \bar{\alpha}_k\gamma$.

In order to guarantee that $x_{k+1} > 0$, we choose $\alpha_k = \beta\alpha_{\max}$, with $0 < \beta < 1$ and

$$\alpha_{\max} = \max\left\{\alpha \mid x_k - \alpha H^{-1}(c - A^T\lambda) \geq 0\right\}.$$

In practice, β is often selected between `0.9` and `0.999`, but other values are also possible. Below, we illustrate the algorithm with $\beta = 0.9$ and $\beta = 0.5$.

For the method to work, we need to select the matrix H. Since the objective function is linear, the choice $H = \nabla^2 f(x_k)$ is not appropriate, as $\nabla^2 f(x_k) = 0$. The method proposed by Dikin (1967) consists in choosing

$$H^{-1} = \operatorname{diag}(x_k)^2 = \begin{pmatrix} (x_k)_1^2 & & 0 \\ & \ddots & \\ 0 & & (x_k)_n^2 \end{pmatrix}.$$

We obtain Algorithm 17.3.

Algorithm 17.3: Dikin's method

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160).

Input

The matrix $A \in \mathbb{R}^{m \times n}$.

The vector $b \in \mathbb{R}^m$.

The vector $c \in \mathbb{R}^n$.

An initial solution x_0 such that $Ax_0 = b$ and $x_0 > 0$.

A parameter β such that $0 < \beta < 1$ (by default, $\beta = 0.9$).

The required precision $\varepsilon \in \mathbb{R}$.

Output

A Boolean indicator U identifying an unbounded problem.

If U is false, an approximation of the optimal solution x^*.

Initialization

$k := 0$.

Repeat

$H^{-1} := \operatorname{diag}(x_k)^2$.

$\lambda := \left(AH^{-1}A^T\right)^{-1} AH^{-1}c$.

$d := -H^{-1}(c - A^T\lambda)$.

For each $i = 1, \ldots, n$, calculate

$$\alpha_i := \begin{cases} \dfrac{-(x_k)_i}{d_i} & \text{if } d_i < 0 \\ +\infty & \text{otherwise.} \end{cases}$$

$\alpha_{max} := \min_i \alpha_i$.

if $\alpha_{max} = \infty$ **then**

the problem is unbounded. U = TRUE. STOP.

$x_{k+1} := x_k + \beta\alpha_{max}d$.

$k := k + 1$.

Until $\|d\| \leq \varepsilon$

$x^* := x_k$.

Example 17.3 (Dikin's method)**.** Consider the problem

$$\min x_1 + 2x_2 + 3x_3$$

subject to

$$x_1 + x_2 + x_3 = 1$$
$$x \geq 0\,.$$

The optimal solution is $x^* = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$. The iterations of Dikin's method starting from $x_0 = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix}^T$ are shown in Figure 17.2 for $\beta = 0.9$ and Figure 17.3 for $\beta = 0.5$.

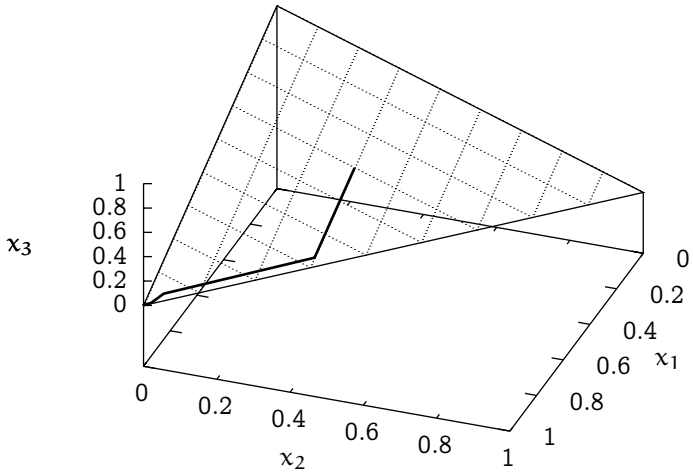


Figure 17.2: Dikin's method for Example 17.3 ($\beta = 0.9$)

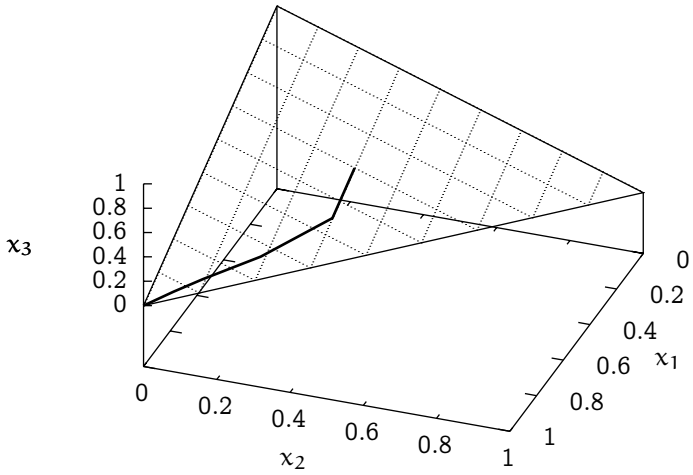


Figure 17.3: Dikin's method for Example 17.3 ($\beta = 0.5$)

Finally, Table 17.4 represents the iterations of Dikin's method for Example 16.7.

Dikin's method is a precursor for *interior point methods*. In fact, all the iterates of this method are interior points (Definition 1.15) in the subspace defined by $Ax = b$. Then, the directions obtained are automatically feasible for the constraints $x \geq 0$. In the following chapter, we study these methods in more detail.

Table 17.4: Iterations of Dikin's method for Example 16.7

x_1	x_2	x_3	x_4	x_5	x_6	$\|d\|$
3.000000e+00	3.000000e+00	3.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	
3.103540e+00	4.099115e+00	4.099115e+00	5.000000e-01	1.495575e+00	1.495575e+00	9.137113e+01
3.956006e+00	3.979477e+00	3.979477e+00	1.260874e-01	1.495575e-01	1.495575e-01	5.325931e+00
3.944714e+00	4.010669e+00	4.010669e+00	1.260874e-02	7.856374e-02	7.856374e-02	7.977189e-02
3.998821e+00	3.998167e+00	3.998167e+00	8.510000e-03	7.856374e-03	7.856374e-03	1.607856e-02
3.996546e+00	4.000651e+00	4.000651e+00	8.510000e-04	4.955183e-03	4.955183e-03	3.282717e-04
3.999937e+00	3.999877e+00	3.999877e+00	5.550455e-04	4.955183e-04	4.955183e-04	6.386205e-05
3.999778e+00	4.000042e+00	4.000042e+00	5.550455e-05	3.185682e-04	3.185682e-04	1.388727e-06
3.999996e+00	3.999992e+00	3.999992e+00	3.592153e-05	3.185682e-05	3.185682e-05	2.638521e-07

17.4 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of this project is, first, to implement and analyze the preconditioned gradient method and, second, to implement Dikin's method and compare it with the simplex method.

Approach

Projected gradient. The algorithm of the preconditioned projected gradient (Algorithm 17.2) is applied to the non linear problem described below. The idea is to analyze the behavior of the algorithm for the following families of preconditioners:

1. $H_k = I$ for any k.
2. $H_k = (\nabla^2 f(x_k) + \tau I)^{-1}$, where τ is selected so that H_k is positive definite (use the modified Cholesky factorization, Algorithm 11.7).
3. $H_k = \operatorname{diag}\left(\min\left(1, 1/\left|(x_k)_1\right|\right), \ldots, \min\left(1, 1/\left|(x_k)_n\right|\right)\right)$ is a diagonal matrix.

It is interesting to modify the value of the step γ and test, for instance, $\gamma = 0.1,\ 1,\ 10$.

Dikin. Dikin's method (Algorithm 17.3) is implemented with several values of β, for instance, $\beta = 0.1, 0.5, 0.9, 0.99$ and 0.999. Compare the performance of Dikin's method and of the simplex method for different values of n in the linear problems described below. In particular, try a value of n that is as large as possible.

Algorithms

Algorithms 17.2, 17.3, and 11.7.

Problems

Exercise 17.1. *Non linear* (Jansson and Knüppel, 1992):

$$\min_{x \in \mathbb{R}^2} x_1^2 - 12\,x_1 + 10\,\cos\left(\pi \frac{x_1}{2}\right) + 8\sin(\pi 5 x_1) - \frac{\exp\left(-(x_2 - 5)^2/2\right)}{\sqrt{5}}$$

subject to

$$-30 \leq x_1 \leq 30$$
$$-10 \leq x_2 \leq 10\,.$$

Exercise 17.2. *Linear* (i):

$$\min - \sum_{i=1}^{n} 2^{n-i} x_i$$

subject to

$$\begin{aligned} x_1 &\leq 5 \\ 4x_1 + x_2 &\leq 25 \\ 8x_1 + 4x_2 + x_3 &\leq 125 \\ &\vdots \\ 2^n x_1 + 2^{n-1} x_2 + \cdots + 4x_{n-1} + x_n &\leq 5^n \\ x_1, x_2, \ldots, x_n &\geq 0\,. \end{aligned}$$

Use the starting point $\begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix}^T$. The optimal solution to this problem is $\begin{pmatrix} 0 & 0 & \ldots & 0 & 5^n \end{pmatrix}^T$.

Exercise 17.3. *Linear* (ii):

$$\min -x_n$$

subject to

$$\begin{aligned} \varepsilon \leq x_1 &\leq 1 \\ \varepsilon x_{i-1} \leq x_i &\leq 1 - \varepsilon x_{i-1}\,, \qquad i = 2, \ldots, n\,. \end{aligned}$$

Use the starting point

$$\begin{aligned} x_1 &= \frac{1+\varepsilon}{2} \\ x_i &= \frac{1}{2}\,, \qquad i = 2, \ldots, n\,, \end{aligned}$$

with $0 < \varepsilon < 1/2$.

Chapter 18

Interior point methods

Contents

18.1 Barrier methods

The simplex method moves from one vertex of the constraint polyhedron to the next. It can, in extreme cases, require a large number of iterations. In the context of the theory of algorithm complexity, we speak of an *exponentially large* number of iterations, because the number of iterations in the worst case increases exponentially with the size of the problem to be solved. Interior point methods, as their name indicates, methodically avoid the border of the feasible set and thus do not suffer from the combinatorial aspect inherent to the simplex method. Khachiyan (1979) was the first to propose an algorithm which he showed to be *polynomial*, i.e., for which the number of iterations increases polynomially with the size of the problem. Unfortunately, this algorithm proved ineffective in practice and it took the work of Karmarkar (1984) to create enthusiasm around interior point methods.

Currently, the importance of these methods exceeds the framework of linear optimization and they are widely used in the context of convex optimization thanks to the work by Nesterov and Nemirovsky (1994). The reader is referred to Boyd and Vandenberghe (2004) for more details.

Here, we motivate the development of interior point methods through barrier methods, as they provide an intuitive interpretation.

Consider the problem

$$\min f(x) \qquad \text{subject to} \quad x \in X,\ g(x) \leq 0, \tag{18.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ and X is a closed set. The set of feasible points is

$$\mathcal{F} = \{x \in \mathbb{R}^n \mid x \in X,\ g(x) \leq 0\}. \tag{18.2}$$

In this context, we call the set of interior points the set $\mathcal{S}$ defined by

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid x \in X,\ g(x) < 0\}, \tag{18.3}$$

and we assume that it is not empty. Note that the points in $\mathcal{S}$ are not interior points of the set $\mathcal{F}$ (in the sense of Definition 1.15). They are interior within the set X, with respect to the constraint $g(x) \leq 0$.

We assume that

- $\mathcal{S} \neq \emptyset$,
- any feasible point can be arbitrarily well approximated by an interior point, i.e., for any $x \in \mathcal{F}$ and any $\varepsilon > 0$, there exists $\widetilde{x} \in \mathcal{S}$ such that

$$\|\widetilde{x} - x\| \leq \varepsilon.$$

If X is a convex set and g is a convex function, this hypothesis is always satisfied.

Lemma 18.1. *Let $X \subset \mathbb{R}^n$ be a closed convex set. Let $g : \mathbb{R}^n \to \mathbb{R}^m$ be a convex function. Let $\mathcal{F}$ be defined by (18.2) and $\mathcal{S}$ defined by (18.3). For all $x \in \mathcal{F}$ and for all $\varepsilon > 0$, there exists $\widetilde{x} \in \mathcal{S}$ such that*

$$\|\widetilde{x} - x\| \leq \varepsilon.$$

Proof. If $x \in \mathcal{S}$, the property is trivially satisfied with $\widetilde{x} = x$. Take $x \in \mathcal{F} \setminus \mathcal{S}$, i.e., so that $x \in X$ and so that there exist indices $i_1, \ldots, i_k$, $k \leq n$, such that $g(x)_{i_j} = 0$ for $j = 1, \ldots, k$. Without loss of generality, we assume that $k = n$. Indeed, the other indices do not pose a problem, and we can always choose $\widetilde{x}_i = x_i$ for them. Then, $g(x) = 0$. Take $y \in \mathcal{S}$. By convexity of X, $\lambda x + (1-\lambda)y \in X$ for any $0 \leq \lambda \leq 1$. Moreover, by convexity of g, we have

$$g\big(\lambda x + (1-\lambda)y\big) \leq \lambda g(x) + (1-\lambda)g(y) = (1-\lambda)g(y).$$

Therefore, since $g(y) < 0$, we have $g\big(\lambda x + (1-\lambda)y\big) < 0$ for any $\lambda < 1$ and $\widetilde{x} = \lambda x + (1-\lambda)y \in \mathcal{S}$. To obtain $\varepsilon \geq \|\widetilde{x} - x\|$, we need

$$\begin{aligned} \varepsilon^2 &\geq \big\|(\lambda - 1)x + (1-\lambda)y\big\|^2 \\ &= (1-\lambda)^2 \|x - y\|^2. \end{aligned}$$

Since $1 - \lambda > 0$, this equals

$$1 - \lambda \leq \frac{\varepsilon}{\|x - y\|}$$

or

$$\lambda \geq 1 - \frac{\varepsilon}{\|x - y\|}.$$

Since $x \neq y$, for all $\varepsilon > 0$, it is possible to find a $\lambda < 1$ such that $\widetilde{x} = \lambda x + (1-\lambda)y \in \mathcal{S}$. □

The interior point methods employ functions known as *barrier functions* in order to force the algorithms to remain in $\mathcal{S}$. A barrier function is defined on $\mathcal{S}$ and is going to infinity as x approaches the border of the set.

Definition 18.2 (Barrier function)**.** Let $X \subset \mathbb{R}^n$ be a closed set. Let $g : \mathbb{R}^n \to \mathbb{R}^m$ be a convex function. Let $\mathcal{S}$ be defined by (18.3). A function $B : S \to \mathbb{R}$ is a barrier function if it is continuous and if

$$\lim_{x \in S,\, g(x) \to 0} B(x) = +\infty\,. \tag{18.4}$$

Example 18.3 (Barrier function)**.** The most used barrier functions are the *logarithmic* function

$$B(x) = -\sum_{j=1}^{m} \ln\bigl(-g_j(x)\bigr) \tag{18.5}$$

and the *inverse* function

$$B(x) = -\sum_{j=1}^{m} \frac{1}{g_j(x)}\,. \tag{18.6}$$

Consider the constraints defined by $1 \leq x \leq 3$. They are defined by $g : \mathbb{R} \to \mathbb{R}^2$, with $g_1(x) = 1 - x$ and $g_2(x) = x - 3$. The logarithmic barrier function for these constraints are written as

$$-\ln(x-1) - \ln(3-x)$$

and the inverse barrier function is

$$\frac{1}{x-1} + \frac{1}{3-x}\,.$$

By multiplying this function by a parameter ε, we can control the height of the barrier, as illustrated in Figure 18.1 for the logarithmic barrier and in Figure 18.2 for the inverse barrier.

A barrier method consists in combining the objective function of the problem with the barrier function and progressively decreasing the height of the latter. We define a set of parameters $(\varepsilon_k)_k$ such that

- $0 < \varepsilon_{k+1} < \varepsilon_k$, $k = 0, 1, \ldots,$
- $\lim_k \varepsilon_k = 0$.

At each iteration, the following problem is solved:

$$x_k \in \operatorname{argmin}_{x \in \mathcal{S}} f(x) + \varepsilon_k B(x)\,. \tag{18.7}$$

At first glance, this technique seems ineffective. In fact, we need to solve a non linear constrained problem *at each iteration*. However, the structure of this problem, and

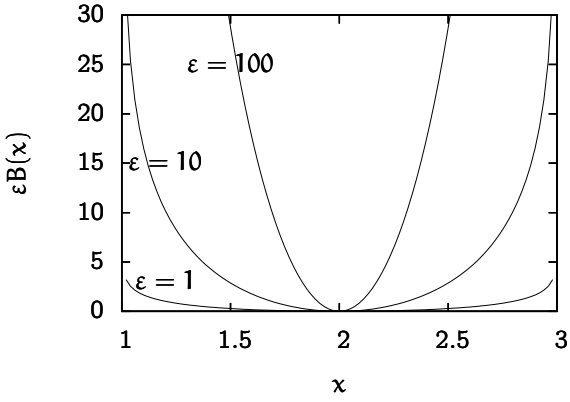


Figure 18.1: Logarithmic barrier

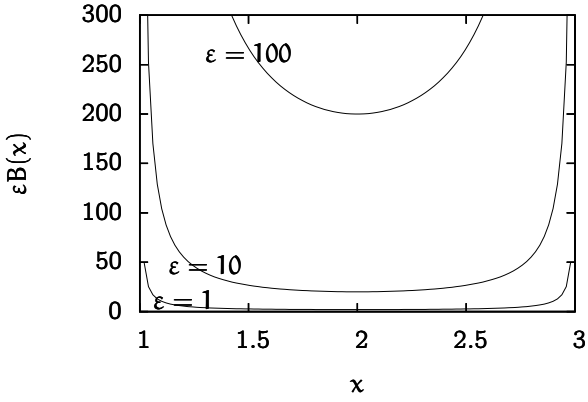


Figure 18.2: Inverse barrier

especially the presence of the barrier function, enable us to employ effective methods. For instance, if $X = \mathbb{R}^n$, the problem to solve is written as

$$x_k \in \operatorname{argmin}_{g(x)<0} f(x) + \varepsilon_k B(x)\,. \tag{18.8}$$

It is possible to solve this problem by using the unconstrained optimization methods described in Part IV. Indeed, since these methods are descent methods, the presence of a sufficiently high barrier and an appropriate selection of the step along the current descent direction prevent the generation of iterates outside of $\mathcal{S}$ and the constraints can be ignored. Similarly, if X is a convex set, the methods described in Chapter 17 can be used.

An essential element for the proper functioning of this type of method is the speed at which the set $(\varepsilon_k)_k$ tends to 0, i.e., the speed at which we decrease the height of

the barrier. If it is reduced too fast, an unconstrained algorithm (or an algorithm for convex constraints) may generate a point outside of $\mathcal{S}$. Good interior point methods are based on a reduction of ε_k that is neither too large, to avoid generating infeasible iterates, nor too small, to avoid a slow convergence of the method.

Example 18.4 (Barrier method)**.** Consider the problem

$$\min f(x) = \frac{1}{2}(x_1^2 + x_2^2)$$

with the constraint

$$x_1 \geq 2\,,$$

for which the optimal solution is $x^* = \begin{pmatrix} 2 & 0 \end{pmatrix}^T$. By taking the logarithmic barrier, (18.7) is written as

$$x_k \in \operatorname{argmin}_{x_1 > 2} \frac{1}{2}(x_1^2 + x_2^2) - \varepsilon_k \ln(x_1 - 2)\,. \tag{18.9}$$

We zero the gradient of the objective function

$$\begin{pmatrix} x_1 - \dfrac{\varepsilon_k}{x_1 - 2} \\ x_2 \end{pmatrix} = 0\,.$$

The first component is zero if $x_1^2 - 2x_1 - \varepsilon_k = 0$, i.e., if

$$x_1 = 1 + \sqrt{1 + \varepsilon_k} \qquad \text{or} \qquad x_1 = 1 - \sqrt{1 + \varepsilon_k}\,.$$

Only the first value is feasible. The second component is zero if $x_2 = 0$. Therefore, the minimum is unique and we obtain

$$x_k = \begin{pmatrix} 1 + \sqrt{1 + \varepsilon_k} \\ 0 \end{pmatrix}.$$

We clearly see that

$$\lim_{k \to \infty} x_k = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

because

$$\lim_{k \to \infty} \varepsilon_k = 0\,.$$

The function to minimize in (18.9) is shown in Figure 18.3 for $\varepsilon = 0.3$. We can follow the evolution of the level curves of this function for different values of ε, as well as the value of the first component of x_k:

ε	$(x_k)_1$	Figure
0.300	2.140175425	18.4(a)
0.150	2.072380529	18.4(b)
0.095	2.046422477	18.4(c)
0.030	2.014889157	18.4(d)
0.003	2.001498877	18.4(e)
0.000	2.000000000	18.4(f)

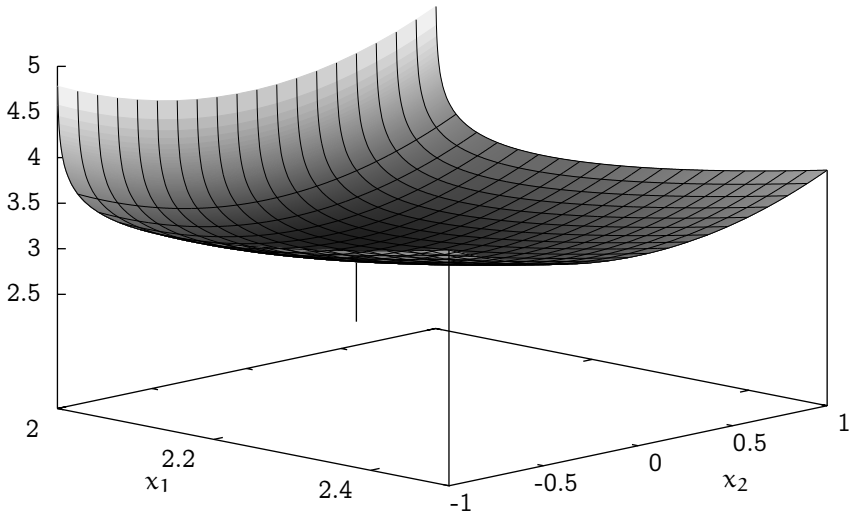


Figure 18.3: $\varepsilon = 0.3$

Theorem 18.5 (Convergence of the barrier method)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be differentiable,* $X \subseteq \mathbb{R}^n$ *closed and* $g : \mathbb{R}^n \to \mathbb{R}^m$ *differentiable. Consider the sets* $\mathcal{F}$ *and* $\mathcal{S}$ *defined by (18.2) and (18.3), respectively. We assume that* $\mathcal{S} \neq \emptyset$ *and that any feasible point can be approximated arbitrarily closely by an interior point, i.e., for any* $x \in \mathcal{F}$ *and for any* $\varepsilon > 0$*, there exists* $\widetilde{x} \in \mathcal{S}$ *such that*

$$\|\widetilde{x} - x\| \leq \varepsilon \,.$$

Consider the set $\left(x_k\right)_k$ *such that*

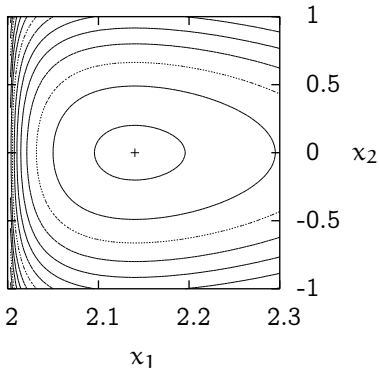
$$x_k \in \operatorname{argmin}_{x \in \mathcal{S}} f(x) + \varepsilon_k B(x) \,,$$

where $B : \mathbb{R}^n \to \mathbb{R}$ *is a barrier function, according to Definition 18.2 and* $\left(\varepsilon_k\right)_k$ *is such that* $\varepsilon_k > \varepsilon_{k+1}$*,* $\forall k$*, with* $\lim_{k\to\infty} \varepsilon_k = 0$*. Then, any limit point of the sequence* $\left(x_k\right)_k$ *is a global minimum of the optimization problem*

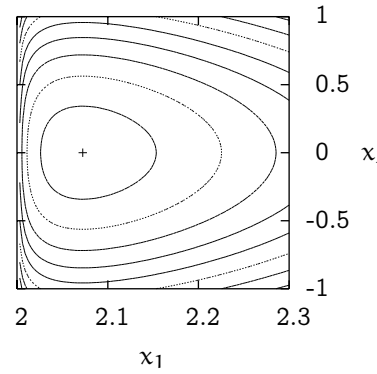
$$\min f(x)$$

subject to

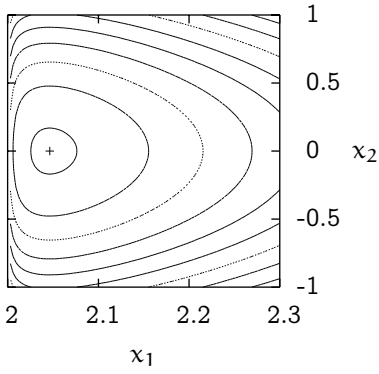
$$x \in \mathcal{F} \,.$$



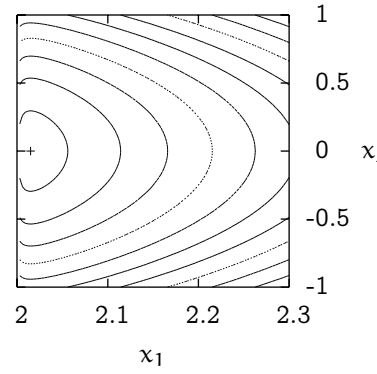
(a) $\varepsilon = 0.3$



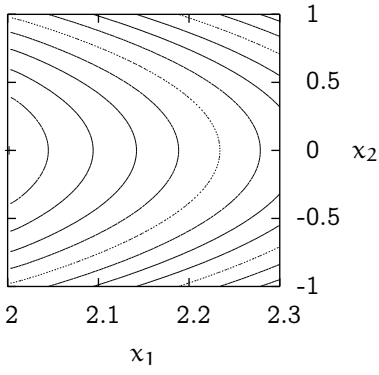
(b) $\varepsilon = 0.15$



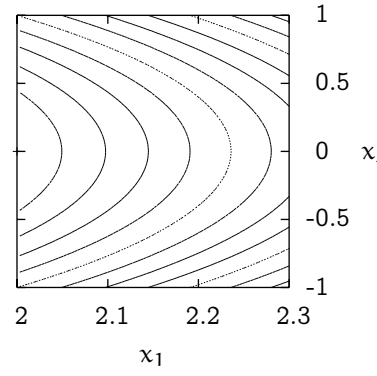
(c) $\varepsilon = 0.095$



(d) $\varepsilon = 0.03$



(e) $\varepsilon = 0.003$



(f) $\varepsilon = 0$

Figure 18.4: Level curves of the function (18.9)

Proof. Let $\bar{x}$ be the limit of a convergent subsequence $(x_k)_{k\in K}$ and then a limit point of the sequence $(x_k)_k$. If $\bar{x} \in \mathcal{S}$, then $B(\bar{x}) < \infty$ and

$$\lim_{k\in K} \varepsilon_k B(x_k) = 0\,.$$

If $\bar{x} \notin \mathcal{S}$, then $\lim_{k\in K} B(x_k) = +\infty$. In both cases, we have

$$\liminf_{k\in K} \varepsilon_k B(x_k) \geq 0\,. \tag{18.10}$$

Then,

$$\liminf_{k\in K} f(x_k) + \varepsilon_k B(x_k) = f(\bar{x}) + \liminf_{k\in K} \varepsilon_k B(x_k) \geq f(\bar{x})\,.$$

Since X is closed and $\bar{x}$ is the limit of points in $\mathcal{S}$, $\bar{x}$ is feasible. We assume by contradiction that it is not a global minimum. Therefore, there exists $x^* \in \mathcal{F}$ such that $f(x^*) < f(\bar{x})$. By assumption, x^* may be approached arbitrarily closely by an interior point. Therefore, there exists $\widetilde{x} \in \mathcal{S}$ such that $f(\widetilde{x}) < f(\bar{x})$.

By definition of x_k, we have

$$f(x_k) + \varepsilon_k B(x_k) \leq f(\widetilde{x}) + \varepsilon_k B(\widetilde{x})\,.$$

When $k \to \infty$, $k \in K$, we obtain

$$f(\bar{x}) + \liminf_{k\in K} \varepsilon_k B(x_k) \leq f(\widetilde{x})\,.$$

According to (18.10), we have $f(\bar{x}) \leq f(\widetilde{x})$, which leads to a contradiction and proves the result. □

In practice, this type of method is rarely used. We now develop these ideas in the specific case of linear optimization.

18.2 Linear optimization

The barrier methods, or interior point methods, have proven particularly effective in the context of convex optimization in general and in the context of linear optimization in particular.

Consider the linear problem

$$\min c^T x$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0\,. \end{aligned}$$

This is a version of (18.1) with $f(x) = c^T x$, $X = \{x \mid Ax = b\}$ and $g(x) = -x$. The set of interior points is defined by $\mathcal{S} = \{x \mid Ax = b\,,\ x > 0\}$, which is assumed to be non empty. By using the logarithmic barrier function, we define

$$x_\varepsilon = \operatorname{argmin}_{x\in\mathcal{S}} c^T x - \varepsilon \sum_{i=1}^{n} \ln x_i\,. \tag{18.11}$$

When ε tends toward infinity, the objective function $c^T x$ plays no role and only the barrier on the constraints is minimized. We then obtain a point x_∞ called the *analytical center* of the constraint polyhedron.

Definition 18.6 (Analytical center)**.** Consider a polyhedron represented in standard form with a non empty interior $\mathcal{P} = \{x \mid Ax = b\,,\ x \geq 0\}$. The analytical center x_∞ of $\mathcal{P}$ is defined by

$$x_\infty = \operatorname{argmin}_{Ax=b,\, x>0} -\sum_{i=1}^{n} \ln x_i\,. \tag{18.12}$$

Example 18.7 (Analytical center)**.** Consider the linear problem:

$$\min x_1 + 2x_2 + 3x_3$$

subject to

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_i &\geq 0\,, \quad i = 1,2,3\,. \end{aligned}$$

The analytical center of the constraint polyhedron is given by

$$x_\infty = \operatorname{argmin}_{x_1+x_2+x_3=1,\, x>0} -\ln x_1 - \ln x_2 - \ln x_3\,.$$

By replacing x_3 with $1 - x_1 - x_2$, we obtain

$$x_\infty = \operatorname{argmin}_{x>0} -\ln x_1 - \ln x_2 - \ln(1 - x_1 - x_2)\,.$$

The gradient of the function is

$$\begin{pmatrix} -\dfrac{1}{x_1} + \dfrac{1}{1 - x_1 - x_2} \\ -\dfrac{1}{x_2} + \dfrac{1}{1 - x_1 - x_2} \end{pmatrix},$$

which is zero at $x_1 = 1/3$ and $x_2 = 1/3$. Since $x_3 = 1 - x_1 - x_2$, we also have $x_3 = 1/3$. Since all the components are positive, we have an analytical center. It is shown in Figure 18.5.

Then, from Theorem 18.5, when ε decreases towards 0, the point x_ε approaches the optimal solution to the linear problem. The trajectory followed by point x_ε is called a *central path*. We note that this concept is used in the following in a broader context, involving dual problems. Therefore, we call it here the *primal central path*. For Example 18.7, the primal central path is shown in Figure 18.6.

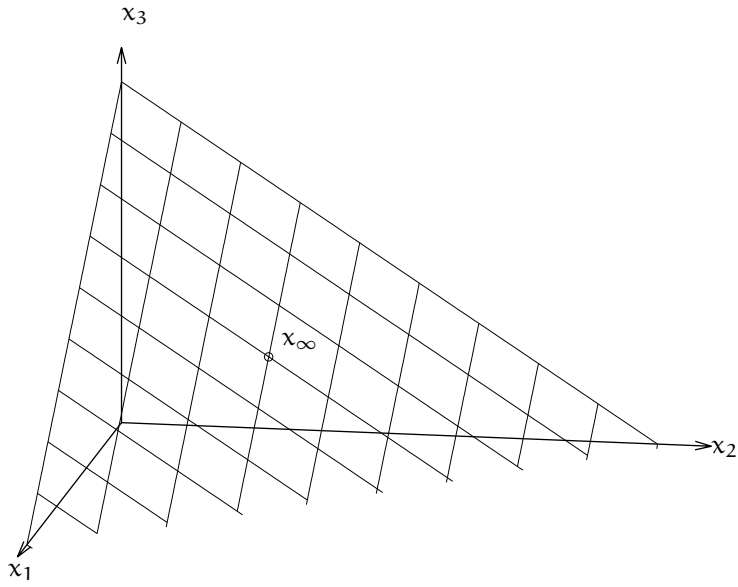


Figure 18.5: Analytical center of Example 18.7

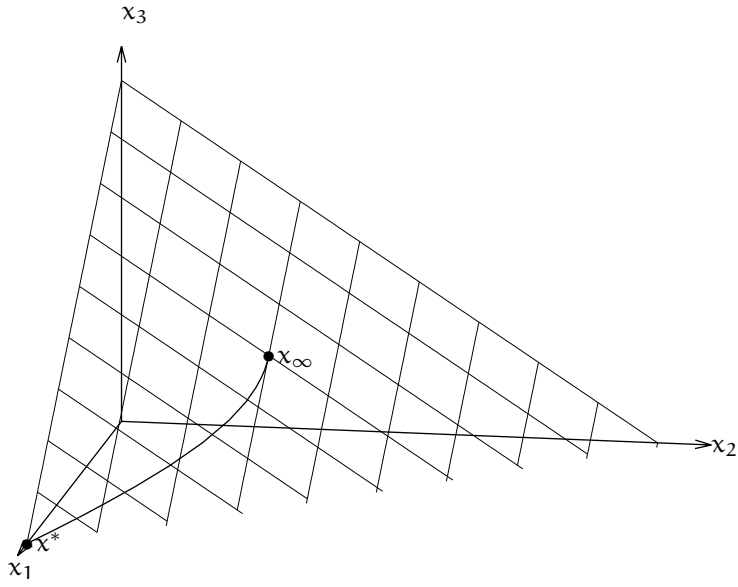


Figure 18.6: Central path for Example 18.7

Definition 18.8 (Primal central path)**.** Consider the linear problem

$$\min c^T x$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0\,. \end{aligned}$$

The primal central path is the curve described by

$$x_\varepsilon = \operatorname{argmin}_{x \in \mathcal{S}} c^T x - \varepsilon \sum_{i=1}^{n} \ln x_i \tag{18.13}$$

and parameterized by $\varepsilon \geq 0$.

It is important to note that the identification of the central path is a difficult problem. For each value of ε, we must indeed solve a non linear optimization problem. Therefore, the interior point algorithms use the central path as an indicator of the direction to progress toward the optimal solution, but without trying to follow it exactly. It means that, for a given ε, the corresponding non linear optimization problem is solved approximately.

We now compare the optimality conditions of the linear problem with those of the barrier problem. By taking the results of Section 6.5, we get the optimality conditions of the linear problem:

$$\begin{aligned} Ax - b = 0 \quad & \text{primal constraint} \\ x \geq 0 \quad & \text{primal constraint} \\ A^T\lambda + \mu - c = 0 \quad & \text{dual constraint (Eq. (6.162))} \\ \mu \geq 0 \quad & \text{dual constraint (Eq. (6.162))} \\ x_i\mu_i = 0 \quad & \text{complementarity constraint (Eq. (6.163))}\,. \end{aligned}$$

If we denote $e = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^T$ and

$$X = \begin{pmatrix} x_1 & 0 & \cdots & 0 & 0 \\ 0 & x_2 & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \cdots & x_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & x_n \end{pmatrix}, \quad S = \begin{pmatrix} \mu_1 & 0 & \cdots & 0 & 0 \\ 0 & \mu_2 & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \cdots & \mu_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & \mu_n \end{pmatrix}, \tag{18.14}$$

we obtain

$$\begin{aligned} Ax - b &= 0 \\ A^T\lambda + \mu - c &= 0 \\ XSe &= 0 \\ x &\geq 0 \\ \mu &\geq 0\,. \end{aligned} \tag{18.15}$$

We now write the optimality conditions for the problem

$$\min c^T x - \varepsilon \sum_{i=1}^{n} \ln x_i$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0\,. \end{aligned}$$

The Lagrangian is

$$L(x, \lambda, \mu) = c^T x - \varepsilon \sum_{i=1}^{n} \ln x_i + \lambda^T (Ax - b) - \mu^T x\,.$$

The first-order optimality condition is written as

$$\nabla L(x, \lambda, \mu) = c - \varepsilon X^{-1} + A^T\lambda - \mu = 0\,, \tag{18.16}$$

and the complementarity condition is

$$XSe = 0\,. \tag{18.17}$$

Incidentally, the optimal solution to the barrier problem is always such that $x > 0$. Therefore, the multipliers μ_i are always zero at the optimum. We keep them in the development in order to draw a parallel with the conditions (18.15). We now define

$$\bar{\mu} = \mu + \varepsilon X^{-1} \qquad \text{and} \qquad \bar{S} = S + \varepsilon X^{-1}\,. \tag{18.18}$$

The condition (18.16) is written as

$$c + A^T\lambda - \bar{\mu} = 0\,.$$

Since

$$X\bar{S}e = XSe + \varepsilon e\,,$$

the condition (18.17) is written as

$$X\bar{S}e = \varepsilon e\,.$$

We thus obtain the following conditions for the barrier problem:

$$\begin{aligned} Ax - b &= 0 \\ A^T\lambda + \bar{\mu} - c &= 0 \\ X\bar{S}e &= \varepsilon e \\ x &\geq 0 \\ \bar{\mu} &\geq 0\,. \end{aligned} \tag{18.19}$$

We note the similarities between the optimality conditions for the original problem (18.15) and those of the barrier problem (18.19). The conditions are the same, except for the third one, where the right hand side is εe instead of 0. In the following, we abandon the notation $\bar{\mu}$ and $\bar{S}$ and use μ and S.

We can thus characterize the optimal solution to the barrier problem which uses dual variables. We consider the primal and dual variables together and work in the space $\mathbb{R}^{n+m+n}$ with the variables (x, λ, μ). In this space, the feasible set is

$$\mathcal{F} = \left\{(x, \lambda, \mu) \mid Ax = b\,,\ A^T\lambda + \mu = c\,,\ x \geq 0\,,\ \mu \geq 0\right\} \tag{18.20}$$

and the set of interior points is

$$\mathcal{S} = \left\{(x, \lambda, \mu) \mid Ax = b\,,\ A^T\lambda + \mu = c\,,\ x > 0\,,\ \mu > 0\right\}, \tag{18.21}$$

again assumed to be non empty. The central path concept is also extended.

Definition 18.9 (Primal-dual central path)**.** Consider the linear problem

$$\min c^T x$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0\,. \end{aligned}$$

The primal-dual central path is the curve described by $(x_\varepsilon, \lambda_\varepsilon, \mu_\varepsilon)$ with $\varepsilon \geq 0$, where $(x_\varepsilon, \lambda_\varepsilon, \mu_\varepsilon)$ solves (18.19), that is such that

$$\begin{aligned} Ax_\varepsilon - b &= 0 \\ A^T\lambda_\varepsilon + \mu_\varepsilon - c &= 0 \\ X_\varepsilon S_\varepsilon e &= \varepsilon e \\ x_\varepsilon &\geq 0 \\ \mu_\varepsilon &\geq 0\,. \end{aligned}$$

Some elements of the primal-dual central path for Example 18.7 are listed in Table 18.1.

The system (18.19) includes two sets of linear equations, a set of slightly non linear equations ($XSe = \varepsilon e$), and two sets of inequations. Based on the same idea as Dikin's method (Algorithm 17.3), we proceed in the following manner:

1. Consider only the iterates in $\mathcal{S}$.
2. Ignore the inequalities and apply (partially) Newton's method (Algorithm 7.3) to the following system of equations, in order to identify a direction for the algorithm to follow.

$$F(x, \lambda, \mu) = \begin{pmatrix} Ax - b \\ A^T\lambda + \mu - c \\ XSe - \varepsilon e \end{pmatrix} = 0\,. \tag{18.22}$$

Table 18.1: Elements of the primal-dual central path for Example 18.7

ε	x_1	x_2	x_3	μ_1	μ_2	μ_3	λ
10000	3.3335e-01	3.3333e-01	3.3332e-01	2.9999e+04	3.0000e+04	3.0001e+04	-2.9998e+04
1000	3.3344e-01	3.3333e-01	3.3322e-01	2.9990e+03	3.0000e+03	3.0010e+03	-2.9980e+03
100	3.3445e-01	3.3333e-01	3.3222e-01	2.9900e+02	3.0000e+02	3.0100e+02	-2.9800e+02
10	3.4457e-01	3.3309e-01	3.2235e-01	2.9022e+01	3.0022e+01	3.1022e+01	-2.8022e+01
1	4.5157e-01	3.1118e-01	2.3724e-01	2.2146e+00	3.2149e+00	4.2147e+00	-1.2146e+00
0.1	8.6295e-01	8.9598e-02	4.7454e-02	1.1585e-01	1.1156e+00	2.1158e+00	8.8426e-01
0.01	9.8484e-01	1.0176e-02	4.9833e-03	1.0151e-02	1.0098e+00	2.0101e+00	9.9008e-01
0.001	9.9840e-01	8.4649e-04	7.5807e-04	1.0020e-03	1.0010e+00	2.0014e+00	9.9931e-01
0.0001	9.9902e-01	9.0048e-04	7.9656e-05	1.0602e-04	1.0007e+00	2.0008e+00	9.9940e-01
0	1.0000e+00	0.0000e+00	0.0000e+00	0.0000e+00	1.0000e+00	2.0000e+00	1.0000e+00

3. Calculate the step along the direction such that no iterate leaves $\mathcal{S}$.

The Jacobian matrix of the system (18.22) is

$$J(x,\lambda,\mu) = \nabla F(x,\lambda,\mu)^T = \begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix}.$$

If $(x,\lambda,\mu)^T \in \mathcal{S}$, then this point is feasible and

$$F(x,\lambda,\mu) = \begin{pmatrix} 0 \\ 0 \\ XSe - \varepsilon e \end{pmatrix}. \tag{18.23}$$

Therefore, the Newton equations (7.16) for an iterate (x,λ,μ) are written as

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{pmatrix} \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -XSe + \varepsilon e \end{pmatrix} \tag{18.24}$$

and an iteration consists in

$$\begin{pmatrix} x^+ \\ \lambda^+ \\ \mu^+ \end{pmatrix} = \begin{pmatrix} x \\ \lambda \\ \mu \end{pmatrix} + \alpha \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix}, \tag{18.25}$$

where $0 < \alpha \leq 1$ is chosen such that $\left(x^+,\lambda^+,\mu^+\right)^T \in \mathcal{S}$.

For a given value of ε, i.e., for a given barrier height, the Newton iterations can be applied until convergence. In this case, we identify the (primal-dual) central path element corresponding to this value of ε.

We can thus present a generic primal-dual algorithm of interior points to solve a linear optimization problem: Algorithm 18.1.

For the algorithm to be well defined, we need to specify two more things:

1. the calculation of the step α_k (step 16) and
2. the handling of the barrier height ε_k (step 17).

These two things are actually closely related, as explained below. But first, we analyze the stopping criterion and the distance to the central path. The necessary and sufficient condition for (x_k,λ_k,μ_k) to be an optimal solution to the initial problem is that it solves the system (18.15). As all the iterates are in $\mathcal{S}$, $Ax_k - b = 0$, $A^T\lambda_k + \mu_k - c = 0$, $x_k > 0$, $\mu_k > 0$ for any k. Therefore, the algorithm identifies an optimal solution when $\left(x_k\right)_i\left(\mu_k\right)_i = 0$, for $i = 1,\ldots,n$. The stopping criterion is in a sense based on the average distance to optimality

$$\nu_k = \frac{1}{n}\sum_{i=1}^{n}\left(x_k\right)_i\left(\mu_k\right)_i = \frac{1}{n}x_k^T\mu_k\,,$$

as every term is positive (as all the iterates are in the interior set $\mathcal{S}$), and zero at the optimal solution. We call this quantity the *duality measure*.

Algorithm 18.1: Generic interior points algorithm

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160), i.e.,

$$\min_{x \in \mathbb{R}^n} c^T x \qquad \text{subject to} \quad Ax = b\,,\ x \geq 0\,.$$

Input

The matrix $A \in \mathbb{R}^{m \times n}$.

The vector $b \in \mathbb{R}^m$.

The vector $c \in \mathbb{R}^n$.

An initial feasible solution $(x_0, \lambda_0, \mu_0)^T$ such that $Ax_0 = b$, $A^T\lambda_0 + \mu_0 = c$, $x_0 > 0$ and $\mu_0 > 0$.

An initial value for the height of the barrier $\varepsilon_0 > 0$.

The required precision $\bar{\varepsilon} \in \mathbb{R}$.

Output

An approximation of the optimal solution x^*.

Initialization

$k := 0$.

Repeat

Calculate (d_x, d_λ, d_μ) by solving

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X_k S_k e + \varepsilon_k e \end{pmatrix}, \tag{18.26}$$

where X_k and S_k are defined by (18.14).

Calculate a step $0 < \alpha_k \leq 1$ such that

$$\begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \\ \mu_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ \lambda_k \\ \mu_k \end{pmatrix} + \alpha_k \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} \tag{18.27}$$

is strictly feasible, i.e., in $\mathcal{S}$.

Update the height of the barrier by defining ε_{k+1}.

$k := k + 1$.

Until $\frac{1}{n} x_k^T \mu_k \leq \bar{\varepsilon}$.

In order to discuss the choice of the barrier parameter ε, we note that it is wise to have high barriers when we are far from the optimal solution, while they should be lower when we are close to it. It is appropriate to define the height of the barrier as a function of the duality measure, used as a stopping criterion in the algorithm. Then, we define

$$\varepsilon = \sigma\nu\,, \tag{18.28}$$

where $\nu = x^T\mu/n$ and σ, called the *centering parameter*, is such that $0 \leq \sigma \leq 1$. In order to understand the role of this parameter, let us analyze its two extreme values.

$\sigma = 0$ The barrier is absent in this case, as $\varepsilon = 0$. The direction obtained in the algorithm aims to directly solve the optimality conditions (18.15) of the initial problem. It is thus necessary to calculate the step in order to stay within $\mathcal{S}$. This method is a version of Dikin's method (Algorithm 17.3) in the primal-dual space.

$\sigma = 1$ If ν is fixed, the generic algorithm converges toward the point of the central path corresponding to $\varepsilon = \nu$. Each iteration with $\sigma = 1$ enables the central path to be approached.

The method faces a dilemma. The further the iterates are from the constraints, the more flexible the method is, as it can make longer steps in any direction. At the same time, the further it is from the optimal solution, which is at a vertex that is on the border. The role of the parameter σ is to handle this dilemma.

Example 18.10 (Centering parameter)**.** Consider Example 18.7 again and calculate the direction (d_x, d_λ, d_μ) in several points, with two values for σ. For each point, the values of the dual variables are $\lambda = 0$ and $\mu = c$.

The results are presented in Tables 18.2, 18.3, and 18.4, and illustrated in Figures 18.9, 18.7, and 18.8. The case presented in Figure 18.8 illustrates particularly well the difference between the two extreme values of the σ parameter. While the direction generated with $\sigma = 0$ points more or less in the direction of the optimal solution of the problem, the direction generated with $\sigma = 1$ points toward the central path.

Table 18.2: Primal-dual directions at $x_0 = \begin{pmatrix}0.6 & 0.2 & 0.2\end{pmatrix}^T$

$\nu = 0.53333$

$\sigma = 1$			$\sigma = 0$		
d_x	d_μ	d_λ	d_x	d_μ	d_λ
-0.049275	-0.028986		0.069739	-0.498138	
0.069565	-0.028986	0.028986	-0.026567	-0.498138	0.498138
-0.020290	-0.028986		-0.043172	-0.498138	

Table 18.3: Primal-dual directions at $x_0 = \begin{pmatrix} 0.2 & 0.6 & 0.2 \end{pmatrix}^T$

$\nu = 0.66667$

$\sigma = 1$			$\sigma = 0$		
d_x	d_μ	d_λ	d_x	d_μ	d_λ
0.4061814	0.4102842		0.043272	-0.499296	
-0.4020786	0.4102842	-0.4102842	-0.019972	-0.499296	0.499296
-0.0041028	0.4102842		-0.023300	-0.499296	

Table 18.4: Primal-dual directions at $x_0 = \begin{pmatrix} 0.2 & 0.2 & 0.6 \end{pmatrix}^T$

$\nu = 0.8$

$\sigma = 1$			$\sigma = 0$		
d_x	d_μ	d_λ	d_x	d_μ	d_λ
0.208312	0.470382		0.2	-2	
0.053758	0.470382	-0.470382	0.2	-2	
-0.262070	0.470382	0	-2		
	2	-0.2	-2		

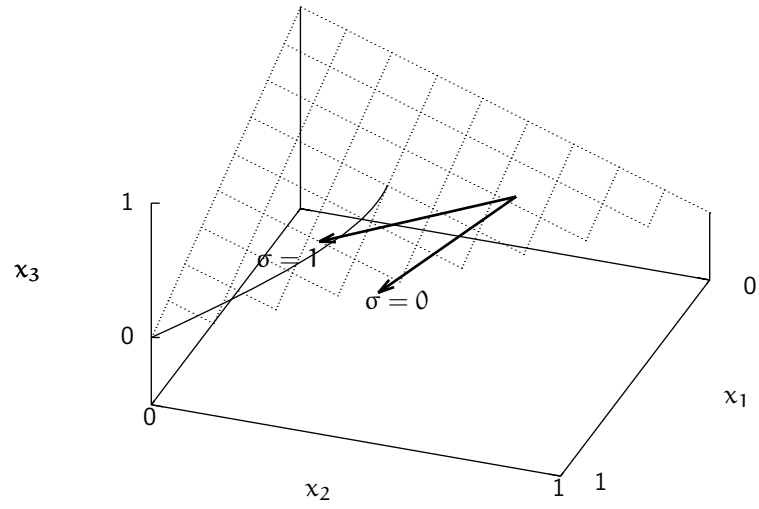


Figure 18.7: Newton directions for the point $\begin{pmatrix} 0.2 & 0.6 & 0.2 \end{pmatrix}^T$

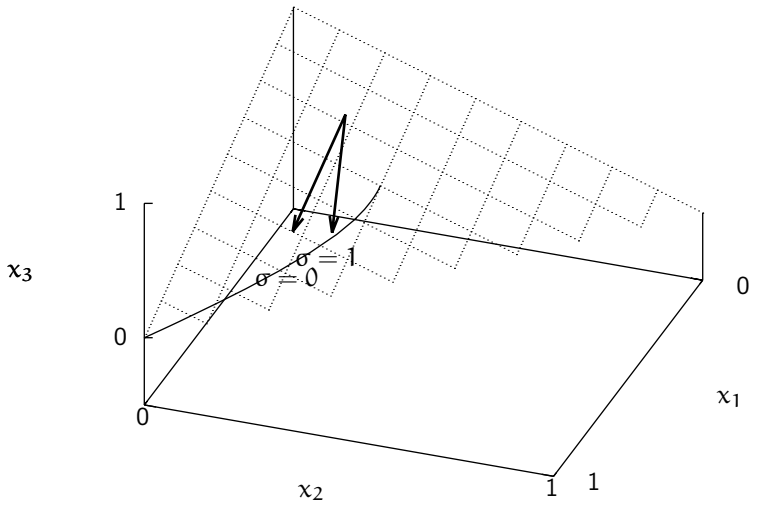


Figure 18.8: Newton directions for the point $\begin{pmatrix} 0.2 & 0.2 & 0.6 \end{pmatrix}^T$

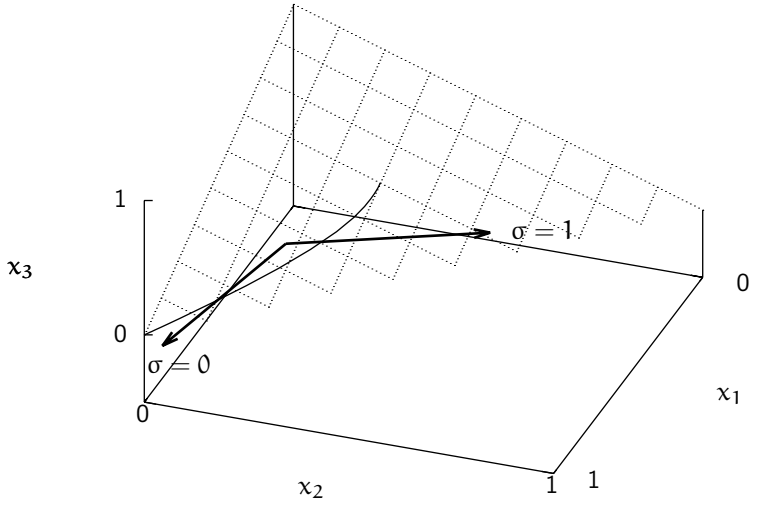


Figure 18.9: Newton directions for the point $\begin{pmatrix} 0.6 & 0.2 & 0.2 \end{pmatrix}^T$

In order to "follow" the central path, the algorithm needs a measure of the distance from one iterate to the path. For $\varepsilon > 0$, the point on the central path corresponding to ε is such that each product $x_i\mu_i$ is equal to the barrier parameter, i.e., that $x_1\mu_1 = x_2\mu_2 = \ldots = x_n\mu_n = \varepsilon$ (see Eq. (18.19)). Therefore, an indicator of the proximity to the central path is the difference between each individual product and their average value ν:

$$\frac{1}{\nu}\left\| \begin{pmatrix} x_1\mu_1 \\ \vdots \\ x_n\mu_n \end{pmatrix} - \begin{pmatrix} \nu \\ \vdots \\ \nu \end{pmatrix} \right\| = \frac{1}{\nu}\left\| XSe - \nu e \right\|, \tag{18.29}$$

where $\|\cdot\|$ is a norm in $\mathbb{R}^n$. Thanks to this measure, we can define the neighborhoods around the central path. By using the norm 2, we obtain a *restricted neighborhood*.

Definition 18.11 (Restricted neighborhood of the central path)**.** Consider a linear problem

$$\min c^T x$$

subject to

$$\begin{aligned} Ax &= b \\ x &\geq 0\,. \end{aligned}$$

The θ-restricted neighborhood of the primal-dual central path is the set

$$\mathcal{V}_2(\theta) = \left\{ (x, \lambda, \mu) \in \mathcal{S} \;\middle|\; \frac{1}{\nu}\left\| XSe - \nu e \right\|_2 \leq \theta \right\}, \tag{18.30}$$

where $\mathcal{S}$ is the set of primal-dual interior points (18.21) and $0 \leq \theta < 1$.

We can define a large neighborhood using the norm ∞. In this case, we obtain

$$\begin{aligned} \mathcal{V}_\infty(\theta) &= \left\{ (x, \lambda, \mu) \in \mathcal{S} \;\middle|\; -\theta\nu \leq x_i\mu_i - \nu \leq \theta\nu\,,\ i = i, \ldots n \right\} \\ &= \left\{ (x, \lambda, \mu) \in \mathcal{S} \mid (1-\theta)\nu \leq x_i\mu_i \leq (1+\theta)\nu\,,\ i = i, \ldots n \right\}. \end{aligned}$$

It is possible to extend this interval even further, in order to give the algorithms more flexibility. Indeed, the fact that the product $x_i\mu_i$ takes large values is not important. However, we wish to avoid too small values of these products in relation to their average value ν and that x_i and μ_i approach 0 too rapidly. Indeed, it this case, the iterate would be too close to the constraints, and the algorithm would not benefit from being in the interior anymore. We define a large neighborhood ignoring the upper bound and defining $\gamma = 1 - \theta$. As θ is between 0 and 1, so is γ. The designation $\mathcal{V}_{-\infty}$ is used here to highlight the fact that, contrary to $\mathcal{V}_\infty$, only the lower bound is taken into account.

Definition 18.12 (Large neighborhood of the central path)**.** Consider the linear problem

$$\min c^T x$$

subject to

$$Ax = b$$
$$x \geq 0\,.$$

The γ-large neighborhood of the primal-dual central path is the set

$$\mathcal{V}_{-\infty}(\gamma) = \left\{(x, \lambda, \mu) \in \mathcal{S} \mid x_i \mu_i \geq \gamma \nu\,,\ i = 1, \ldots, n\right\}, \tag{18.31}$$

where $\mathcal{S}$ is the set of primal-dual interior points (18.21) and $0 < \gamma < 1$.

Many variants of this algorithm have been proposed in the literature. We present here three algorithms that follow the primal-dual central path, using different strategies.

Restricted step algorithm This algorithm, presented as Algorithm 18.2, chooses the centering parameter so that the step $\alpha_k = 1$ in Algorithm 18.1 generates only iterates located in the restricted neighborhood of the central path. The values of θ and σ guarantee that each Newton step generates an iterate in the neighborhood $\mathcal{V}_2(\theta)$. Compared to the generic algorithm (Algorithm 18.1), we have $\varepsilon_k = \nu_k \sigma$, and the step along the direction is always $\alpha_k = 1$.

Prediction-correction algorithm This algorithm (Algorithm 18.3) combines the two extreme cases discussed above:

- the prediction step investigates where the optimal solution can be, by setting the centering parameter σ to 0. Still, the step length is calculated so that the next iterate lies in the restricted neighborhood of the central path;
- the correction step is an iteration of the restricted step algorithm (Algorithm 18.2), that focuses on moving the iterations back toward the central path using the value $\sigma = 1$ for the centering parameter.

Long step algorithm This algorithm fixes the centering parameter to an intermediary value (not 0, not 1) and selects the step α_k in Algorithm 18.1 so that each iterate is situated in the large neighborhood of the central path.

The theoretical foundation of these algorithms is technical and beyond the scope of this book. We refer the interested reader to the excellent presentation proposed by Wright (1997). We present an illustration of these algorithms on Example 18.7. On such small examples, the interior point methods are definitely not efficient. But the iterations can be represented on a figure, in order to have some insight on their general behavior. Comparing the three versions, the flexibility of the long step algorithm seems to pay off on this example. Although it cannot be formally generalized, it is the version that should be preferred in general.

Algorithm 18.2: Interior point algorithm with restricted steps

Objective

To find a global minimum of a linear optimization problem in standard form (6.159)–(6.160), i.e.,

$$\min_{x\in\mathbb{R}^n} c^T x \qquad \text{subject to} \quad Ax = b\,,\ x \geq 0\,.$$

Input

The matrix $A \in \mathbb{R}^{m\times n}$.
The vector $b \in \mathbb{R}^m$.
The vector $c \in \mathbb{R}^n$.
$\theta = 0.4$.
An initial feasible solution $(x_0, \lambda_0, \mu_0) \in \mathcal{V}_2(\theta)$.
$\sigma = 1 - \theta/\sqrt{n}$.
The required precision $\bar{\varepsilon} \in \mathbb{R}$.

Output

An approximation of the optimal solution (x^*, λ^*, μ^*).

Initialization

$k := 0$.

Repeat

$\nu_k = \dfrac{1}{n} x_k^T \mu_k$.
Calculate $(d_x, d_\lambda, d_\mu)^T$ by solving

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X_k S_k e + \nu_k \sigma e \end{pmatrix}, \tag{18.32}$$

where X_k and S_k are defined by (18.14).
$(x_{k+1}, \lambda_{k+1}, \mu_{k+1})^T := (x_k, \lambda_k, \mu_k)^T + (d_x, d_\lambda, d_\mu)^T$.
$k := k + 1$.

Until $\nu_k \leq \bar{\varepsilon}$.

Algorithm 18.3: Predictor-corrector interior point algorithm

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160), i.e., $\min_{x\in\mathbb{R}^n} c^T x$ subject to $Ax = b,\ x \geq 0$.

Input

$A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$.

$\theta_{pred} = 0.5$, $\theta_{corr} = 0.25$.

An initial feasible solution $(x_0, \lambda_0, \mu_0) \in \mathcal{V}_2(\theta_{corr})$.

The required precision $\bar{\varepsilon} \in \mathbb{R}$.

Output

An approximation of the optimal solution (x^*, λ^*, μ^*).

Initialization

$k := 0$.

Repeat

Prediction: $\sigma_k = 0$, **no barrier.**

$\nu_k = \dfrac{1}{n} x_k^T \mu_k$.

Calculate $(d_x, d_\lambda, d_\mu)^T$ by solving

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X_k S_k e \end{pmatrix},$$

where X_k and S_k are defined by (18.14).

$\alpha := 1$.

Repeat

$(x_{k+1}, \lambda_{k+1}, \mu_{k+1})^T := (x_k, \lambda_k, \mu_k)^T + \alpha(d_x, d_\lambda, d_\mu)^T$.

$\alpha := \alpha/2$.

Until $(x_{k+1}, \lambda_{k+1}, \mu_{k+1})^T \in \mathcal{V}_2(\theta_{pred})$

$k := k + 1$.

Correction: $\sigma_k = 1$

$\nu_k = \dfrac{1}{n} x_k^T \mu_k$.

Calculate $(d_x, d_\lambda, d_\mu)^T$ by solving

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X_k S_k e + \nu_k e \end{pmatrix},$$

where X_k and S_k are defined by (18.14).

$(x_{k+1}, \lambda_{k+1}, \mu_{k+1})^T := (x_k, \lambda_k, \mu_k)^T + (d_x, d_\lambda, d_\mu)^T$.

$k := k + 1$.

Until $\nu_k \leq \bar{\varepsilon}$.

Example 18.13 (Restricted step algorithm)**.** Consider Example 18.7 again and apply Algorithm 18.2 from the starting point $x = \begin{pmatrix} 0.6 & 0.2 & 02 \end{pmatrix}^T$, $\lambda = 0$ and $\mu = c$. A few iterations are listed in Table 18.5. We observe how slow the method is, due to its inability to take large steps.

Table 18.5: Iterations for the interior point algorithm with restricted steps (Algorithm 18.2) for Example 18.7

k	$\left\|X_k S_k e - \nu_k e\right\|_2 / \nu_k$	$\|d_k\|$	ν_k
0	3.061862e-01		5.333333e-01
1	4.433431e-02	6.494725e-01	4.101653e-01
2	3.892141e-02	4.203992e-01	3.154417e-01
3	3.940146e-02	2.830158e-01	2.425935e-01
4	3.305109e-02	1.976729e-01	1.865690e-01
5	2.615063e-02	1.416567e-01	1.434827e-01
⋮			
20	4.730746e-04	2.303030e-03	2.793843e-03
21	3.635129e-04	1.770192e-03	2.148633e-03
22	2.793793e-04	1.360809e-03	1.652427e-03
23	2.147504e-04	1.046205e-03	1.270815e-03
24	1.650912e-04	8.043948e-04	9.773332e-04
25	1.269267e-04	6.185101e-04	7.516277e-04
⋮			
35	9.178012e-06	4.473980e-05	5.440069e-05
36	7.058322e-06	3.440722e-05	4.183739e-05
37	5.428202e-06	2.646100e-05	3.217546e-05
38	4.174570e-06	2.034997e-05	2.474486e-05
39	3.210470e-06	1.565027e-05	1.903028e-05
⋮			
55	4.807511e-08	2.343568e-07	2.849756e-07
56	3.697263e-08	1.802344e-07	2.191633e-07
57	2.843417e-08	1.386111e-07	1.685497e-07
58	2.186758e-08	1.066002e-07	1.296248e-07
59	1.681748e-08	8.198194e-08	9.968925e-08

Example 18.14 (Predictor-corrector algorithm)**.** Consider Example 18.7 again and apply Algorithm 18.3 from the starting point[1] $x = \begin{pmatrix} 0.5 & 0.3 & 0.2 \end{pmatrix}^T$, $\lambda = 0$ and $\mu = c$. A few iterations are listed in Table 18.6. It is interesting to note that the lengths of the step and of the correction are of the same order of magnitude as the distance to the central path. Moreover, the prediction step has the effect of moving the iterates further from the central path, which requires a correction iteration.

Table 18.6: Iterations of the predictor-corrector interior point algorithm (Algorithm 18.3) for Example 18.7

k	$\left\|X_k S_k e - \nu_k e\right\|_2/\nu_k$	$\|\alpha d_k\|$	ν_k	α	type
0	1.440876e-01		5.666667e-01		
1	4.423867e-01	1.400602e+00	2.833333e-01	0.5	pred
2	5.054768e-02	2.736081e-01	2.833333e-01	1.0	corr
3	1.986016e-01	5.684548e-01	1.416667e-01	0.5	pred
4	4.375390e-03	5.605598e-02	1.416667e-01	1.0	corr
5	1.392922e-01	2.300908e-01	7.083333e-02	0.5	pred
6	1.031609e-03	1.890433e-02	7.083333e-02	1.0	corr
⋮					
25	1.294342e-04	1.894595e-04	6.917318e-05	0.5	pred
26	8.047593e-13	1.631807e-08	6.917318e-05	1.0	corr
27	6.471134e-05	9.472452e-05	3.458659e-05	0.5	pred
28	1.005798e-13	4.079063e-09	3.458659e-05	1.0	corr
29	3.235423e-05	4.736095e-05	1.729329e-05	0.5	pred
30	1.257720e-14	1.019709e-09	1.729329e-05	1.0	corr
⋮					
39	1.011026e-06	1.479990e-06	5.404154e-07	0.5	pred
40	1.959217e-16	9.957556e-13	5.404154e-07	1.0	corr
41	5.055127e-07	7.399946e-07	2.702077e-07	0.5	pred
42	0.000000e+00	2.489387e-13	2.702077e-07	1.0	corr
43	2.527563e-07	3.699972e-07	1.351039e-07	0.5	pred
44	2.770751e-16	6.223464e-14	1.351039e-07	1.0	corr
45	1.263781e-07	1.849986e-07	6.755193e-08	0.5	pred

[1] The starting point $x = \begin{pmatrix} 0.6 & 0.2 & 0.2 \end{pmatrix}^T$, used in the previous example, does not belong to $\mathcal{V}_2(0.25)$.

Example 18.15 (Long step algorithm). Consider Example 18.7 again and apply Algorithm 18.4 from the starting point $x = \begin{pmatrix} 0.6 & 0.2 & 0.2 \end{pmatrix}^T$, $\lambda = 0$ and $\mu = c$. The iterations are listed in Table 18.7 and illustrated in Figures 18.10, 18.11 and 18.12. This algorithm clearly performs better than the two others on this example. The fact that it only loosely follows the central path provides it with more flexibility to move toward the optimal solution.

Algorithm 18.4: Long step interior point algorithm

Objective

To find the global minimum of a linear optimization problem in standard form (6.159)–(6.160), i.e., $\min_{x \in \mathbb{R}^n} c^T x$ subject to $Ax = b,\ x \geq 0$.

Input

$A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$.

$\gamma > 0$ (for instance, $\gamma = 10^{-3}$).

$0 < \sigma < 1$ (for instance $\sigma = 0.1$).

An initial feasible solution $(x_0, \lambda_0, \mu_0) \in \mathcal{V}_{-\infty}(\gamma)$.

The required precision $\bar{\varepsilon} \in \mathbb{R}$.

Output

An approximation of the optimal solution (x^*, λ^*, μ^*).

Initialization

$k := 0$.

Repeat

$\nu_k = \frac{1}{n} x_k^T \mu_k$

Calculate $(d_x, d_\lambda, d_\mu)^T$ by solving

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} d_x \\ d_\lambda \\ d_\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X_k S_k e + \nu_k \sigma e \end{pmatrix},$$

where X_k and S_k are defined by (18.14).

$\alpha := 1$

Repeat

$(x_{k+1}, \lambda_{k+1}, \mu_{k+1})^T := (x_k, \lambda_k, \mu_k)^T + \alpha (d_x, d_\lambda, d_\mu)^T$.

$\alpha := \alpha / 2$.

Until $(x_{k+1}, \lambda_{k+1}, \mu_{k+1})^T \in \mathcal{V}_{-\infty}(\gamma)$.

$k := k + 1$.

Until $\nu_k \leq \bar{\varepsilon}$.

Table 18.7: Iterations of the long step interior point algorithm (Algorithm 18.4) for Example 18.7

k	$(\max_{i=1,\ldots,n} \mu_i x_i)/\nu_k$	$\|\alpha d_k\|$	ν_k	α
0	1.125000e+00		5.333333e-01	
1	1.217712e+00	1.180976e+00	2.933333e-01	0.5
2	1.278088e+00	5.030498e-01	1.613333e-01	0.5
3	1.294039e+00	2.119436e-01	8.873333e-02	0.5
4	1.561739e+00	2.068096e-01	8.873333e-03	1.0
5	1.014971e+00	2.363846e-02	8.873333e-04	1.0
6	1.007229e+00	2.198411e-03	8.873333e-05	1.0
7	1.000716e+00	2.180253e-04	8.873333e-06	1.0
8	1.000072e+00	2.186378e-05	8.873333e-07	1.0

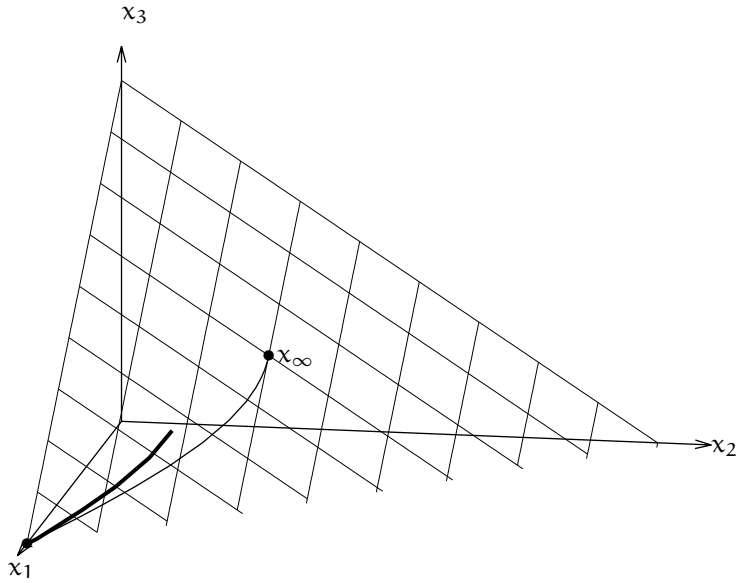


Figure 18.10: Iterations of the long step interior point algorithm (Algorithm 18.4) for Example 18.7

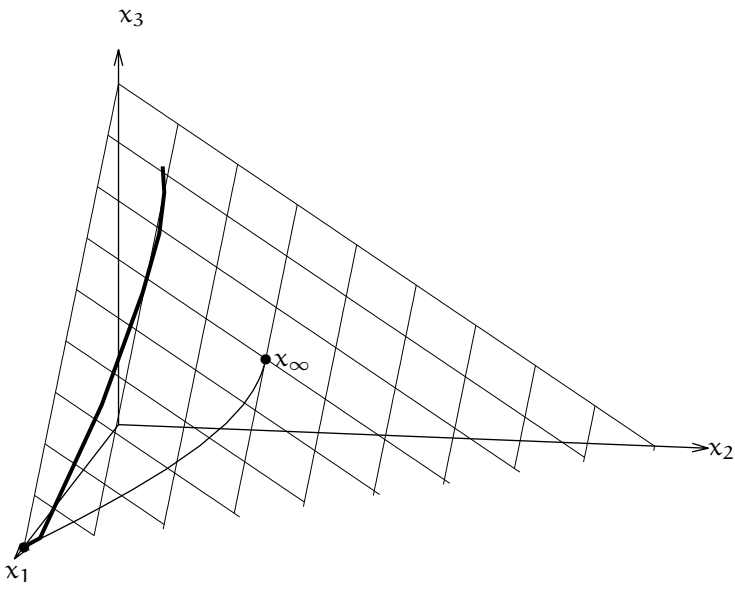


Figure 18.11: Iterations of the long step interior point algorithm (Algorithm 18.4) for Example 18.7, with $x_0 = \begin{pmatrix} 0.1 & 0.1 & 0.8 \end{pmatrix}^T$

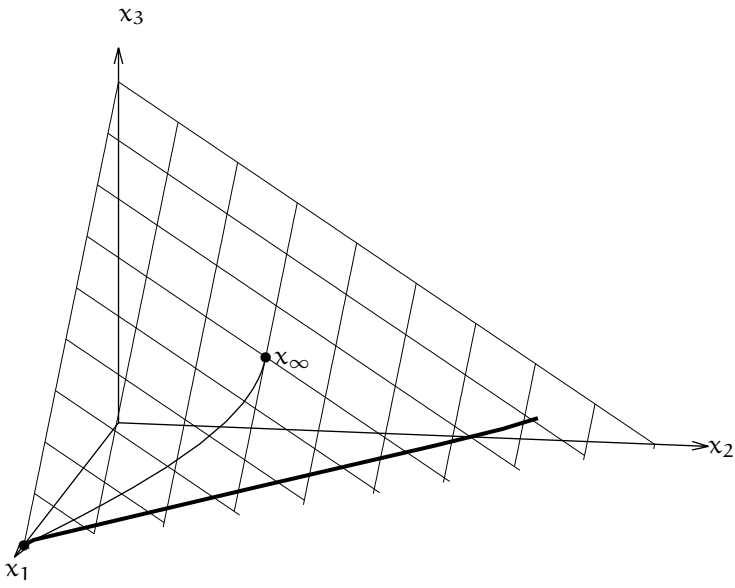


Figure 18.12: Iterations of the long step interior point algorithm (Algorithm 18.4) for Example 18.7, with $x_0 = \begin{pmatrix} 0.1 & 0.8 & 0.1 \end{pmatrix}^T$

The algorithms described in this chapter require an initial feasible solution $(x_0, \lambda_0, \mu_0)^T$ such that $Ax_0 = b$, $A^T\lambda_0 + \mu_0 = c$, $x_0 > 0$ and $\mu_0 > 0$. It means that a "phase I" procedure is required, similar to the procedure described for the simplex algorithm in Section 16.3, where an auxiliary problem is solved to generate the feasible solution of the original problem (see Huhn, 1999 for an analysis of such techniques in the context of interior point methods). Note that some implementations do not require beginning with a feasible starting point. These so called *infeasible interior point* methods are more complex, but may be quite effective in practice (see for instance Zhang, 1994).

18.3 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of this project is to implement interior point algorithms and compare them with the simplex method (Chapter 16) and Dikin's method (Section 17.3).

Approach

Apply the algorithms to the problems described below, for different values of n. Caution! If the starting point is not in the neighborhood corresponding to the algorithm, first apply centering iterations ($\sigma = 1$) to obtain a point in the neighborhood.

The following variations of the algorithms can be tested.

Algorithm 18.2. Vary the value of $\theta = 0.1, 0.4, 0.8$.

Algorithm 18.3. $\theta_{pred} = 0.1, 0.5, 0.9$ and $\theta_{corr} = \theta_{pred}/2$.

Algorithm 18.4. $\gamma = 10^{-5}, 10^{-3}, 1$, $\sigma = 0.1, 0.5, 0.9$.

Algorithms

Algorithms 16.5, 18.2, 18.3, and 18.4.

Problems

Exercise 18.1. The problem

$$\min -\sum_{i=1}^{n} 2^{n-i} x_i$$

subject to

$$\begin{aligned} x_1 &\le 5 \\ 4x_1 + x_2 &\le 25 \\ 8x_1 + 4x_2 + x_3 &\le 125 \\ &\vdots \\ 2^n x_1 + 2^{n-1} x_2 + \cdots + 4x_{n-1} + x_n &\le 5^n \\ x_1, x_2, \ldots, x_n &\ge 0. \end{aligned}$$

Use the starting point $\begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix}^T$. The optimal solution to this problem is $\begin{pmatrix} 0 & 0 & \ldots & 0 & 5^n \end{pmatrix}^T$.

Exercise 18.2. The problem

$$\min -x_n$$

subject to

$$\begin{aligned} \varepsilon \le x_1 &\le 1 \\ \varepsilon x_{i-1} \le x_i &\le 1 - \varepsilon x_{i-1}, \qquad i = 2, \ldots, n, \end{aligned}$$

where $0 < \varepsilon < 1/2$. Use the starting point

$$\begin{aligned} x_1 &= \frac{1+\varepsilon}{2} \\ x_i &= \frac{1}{2}, \qquad i = 2, \ldots, n. \end{aligned}$$

Chapter 19

Augmented Lagrangian method

A major difficulty with constrained optimization problems is to combine two goals that are often conflicting:

1. to reduce the value of the objective function and
2. to verify the constraints.

Interior point methods start by giving significant weight to the second criterion at the expense of the first, and then rebalance the two during the iterations by lowering the barrier. The augmented Lagrangian method described in this chapter works in the opposite way. It tries to reduce the objective function, possibly by violating the constraints. Subsequently, the feasibility is restored progressively as the iterations proceed.

Contents

In this chapter, and in the next, we consider the optimization problem (1.71)–(1.72), i.e.,

$$\min_{x \in \mathbb{R}^n} f(x) \tag{19.1}$$

subject to

$$h(x) = 0\,, \tag{19.2}$$

where f is a function of $\mathbb{R}^n$ in $\mathbb{R}$ and h is a function of $\mathbb{R}^n$ in $\mathbb{R}^m$. We keep in mind that it is always possible to transform an inequality constraint

$$g_i(x) \leq 0$$

with $g_i : \mathbb{R}^n \to \mathbb{R}$ into an equality constraint, by introducing slack variables $z_i \in \mathbb{R}$ (see Section 1.2.2 and Example 6.17), to obtain

$$g_i(x) + z_i^2 = 0\,.$$

The augmented Lagrangian method is directly inspired by the Karush-Kuhn-Tucker optimality conditions and especially the proof of Theorem 6.10. The basic idea consists in transforming a constrained problem into a sequence of unconstrained problems, by penalizing more and more the possible violation of the constraints. It is interesting to note that this approach is the opposite of the one used in the context of interior point methods. Indeed, the latter mainly focuses on the generation of feasible iterates. During the first iterations, this comes at the expense of optimality. As the height of the barriers decreases, the interior point algorithms converge toward the minimum. For the augmented Lagrangian method, it is the opposite. The main focus is on the identification of optimal solutions to the subproblems, if required by violating the constraints. Successive iterations try to restore the feasibility of the iterates.

Two types of penalties for violating the constraints are considered:

- a Lagrangian penalty, or Lagrangian relaxation, as presented in the introduction of duality, discussed in Chapter 4,
- a quadratic penalty, as in the proof of Theorem 6.10.

This involves combining the Lagrangian (Definition 4.3) of the problem, i.e.,

$$L(x, \lambda) = f(x) + \lambda^T h(x)$$

with a quadratic penalty

$$\frac{c}{2} \left\| h(x) \right\|^2 ,$$

where $c \in \mathbb{R}$, $c > 0$. We obtain the *augmented Lagrangian* (Definition 6.18)

$$L_c(x, \lambda) = f(x) + \lambda^T h(x) + \frac{c}{2} \left\| h(x) \right\|^2 , \tag{19.3}$$

for which the derivatives with respect to x are

$$\nabla_x L_c(x, \lambda) = \nabla f(x) + \nabla h(x)\lambda + c\nabla h(x)h(x) \tag{19.4}$$

and

$$\begin{aligned} \nabla^2_{xx} L_c(x, \lambda) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 h_i(x) \\ + c\nabla h(x)\nabla h(x)^T + c \sum_{i=1}^{m} h_i(x)\nabla^2 h_i(x)\,. \end{aligned} \tag{19.5}$$

We start by analyzing each penalty separately.

19.1 Lagrangian penalty

As described in the proof of Theorem 6.19, if x^* and λ^* satisfy the sufficient optimality conditions (6.115)

$$\nabla L(x^*, \lambda^*) = 0$$

and (6.116)

$$y^T \nabla^2_{xx} L(x^*, \lambda^*) y > 0\,, \qquad \forall y \in \mathcal{D}(x^*)\,,\ y \neq 0,$$

where $\mathcal{D}(x^*)$ is the linearized cone in x^* (Definition 3.23), then x^* is a strict local minimum of the problem

$$\min_{x \in \mathbb{R}^n} L_c(x, \lambda^*) \tag{19.6}$$

with a sufficiently large c.

Example 19.1 (Lagrangian penalty)**.** Consider the problem

$$\min_{x \in \mathbb{R}^2} \frac{1}{2}\left(-x_1^2 + x_2^2\right)$$

subject to

$$x_1 = 1\,.$$

The Lagrangian is

$$L(x, \lambda) = \frac{1}{2}\left(-x_1^2 + x_2^2\right) + \lambda(x_1 - 1)$$

and

$$\nabla_x L(x, \lambda) = \begin{pmatrix} -x_1 + \lambda \\ x_2 \end{pmatrix}$$

$$\nabla^2_{xx} L(x, \lambda) = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Then, $x^* = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ and $\lambda^* = 1$ satisfy the sufficient optimality conditions. Indeed, $\nabla_x L(x^*, \lambda^*) = 0$ and (6.23) are satisfied. One direction y is in the linearized cone $\mathcal{D}(x^*)$ if and only if $y^T \nabla h(x^*) = 0$, that is, if $y_1 \cdot 1 + y_2 \cdot 0 = y_1 = 0$. Therefore, if $y \neq 0$,

$$\begin{pmatrix} 0 & y_2 \end{pmatrix} \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ y_2 \end{pmatrix} = y_2^2 > 0$$

and the second order KKT condition (6.24) is satisfied. The augmented Lagrangian is

$$L_c(x, \lambda) = \frac{1}{2}\left(-x_1^2 + x_2^2\right) + \lambda(x_1 - 1) + \frac{c}{2}(x_1 - 1)^2\,.$$

We have

$$\nabla_x L_c(x, \lambda) = \begin{pmatrix} (c-1)x_1 + \lambda - c \\ x_2 \end{pmatrix},$$

which is zero at

$$x = \begin{pmatrix} \dfrac{c - \lambda}{c - 1} \\ 0 \end{pmatrix}$$

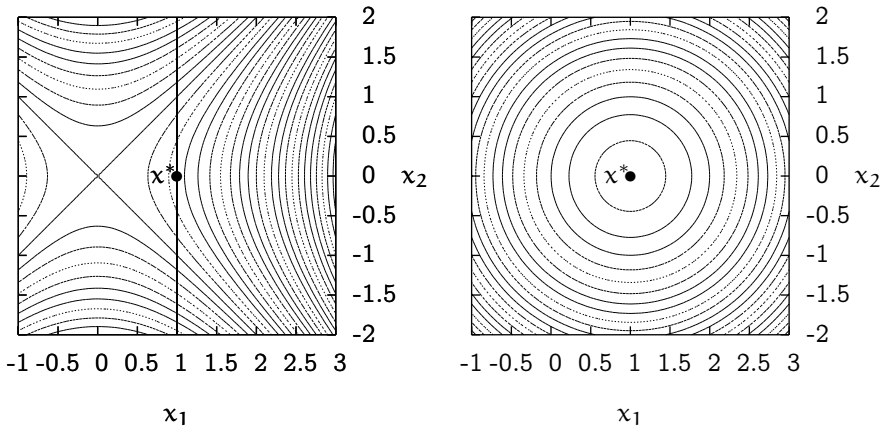
and

$$\nabla^2_{xx} L_c(x, \lambda) = \begin{pmatrix} c-1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{19.7}$$

We immediately note that x is not defined if $c = 1$. Moreover, if $c < 1$, the second derivatives matrix is not positive semidefinite, the necessary optimality conditions are not satisfied and the point x is not a local minimum of the augmented Lagrangian. If $c > 1$, then x is a strict local minimum of the augmented Lagrangian. Now, consider $\lambda = \lambda^* = 1$. For any $c \neq 1$, we get

$$x = \begin{pmatrix} \frac{c-\lambda^*}{c-1} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{c-1}{c-1} \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = x^*.$$

This illustrates that if the optimal value of the dual variable λ^* is known, and if the parameter c is large enough, minimizing the augmented Lagrangian identifies also a local minimum of the constrained problem. The constrained problem is represented in Figure 19.1(a). Figures 19.1(b) and 19.2(b) display the level curves of the augmented Lagrangian (with $\lambda = \lambda^*$) for values of c above 1. We note that the minimum without constraint is x^*. However, when $c < 1$ (Figure 19.2(a)), x^* is a saddle point of the augmented Lagrangian and not a minimum.

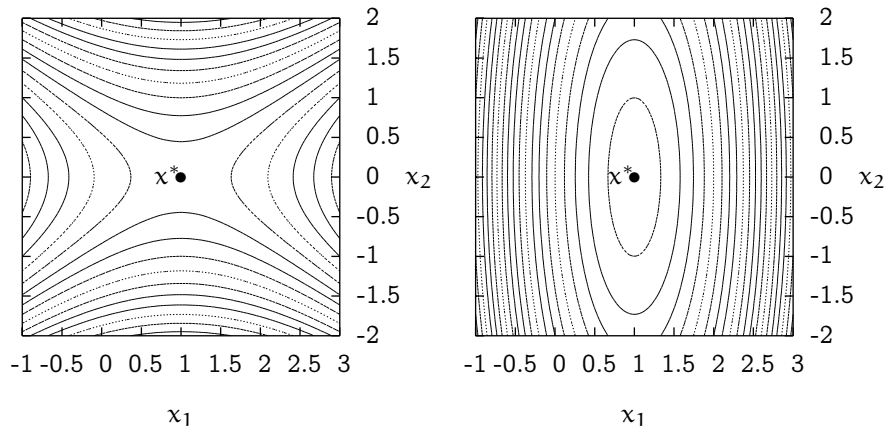


(a) Constrained problem

(b) Augmented Lagrangian with $c = 2$

Figure 19.1: Level curves for Example 19.1

Then, if the value of λ^* is known, the constrained optimization problem (19.1)–(19.2) reduces to the unconstrained optimization problem (19.6) and the methods presented in Part IV can be used. Unfortunately, in practice, the value of λ^* is as complicated to obtain as the value of x^*, and this method cannot be directly used. Consequently, the second penalty plays an important role.



(a) Augmented Lagrangian with $c = 1/2$ (b) Augmented Lagrangian with $c = 10$

Figure 19.2: Level curves for Example 19.1

19.2 Quadratic penalty

The quadratic penalty amounts to making c sufficiently large that an infeasible point of the initial problem cannot be optimal when minimizing the augmented Lagrangian. This idea is the basis for the proof of Theorem 6.10.

If we take Example 19.1, the minimum of the augmented Lagrangian is

$$x = \begin{pmatrix} \dfrac{c-\lambda}{c-1} \\ 0 \end{pmatrix}.$$

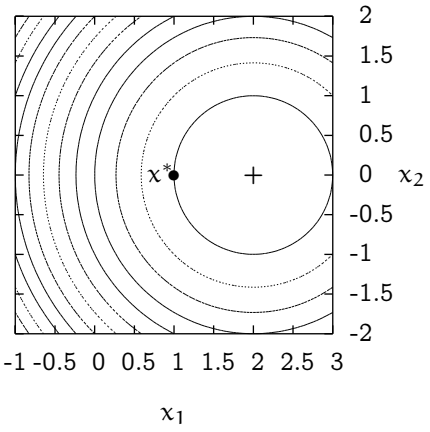
When c tends toward infinity, x tends toward $x^* = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$, and this regardless of the value of λ. The level curves for several values of c are presented in Figure 19.3.

A possible algorithm would consist in solving a sequence of unconstrained problems

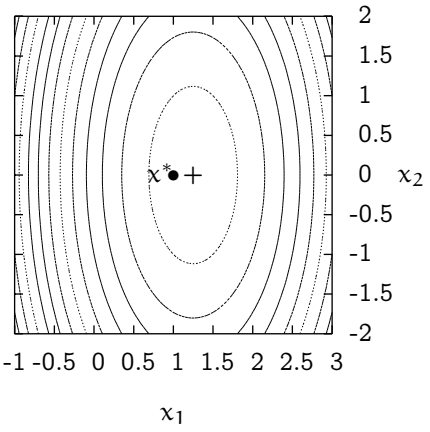
$$x_k \in \operatorname{argmin}_{x \in \mathbb{R}^n} L_{c_k}(x, \lambda), \tag{19.8}$$

where λ is given and c_k is a sequence of real numbers such that $\lim_{k\to\infty} c_k = +\infty$. In general, the final solution to the problem at step k serves as the starting point for the calculation of the optimal solution in step $k+1$. Note that if several minima belongs $\operatorname{argmin}_{x \in \mathbb{R}^n} L_{c_k}(x, \lambda)$, we consider only one, that is the solution produced by the considered unconstrained optimization algorithm.

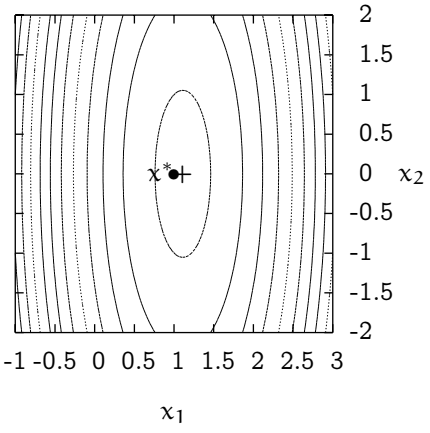
Such an algorithm would work regardless of the value of λ (this is proved below). Unfortunately, the unconstrained minimization problem becomes increasingly ill-conditioned as c_k increases. This can be seen in Example 19.1. Indeed, the level curves become increasingly stretched (Figure 19.3(d)).



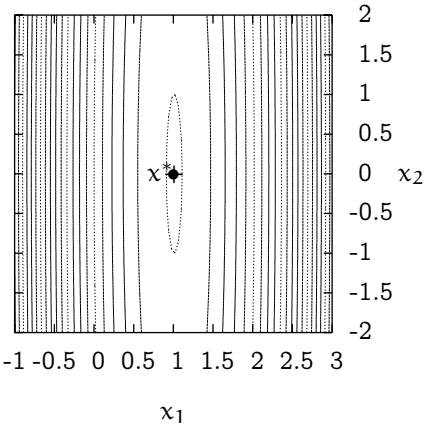
(a) Augmented Lagrangian with $c = 2$



(b) Augmented Lagrangian with $c = 5$



(c) Augmented Lagrangian with $c = 10$



(d) Augmented Lagrangian with $c = 100$

Figure 19.3: Level curves for Example 19.1, with $\lambda = 0$

We can also see that one of the eigenvalues of the Hessian matrix (19.7) is $c-1$, which tends toward infinity with c. This situation causes significant numerical difficulties for solving the unconstrained problems. Therefore, it is necessary to combine the two types of penalties in order to obtain an efficient algorithm.

19.3 Double penalty

By using both penalty types, we can obtain the advantages of the two approaches. It is important to note that the quadratic penalty method works better when the value of λ is close to λ^*. This is shown in Figure 19.4, where the first component of x_k,

obtained by solving (19.8), is reproduced for various values of c and λ. Clearly, when c is increased, the value tends faster toward 1 when λ is close to $\lambda^* = 1$.

It is therefore important to be able to obtain a good approximation of λ^*. The algorithm presented here is based not only on a sequence of penalty parameters $(c_k)_k$ such that $c_k \to \infty$, but also on a set of vectors $(\lambda_k)_k$ approximating λ^*. From a theoretical point of view, it is sufficient that the sequence $(\lambda_k)_k$ is bounded in order for the method to work.

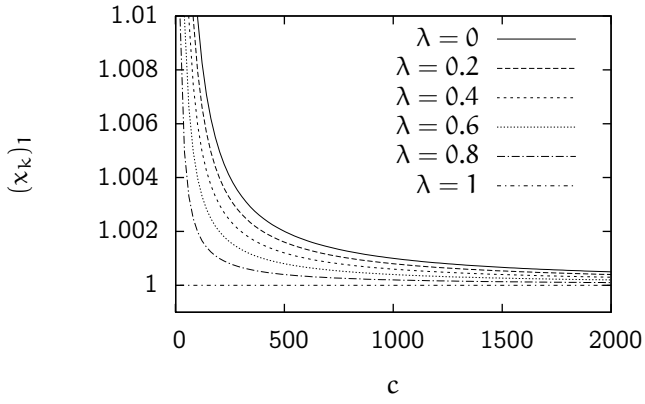


Figure 19.4: Efficiency of the quadratic penalty as a function of λ

Theorem 19.2 (Augmented Lagrangian method). *Let $f : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ be two continuous functions. Let $X \subset \mathbb{R}^n$ be a closed subset of $\mathbb{R}^n$ such that the set $\{x \in X \mid h(x) = 0\}$ is non empty. Consider a sequence $(c_k)_k$ such that, for all k, $c_k \in \mathbb{R}$, $0 < c_k < c_{k+1}$, and $\lim_{k\to\infty} c_k = +\infty$. Consider a bounded sequence $(\lambda_k)_k$ such that $\lambda_k \in \mathbb{R}^m$ for all k. Let x_k be the global minimum of augmented Lagrangian, that is*

$$x_k \in \operatorname{argmin}_{x\in X} L_{c_k}(x, \lambda_k) = f(x) + \lambda_k^T h(x) + \frac{c_k}{2} \left\|h(x)\right\|^2 . \tag{19.9}$$

Then, each limit point of the sequence $(x_k)_k$ is a global minimum of the problem $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$ and $x \in X$.

Proof. Let f^* be the optimal value of the constrained problem and let k be arbitrary.

$$\begin{aligned} f^* &= \min_{h(x)=0,\, x\in X} f(x) \\ &= \min_{h(x)=0,\, x\in X} f(x) + \lambda_k^T h(x) + \frac{c_k}{2} \left\|h(x)\right\|^2 \\ &= \min_{h(x)=0,\, x\in X} L_{c_k}(x, \lambda_k) . \end{aligned}$$

Since X is closed, $\{x \in X \mid h(x) = 0\}$ is non empty and f is continuous, f^* is finite.

By definition of x_k, a global minimum of (19.9), we have

$$L_{c_k}(x_k, \lambda_k) \leq L_{c_k}(x, \lambda_k), \qquad \forall x \in X. \tag{19.10}$$

Then, taking the minimum,

$$L_{c_k}(x_k, \lambda_k) \leq \min_{h(x)=0, x\in X} L_{c_k}(x, \lambda_k) = f^*,$$

and this for all k. When k tends toward infinity, $L_{c_k}(x_k, \lambda_k)$ is finite since this is also the case for f^*. Let $\bar{x}$ be a limit point of the sequence $(x_k)_k$ and $\bar{\lambda}$ a limit point of the sequence $(\lambda_k)_k$ (it exists because the sequence is bounded). Then, when going to the upper limit and taking into account the fact that the functions f and h are continuous, we get

$$\begin{aligned}
\limsup_{k\to\infty} L_{c_k}(x_k, \lambda_k) &\leq f^* \\
\limsup_{k\to\infty} f(x_k) + \lambda_k^T h(x_k) + \frac{c_k}{2}\|h(x_k)\|^2 &\leq f^* \\
f(\bar{x}) + \bar{\lambda}^T h(\bar{x}) + \limsup_{k\to\infty} \frac{c_k}{2}\|h(x_k)\|^2 &\leq f^*.
\end{aligned}$$

In order for the left term to remain finite, while $(c_k)_k \to \infty$, we require $(h(x_k))_k \to 0$ and then $h(\bar{x}) = 0$. The above expression simplifies to

$$f(\bar{x}) \leq f^*.$$

Moreover, since X is closed, we also have $\bar{x} \in X$. Therefore, $\bar{x}$ is indeed a global minimum to the problem. □

Note that if x^* is a local minimum to the constrained problem (19.1)–(19.2), then there exists a closed neighborhood $X \subseteq \mathbb{R}^n$, such that x^* is a global minimum for the problem $\min_{x\in X} f(x)$ subject to $h(x) = 0$, and the theorem applies. We have shown that the quadratic penalty enables us to solve the problem, as was demonstrated above. The following result enables us to find an approximation of λ^*.

Theorem 19.3 (Approximation of Lagrange multipliers). *Let f and h be continuously differentiable. Consider a sequence $(c_k)_k$ such that, for all k, $c_k \in \mathbb{R}$ and $0 < c_k < c_{k+1}$. Moreover, let us assume $\lim_{k\to\infty} c_k = +\infty$. Let $(\lambda_k)_k$ be a bounded sequence such that $\lambda_k \in \mathbb{R}^m$ for all k. Let $(\varepsilon_k)_k$ be a sequence such that $\varepsilon_k > 0$ for all k and $\lim_{k\to\infty} \varepsilon_k = 0$. Let $(x_k)_k$ be a sequence such that*

$$\|\nabla_x L_{c_k}(x_k, \lambda_k)\| \leq \varepsilon_k. \tag{19.11}$$

Let $(x_k)_{k\in K}$ be a subsequence of the sequence $(x_k)_k$ converging toward x^*. If $\nabla h(x^*)$ is of full rank, then

$$\lim_{k\in K, k\to\infty} \lambda_k + c_k h(x_k) = \lambda^*, \tag{19.12}$$

where x^* and λ^* satisfy the necessary first-order optimality conditions (6.23), i.e.,

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* = 0 \tag{19.13}$$
$$h(x^*) = 0\,. \tag{19.14}$$

Proof. We assume, without loss of generality, that the sequence $(x_k)_k$ converges toward x^* (by eliminating all terms such that $k \notin K$). We denote

$$\ell_k = \lambda_k + c_k h(x_k)\,. \tag{19.15}$$

From (19.3), we have

$$\begin{aligned}\nabla_x L_{c_k}(x_k, \lambda_k) &= \nabla f(x_k) + \nabla h(x_k)\lambda_k + c_k \nabla h(x_k) h(x_k) \\ &= \nabla f(x_k) + \nabla h(x_k)\big(\lambda_k + c_k h(x_k)\big)\end{aligned}$$

and using (19.15),

$$\nabla_x L_{c_k}(x_k, \lambda_k) = \nabla f(x_k) + \nabla h(x_k)\ell_k\,. \tag{19.16}$$

By continuity, since $\nabla h(x^*)$ is of full rank, starting from a sufficiently large k, $\nabla h(x_k)$ is also of full rank. Therefore,

$$\begin{aligned}\nabla_x L_{c_k}(x_k, \lambda_k) &= \nabla f(x_k) + \nabla h(x_k)\ell_k \\ \nabla h(x_k)^T \nabla_x L_{c_k}(x_k, \lambda_k) &= \nabla h(x_k)^T \nabla f(x_k) + \nabla h(x_k)^T \nabla h(x_k)\ell_k\end{aligned}$$

and

$$\ell_k = \big(\nabla h(x_k)^T \nabla h(x_k)\big)^{-1} \nabla h(x_k)^T \big(\nabla_x L_{c_k}(x_k, \lambda_k) - \nabla f(x_k)\big)\,.$$

According to (19.11), since $\varepsilon_k \to 0$, we have $\nabla_x L_{c_k}(x_k, \lambda_k) \to 0$. When k tends toward infinity, we obtain

$$\lambda^* = \lim_{k\to\infty} \ell_k = -\big(\nabla h(x^*)^T \nabla h(x^*)\big)^{-1} \nabla h(x^*)^T \nabla f(x^*)\,.$$

By making k tend toward infinity in (19.16), as $\nabla_x L_{c_k}(x_k, \lambda_k) \to 0$, we obtain (19.13)

$$\nabla f(x^*) + \nabla h(x^*)\lambda^* = 0\,.$$

Since $\ell_k = \lambda_k + c_k h(x_k) \to \lambda^*$ and $(\lambda_k)_k$ is bounded, then $\big(c_k h(x_k)\big)_k$ is also bounded. Since $c_k \to \infty$, then $h(x_k) \to 0$ and we get (19.14),

$$h(x^*) = 0\,.$$

The arguments used in this proof are similar to those used to prove Theorem 6.10. □

This result enables us to define the sequence $(\lambda_k)_k$ as follows:

$$\lambda_{k+1} = \lambda_k + c_k h(x_k)\,. \tag{19.17}$$

Algorithm 19.1: Augmented Lagrangian algorithm

Objective

To find a local minimum of the problem (1.71)–(1.72):
$\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$.

Input

The twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
The Hessian of the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.
The twice differentiable constraint $h : \mathbb{R}^n \to \mathbb{R}^m$.
The gradient matrix of the constraint $\nabla h : \mathbb{R}^n \to \mathbb{R}^{n\times m}$.
The Hessian $\nabla^2 h_i : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ of each constraint $i = 1, \ldots, m$.
An initial feasible solution (x_0, λ_0).
An initial penalty parameter c_0 (by default $c_0 = 10$).
The required precision $\varepsilon > 0$.

Output

An approximation of the optimal solution (x^*, λ^*).

Initialization

$k := 0$.
$\widehat{\eta}_0 := 0.1258925$. `Value chosen so that` $\eta_0 = 0.1$.
$\tau := 10$.
$\alpha := 0.1$.
$\beta := 0.9$.
$\varepsilon_k := 1/c_0$.
$\eta_k := \widehat{\eta}_0/c_0^{\alpha}$.

Repeat

Use Newton's method with line search (Algorithm 11.8) or with trust region (Algorithm 12.4) to solve

$$x_{k+1} \in \operatorname{argmin}_{x\in\mathbb{R}^n} L_{c_k}(x, \lambda_k) = f(x) + \lambda_k h(x) + \frac{c_k}{2}\left\|h(x)\right\|^2, \quad (19.18)$$

by using x_k as the starting point and ε_k as the precision.

if $\left\|h(x_k)\right\| \le \eta_k$. **then** we update the multipliers
  $\lambda_{k+1} := \lambda_k + c_k h(x_k)$.
  $c_{k+1} := c_k$, `` `c_k` is not modified. ``
  $\varepsilon_{k+1} := \varepsilon_k/c_k$, `precision is increased.`
  $\eta_{k+1} := \eta_k/c_k^{\beta}$, `feasibility requirement is increased.`
else we update the penalty parameter
  $\lambda_{k+1} := \lambda_k$, `` `λ_k` is not modified. ``
  $c_{k+1} := \tau c_k$, `penalty is increased.`
  $\varepsilon_{k+1} := \varepsilon_0/c_{k+1}$, `precision reset.`
  $\eta_{k+1} := \widehat{\eta}_0/c_{k+1}^{\alpha}$, `feasibility requirement reset.`
$k := k + 1$.

Until $\left\|\nabla L(x_k, \lambda_k)\right\| \le \varepsilon$ *and* $\left\|h(x_k)\right\|^2 \le \varepsilon$

Example 19.4 (Lagrangian penalty – cont.). Consider Example 19.1 again and apply the update (19.17) to obtain

$$\lambda_{k+1} = \lambda_k + c_k \left(\frac{c_k - \lambda_k}{c_k - 1} - 1 \right) = \frac{-\lambda_k + c_k}{c_k - 1}.$$

We examine the convergence of this sequence toward $\lambda^* = 1$.

$$\lambda_{k+1} - \lambda^* = \frac{-\lambda_k + c_k - \lambda^*(c_k - 1)}{c_k - 1} = \frac{-\lambda_k + \lambda^*}{c_k - 1}.$$

Therefore, for λ_{k+1} to be closer to λ^* than λ_k, so that the sequence converges, we need $c_k > 2$. We see again that, in order for the method to work, the value of c_k should be sufficiently large.

Theorem 19.3 is now used as a basis to define the algorithm. Indeed, we need to specify the sequences of penalty parameters $(c_k)_k$, of parameter $(\lambda_k)_k$, and of parameters ε_k to obtain an algorithm. At each iteration,

1. we solve $\min_x L_c(x, \lambda_k)$ at a precision ε_k, by using an appropriate algorithm shown in Chapters 11, 12 or 13 to obtain x_{k+1};
2. if x_{k+1} is "sufficiently" feasible, we update the Lagrange multipliers λ_k, by using (19.12);
3. otherwise, we increase the penalty parameter c_k.

The values of the parameters proposed in Algorithm 19.1 are taken from the LANCELOT software (Conn et al., 1992).

Example 19.5 (Augmented Lagrangian). Consider the problem

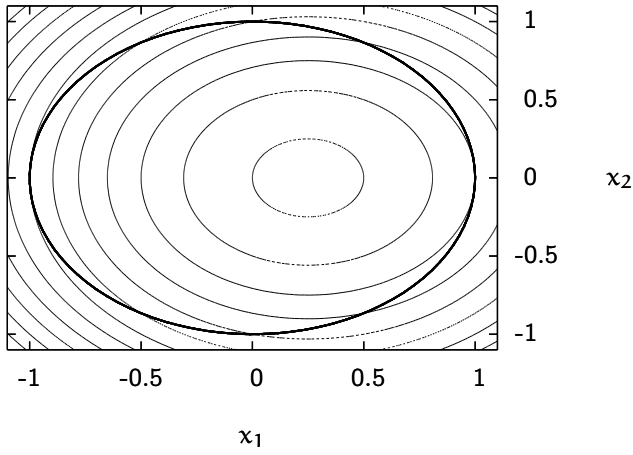
$$\min_{x \in \mathbb{R}^2} 2(x_1^2 + x_2^2 - 1) - x_1$$

subject to

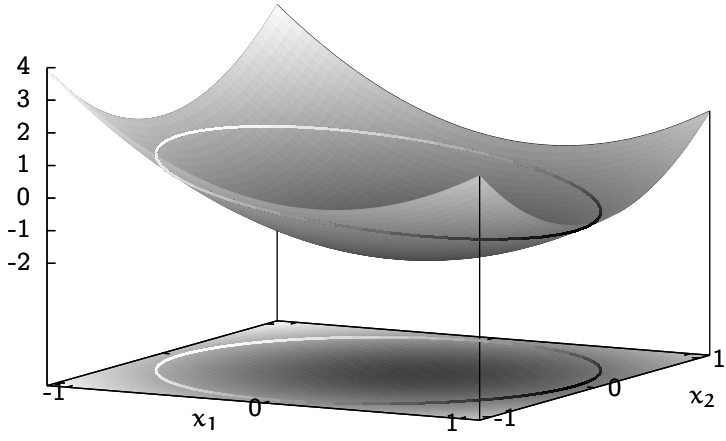
$$x_1^2 + x_2^2 = 1$$

shown in Figure 19.5. Table 19.1 lists the values of the iterates, as well as the norm of the gradient of the Lagrangian. Table 19.2 lists the values of c_k, $\left\|\nabla_x L_{c_k}(x_k, \lambda_k)\right\|$, ε_k, $\left\|h(x_k)\right\|$ and η_k during the iterations. The last column gives the number of iterations required to solve the problem (19.18). We can see that the penalty parameter is increased during the first iteration, because the constraint satisfaction was insufficient ($\left\|h(x_k)\right\| = 1.45292\text{e-}01$, while $\eta_k = 1\text{e-}01$).

At the subsequent iterations, the value of the multiplier λ_k has been updated. The path of the algorithm is presented in solid lines starting from $x_0 = \begin{pmatrix} -1 & 0.1 \end{pmatrix}^T$ and in dashed lines starting from $\bar{x}_0 = \begin{pmatrix} 0 & -0.1 \end{pmatrix}^T$, in Figure 19.6. It is interesting to note the way it approximately "follows" the constraint. Figure 19.7 shows the evolution of the level curves of the augmented Lagrangian around the optimal solution $x^* = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ during the 4 first iterations.



(a) Level curves



(b) Function

Figure 19.5: Problem for Example 19.5

Table 19.1: Iterates of the augmented Lagrangian method for Example 19.5

k	x_1	x_2	λ	$\|\nabla_x L(x_k,\lambda_k)\|$	$\|h(x_k)\|$
0	-1.00000e+00	1.00000e-01	0.00000e+00	5.01597e+00	1.00000e-02
1	9.24487e-01	5.51255e-03	0.00000e+00	2.69804e+00	1.45292e-01
2	9.92491e-01	-1.83076e-05	-1.49616e+00	1.07375e-04	1.49616e-02
3	9.99981e-01	1.03829e-07	-1.49999e+00	5.44820e-07	3.82637e-05
4	1.00000e+00	-7.98650e-10	-1.50000e+00	2.18509e-07	9.70011e-08
5	1.00000e+00	1.52816e-14	-1.50000e+00	1.36610e-12	1.33171e-09

Table 19.2: Iterates of the augmented Lagrangian method for Example 19.5 (cont.)

k	c_k	$\|\nabla_x L_{c_k}(x_k,\lambda_k)\|$	ε_k	$\|h(x_k)\|$	η_k	
1	10	1.30108e-02	1e-01	1.45292e-01	1.00000e-01	15
2	100	1.07375e-04	1e-02	1.49616e-02	7.94328e-02	8
3	100	5.44820e-07	1e-04	3.82637e-05	1.25892e-03	4
4	100	2.18509e-07	1e-06	9.70011e-08	1.99526e-05	1
5	100	1.36628e-12	1e-08	1.33171e-09	3.16228e-07	1
6	100	1.33227e-15	1e-10	3.32778e-12	5.01187e-09	1

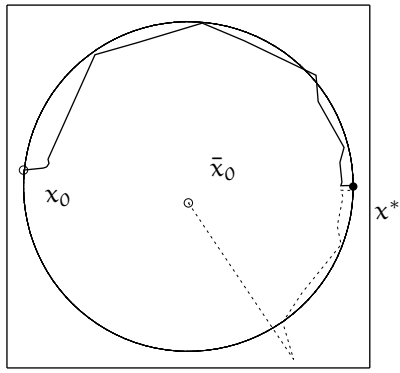
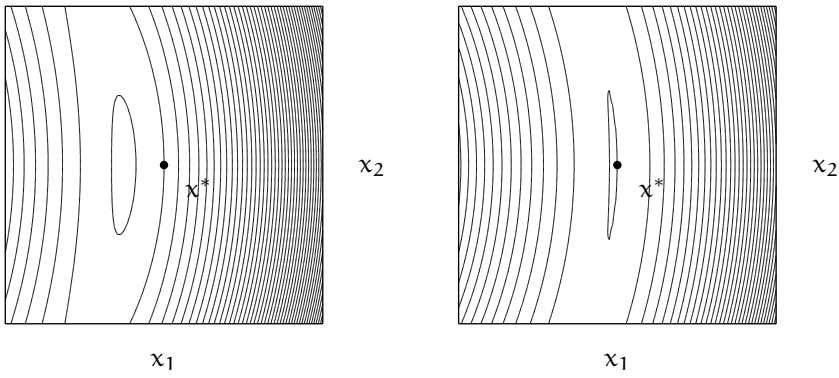
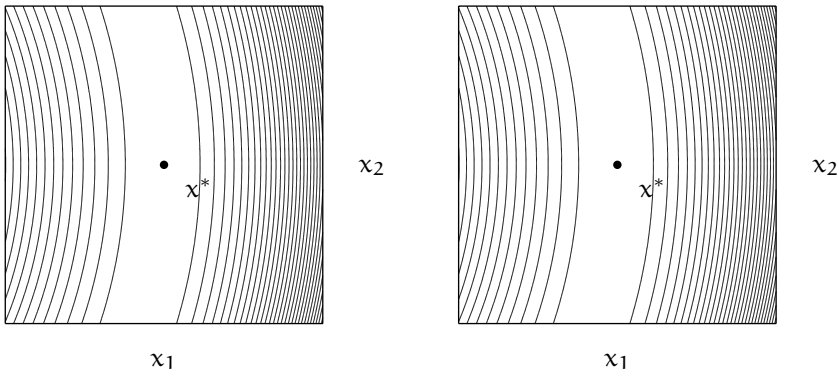


Figure 19.6: Augmented Lagrangian: iterations for Example 19.5



(a) $k = 1$, $c = 10.0$, $\lambda = 0$ (b) $k = 2$, $c = 100.0$,$\lambda = 0$



(c) $k = 3$, $c = 100.0$, $\lambda = -1.49616$ (d) $k = 4$, $c = 100.0$, $\lambda = -1.49998$

Figure 19.7: Level curves for the augmented Lagrangian for Example 19.5

Example 19.6 (Augmented Lagrangian with the constrained Rosenbrock problem)**.** The Rosenbrock problem (Chapter 11.6) is difficult. We consider a constrained version of this problem

$$\min_{x \in \mathbb{R}^2} 100 \left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

subject to

$$x_1 - x_2^2 - \frac{1}{2} = 0\,,$$

shown in Figure 19.8. The iterations are listed in Table 19.3 and the evolution of the parameters in Table 19.4.

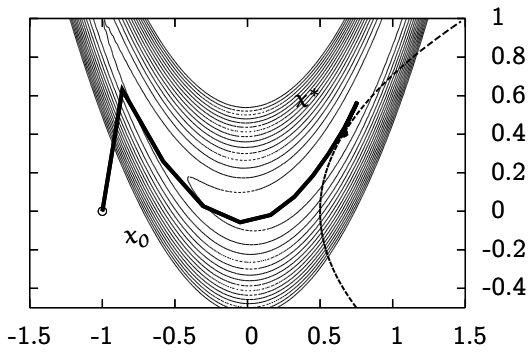


Figure 19.8: Augmented Lagrangian: iterations for Example 19.6

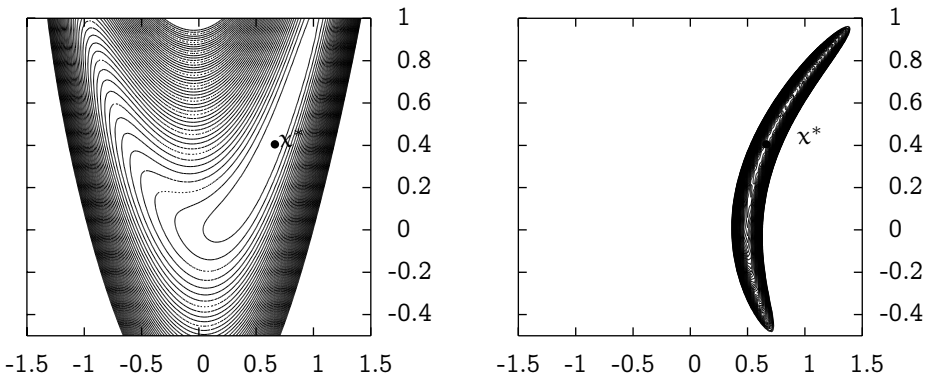
Table 19.3: Iterates of the augmented Lagrangian method for Example 19.6

k	x_1	x_2	λ	$\|\nabla_x L(x_k,\lambda_k)\|$	$\|h(x_k)\|$
0	-1.00000e+00	0.00000e+00	0.00000e+00	4.50795e+02	1.50000e+00
1	7.56394e-01	5.68279e-01	-6.65475e-01	1.90681e-02	6.65475e-02
2	7.23548e-01	5.17764e-01	-6.65475e-01	6.43621e-01	4.45321e-02
3	6.80054e-01	4.49567e-01	-2.87105e+00	1.41250e-04	2.20558e-02
4	6.69717e-01	4.29750e-01	-2.87105e+00	1.97370e+00	1.49681e-02
5	6.64093e-01	4.10635e-01	-7.39946e+00	1.78647e-04	4.52841e-03
6	6.63957e-01	4.06281e-01	-7.39946e+00	1.42593e+00	1.10665e-03
7	6.64017e-01	4.05166e-01	-8.82418e+00	1.44475e-08	1.42472e-04
8	6.64029e-01	4.05011e-01	-8.86988e+00	7.57362e-13	4.56917e-06

Table 19.4: Iterates of the augmented Lagrangian method for Example 19.6 cont.

k	c_k	$\|\nabla_x L_{c_k}(x_k,\lambda_k)\|$	ε_k	$\|h(x_k)\|$	η_k	
1	10	1.90681e-02	1e-01	6.65475e-02	1.00000e-01	11
2	10	2.74456e-03	1e-02	4.45321e-02	1.25892e-02	2
3	100	1.41250e-04	1e-02	2.20558e-02	7.94328e-02	3
4	100	2.61580e-05	1e-04	1.49681e-02	1.25892e-03	2
5	1000	1.78647e-04	1e-03	4.52841e-03	6.30957e-02	3
6	1000	1.30511e-08	1e-06	1.10665e-03	1.25892e-04	3
7	10000	1.44475e-08	1e-04	1.42472e-04	5.01187e-02	3
8	10000	7.57636e-13	1e-08	4.56917e-06	1.25892e-05	3

It appears that the algorithm quickly identifies the neighborhood of the optimal solution, but that it cannot satisfy the constraint with high precision. It must thus increase c_k, which affects the conditioning of the problem. The close-packed level curves of the augmented Lagrangian, shown in Figure 19.9(b), highlight this phenomenon.



(a) $c_k = 10$, $\lambda_k = 0$ (b) $c_k = 10\,000$, $\lambda_k = -8.86988$

Figure 19.9: Level curves of the augmented Lagrangian for Example 19.6

The presentation of the proof for Theorems 19.2 and 19.3, as well as Example 19.1, was inspired by Bertsekas (1999).

19.4 Project

The general organization of the projects are described in Appendix D.

Objective

The aim of this project is to implement the augmented Lagrangian to solve various problems and analyze the role of different parameters on its efficiency.

Approach

Analyze the impact of the following parameters.

1. Penalty parameter: utilize the initial values $c_0 = 1$, 10, and 100 and the augmentation rates $\tau = 1$, 2, 10 and 100. Analyze the impact of the algorithm itself and also on the behavior of the unconstrained algorithm used to solve the subproblem of step 11.
2. Precision of the constraints: use the values $\beta = 0.1$, 0.5, and 0.9. Analyze the impact on the approximation of the dual variables.

Algorithm

Algorithm 19.1.

Problems

Exercise 19.1. The problem

$$\min e^{x_1 - 2x_2}$$

subject to

$$\begin{aligned} \sin(-x_1 + x_2 - 1) &= 0 \\ -2 \le x_1 &\le 2 \\ -1.5 \le x_2 &\le 1.5\,. \end{aligned}$$

Exercise 19.2. The problem

$$\min_{x \in \mathbb{R}^{10}} \sum_{i=1}^{10} e^{x_i} \left(c_i + x_i - \ln \left(\sum_{k=1}^{10} e^{x_k} \right) \right)$$

subject to

$$\begin{aligned} e^{x_1} + 2\,e^{x_2} + 2\,e^{x_3} + e^{x_6} + e^{x_{10}} &= 2 \\ e^{x_4} + 2\,e^{x_5} + e^{x_6} + e^{x_7} &= 1 \\ e^{x_3} + e^{x_7} + e^{x_8} + 2\,e^{x_9} + e^{x_{10}} &= 1 \\ -100 \le x_i \le 100\,, \quad i &= 1, \ldots, 10\,, \end{aligned}$$

with

$$\begin{aligned} c_1 &= -6.089 & c_6 &= -14.986 \\ c_2 &= -17.164 & c_7 &= -24.1 \\ c_3 &= -34.054 & c_8 &= -10.708 \\ c_4 &= -5.914 & c_9 &= -26.662 \\ c_5 &= -24.721 & c_{10} &= -22.179\,. \end{aligned}$$

Solution:

$$\begin{aligned} x_1^* &= -3.40629 & x_6^* &= -4.44283 \\ x_2^* &= -0.596369 & x_7^* &= -1.41264 \\ x_3^* &= -1.1912 & x_8^* &= -21.6066 \\ x_4^* &= -4.62689 & x_9^* &= -2.26867 \\ x_5^* &= -1.0011 & x_{10}^* &= -1.40346\,. \end{aligned}$$

Proposed by Hock and Schittkowski (1981).

Exercise 19.3. The problem

$$\min_{x \in \mathbb{R}^2} \ln(1 + x_1^2) - x_2$$

subject to

$$\begin{aligned} \left(1 + x_1^2\right)^2 + x_2^2 &= 4 \\ -4 \le x_1 &\le 4 \\ -4 \le x_2 &\le 4\,. \end{aligned}$$

Please note: do not forget to transform the formulation of the problems so that they are compatible with (19.1)–(19.2).

Chapter 20

Sequential quadratic programming

Contents

In this chapter, we consider the optimization problem (1.71)-(1.72), i.e.,

$$\min_{x \in \mathbb{R}^n} f(x) \tag{20.1}$$

subject to

$$h(x) = 0\,, \tag{20.2}$$

where f is a function of $\mathbb{R}^n$ in $\mathbb{R}$ and h a function of $\mathbb{R}^n$ in $\mathbb{R}^m$. Remember that it is always possible to transform an inequality constraint

$$g_i(x) \leq 0$$

with $g_i : \mathbb{R}^n \to \mathbb{R}$ into an equality constraint by introducing slack variables $z_i \in \mathbb{R}$ (see Section 1.2.2 and Example 6.17), to get

$$g_i(x) + z_i^2 = 0\,.$$

The basic idea of the algorithm that we develop is simple. Just as for unconstrained optimization, the necessary optimality conditions (6.23) constitute a system of non linear equations (Theorem 6.10). The methods presented in Chapters 7 and 8 are relevant in this context. We start by applying Newton's local method.

20.1 Local sequential quadratic programming

We need to find primal variables x^* and dual variables λ^* such that the gradient of the Lagrangian of the problem (Definition 4.3) is zero, i.e.,

$$\nabla L(x^*, \lambda^*) = 0\,,$$

with

$$L(x, \lambda) = f(x) + \lambda^T h(x)\,.$$

We have

$$\begin{aligned}
\nabla L(x, \lambda) &= \begin{pmatrix} \nabla f(x) + \nabla h(x)\lambda \\ h(x) \end{pmatrix} = \begin{pmatrix} \nabla_x L(x, \lambda) \\ h(x) \end{pmatrix} \qquad (20.3)\\
\nabla^2 L(x, \lambda) &= \begin{pmatrix} \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 h_i(x) & \nabla h(x) \\ \nabla h(x)^T & 0 \end{pmatrix} \\
&= \begin{pmatrix} \nabla^2_{xx} L(x, \lambda) & \nabla h(x) \\ \nabla h(x)^T & 0 \end{pmatrix}.
\end{aligned}$$

Then, an iteration k of Newton's method consists in finding a direction $d \in \mathbb{R}^{n+m}$ such that

$$\nabla^2 L(x, \lambda) d = -\nabla L(x, \lambda)\,,$$

i.e., finding $d_x \in \mathbb{R}^n$ and $d_\lambda \in \mathbb{R}^m$ such that

$$\begin{aligned}
\nabla^2_{xx} L(x, \lambda) d_x + \nabla h(x) d_\lambda &= -\nabla_x L(x, \lambda) \\
\nabla h(x)^T d_x &= -h(x)\,.
\end{aligned} \qquad (20.4)$$

It is interesting to note that Equations (20.4) are the optimality conditions of the following quadratic optimization problem:

$$\min_d \nabla_x L(x_k, \lambda_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d \qquad (20.5)$$

subject to

$$\nabla h(x_k)^T d + h(x_k) = 0\,. \qquad (20.6)$$

If $\ell \in \mathbb{R}^m$ is the vector of dual variables for the above problem, the Lagrangian of (20.5)–(20.6) is

$$L^{PQ}(d, \ell) = \nabla_x L(x_k, \lambda_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d + \ell^T (\nabla h(x_k)^T d + h(x_k))$$

and the necessary optimality conditions are

$$\begin{aligned}
\nabla_d L^{PQ}(d^*, \ell^*) &= \nabla_x L(x_k, \lambda_k) + \nabla^2_{xx} L(x_k, \lambda_k) d^* + \nabla h(x_k) \ell^* = 0 \\
\nabla_l L^{PQ}(d^*, \ell^*) &= \nabla h(x_k)^T d^* + h(x_k) = 0\,.
\end{aligned}$$

We now need to take $d^* = d_x$ and $\ell^* = d_\lambda$ to obtain (20.4).

In the context of unconstrained optimization, we have seen that the calculation of the Newton step amounted to the optimization of a quadratic model of the function (Algorithm 10.2). In the context of constrained optimization, the calculation of the Newton step amounts to the optimization of a quadratic function with linear constraints. This is the problem (20.5)–(20.6).

We now simplify the formulation somewhat. By using (20.3), the Newton equations (20.4) can be written as

$$\begin{aligned}\nabla^2_{xx}L(x_k,\lambda_k)d_x + \nabla h(x_k)d_\lambda &= -\nabla f(x_k) - \nabla h(x_k)\lambda_k\\ \nabla h(x_k)^T d_x &= -h(x_k)\,.\end{aligned}$$

Define $\widehat{d}_\lambda = d_\lambda + \lambda_k$ to obtain

$$\nabla^2_{xx}L(x_k,\lambda_k)d_x + \nabla h(x_k)\widehat{d}_\lambda = -\nabla f(x_k) \tag{20.7}$$

$$\nabla h(x_k)^T d_x = -h(x_k)\,. \tag{20.8}$$

We are here also dealing with optimality conditions for a quadratic problem

$$\min_d \nabla f(x_k)^T d + \frac{1}{2}\, d^T \nabla^2_{xx}L(x_k,\lambda_k)d \tag{20.9}$$

subject to

$$\nabla h(x_k)^T d + h(x_k) = 0\,. \tag{20.10}$$

According to Theorem 6.38, the optimal solution to this quadratic problem is

$$\lambda^* = H^{-1}\left(h(x_k) - \nabla h(x_k)^T \nabla^2_{xx}L(x_k,\lambda_k)^{-1}\nabla f(x_k)\right), \tag{20.11}$$

with $H = \nabla h(x_k)^T \nabla^2_{xx}L(x_k,\lambda_k)^{-1}\nabla h(x_k)$ and

$$x^* = -\nabla^2_{xx}L(x_k,\lambda_k)^{-1}\left(\nabla h(x_k)\lambda^* + \nabla f(x_k)\right). \tag{20.12}$$

It is important to note that, in practice, specialized algorithms should be used to solve the quadratic problem.[1] The analytic solution is computationally intense, and numerical issues may occur.

Example 20.1 (Quadratic problem in SQP)**.** Consider the problem

$$\min f(x) = x_1 + x_2$$

subject to

$$h(x) = x_1^2 + (x_2 - 1)^2 - 1 = 0\,.$$

The Lagrangian is

$$L(x,\lambda) = x_1 + x_2 + \lambda x_1^2 + \lambda x_2^2 - 2\lambda x_2\,.$$

[1] We refer the interested reader to more detailed discussion on SQP methods in the literature, such as Gould and Toint (2000), and Gill and Wong (2012).

Algorithm 20.1: Local SQP algorithm

Objective

To find a local minimum of the problem (1.71)–(1.72),

$$\min_{x\in\mathbb{R}^n} f(x) \qquad \text{subject to} \quad h(x) = 0\,.$$

Input

The twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$.
The gradient of the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$.
The Hessian of the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.
The differentiable constraint $h : \mathbb{R}^n \to \mathbb{R}^m$.
The gradient matrix of the constraint $\nabla h : \mathbb{R}^n \to \mathbb{R}^{n\times m}$.
The Hessian $\nabla^2 h_i : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ of each constraint $i = 1, \ldots, m$.
An initial solution (x_0, λ_0).
The required precision $\varepsilon > 0$.

Output

An approximation of the optimal solution (x^*, λ^*).

Initialization

$k := 0$.

Repeat

Calculate $\nabla^2_{xx} L(x_k, \lambda_k) = \nabla^2 f(x_k) + \sum_{i=1}^{m} (\lambda_k)_i \nabla^2 h_i(x_k)\,.$

Obtain d_x and d_λ by solving the quadratic problem

$$\min_d \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d$$

subject to

$$\nabla h(x_k)^T d + h(x_k) = 0\,,$$

with an appropriate algorithm. To illustrate the algorithm, we can use (20.11) and (20.12).

$x_{k+1} := x_k + d_x$.
$\lambda_{k+1} := d_\lambda$.
$k := k + 1$.

Until $\|\nabla L(x_k, \lambda_k)\| \le \varepsilon$.

Then,

$$\nabla L(x, \lambda) = \begin{pmatrix} 1 + 2\lambda x_1 \\ 1 + 2\lambda x_2 - 2\lambda \\ x_1^2 + x_2^2 - 2x_2 \end{pmatrix}$$

$$\nabla^2 L(x, \lambda) = \begin{pmatrix} 2\lambda & 0 & 2x_1 \\ 0 & 2\lambda & 2x_2 - \lambda \\ 2x_1 & 2x_2 - 2 & 0 \end{pmatrix}.$$

The quadratic problem to be solved at each iteration is

$$\min_{d \in \mathbb{R}^2} d_1 + d_2 + \lambda_k d_1^2 + \lambda_k d_2^2$$

subject to

$$2x_1 d_1 + (2x_2 - 2)\, d_2 + x_1^2 + (x_2 - 1)^2 - 1 = 0\,.$$

Since the solving of Karush-Kuhn-Tucker equations by Newton's method consists in solving a sequence of problems, or quadratic *programs*, the thus-obtained algorithm is called the *sequential quadratic programming* (SQP) algorithm. Evidently, this method has the same characteristics as Newton's method and is not globally convergent. We add the adjective "local" to describe it in Algorithm 20.1.

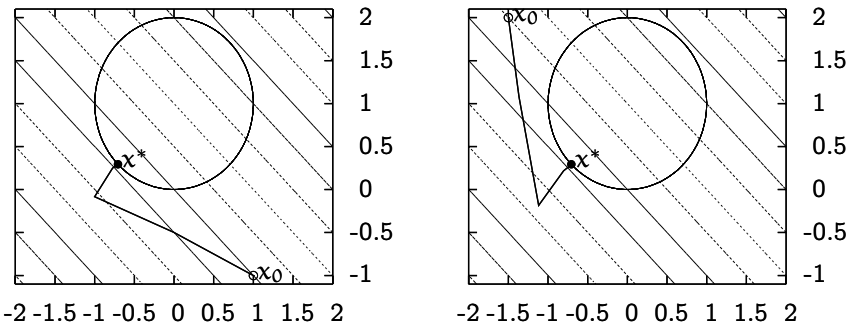
Example 20.2 (Local SQP algorithm – I)**.** Consider the problem

$$\min f(x) = x_1 + x_2$$

subject to

$$h(x) = x_1^2 + (x_2 - 1)^2 - 1 = 0\,.$$

Figure 20.1 and Tables 20.1 and 20.2 demonstrate the application of the local SQP algorithm to this problem. We note that the algorithm quickly finds an optimal solution. It has the speed of convergence of Newton's method. The algorithm also suffers from the drawbacks of this method.

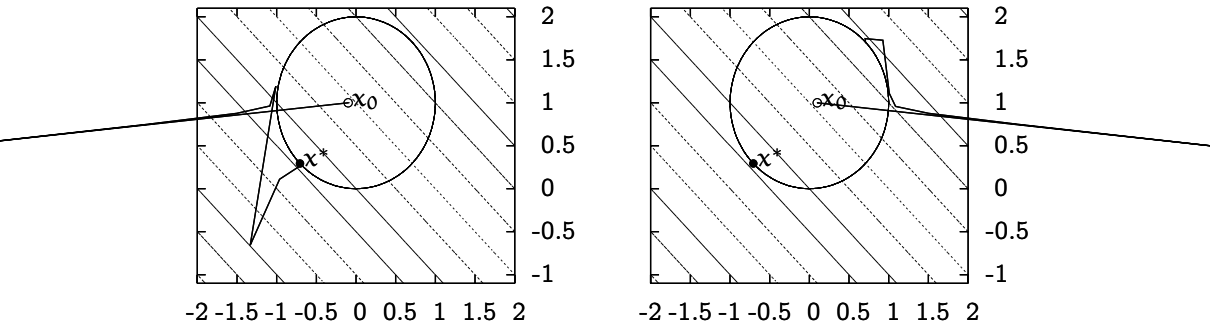


(a) $x_0 = (1 \quad -1)^T$ (b) $x_0 = (-3/2 \quad 2)^T$

Figure 20.1: Illustrations of the local SQP algorithm for Example 20.2

Table 20.1: Illustration of the local SQP algorithm for Example 20.2; $x_0 = \begin{pmatrix} 1 & -1 \end{pmatrix}^T$

k	x_1	x_2	λ	$\|\nabla L_{xx}\|$
0	1.00000e+00	-1.00000e+00	1.00000e+00	5.83095e+00
1	0.00000e+00	-5.00000e-01	5.00000e-01	1.67705e+00
2	-1.00000e+00	-8.33333e-02	4.72222e-01	1.17515e+00
3	-7.74009e-01	2.49726e-01	6.06718e-01	1.94849e-01
4	-7.07425e-01	2.88997e-01	6.98180e-01	1.53511e-02
5	-7.07135e-01	2.92911e-01	7.07075e-01	7.14908e-05
6	-7.07107e-01	2.92893e-01	7.07107e-01	2.38157e-09



(a) $x_0 = \begin{pmatrix} -0.1 & 1 \end{pmatrix}^T$ (b) $x_0 = \begin{pmatrix} 0.1 & 1 \end{pmatrix}^T$

Figure 20.2: Illustrations of the local SQP algorithm for Example 20.2

Table 20.2: Illustration of the local SQP algorithm for Example 20.2; $x_0 = \begin{pmatrix} -3/2 & 2 \end{pmatrix}^T$

k	x_1	x_2	λ	$\|\nabla L_{xx}\|$
0	-1.50000e+00	2.00000e+00	1.00000e+00	4.25000e+00
1	-1.36538e+00	1.07692e+00	4.23077e-01	1.38412e+00
2	-1.11784e+00	-1.85423e-01	4.42901e-01	1.65558e+00
3	-8.03520e-01	2.16153e-01	5.71828e-01	2.91415e-01
4	-7.09901e-01	2.86072e-01	6.88886e-01	3.05735e-02
5	-7.07179e-01	2.92927e-01	7.06965e-01	2.72185e-04
6	-7.07107e-01	2.92893e-01	7.07107e-01	2.34061e-08

For instance, in the center of the circle of constraints, i.e., at the point $(0, 1)$, the matrix $\nabla h(x)$ is zero. Then, the matrix H in (20.11) is zero and consequently not invertible. Note that, when we start the algorithm from a point close to $(0, 1)$, it has a tendency to take big steps (Figure 20.2(a) and Table 20.3). Finally, note that there is no guarantee that a minimum can be found, as shown in Figure 20.2(b) and Table 20.4. Indeed, the only objective of the algorithm is to zero the gradient of the Lagrangian.

Table 20.3: Illustration of the local SQP algorithm for Example 20.2; $x_0 = \begin{pmatrix} -0.1 & 1 \end{pmatrix}^T$

k	x_1	x_2	λ	$\|\nabla L_{xx}\|$
0	-1.00000e-01	1.00000e+00	1.00000e+00	1.61867e+00
1	-5.05000e+00	5.00000e-01	-4.45000e+01	4.53418e+02
2	-2.62404e+00	7.50317e-01	-2.12782e+01	1.13424e+02
3	-1.50286e+00	8.78262e-01	-8.90106e+00	2.79633e+01
4	-1.08612e+00	9.63643e-01	-2.13558e+00	5.75895e+00
5	-1.01047e+00	1.19247e+00	3.11609e-01	1.18100e+00
6	-1.33383e+00	-6.56144e-01	3.95100e-01	3.53584e+00
7	-9.63788e-01	1.09118e-01	4.84472e-01	7.38361e-01
8	-7.22734e-01	2.53867e-01	6.39958e-01	1.17880e-01
9	-7.08899e-01	2.93445e-01	7.04068e-01	5.65591e-03
10	-7.07103e-01	2.92887e-01	7.07103e-01	1.19518e-05
11	-7.07107e-01	2.92893e-01	7.07107e-01	7.90184e-11

Table 20.4: Illustration of the local SQP algorithm for Example 20.2; $x_0 = \begin{pmatrix} 0.1 & 1 \end{pmatrix}^T$

k	x_1	x_2	λ	$\|\nabla L_{xx}\|$
0	1.00000e-01	1.00000e+00	1.00000e+00	1.84935e+00
1	5.05000e+00	5.00000e-01	-5.45000e+01	5.52800e+02
2	2.62404e+00	7.50277e-01	-2.62802e+01	1.37776e+02
3	1.50282e+00	8.77817e-01	-1.14197e+01	3.35627e+01
4	1.08576e+00	9.59069e-01	-3.50192e+00	6.73105e+00
5	1.00831e+00	1.11015e+00	-7.10297e-01	9.48330e-01
6	9.26496e-01	1.72824e+00	-5.53513e-01	4.35129e-01
7	7.02912e-01	1.74580e+00	-6.73242e-01	7.35796e-02
8	7.08697e-01	1.70662e+00	-7.05786e-01	3.01674e-03
9	7.07106e-01	1.70711e+00	-7.07105e-01	5.18652e-06
10	7.07107e-01	1.70711e+00	-7.07107e-01	1.55080e-11

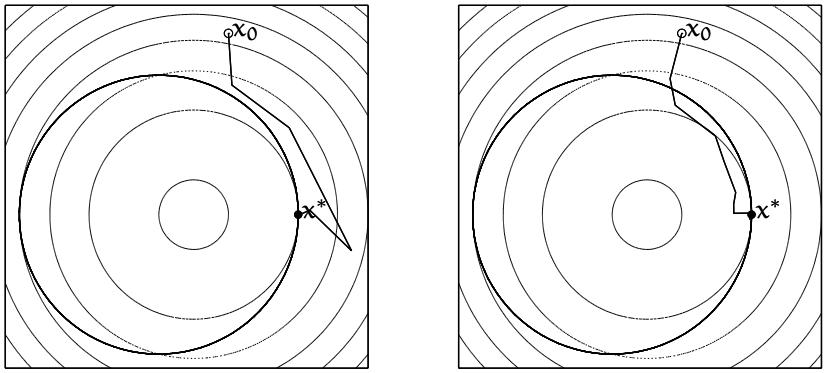
Example 20.3 (Local SQP algorithm – II)**.** We apply the local SQP algorithm to the problem in Example 19.5, i.e.,

$$\min_{x\in\mathbb{R}^2} 2(x_1^2 + x_2^2 - 1) - x_1$$

subject to

$$x_1^2 + x_2^2 = 1$$

shown in Figure 19.5. The iterations are listed in Table 20.5 and presented in Figure 20.3(a). It is interesting to compare the iterations with those of the augmented Lagrangian method (Figure 20.3(b)).



(a) SQP (b) Augmented Lagrangian

Figure 20.3: SQP and augmented Lagrangian: iterations for Example 20.3

Table 20.5: Illustration of the local SQP algorithm for Example 20.3; $x_0 = \begin{pmatrix}0.5 & 1.3\end{pmatrix}^T$

k	x_1	x_2	λ	$\|\nabla L_{xx}\|$
0	5.00000e-01	1.30000e+00	1.00000e+00	8.10701e+00
1	5.24055e-01	9.29210e-01	-1.14433e+00	1.59951e+00
2	9.35594e-01	6.22819e-01	-1.71786e+00	6.44709e-01
3	1.38229e+00	-2.59531e-01	-1.60029e+00	1.00534e+00
4	1.08314e+00	3.14974e-02	-1.55178e+00	1.78831e-01
5	1.00374e+00	-3.25035e-03	-1.50552e+00	1.09864e-02
6	1.00001e+00	3.61321e-05	-1.50003e+00	5.99917e-05
7	1.00000e+00	-1.92868e-09	-1.50000e+00	2.50640e-09
8	1.00000e+00	2.67876e-18	-1.50000e+00	2.67876e-18

We note that the latter attempts to follow the constraint, thereby requiring more iterations. The SQP method is much faster in this case. However, let us keep in mind that the local SQP method is not globally convergent.

Example 20.4 (SQP with the constrained Rosenbrock problem)**.** Consider again the problem of Example 19.6, i.e.,

$$\min_{x\in\mathbb{R}^2} 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

subject to

$$x_1 - x_2^2 - \frac{1}{2} = 0\,,$$

and use the SQP method to solve it. The iterations are listed in Table 20.6 and illustrated in Figure 20.4.

Table 20.6: Illustration of the local SQP algorithm for Example 20.4; $x_0 = \begin{pmatrix}-1 & 0\end{pmatrix}^T$, $\lambda_0 = 1$

k	x_1	x_2	λ	$\|\nabla L_{xx}\|$
0	-1.00000e+00	0.00000e+00	1.00000e+00	4.49901e+02
1	5.00000e-01	-2.02020e+00	-5.90919e+02	2.84494e+03
2	4.86136e-01	-1.00667e+00	-2.34944e+02	7.21646e+02
3	3.95205e-01	-4.51284e-01	-7.00969e+01	1.86409e+02
4	3.54255e-01	-6.41642e-02	-1.84751e+01	4.09255e+01
5	4.73514e-01	1.74309e-01	-1.30510e+01	7.15123e+00
6	5.71075e-01	2.91031e-01	-5.93423e+00	3.76956e+00
7	6.39843e-01	3.85770e-01	-4.56241e+00	1.42789e+00
8	6.76129e-01	4.21168e-01	-8.56764e+00	5.16695e-01
9	6.63972e-01	4.05248e-01	-8.74417e+00	5.46174e-02
10	6.64028e-01	4.05004e-01	-8.87130e+00	6.31475e-05
11	6.64029e-01	4.05006e-01	-8.87139e+00	7.79619e-10
12	6.64029e-01	4.05006e-01	-8.87139e+00	8.88178e-15
13	6.64029e-01	4.05006e-01	-8.87139e+00	0.00000e+00

It is interesting to note that, contrary to the augmented Lagrangian method (Example 19.6), the SQP method is able to solve the problem with high precision.

20.2 Globally convergent algorithm

In order to render the SQP method globally convergent, we take inspiration from the descent methods of Chapter 11. In the context of unconstrained optimization, the aim was to identify a descent direction and calculate an appropriate step in this direction, in order for the new iterate to be "significantly better" than the last one. In this context, the concept of "significantly better" corresponds to a sufficient decrease of the objective function. In the context of constrained optimization, it is not so simple. Indeed, an iterate can be better than the previous one for two reasons: the value of the objective function is lower, or the iterate is closer to the feasible set. These two objectives are often conflicting, in the sense that we generally have to increase the

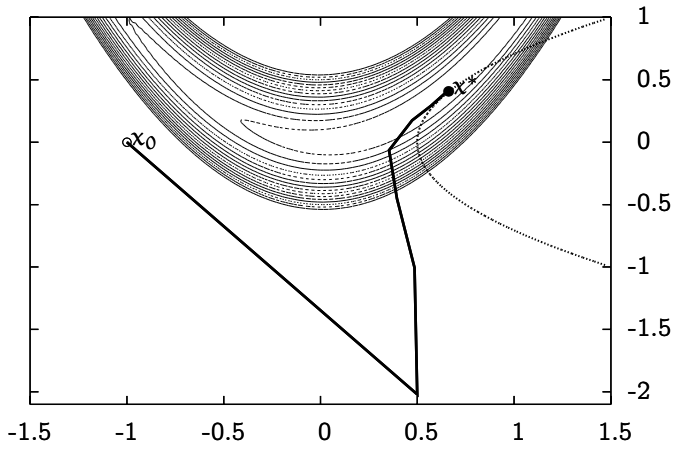


Figure 20.4: Iterations of the SQP method for Example 20.4

value of the objective function in order to satisfy the constraints. To identify whether an iterate is "significantly better," we have to combine the two aspects in a function called the *merit* function. This is similar to the idea developed in the context of the augmented Lagrangian algorithm. It is referred to as *exact* if the optimal solution to the constrained optimization problem (20.1)–(20.2) is a local minimum of the merit function.

Definition 20.5 (Exact merit function)**.** Consider the constrained optimization problem (1.71)-(1.74). A function $\phi : \mathbb{R}^n \to \mathbb{R}$ is an exact merit function of the problem if each local minimum x^* of the problem (1.71)–(1.74) is also a local minimum of the unconstrained function ϕ.

For the problem (20.1)-(20.2), the exact merit function that is used the most is

$$\phi_c(x) = f(x) + c\big\|h(x)\big\|_1 = f(x) + c\sum_{i=1}^{m}\big|h_i(x)\big|\,. \tag{20.13}$$

We demonstrate that this is an exact merit function, at least when c is sufficiently large.

Theorem 20.6 (Exact merit function ℓ_1). *Let $f : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ be twice differentiable and let us take the optimization problem (20.1)–(20.2), $\min_{x \in \mathbb{R}^n} f(x)$ subject to $h(x) = 0$. Let x^* and λ^* satisfy the sufficient optimality conditions (6.23)–(6.24). If*

$$c > \max_{i=1,\dots,m} \left|\lambda_i^*\right|, \tag{20.14}$$

the function (20.13) is an exact merit function for this problem.

Proof. Take $\varepsilon > 0$ such that $f(x^*) \leq f(x)$ for all x such that $h(x) = 0$ and $\|x - x^*\| \leq \varepsilon$. We define the following optimization problems:

Perturbed problem. Take $\delta \in \mathbb{R}^m$. The perturbed problem is

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

subject to

$$\begin{aligned} h(x) &= \delta \\ \|x - x^*\| &\leq \varepsilon, \end{aligned}$$

for which the optimal value is denoted by $p(\delta)$. According to the sensitivity theorem (Theorem 6.24), we have

$$\nabla p(0) = -\lambda^*. \tag{20.15}$$

Relaxed problem. Take $\delta \in \mathbb{R}^m$ and $c > 0$. The relaxed problem is

$$\min_{x \in \mathbb{R}^n} \phi_c(x) = f(x) + c \sum_{i=1}^{m} \left|h_i(x)\right|$$

subject to

$$\begin{aligned} h(x) &= \delta \\ \|x - x^*\| &\leq \varepsilon, \end{aligned}$$

for which the optimal value is denoted by $p_c(\delta)$. We can also write the objective function

$$\min_{x \in \mathbb{R}^n} f(x) + c \sum_{i=1}^{m} |\delta_i|.$$

As $c \sum_{i=1}^{m} |\delta_i|$ does not depend on x, the relaxed problem is equivalent to the perturbed problem, up to a shift of the objective function. Therefore,

$$p_c(\delta) = p(\delta) + c \sum_{i=1}^{m} |\delta_i|. \tag{20.16}$$

In particular, we have

$$p_c(0) = p(0). \tag{20.17}$$

Auxiliary problem. Take $c > 0$ and $\Delta(\varepsilon) = \{\delta \mid \exists x \text{ such that } h(x) = \delta \text{ and } \|x - x^*\| < \varepsilon\}$. The auxiliary problem is

$$\min_{\delta \in \mathbb{R}^m} p_c(\delta)$$

subject to $\delta \in \Delta(\varepsilon)$.

We first demonstrate that $\delta = 0$ is an optimal solution to the auxiliary problem. Using Taylor's theorem (Theorem C.2), we have for any δ feasible for the auxiliary problem

$$\begin{aligned} p_c(\delta) &= p(\delta) + c\sum_{i=1}^{m} |\delta_i| && \text{from (20.16)} \\ &= p(0) + \delta^T \nabla p(0) + \frac{1}{2}\delta^T \nabla^2 p(\bar{\alpha}\delta)\delta + c\sum_{i=1}^{m} |\delta_i| && \text{using (C.4)} \\ &= p(0) - \delta^T \lambda^* + \frac{1}{2}\delta^T \nabla^2 p(\bar{\alpha}\delta)\delta + c\sum_{i=1}^{m} |\delta_i| && \text{from (20.15)} \end{aligned}$$

where $0 \le \bar{\alpha} \le 1$. Take $\gamma > 0$ and $c \ge \max_{i=1,\ldots,m} |\lambda_i^*| + \gamma$. Then,

$$\begin{aligned} c\sum_{i=1}^{m} |\delta_i| &\ge \max |\lambda_i^*| \sum_i |\delta_i| + \gamma \sum_i |\delta_i| \\ &\ge \sum_i \delta_i \lambda_i^* + \gamma \sum_i |\delta_i| \\ &= \delta^T \lambda^* + \gamma \sum_i |\delta_i| . \end{aligned}$$

Then,

$$p_c(\delta) \ge p(0) - \delta^T \lambda^* + \frac{1}{2}\delta^T \nabla^2 p(\bar{\alpha}\delta)\delta + \delta^T \lambda^* + \gamma \sum_i |\delta_i|$$

or

$$p_c(\delta) \ge p(0) + \frac{1}{2}\delta^T \nabla^2 p(\bar{\alpha}\delta)\delta + \gamma \sum_i |\delta_i| .$$

We can make δ sufficiently close to 0, so that $\gamma \sum_i |\delta_i|$ dominates $\frac{1}{2}\delta^T \nabla^2 p(\bar{\alpha}\delta)\delta$, so that

$$\frac{1}{2}\delta^T \nabla^2 p(\bar{\alpha}\delta)\delta + \gamma \sum_i |\delta_i| > 0,$$

and

$$p_c(\delta) > p(0).$$

Using (20.17), we have

$$p_c(\delta) > p(0) = p_c(0), \tag{20.18}$$

and $\delta = 0$ is the optimal solution of the auxiliary problem.

We assume that $\delta \neq 0$, but sufficiently close to 0, and x such that $h(x) = \delta$ (then, x is infeasible for the initial problem) and satisfying $\|x - x^*\| < \varepsilon$. Since $p_c(\delta)$ is the optimal value of the relaxed problem, we have

$$p_c(\delta) \leq f(x) + c \sum_{i=1}^{m} |h_i(x)| .$$

According to (20.18), we have

$$\phi_c(x^*) = f(x^*) = p_c(0) < f(x) + c \sum_{i=1}^{m} |h_i(x)| = \phi_c(x)$$

and $\phi_c(x^*)$ is better that all the infeasible x in a neighborhood of x^*.

When x is feasible, i.e., $h(x) = 0$, then $\phi_c(x) = f(x)$. Since x^* is a local minimum of the initial problem, we have $\phi_c(x^*) = f(x^*) \leq f(x) = \phi_c(x)$ if $\|x - x^*\| < \varepsilon$ and x^* is indeed a local minimum of $\phi_c(x)$. □

It is interesting to graphically analyze this function. The level curves of the merit function for the problem in Example 20.2 are shown in Figure 20.5. When $c > |\lambda^*| = \sqrt{2}/2 \approx 0.707$, the minimum of the merit function (x_m in the graph) corresponds to the optimal solution of the initial problem

$$x^* = \begin{pmatrix} \sqrt{2}/2 & 1 + \sqrt{2}/2 \end{pmatrix}^T .$$

The level curves of the merit function for the problem in Example 20.3 are shown in Figure 20.6. When $c > |\lambda^*| = 1.5$, the minimum of the merit function corresponds to the optimal solution of the initial problem $x^* = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$.

In order to render Algorithm 20.1 globally convergent, we use the same ideas as in the unconstrained case, where the merit function plays the role of the objective function when the notion of "significantly better" is required. The line search methods (Chapter 11) based on the Wolfe conditions and the trust region methods (Chapter 12) can be used in this context. We give a detailed description of the algorithm based on line search.

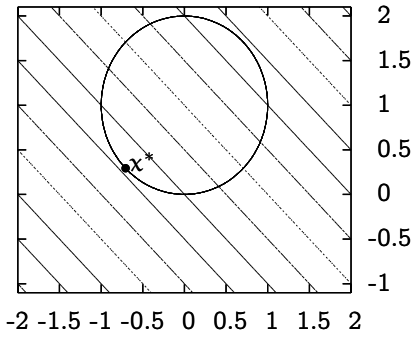
The Wolfe conditions (11.45) and (11.47) should here be translated as

$$\phi_c(x_k + \alpha_k d_k) \leq \phi_c(x_k) + \alpha_k \beta_1 \nabla \phi_c(x_k)^T d_k$$

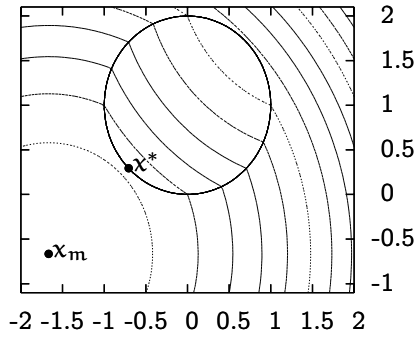
and

$$\nabla \phi_c(x_k + \alpha_k d_k)^T d_k \geq \beta_2 \nabla \phi_c(x_k)^T d_k$$

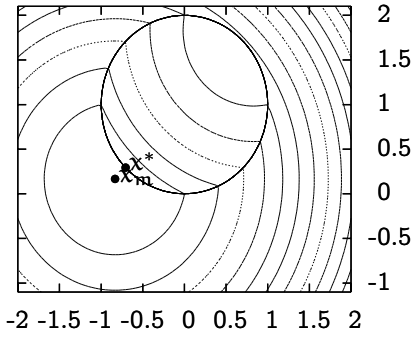
with $0 < \beta_1 < \beta_2 < 1$. Unfortunately, the merit function (20.13) is not differentiable, especially when x is feasible. It is not permitted to use $\nabla \phi_c(x_k)$, which does not exist everywhere. However, we do not need the gradient itself, but only the directional derivative. And it is important that the latter is negative, in order for d_k to be a descent direction for the merit function.



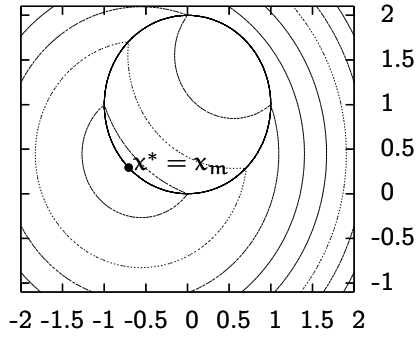
(a) $c = 0$



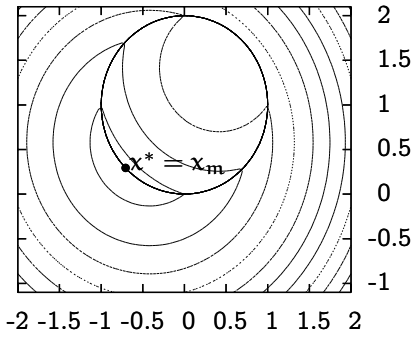
(b) $c = 0.3$



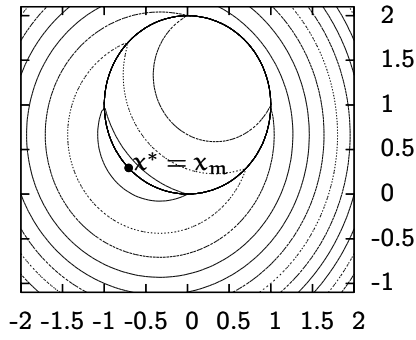
(c) $c = 0.6$



(d) $c = 0.9$

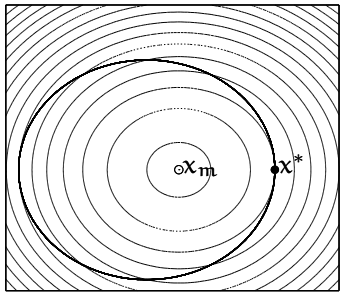


(e) $c = 1.2$

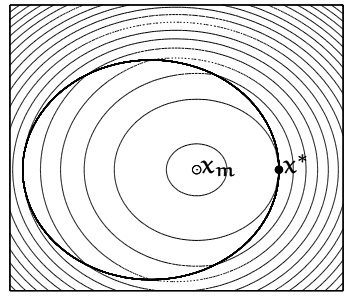


(f) $c = 1.5$

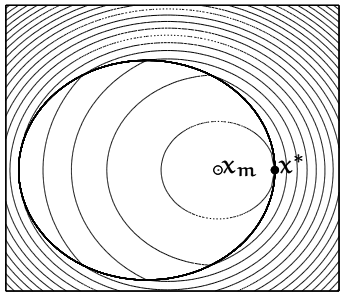
Figure 20.5: Merit function for Example 20.2



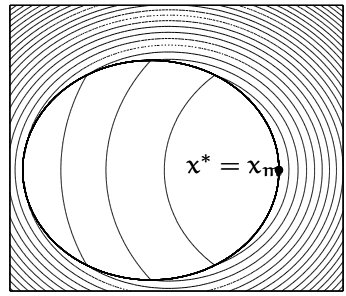
(a) $c = 0$



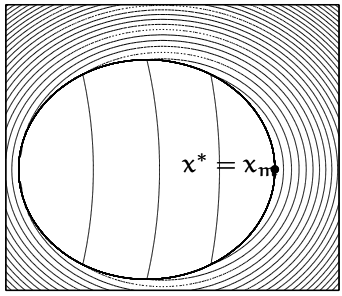
(b) $c = 0.6$



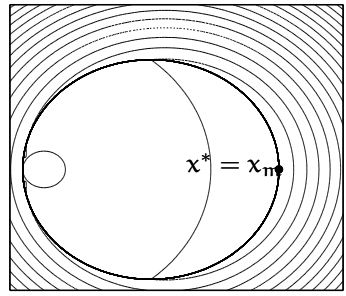
(c) $c = 1.1$



(d) $c = 1.6$



(e) $c = 2.1$



(f) $c = 2.6$

Figure 20.6: Merit function for Example 20.3

Theorem 20.7 (Directional derivative of the merit function). *Let* d_x *and* d_λ *satisfy the conditions (20.7) and (20.8). Then, the directional derivative of* ϕ_c *in the direction* d_x *is*

$$\phi_c'(x_k; d_x) = \nabla f(x_k)^T d_x - c\left\|h(x_k)\right\|_1 .$$

Proof. According to Taylor's theorem (Theorem C.2), there exist α_f and α_{h_i} such that

$$f(x_k + \alpha d_x) = f(x_k) + \alpha d_x^T \nabla f(x_k) + \frac{1}{2}\alpha^2 d_x^T \nabla^2 f(x_k + \alpha\alpha_f d) d_x$$

and, for all $i = 1, \ldots, m$,

$$h_i(x_k + \alpha d_x) = h_i(x_k) + \alpha d_x^T \nabla h_i(x_k) + \frac{1}{2}\alpha^2 d_x^T \nabla^2 h_i(x + \alpha\alpha_{h_i} d) d_x .$$

Let M be an upper bound on the eigenvalues of $\nabla^2 f(x_k + \alpha\alpha_f d)$ and $\nabla^2 h_i(x + \alpha\alpha_{h_i} d)$, $i = 1, \ldots, m$, such that

$$-M\|d_x\|^2 \le d_x^T \nabla^2 f(x_k + \alpha\alpha_f d) d_x \le M\|d_x\|^2 \tag{20.19}$$

and

$$-M\|d_x\|^2 \le d_x^T \nabla^2 h_i(x_k + \alpha\alpha_{h_i} d) d_x \le M\|d_x\|^2 . \tag{20.20}$$

Then,

$$\begin{aligned}
\phi_c(x_k + \alpha d_x) - \phi_c(x_k) &= f(x_k + \alpha d_x) + c\left\|h(x_k + \alpha d_k)\right\|_1 \\
&\quad - f(x_k) - c\left\|h(x_k)\right\|_1 \\
&\le f(x_k) + \alpha \nabla f(x_k)^T d_x + \frac{1}{2}\alpha^2 M\|d_x\|^2 \\
&\quad + c\left\|h(x_k) + \alpha \nabla h(x_k)^T d_x\right\|_1 + \frac{1}{2}\alpha^2 M\|d_x\|^2 \\
&\quad - f(x_k) - c\left\|h(x_k)\right\|_1 \\
&= \alpha \nabla f(x_k)^T d_x + \alpha^2 M\|d_x\|^2 \\
&\quad + c\left\|h(x_k) + \alpha \nabla h(x_k)^T d_x\right\|_1 - c\left\|h(x_k)\right\|_1 .
\end{aligned}$$

By using (20.8), we obtain

$$\begin{aligned}
\phi_c(x_k + \alpha d_x) - \phi_c(x_k) \le{}& \alpha \nabla f(x_k)^T d_x + \alpha^2 M\|d_x\|^2 \\
&+ c\left\|h(x_k) - \alpha h(x_k)\right\|_1 - c\left\|h(x_k)\right\|_1 .
\end{aligned}$$

If we assume that $\alpha < 1$, such that $1 - \alpha > 0$, we get

$$\phi_c(x_k + \alpha d_x) - \phi_c(x_k) \le \alpha\left(\nabla f(x_k)^T d_x - c\left\|h(x_k)\right\|_1\right) + M\alpha^2\|d_x\|^2 .$$

By using the lower bounds of (20.19) and (20.20) instead of the upper bounds, we similarly obtain

$$\phi_c(x_k + \alpha d_x) - \phi_c(x_k) \ge \alpha\left(\nabla f(x_k)^T d_x - c\left\|h(x_k)\right\|_1\right) - M\alpha^2\|d_x\|^2 .$$

Therefore,

$$\begin{aligned}\nabla f(x_k)^T d_x - c\|h(x_k)\|_1 - M\alpha\|d_x\|^2 &\leq \frac{\phi_c(x_k + \alpha d_x) - \phi_c(x_k)}{\alpha} \\ &\leq \nabla f(x_k)^T d_x - c\|h(x_k)\|_1 + M\alpha\|d_x\|^2 .\end{aligned}$$

When $\alpha \to 0$, we obtain

$$\phi_c'(x_k; d_x) = \nabla f(x_k)^T d_x - c\|h(x_k)\|_1 . \tag{20.21}$$

□

Therefore, if the parameter c is chosen such that

$$c > \frac{\nabla f(x_k)^T d_x}{\|h(x_k)\|_1},$$

the direction d_x is a descent direction for the merit function. Unfortunately, this condition may generate large values for c. We perform a detailed analysis of this directional derivative in order to find another value of c that ensures that d_x is a descent direction for ϕ_c.

Theorem 20.8 (Descent direction for the merit function). *Let d_x and d_λ satisfy the conditions (20.7) and (20.8). Then, the directional derivative of ϕ_c in the direction d_x, denoted by $\phi_c'(x_k; d_x)$ is such that*

$$\phi_c'(x_k; d_x) \leq -d_x^T \nabla^2 L(x_k, \lambda_k) d_x - \left(c - \|d_\lambda\|_\infty\right) \|h(x_k)\|_1 .$$

Proof. According to Theorem 20.7, we have

$$\phi_c'(x_k; d_x) = \nabla f(x_k)^T d_x - c\|h(x_k)\|_1 .$$

By using (20.7), we obtain

$$\nabla f(x_k)^T d_x = -d_x^T \nabla_{xx}^2 L(x_k, \lambda_k) d_x - d_x^T \nabla h(x_k) d_\lambda .$$

According to (20.8), this gives

$$\nabla f(x_k)^T d_x = -d_x^T \nabla_{xx}^2 L(x_k, \lambda_k) d_x + h(x_k)^T d_\lambda$$

and

$$\phi_c'(x_k; d_x) = -d_x^T \nabla_{xx}^2 L(x_k, \lambda_k) d_x + h(x_k)^T d_\lambda - c\|h(x_k)\|_1 .$$

Applying the Cauchy-Schwartz inequality (C.20):

$$h(x_k)^T d_\lambda \leq \|h(x_k)\|_1 \|d_\lambda\|_\infty$$

produces the result. □

Algorithm 20.2: Globalized SQP algorithm

Objective

To find a local minimum of the problem (1.71)–(1.72), $\min_{x\in\mathbb{R}^n} f(x)$ subject to $h(x) = 0$.

Input

$f : \mathbb{R}^n \to \mathbb{R}$, $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$, $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$.

$h : \mathbb{R}^n \to \mathbb{R}^m$, $\nabla h : \mathbb{R}^n \to \mathbb{R}^{n\times m}$, $\nabla^2 h_i : \mathbb{R}^n \to \mathbb{R}^{n\times n}$, $i = 1, \ldots, m$.

A parameter $0 < \beta_1 < 1$ (by default: $\beta_1 = 0.3$).

A parameter $\bar{c} > 0$ (by default: $\bar{c} = 0.1$).

An initial solution (x_0, λ_0).

The required precision $\varepsilon > 0$.

Output

An approximation of the optimal solution (x^*, λ^*).

Initialization

$k := 0$.

$c_0 := \|\lambda_0\|_\infty + \bar{c}$.

Repeat

Calculate $\nabla^2_{xx} L(x_k, \lambda_k) = \nabla^2 f(x_k) + \sum_{i=1}^{m} (\lambda_k)_i \nabla^2 h_i(x_k)$.

Find an positive definite approximation H_k of $\nabla^2_{xx} L(x_k, \lambda_k)$ (e.g. using Algorithm 11.7).

Obtain d_x and d_λ by solving the quadratic problem $\min_d \nabla f(x_k)^T d + \frac{1}{2} d^T H_k d$ subject to $\nabla h(x_k)^T d + h(x_k) = 0$ with an appropriate algorithm (to illustrate the method, we may use (20.11) and (20.12)).

$c^+ := \|d_\lambda\|_\infty + \bar{c}$.

Update the penalty parameter

If $c_k \geq 1.1\, c^+$, $c_{k+1} := \frac{1}{2}(c_k + c^+)$.

If $c^+ \leq c_k < 1.1\, c^+$, $c_{k+1} := c_k$.

If $c_k < c^+$, $c_{k+1} := \max(1.5\, c_k, c^+)$.

$\phi'_{c_k}(x_k; d_x) := \nabla f(x_k)^T d_x - c_k \|h(x_k)\|_1$.

Calculate the step

$i = 0$, $\alpha_i = 1$.

while $\phi_{c_k}(x_k + \alpha_i d_k) > \phi_{c_k}(x_k) + \alpha_i \beta_1 \phi'_{c_k}(x_k; d_x)$ **do**

$\alpha_{i+1} := \alpha_i / 2$

$i := i + 1$

$\alpha := \alpha_i$.

$x_{k+1} := x_k + \alpha d_x$.

$\lambda_{k+1} := d_\lambda$.

$k := k + 1$.

Until $\|\nabla L(x_k, \lambda_k)\| \leq \varepsilon$.

Then, d_x is a descent direction for the merit function if

$$c > \|d_\lambda\|_\infty - \frac{d_x^T \nabla^2_{xx} L(x_k, \lambda_k) d_x}{\|h(x_k)\|_1} .$$

If the matrix $\nabla^2_{xx} L(x_k, \lambda_k)$ is positive definite, we need only

$$c > \|d_\lambda\|_\infty . \tag{20.22}$$

This condition is consistent with (20.14).

In practice, the matrix $\nabla^2_{xx} L(x_k, \lambda_k)$ is not always positive definite. Therefore, as presented in Chapter 11, we replace this matrix with a positive definite approximation, using for instance a modified Cholesky factorization (Algorithm 11.7).

In practice, the choice of c is delicate. We here adopt the procedure presented by Bonnans et al. (1997), where the parameter c_k is updated at each iteration. Take $c^+ = \|d_\lambda\|_\infty + \bar{c}$, where $\bar{c}$ is a positive constant. The value of the parameter is chosen in line with (20.22).

- If $c_{k-1} \geq 1.1\, c^+$, then the parameter is too large, and is reduced to the average value between c_{k-1} and c^+, i.e.,

$$c_k = \frac{1}{2} (c_{k-1} + c^+) .$$

- If $1.1\, c^+ \geq c_{k-1} \geq c^+$, then the value of the parameter is good, and we leave it as it is, i.e.,

$$c_k = c_{k-1} .$$

- In the other cases, the parameter has to be increased. In order to significantly increase it, we impose a minimum augmentation of 50 %, i.e.,

$$c_k = \max(1.5\, c_{k-1}, c^+) .$$

The algorithm is described as Algorithm 20.2.

Comments

- The second Wolfe condition has not been used here, first of all in order to simplify the description of the algorithm and secondly because it is not necessary in practice. Moreover, it requires calculations of the directional derivative for each candidate.
- The matrix H_k can also be constructed by using the update formulas defined in Chapter 13. If the BFGS method is used, it is important to note that the condition $d^T y > 0$ (Theorem 13.2) is not automatically satisfied in this context.

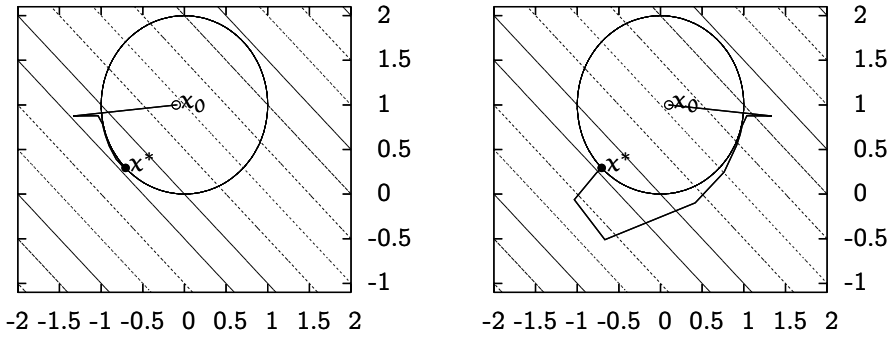
We apply this algorithm to Example 20.2. We keep in mind that the local SQP algorithm does not always converge toward a local minimum, as illustrated in Figure 20.2. The globalized algorithm, on the other hand, converges toward a local minimum for the two starting points (Figure 20.7). Tables 20.7 and 20.8 provide a detailed list of the iterations, where the parameter τ indicates the multiple of the identity that had to be added to the matrix $\nabla^2_{xx} L$ for it to be positive definite. It is interesting to note that, during the last iterations, $\tau = 0$ and $\alpha = 1$ and these iterations are thus equivalent to those of the local SQP method.

Table 20.7: Illustration of the globalized SQP algorithm for Example 20.2; $x_0 = \begin{pmatrix} -0.1 & 1 \end{pmatrix}^T$

k	x_1	x_2	λ	c	α	τ	$\|\nabla L_{xx}\|$
0	-1.00000e-01	1.00000e+00	1.00000e+00	4.5e+01	2.5e-01	0.0e+00	1.61867e+00
1	-1.33750e+00	8.75000e-01	-4.45000e+01	2.5e+01	1.0e+00	1.3e+02	1.20651e+02
2	-1.03707e+00	8.78487e-01	4.51420e+00	1.3e+01	1.0e+00	0.0e+00	8.36410e+00
3	-9.82831e-01	7.87058e-01	7.18209e-01	6.7e+00	1.2e-01	0.0e+00	8.07145e-01
4	-9.68037e-01	7.22096e-01	5.95218e-01	3.7e+00	1.2e-01	0.0e+00	6.86454e-01
5	-9.47328e-01	6.53182e-01	6.18375e-01	2.2e+00	2.5e-01	0.0e+00	5.96563e-01
6	-9.03900e-01	5.40943e-01	6.41192e-01	1.5e+00	5.0e-01	0.0e+00	4.41901e-01
7	-8.20325e-01	3.91504e-01	6.71728e-01	1.1e+00	1.0e+00	0.0e+00	2.13531e-01
8	-7.11368e-01	2.80115e-01	6.98734e-01	9.8e-01	1.0e+00	0.0e+00	2.56963e-02
9	-7.07221e-01	2.92880e-01	7.06945e-01	8.9e-01	1.0e+00	0.0e+00	2.84644e-04
10	-7.07107e-01	2.92893e-01	7.07107e-01	8.5e-01	1.0e+00	0.0e+00	3.93262e-08
11	-7.07107e-01	2.92893e-01	7.07107e-01	8.5e-01	1.0e+00	0.0e+00	1.42178e-15

Table 20.8: Illustration of the globalized SQP algorithm for Example 20.2; $x_0 = \begin{pmatrix} 0.1 & 1 \end{pmatrix}^T$

k	x_1	x_2	λ	c	α	τ	$\|\nabla L_{xx}\|$
0	1.00000e-01	1.00000e+00	1.00000e+00	5.5e+01	2.5e-01	0.0e+00	1.84935e+00
1	1.33750e+00	8.75000e-01	-5.45000e+01	3.0e+01	1.0e+00	1.5e+02	1.45526e+02
2	1.03710e+00	8.78856e-01	4.69637e+00	1.5e+01	1.0e+00	0.0e+00	1.07425e+01
3	9.80472e-01	7.66569e-01	-2.25676e-01	7.7e+00	1.6e-02	6.4e-01	1.23808e+00
4	9.57039e-01	6.68675e-01	-3.66978e-01	4.1e+00	3.1e-02	1.0e+00	1.27856e+00
5	9.13886e-01	5.45237e-01	-3.03114e-01	2.2e+00	6.2e-02	8.6e-01	1.35205e+00
6	7.64062e-01	2.47039e-01	-2.17779e-01	1.1e+00	6.2e-02	6.2e-01	1.49377e+00
7	4.17063e-01	-9.88200e-02	1.08475e-03	7.5e-01	2.0e-03	0.0e+00	1.46372e+00
8	-6.68621e-01	-5.10558e-01	2.46922e-01	6.6e-01	5.0e-01	0.0e+00	1.87138e+00
9	-1.03459e+00	-6.24439e-02	4.77505e-01	9.9e-01	1.0e+00	0.0e+00	1.19931e+00
10	-7.66609e-01	2.40944e-01	6.06968e-01	9.0e-01	1.0e+00	0.0e+00	1.94510e-01
11	-7.08587e-01	2.90278e-01	6.98162e-01	8.5e-01	1.0e+00	0.0e+00	1.50533e-02
12	-7.07117e-01	2.92897e-01	7.07078e-01	8.5e-01	1.0e+00	0.0e+00	5.43054e-05
13	-7.07107e-01	2.92893e-01	7.07107e-01	8.5e-01	1.0e+00	0.0e+00	6.60568e-10
14	-7.07107e-01	2.92893e-01	7.07107e-01	8.5e-01	1.0e+00	0.0e+00	4.96507e-16



(a) $x_0 = \begin{pmatrix} -0.1 & 1 \end{pmatrix}^T$ (b) $x_0 = \begin{pmatrix} 0.1 & 1 \end{pmatrix}^T$

Figure 20.7: Illustrations of the globalized SQP algorithm for Example 20.2

20.3 Project

The general organization of the projects is described in Appendix D.

Objective

The aim of this project is to implement the SQP algorithm, to test it on several problems, and to compare the local version with its globalized counterpart.

Approach

1. Solve the problems with Algorithm 20.2. Let x^* be a local optimum.
2. Randomly generate several starting points in a central ball x^* of radius ε with, for instance, $\varepsilon = 1$, 10, 100, 1,000.
3. For each value of ε, perform statistics on the number of times that the local algorithm converges.
4. For each starting point from which the local algorithm converges, compare the number of iterations for the two algorithms. What is the impact of the globalization on the efficiency of the method?

Algorithms

Algorithms 20.1 and 20.2.

Problems

Exercise 20.1. The problem

$$\min e^{x_1 - 2x_2}$$

subject to

$$\begin{aligned} \sin(-x_1 + x_2 - 1) &= 0 \\ -2 \le x_1 &\le 2 \\ -1.5 \le x_2 &\le 1.5\,. \end{aligned}$$

Exercise 20.2. The problem

$$\min_{x \in \mathbb{R}^{10}} \sum_{i=1}^{10} e^{x_i} \left(c_i + x_i - \ln \left(\sum_{k=1}^{10} e^{x_k} \right) \right)$$

subject to

$$\begin{aligned} e^{x_1} + 2e^{x_2} + 2e^{x_3} + e^{x_6} + e^{x_{10}} &= 2 \\ e^{x_4} + 2e^{x_5} + e^{x_6} + e^{x_7} &= 1 \\ e^{x_3} + e^{x_7} + e^{x_8} + 2e^{x_9} + e^{x_{10}} &= 1 \\ -100 \le x_i \le 100\,, &\qquad i = 1, \dots, 10\,, \end{aligned}$$

with

$$\begin{aligned} c_1 &= -6.089 & c_6 &= -14.986 \\ c_2 &= -17.164 & c_7 &= -24.1 \\ c_3 &= -34.054 & c_8 &= -10.708 \\ c_4 &= -5.914 & c_9 &= -26.662 \\ c_5 &= -24.721 & c_{10} &= -22.179\,. \end{aligned}$$

Solution:

$$\begin{aligned} x_1^* &= -3.40629 & x_6^* &= -4.44283 \\ x_2^* &= -0.596369 & x_7^* &= -1.41264 \\ x_3^* &= -1.1912 & x_8^* &= -21.6066 \\ x_4^* &= -4.62689 & x_9^* &= -2.26867 \\ x_5^* &= -1.0011 & x_{10}^* &= -1.40346\,. \end{aligned}$$

Proposed by Hock and Schittkowski (1981).

Exercise 20.3. The problem

$$\min_{x \in \mathbb{R}^2} \ln(1 + x_1^2) - x_2$$

subject to

$$\begin{aligned} \left(1 + x_1^2\right)^2 + x_2^2 &= 4 \\ -4 \le x_1 &\le 4 \\ -4 \le x_2 &\le 4\,. \end{aligned}$$

Please note: do not forget to transform the formulation of the problems so that they are compatible with (20.1)–(20.2).

Part VI

Networks

The richest people in the world look for and build networks, everyone else just looks for work.

Robert Kiyosaki

Our daily life is full of networks. We drive on a network of roads and highways. We receive water and electricity at home through the corresponding supply networks. Our houses are connected to a network of sewers for the evacuation of waste water. Our computers communicate over the internet, and our wireless phones are connected through a network of antennas. We exchange messages and pictures with our friends on social networks. We participate in professional meetings for the sake of "networking." Our brain is generating ideas and emotions from a network of neurons. In this book, we define a network as a mathematical object, with interesting properties that are exploited to solve optimization problems. The analogy with "real" networks allows us to develop intuitions about these properties. But the mathematical abstraction is also useful for applications that have nothing to do with networks in real life.

Chapter 21

Introduction and definitions

Contents

Generally speaking, a network is a system of interconnected people or things. The main feature of most networks is that the global complexity of the network is high, while the local complexity is low. For instance, you need a simple connection to the nearest WiFi antenna to connect your smartphone to the entire World Wide Web. When your distribution network operator has connected your new house to the grid, you can consume electricity that may have been produced in a different country. If you create a funny video and post it on YouTube, you can create a "buzz" and reach potentially hundreds of thousands of people around the world.

The mathematical object called "network" shares the same feature: it is able to capture complex structures using simple elements. The analogy between the mathematical object and real networks is useful to develop intuitions. However, a consequence is that the vocabulary and definitions used in the literature vary slightly from one reference to the next. In order to avoid any ambiguity, we provide here the definitions of the concepts that are used in this part of the book.

21.1 Graphs

The element defining the structure of a network is called a *graph*, composed of *vertices*, or *nodes* and *edges*, or *arcs*. The vertices are the entities that are interconnected, and the edges represent the connections. For instance, in the water supply network, the vertices are the houses of the customers, the tanks where the water is stored, the water treatment plants, the pumping stations, and so on. The edges are the physical pipes connecting these entities. On Facebook, the vertices are the registered individuals, and an edge represents a friendship connection between two persons. The relationship between the edges and the vertices that they connect is captured by a function called the *incidence function*. A graph is defined by a set of vertices, a set of edges, and an incidence function.

Definition 21.1 (Graph)**.** A graph is a triple $(\mathcal{V}, \mathcal{E}, \phi)$, where $\mathcal{V}$ is a finite set of elements called *vertices*, $\mathcal{E}$ is a finite set of elements called *edges*, and $\phi : \mathcal{E} \to \mathcal{P}_2(\mathcal{V})$ is the *incidence function*, mapping the set of edges into the set $\mathcal{P}_2(\mathcal{V})$ of all 2-element subsets of vertices.

It is common to represent a graph using a picture where vertices are represented by dots or circles, and edges by lines or arcs connecting the vertices. Consider the graph with

$$\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\},$$
$$\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$$

and the incidence function defined by

$$\begin{aligned} \phi(e_1) &= \{v_1, v_2\}, \quad \phi(e_2) = \{v_2, v_3\}, \\ \phi(e_3) &= \{v_1, v_3\}, \quad \phi(e_4) = \{v_3, v_5\}, \\ \phi(e_5) &= \{v_2, v_4\}, \quad \phi(e_6) = \{v_2, v_4\}, \\ \phi(e_7) &= \{v_4, v_5\}, \quad \phi(e_8) = \{v_4, v_5\}. \end{aligned}$$

The graph is illustrated in Figure 21.1.

A graph defined by a subset of vertices and edges of another graph is called a *subgraph*.

Definition 21.2 (Subgraph)**.** The graph $(\mathcal{V}', \mathcal{E}', \phi')$ is a subgraph of $(\mathcal{V}, \mathcal{E}, \phi)$ if

- $\mathcal{V}' \subseteq \mathcal{V}$,
- $\mathcal{E}' \subseteq \mathcal{E}$,
- $\phi'(e) = \phi(e)$, for each $e \in \mathcal{E}'$,
- for each $e \in \mathcal{E}'$, if $\phi'(e) = \{i, j\}$, then i and j both belong to $\mathcal{V}'$.

In the definition 21.1 of a graph, an edge connects two vertices, and their order is not specified. The names "arc" and "node," used instead of "edge" and "vertex," imply

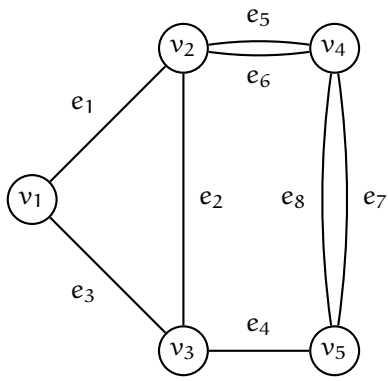


Figure 21.1: An example of a graph

that the underlying connection is directed, meaning that the arc connecting node i to node j is different from the arc that connects node j to node i (if there is one). The graph is then said to be *directed*. In this book, we consider mainly directed graphs, with the exception of Section 21.7 on minimum spanning trees, where undirected graphs are considered.

Definition 21.3 (Directed graph)**.** A directed graph is a triple $(\mathcal{N}, \mathcal{A}, \phi)$, where $\mathcal{N}$ is a finite set of elements called *nodes*, $\mathcal{A}$ is a finite set of elements called *arcs*, and $\phi : \mathcal{A} \to \mathcal{N} \times \mathcal{N}$ is the *incidence function*, mapping the set of arcs into the set of pairs of nodes.

Note that this definition potentially allows several arcs to connect the same pair of nodes. In this book, we focus on networks such that the incidence function is injective.[1] It means that if we select two distinct arcs $a_1, a_2 \in \mathcal{A}$, $a_1 \neq a_2$, then the pairs of nodes incident to these two arcs must be different, that is $\phi(a_1) \neq \phi(a_2)$. In other words, when the incidence function is injective, each ordered pair of nodes is connected by either 0 or 1 arc. In this case, we can use the notation (i, j) without ambiguity to identify the arc representing the connection between node i and node j. When we mention the arc (i, j), we refer to the arc a such that $\phi(a) = (i, j)$. As ϕ is injective, if such an arc exists, it is unique. We say that i is the *upstream node* of arc (i, j), and j its *downstream node*. It is said that the arc (i, j) is *incident* to node i and to node j, irrespectively of the orientation of the arc. The *degree* d_i of node i is the number of incident arcs. The *indegree* d_i^- of node i is the number of arcs incident to i such that i is their downstream node. The *outdegree* d_i^+ of node i is the number of arcs incident to i such that i is their upstream node. For each node i, we have $d_i = d_i^- + d_i^+$. Also, if the arc (i, j) exists, nodes i and j are said

[1] The incidence function of the graph represented in Figure 21.1 is not injective, as there are two edges connecting the vertices v_2 and v_4, as well as the vertices v_3 and v_5.

to be *adjacent* to each other, irrespectively of the orientation of the arc. Figure 21.2 represents a directed graph with 8 nodes and 10 arcs. The indegree, outdegree, and degree of each node are:

$$\begin{array}{cccccccc} d_1^-=1, & d_2^-=2, & d_3^-=1, & d_4^-=2, & d_5^-=2, & d_6^-=1, & d_7^-=1, & d_8^-=0, \\ d_1^+=1, & d_2^+=2, & d_3^+=2, & d_4^+=2, & d_5^+=1, & d_6^+=1, & d_7^+=1, & d_8^+=0, \\ d_1=2, & d_2=4, & d_3=3, & d_4=4, & d_5=3, & d_6=2, & d_7=2, & d_8=0. \end{array}$$

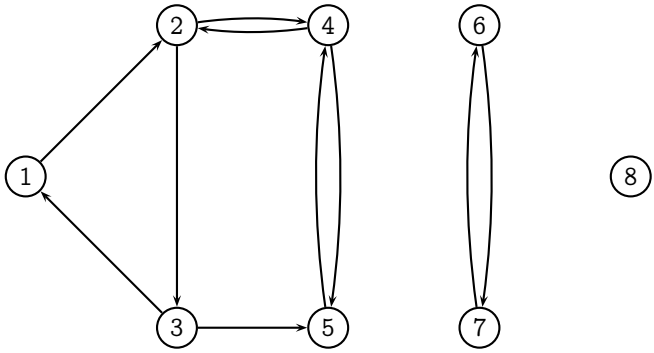


Figure 21.2: An example of a directed graph

21.2 Cuts

Just as cities can be separated into two banks by a river, it may be convenient to separate a directed graph into two sets of nodes. Such a separation is called a *cut*.

Definition 21.4 (Cut)**.** Consider a directed graph where $\mathcal{N}$ is the set of nodes. A cut Γ is an ordered partition of the nodes into two non empty subsets:

$$\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M}), \tag{21.1}$$

where $\mathcal{M} \subset \mathcal{N}$ is a subset of all the nodes of the graph.

We say that the cut Γ separates i from j if $i \in \mathcal{M}$ and $j \notin \mathcal{M}$. Using the analogy of a city divided by a river, $\mathcal{M}$ can be considered as the left bank and $\mathcal{N} \setminus \mathcal{M}$ as the right bank of the river. The arcs with their upstream node in the left bank, and their downstream node in the right bank may represent bridges on the river. The bridges connecting the left bank to the right bank constitute the set of *forward arcs* of the cut:

$$\Gamma^{\rightarrow} = \{(i,j) \in \mathcal{A} | i \in \mathcal{M}, j \notin \mathcal{M}\}. \tag{21.2}$$

Similarly, the bridges connecting the right bank to the left form the set of *backward arcs* of the cut:

$$\Gamma^{\leftarrow} = \{(i,j) \in \mathcal{A} | i \notin \mathcal{M}, j \in \mathcal{M}\}. \tag{21.3}$$

Note that one or both of these sets may happen to be empty. When convenient to do so, we say that $(i,j) \in \Gamma$ if $(i,j) \in \Gamma^{\rightarrow} \cup \Gamma^{\leftarrow}$.

Note from these definitions that the cut based on the partition $(\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$ is different from the cut based on the partition $(\mathcal{N} \setminus \mathcal{M}, \mathcal{M})$. This is what is meant by *ordered partition* in Definition 21.4. Using the bridge analogy again, it means that we explicitly distinguish the left bank from the right bank of the river.

To illustrate the concept, Figure 21.3 represents the cut based on the subset of nodes $\mathcal{M} = \{1,2,3,5,7\}$, and the cut is

$$\Gamma = (\{1,2,3,5,7\}, \{4,6,8\}).$$

The forward arcs of the cut are

$$\Gamma^{\rightarrow} = \{(2,4),(5,4),(7,6)\}.$$

The backward arcs of the cut are

$$\Gamma^{\leftarrow} = \{(4,2),(4,5),(6,7)\}.$$

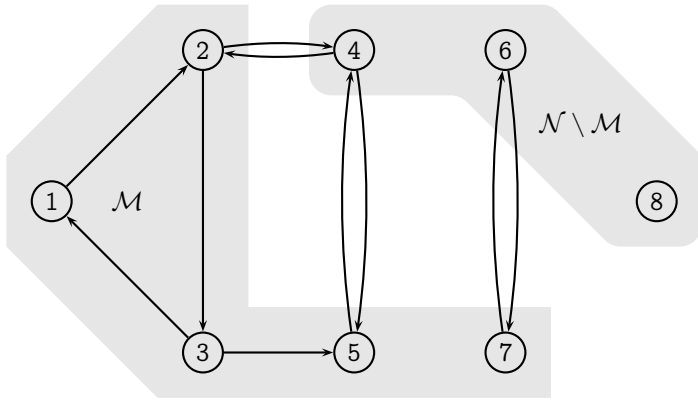


Figure 21.3: Example of a cut

21.3 Paths

A *path* in a graph $(\mathcal{V}, \mathcal{E}, \phi)$ is a finite sequence of edges $e_1, \ldots, e_{p-1}$ for which there is a sequence $v_1, \ldots, v_p$ of vertices such that $\phi(e_k) = \{v_k, v_{k+1}\}$, for $k = 1, \ldots, p-1$. Similarly, a *path* in a directed graph $(\mathcal{N}, \mathcal{A}, \phi)$ is a sequence of arcs $a_1, \ldots, a_{p-1}$ for which there is a sequence $i_1, \ldots, i_p$ of nodes such that $\phi(a_k) = (i_k, i_{k+1})$ or $\phi(a_k) = (i_{k+1}, i_k)$, for $k = 1, \ldots, p-1$.

In the context of a directed graph with an injective incidence function, we denote a path by the sequence of nodes, each pair of consecutive nodes being directed either forward ($\rightarrow$) or backward ($\leftarrow$). If a pair $i \rightarrow j$ is in the path, it means that the two nodes are connected by the arc (i,j) in the forward direction. If the pair $i \leftarrow j$ is in the path, it means that the two nodes are connected by arc (j,i) in the backward direction. To be a valid path, each arc in the path must belong to $\mathcal{A}$. The first node of a path P is called its *origin*. The last node is called its *destination*. A path such that its origin coincides with its destination is called a *cycle*. A path is *simple* if it contains no repeated nodes. A cycle is *simple* if it contains no repeated nodes, with the exception of the origin and the destination that coincide. We denote $P^{\rightarrow}$ the set of forward arcs of path P and $P^{\leftarrow}$ the set of backward arcs. A path is a *forward path* if its set of backward arcs is empty. Here are some examples of paths in the directed graph represented in Figure 21.2:

- $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ is a simple forward path from node 1 to node 5 containing only forward arcs $[P^{\rightarrow} = (1,2),(2,4),(4,5),\ P^{\leftarrow} = \emptyset]$;
- $1 \rightarrow 2 \rightarrow 4 \rightarrow 2 \rightarrow 3$ is a forward path from node 1 to node 3 (note that node 2 is repeated, so that it is not a simple path) $[P^{\rightarrow} = (1,2),\ (2,4),\ (4,2),\ (2,3),\ P^{\leftarrow} = \emptyset]$;
- $1 \rightarrow 2 \leftarrow 4 \rightarrow 5$ is a simple path from node 1 to node 5 that uses arc $(4,2)$ in the reverse direction $[P^{\rightarrow} = (1,2),(4,5),\ P^{\leftarrow} = (4,2)\ ]$;
- $1 \rightarrow 2 \leftarrow 4 \rightarrow 5 \leftarrow 3 \rightarrow 1$ is a simple cycle $[P^{\rightarrow} = (1,2),(4,5),(3,1),\ P^{\leftarrow} = (4,2),(3,5)]$;
- $4 \rightarrow 6 \rightarrow 7$ is an invalid path as arc $(4,6)$ does not exist.

Lemma 21.5. *[Longest simple path] Consider a directed graph with m nodes. The number of arcs in any simple path is no more than $m-1$.*

Proof. A simple path that visits all the nodes of the graph has $m-1$ arcs. If this path is extended by one more arc, the downstream node of this arc would be visited twice by the path, which would not be simple. □

Lemma 21.6. *[Finite number of simple paths] Consider a directed graph with m nodes, $m \geq 2$, and two nodes o and d. There is a finite number of simple paths between o and d.*

Proof. Consider k such that $2 \leq k \leq m$. The number of simple paths containing k nodes is finite. Indeed, the number of simple paths is bounded above by the number of permutations of $k-2$ nodes (some of these permutations do not correspond to a valid path). As the value of k is bounded above by m, the total number of simple paths is bounded above. □

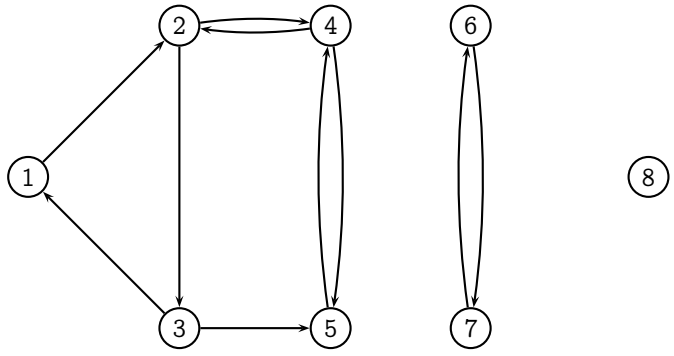
When every pair of nodes in the directed graph is connected with a path, the graph is said to be *connected*. If every pair of nodes is connected with a path containing

only forward arcs, the graph is said to be *strongly connected*. A graph containing a single node and no arc is considered to be strongly connected, too. The graph represented in Figure 21.2 is not connected, as there are several pairs of nodes with no path connecting them, such as nodes 1 and 8, for instance. Actually, with respect to connectivity, this graph appears to have three connected subgraphs, defined by the sets of nodes: $\{1, 2, 3, 4, 5\}$, $\{6, 7\}$, and $\{8\}$, and the arcs they are incident to. These three connected subgraphs are called *connected components*. The definition of connected components is based on equivalence classes on the set of nodes. Indeed, the relation "is connected with" defines an equivalence relation on the set of nodes, as it is reflexive, symmetric, and transitive (see Definition B.24). Note that the relation "is strongly connected with" does not define an equivalence relation, as it is not symmetric.

Definition 21.7 (Connected component)**.** The subgraph $G' = (\mathcal{N}', \mathcal{A}', \phi')$ of the graph $G = (\mathcal{N}, \mathcal{A}, \phi)$ is a connected component of G if

- $\mathcal{N}'$ is an equivalence class on $\mathcal{N}$ for the relation "is connected with,"
- for each $(i, j) \in \mathcal{A}$, if $i \in \mathcal{N}'$ then $(i, j) \in \mathcal{A}'$, and
- $\phi'(a) = \phi(a)$, for each $a \in \mathcal{A}'$.

In the second part of the definition, the existence of the arc (i, j) implies that i and j belong to the same equivalence class. Consequently, if $i \in \mathcal{N}'$ then j belongs to $\mathcal{N}'$ too.



(a) First (b) Second (c) Third

Figure 21.4: Connected components of the graph represented in Figure 21.2

21.4 Trees

There is a family of graphs called *trees* that are particularly useful both for the algorithms that we describe in this book, and in various other applications.

Definition 21.8 (Tree)**.** A tree is a connected graph without cycles.

Figure 21.5 represents an example of a tree. A *leaf* is a node of degree 1 in a tree.

Lemma 21.9. *Every tree with at least one arc has at least two leaves.*

Proof. Consider the path $\mathcal{P}$ of the tree with the largest number of arcs. Call its origin o and its destination d. By construction, the degree of o and d is larger or equal to 1. If the degree of o is strictly larger than 1, it means that there is another arc, not in the path, and incident to o. Therefore, a path with one more arc than $\mathcal{P}$ exists, which is not possible by the definition of $\mathcal{P}$. Therefore, o is a leaf. The same argument is used to determine that d is a leaf, too. □

A tree can be characterized in several different ways.

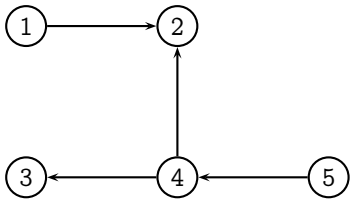


Figure 21.5: Example of a tree

Theorem 21.10 (Characterization of a tree)**.** *Let* $G = (\mathcal{N}, \mathcal{A}, \phi)$ *be a directed graph with* m *nodes and* n *arcs. The following statements are all equivalent.*

1. G *is a tree;*
2. G *is connected and without cycles;*
3. *there is a unique simple path connecting any two nodes;*
4. G *has no cycles, and a simple cycle is formed if any arc is added;*
5. G *is connected and the removal of any single arc disconnects the graph;*
6. G *is connected and* $n = m - 1$*;*
7. G *has no simple cycles and* $n = m - 1$.

Proof. **1** $\iff$ **2** By Definition 21.8.

2 $\implies$ **3** Consider nodes i and j. As the graph is connected, there is at least one path connecting i to j. Suppose by contradiction that there are two distinct paths connecting i to j. Together, these two paths form a cycle, contradicting the assumption. Therefore, there is exactly one path between i and j. As the graph does not contain any cycle, the path is simple.

3 $\implies$ **4** Assume that the graph contains a cycle involving nodes i and j. Then there are two paths connecting i and j, which contradicts the assumption, and proves that the graph has no cycle. Consider now any two nodes i and j that are not connected by an arc. By assumption, there is a simple path connecting i and j. Therefore, if the arc (i, j) is added, it closes the path and forms a simple cycle.

3 $\implies$ **5** Consider any arc (i, j). It is the only path connecting i to j. Therefore, if the arc is removed, node i is disconnected from j.

5 $\implies$ **2** Assume by contradiction that the graph contains a cycle. Removing an arc from the cycle does not disconnect the graph, contradicting this assumption, and proving the result.

4 $\implies$ **3** Consider two nodes i and j in the graph. If the arc (i, j) exists, there is one path connecting i and j. As there are no cycles, it is the only one. If arc (i, j) does not exist, add it to the graph. By assumption, it forms a simple cycle. Therefore, the path obtained by removing arc (i, j) from the cycle is a simple path between i and j. As the original graph has no cycle, it is unique.

4 $\implies$ **2** We need to show that the graph is connected. But as condition 4 implies condition 3 (see above), any pair of node is connected.

1 $\implies$ **6 and 1** $\implies$ **7** We show that $n = m - 1$ by induction. If there is only one arc in the tree, that is $n = 1$, Lemma 21.9 states that there are at least two nodes. If there were 3 nodes or more, one of them would be disconnected, as there is only one arc. As this is not possible in a tree, the tree has exactly 2 nodes. Suppose the property to be true for a tree with m nodes, and consider a tree with $m' = m + 1$ nodes. We must show that this tree has $n' = m' - 1 = m$ arcs. Consider one leaf of this tree (it exists by Lemma 21.9). If we remove the leaf as well as the unique incident arc, we obtain a tree with $m' - 1 = m$ nodes and $n' - 1 = n$ arcs. As $n = m - 1$, we have $n' = 1 + n = 1 + m - 1 = m$.

7 $\implies$ **6** Consider the K connected components of the graph. Each of them is connected and has no cycles, creating a tree. As a consequence, condition 6 is verified for each component (see proof above). If m_k is the number of nodes in the connected component k, then $m_k - 1$ is the number of arcs in the component. Therefore, the total number of arcs is

$$n = \sum_{k=1}^{K} (m_k - 1) = \sum_{k=1}^{K} m_k - K = m - K.$$

As $n = m - 1$, then $K = 1$, meaning that there is only one connected component, and the graph is connected.

6 $\Longrightarrow$ 2 and 6 $\Longrightarrow$ 7 It is immediate for $m = 1$ and $m = 2$ that the graph has no cycle. Assume now by induction that it is true for $m-1$, and consider a connected graph with m nodes and $m-1$ arcs. As each arc is incident to two nodes, the average degree in this graph is

$$\frac{\sum_{i=1}^{m} d_i}{m} = \frac{2n}{m} = \frac{2(m-1)}{m} = 2 - \frac{2}{m}.$$

This is strictly lower than 2 for any m. Therefore, there is at least one node with degree strictly less than 2. Moreover, as the graph is connected, no node with degree 0 exists. Therefore, there is at least one node i with degree 1. This node cannot be part of a cycle. If we remove i and the incident arc, the remaining graph has no cycle either, by induction. Therefore, the graph has no cycle.

In order to combine these implications, construct a directed graph where each node corresponds to one of the conditions, and each arc to an implication proved above (see Figure 21.6). The equivalence of all the conditions is equivalent to the strong connectivity of this graph, which can easily be verified.

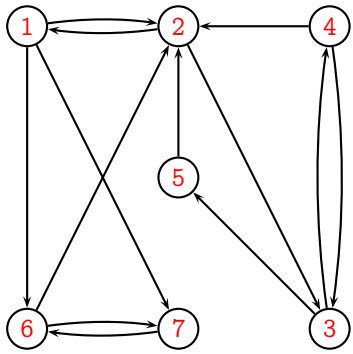
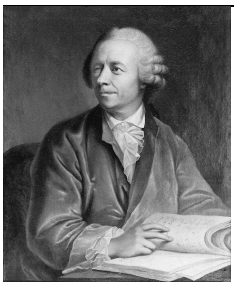


Figure 21.6: Proven implications for Theorem 21.10

□

Trees play an important role in network optimization. In particular, it is common to construct a tree that connects all the nodes of a network. Such a tree is called a *spanning tree*.

Definition 21.11 (Spanning tree)**.** Consider the graph $(\mathcal{V}, \mathcal{E}, \phi)$. The subgraph $(\mathcal{V}, \mathcal{E}', \phi')$, where $\mathcal{E}' \subseteq \mathcal{E}$, and $\phi'(e) = \phi(e)$, for each $e \in \mathcal{E}'$, is a spanning tree of $(\mathcal{V}, \mathcal{E}, \phi)$ if it is a tree.



Leonhard Euler [ɔilr] was born in Basel, Switzerland, on April 15, 1707, and died in St. Petersburg, Russia, on November 18, 1783, of apoplexy. He studied mathematics under the direction of John Bernoulli, and became friend with his two sons, Daniel and Nicholas. In 1727, he joined the Academy of Sciences in St. Petersburg upon the invitation of Empress Catherine I, and in 1741, he became a member of the Academy of Sciences in Berlin, asked by Frederick the Great. During a discussion with the Queen Mother, she found him particularly timid and reserved. "Why, then, will you not talk to me?" she said. "Because Madam," he replied, "I have just come from a country where people are hanged if they talk." His masterpiece is probably *Introductio in analysin infinitorum* (Euler, 1748). The number of things named after Euler is impressively high: conjectures, equations, formulas, theorems, numbers, laws. Euler's identity $e^{i\pi}+1=0$ is an example of mathematical beauty as it involves five fundamental constants, and three basic arithmetic operations appearing exactly once each. He introduced the concept of graphs, when solving the problem known as the Seven Bridges of Königsberg in 1736, that consists in finding a path or a cycle in a graph that uses each edge exactly once.

Figure 21.7: Leonhard Euler

21.5 Networks

It is often useful to associate quantities to nodes and arcs. For instance, in a water network, each house may be associated with a daily consumption of water, each treatment plant may be associated with a daily quantity of water treated, each tank may be associated with a quantity of stored water, and each pipe has a length and a cross section. When quantities are associated with the graph, we call it a *network*.

Definition 21.12 (Network)**.** A network is a 5-uple $(\mathcal{N}, \mathcal{A}, \phi, f_{\mathcal{N}}, f_{\mathcal{A}})$ such that $(\mathcal{N}, \mathcal{A}, \phi)$ is a directed graph, ϕ is an injective incidence function, $f_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}^p$, $p \geq 0$, is a function associating a set of p values with each node, and $f_{\mathcal{A}} : \mathcal{A} \to \mathbb{R}^q$, $q \geq 0$, is a function associating a set of q values with each arc.

In order to simplify the notations, we refer to a network simply as $(\mathcal{N}, \mathcal{A})$. If the arcs are represented by the pair (i, j), the incidence function is implicit. Moreover, the quantities associated with the nodes and the arcs can be represented by vectors of $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively. This section discusses some of these quantities associated with networks.

21.5.1 Flows

A network is often used to transport objects or information. The exact nature of these items varies with the application. The definitions provided here are generic and

do not assume anything about the nature of what is transported. A typical quantity associated with each arc (i, j) is the *flow* on the arc, denoted by x_{ij}. The quantity $x_{ij} \in \mathbb{R}$ is the amount of "things" (water, electricity, information, etc.) that traverses the arc during a given period of time. Note that the concept of flow presented here is static, in the sense that we assume that it represents the total number of "things" traversing the network during a time horizon that is sufficiently large so that all units of flow depart and arrive during this horizon, and the time dimension is irrelevant. The representation of dynamic flows, varying over time, are more complex and out of the scope of this book.

For mathematical convenience, we allow x_{ij} to take on any real value, including negative values. If $x_{ij} < 0$, the interpretation is that the arc (i, j) transports $-x_{ij}$ units of flow from j to i, that is in the opposite direction of the arc. The vector $x \in \mathbb{R}^n$ such that each entry contains the flow on the corresponding arc is called the *flow vector*. Figure 21.8 provides an example of a flow vector, where the flow on each arc is shown next to it. For instance, there are 2.3 units of flow transported from node 1 to node 2. There are 3 units of flow transported from node 4 to node 2 on arc $(4, 2)$, and 2.1 units of flow transported from node 4 to node to 2 on arc $(2, 4)$.

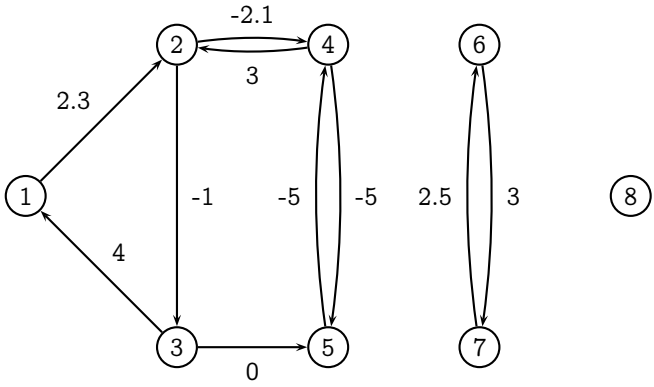


Figure 21.8: A flow vector

Consider now a cut $\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$. The flow through the cut Γ is defined as

$$X(\Gamma) = \sum_{(i,j)\in\Gamma^{\rightarrow}} x_{ij} - \sum_{(i,j)\in\Gamma^{\leftarrow}} x_{ij}, \tag{21.4}$$

where $\Gamma^{\rightarrow}$ is the set of forward arcs, and $\Gamma^{\leftarrow}$ the set of backward arcs of the cut (see Section 21.2). If both $\Gamma^{\rightarrow}$ and $\Gamma^{\leftarrow}$ are empty, the flow is 0.

Paths may also be associated with flows. Suppose that a flow f follows a simple path P from its origin to its destination. The flow vector representing this flow is called a *simple path flow*. It is a vector $x \in \mathbb{R}^n$ such that each component corresponding to a forward arc of the path is equal to f, each component corresponding to a backward

arc is equal to $-f$, and all other components are 0, that is,

$$x_{ij} = \begin{cases} f & \text{if } (i,j) \in P^{\rightarrow} \\ -f & \text{if } (i,j) \in P^{\leftarrow} \\ 0 & \text{otherwise.} \end{cases} \tag{21.5}$$

When the path is a cycle, we refer to a *simple cycle flow*. Figure 21.9 represents a simple path flow for path $1 \rightarrow 2 \leftarrow 4 \rightarrow 5$. It represents f units of flow transported from origin 1 to destination 5 along P.

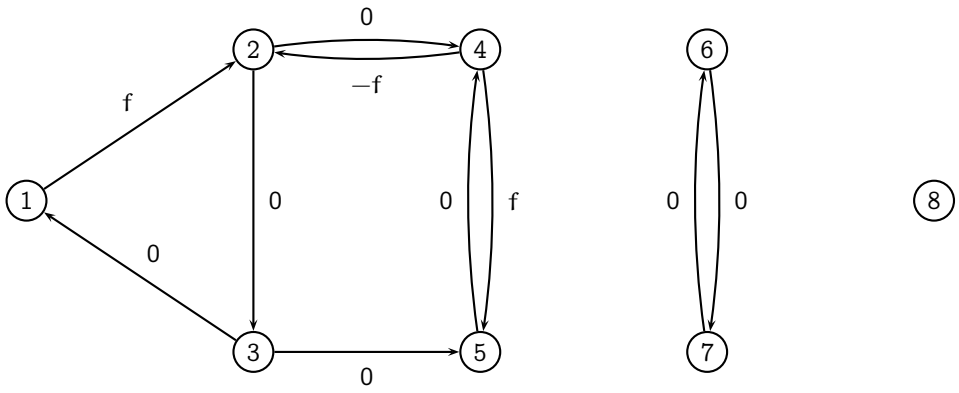


Figure 21.9: Simple path flow for path $1 \rightarrow 2 \leftarrow 4 \rightarrow 5$

21.5.2 Capacities

In many practical applications, the value of the flow cannot exceed some value determined by physical characteristics of the system represented by an arc. For example, the maximum quantity of water per unit of time that a pipe can transport depends on the diameter of the pipe. The maximum number of cars that a highway can transport per unit of time depends on the number and width of lanes. The maximum value of the flow on the arc is called its *capacity*. As we allow x_{ij} to take negative values, both a lower bound ℓ_{ij} and an upper bound u_{ij} on the flow are required. Therefore, we obtain for each arc (i,j) the following constraint:

$$\ell_{ij} \leq x_{ij} \leq u_{ij}. \tag{21.6}$$

There are two common configurations of these bounds in practice. In applications where the direction of flow is constrained to respect the direction of the arc (e.g., one way streets in urban road networks), the value of ℓ_{ij} is set to zero to forbid negative values, and the value of u_{ij} is set to the physical capacity. If the flow is allowed to move in any direction (e.g., in a network transporting electricity), we set $\ell_{ij} = -u_{ij}$ where u_{ij} is set to the physical capacity. However, the framework is general enough to accommodate any value for ℓ_{ij} and u_{ij} such that $\ell_{ij} \leq u_{ij}$.

As we have defined the flow through a cut, the concept of capacity is relevant here as well. Consider a cut $\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$. The capacity of the cut Γ is

$$U(\Gamma) = \sum_{(i,j)\in\Gamma^{\rightarrow}} u_{ij} - \sum_{(i,j)\in\Gamma^{\leftarrow}} \ell_{ij}. \tag{21.7}$$

If both $\Gamma^{\rightarrow}$ and $\Gamma^{\leftarrow}$ are empty, the capacity is 0. For any cut Γ, we always have that the flow through the cut is bounded from above by its capacity, that is

$$X(\Gamma) \leq U(\Gamma). \tag{21.8}$$

If $X(\Gamma) = U(\Gamma)$, the cut is said to be *saturated*, in the sense that no more flow can be sent from set $\mathcal{M}$ (the left bank) to set $\mathcal{N} \setminus \mathcal{M}$ (the right bank).

21.5.3 Supply and demand

Nodes can also be associated with quantities. For instance, the total flow from and towards the node represents the supply and the demand, respectively, of the flows transported on the network. To introduce these quantities, consider a flow vector $x \in \mathbb{R}^n$, and a node i in the network. The quantity of flow leaving node i is given by

$$\sum_{j|(i,j)\in\mathcal{A}} x_{ij}, \tag{21.9}$$

and the quantity of flow that enters node i is given by

$$\sum_{k|(k,i)\in\mathcal{A}} x_{ki}. \tag{21.10}$$

The difference between these two quantities is called the *divergence* of node i.

Definition 21.13 (Divergence)**.** Consider a network with m nodes and n arcs, and a flow vector $x \in \mathbb{R}^n$. For each node i, the divergence of x at node i is defined as the total quantity of flow that leaves the node, minus the total quantity of flow that enters the node:

$$\operatorname{div}(x)_i = \sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki}. \tag{21.11}$$

If this quantity is positive, it means that there are more units leaving the node than units entering it. Units of flow are created at node i. This node is therefore supplying the network with flow. It is a *supply node*. Similarly, if the divergence is negative, it means that there are less units leaving the node than units entering it. It is therefore a node where units of flow are consumed. This is a *demand node*. If the divergence is zero, no flow is generated or consumed at the node. It is a *transit node*. The divergence associated with the flow vector represented in Figure 21.8 is reported in Figure 21.10, where the divergence of node i is denoted by y_i. It is seen

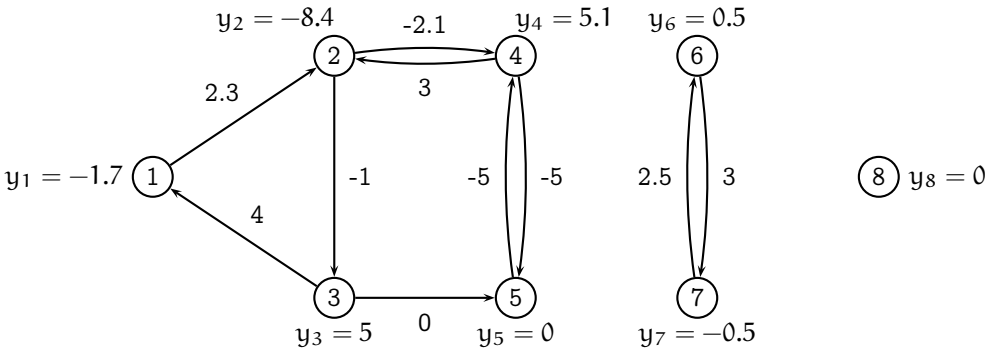


Figure 21.10: Divergence

that nodes 3, 4, and 6 are supply nodes, nodes 1, 2, and 7 are demand nodes, and nodes 5 and 8 are transit nodes.

From (21.11), we obtain that the sum of all divergences is always zero, for any flow vector. Indeed,

$$\begin{aligned} \sum_{i \in \mathcal{N}} \operatorname{div}(x)_i &= \sum_{i \in \mathcal{N}} \sum_{j|(i,j) \in \mathcal{A}} x_{ij} - \sum_{i \in \mathcal{N}} \sum_{k|(k,i) \in \mathcal{A}} x_{ki} \\ &= \sum_{(i,j) \in \mathcal{A}} x_{ij} - \sum_{(k,i) \in \mathcal{A}} x_{ki} \\ &= 0. \end{aligned} \tag{21.12}$$

In other words, every unit of flow that is generated somewhere is consumed somewhere else. A flow vector such that its divergence at *each* node is zero is called a *circulation*, as illustrated in Figure 21.11. In this case, no flow is generated or consumed anywhere.

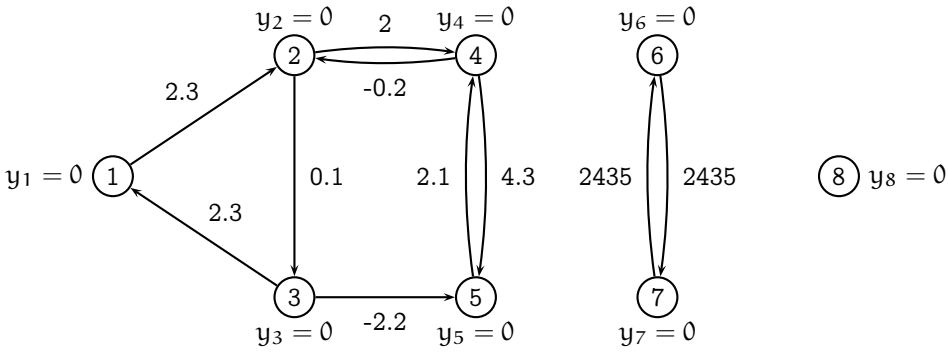


Figure 21.11: Example of a circulation

The following result relates the flow through a cut with the divergence at the nodes.

Theorem 21.14 (Flow through a cut and divergences). *Consider a network with a set $\mathcal{N}$ of m nodes and a set $\mathcal{A}$ of n arcs, a subset of nodes $\mathcal{M} \subset \mathcal{N}$, a cut $\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$ and a flow vector $x \in \mathbb{R}^n$. If $\Gamma^{\rightarrow} \cup \Gamma^{\leftarrow} \neq \emptyset$, then*

$$X(\Gamma) = \sum_{i \in \mathcal{M}} \operatorname{div}(x)_i. \tag{21.13}$$

Proof. From the definition of the flow through a set, and of the sets $\Gamma^{\rightarrow}$ and $\Gamma^{\leftarrow}$, we have

$$X(\Gamma) = \sum_{(i,j)|i \in \mathcal{M}, j \notin \mathcal{M}} x_{ij} - \sum_{(j,i)|j \notin \mathcal{M}, i \in \mathcal{M}} x_{ji}.$$

Note that we have inverted the indices of the second term so that node i always belongs to $\mathcal{M}$ in this expression. Consequently, we can also write

$$X(\Gamma) = \sum_{i \in \mathcal{M}} \left(\sum_{j|(i,j) \in \mathcal{A}, j \notin \mathcal{M}} x_{ij} - \sum_{j|(j,i) \in \mathcal{A}, j \notin \mathcal{M}} x_{ji} \right). \tag{21.14}$$

Now, from (21.11), we have

$$\sum_{i \in \mathcal{M}} \operatorname{div}(x)_i = \sum_{i \in \mathcal{M}} \left(\sum_{j|(i,j) \in \mathcal{A}} x_{ij} - \sum_{j|(j,i) \in \mathcal{A}} x_{ji} \right). \tag{21.15}$$

Consider an arc (k, ℓ) such that both k and ℓ belong to $\mathcal{M}$. In (21.15), the flow $x_{k\ell}$ appears twice, once in the term corresponding to node k with a positive sign, and once for node ℓ with a negative sign:

$$\sum_{i \in \mathcal{M}} \operatorname{div}(x)_i = \cdots + \left(\sum_{j|(k,j) \in \mathcal{A}} x_{kj} - \sum_{j|(j,k) \in \mathcal{A}} x_{jk} \right) + \cdots + \left(\sum_{j|(\ell,j) \in \mathcal{A}} x_{\ell j} - \sum_{j|(j,\ell) \in \mathcal{A}} x_{j\ell} \right). \tag{21.16}$$

Therefore, these two terms cancel out. It means that it is sufficient to consider only $j \notin \mathcal{M}$ in (21.15). Therefore, Equation (21.16) is identical to (21.14), proving the result. □

Consider the cut presented in Figure 21.3 with the flow vector and its divergences presented in Figures 21.8 and 21.10. The flow through the cut is

$$\begin{aligned} x_{24} + x_{54} + x_{76} - x_{42} - x_{45} - x_{67} &= -2.1 - 5 + 2.5 - 3 + 5 - 3 \\ &= -5.6. \end{aligned}$$

The sum of the divergences at the nodes in $\mathcal{M}$ is

$$\begin{aligned} y_1 + y_2 + y_3 + y_5 + y_7 &= -1.7 - 8.4 + 5 + 0 - 0.5 \\ &= -5.6. \end{aligned}$$

21.5.4 Costs

A quantity often associated with an arc (i, j) is a *cost*, which may depend on the amount of flow that traverses the arc. In this book, we focus on linear costs, which are proportional to the flow. The cost to move one unit of flow on arc (i, j) is denoted by c_{ij}, so that the total cost of the arc is $c_{ij}x_{ij}$. The unit of the cost is usually irrelevant, as long as it is the same for every arc in the network. For instance, it can be the actual cost that has to be paid to traverse the arc, expressed in currency units (e.g., a toll road). It can also be the time spent by a unit of flow to traverse the arc. Sometimes, a generalized cost is needed. For instance, both the monetary cost to traverse the arc and the time to traverse it are relevant. In this case, all quantities involved have to be translated into the same unit, for example, a monetary unit, so that they can be added. The valuation of non market resources such as time is referred to by economists as *contingent valuation*. For instance, the value of one hour of travel for commuting car drivers in Switzerland is (on average) CHF 30 (Axhausen et al., 2008). Using this value, if the toll on a road is CHF 10, and the travel time 30 minutes, the total generalized cost per unit of flow would be CHF 25.

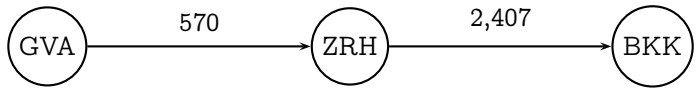
The cost of a path is defined as the sum of the costs of its arcs, that is

$$C(P) = \sum_{(i,j)\in P^{\rightarrow}} c_{ij}x_{ij} - \sum_{(i,j)\in P^{\leftarrow}} c_{ij}x_{ij}. \tag{21.17}$$

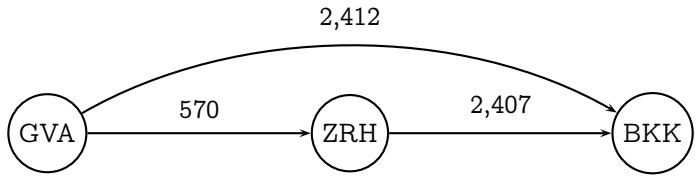
If x is a simple path flow for path P, we have

$$C(P) = \sum_{(i,j)\in P^{\rightarrow}} fc_{ij} - \sum_{(i,j)\in P^{\leftarrow}} fc_{ij} = f\Big(\sum_{(i,j)\in P^{\rightarrow}} c_{ij} - \sum_{(i,j)\in P^{\leftarrow}} c_{ij}\Big). \tag{21.18}$$

As an aside, note that this link-additive assumption may not always correspond to the situation in a real network. For example, if you fly from Geneva Airport (GVA) to Bangkok (BKK) with a transfer at Zurich Airport (ZRH), it costs CHF 2,412. If you fly directly from ZRH to BKK, the cost is CHF 2,407. However, if you buy a ticket from GVA to ZRH, it costs CHF 570. If we use the network representation illustrated in Figure 21.12(a), and send one unit of flow along the path GVA $\rightarrow$ ZRH $\rightarrow$ BKK, the associated cost is 2,977, as a consequence of the link-additivite assumption. It does not correspond to the reality. Another way to model this situation, while keeping the link-additive assumption, is represented in Figure 21.12(b). In this case, the path GVA $\rightarrow$ BKK, with a cost of 2,412, represents passengers buying a ticket from GVA to BKK (regardless of the number of transfers). The path GVA $\rightarrow$ ZRH $\rightarrow$ BKK, with a cost of 2,977, represents passengers that have bought two separate tickets. But that representation ignores the fact that travelers from GVA to ZRH and travelers from GVA to BKK share the same flight (with a limited number of seats) between GVA and ZRH, which may not be satisfactory either. This illustrates that it is important to keep assumptions such as the link-additive assumption in mind when creating the network representation of a real problem.



(a) First model



(b) Second model

Figure 21.12: Network representation for an airline problem

21.5.5 Network representation

In real life, it is common to use a map to look at a network or a representation, as in Figure 21.2. It gives an overview of the overall topology of the network. When dealing with network algorithms, the computer does not have access to this bird's-eye view of the network. Instead, it has access to the set of nodes, the set of arcs, and a representation of the incidence function. From the point of view of the computer, the network is more like a labyrinth, where only local information is available. A common representation of a network is the *adjacency matrix*. The adjacency matrix $A \in \mathbb{R}^{m \times m}$ of a network with m nodes is a $m \times m$ square matrix. Each entry is defined as

$$A(i,j) = \begin{cases} 1 \text{ if } (i,j) \in \mathcal{A}, \\ 0 \text{ otherwise.} \end{cases} \tag{21.19}$$

Note that this representation is valid because we assume that the incidence function of a network is injective. The adjacency matrix of the network represented in Figure 21.2 is

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{21.20}$$

In order to associate quantities with the arcs, an arc numbering convention must be adopted. For instance, arcs can be numbered sequentially as they appear in the

adjacency matrix read row by row. In our example, arc 1 would be $(1,2)$, and arc 10 would be $(7,6)$. In many practical applications, as well as in our simple example, the adjacency matrix is sparse, as it contains a large number of zero entries. There are several techniques for an efficient storage of sparse matrices (see, for instance, Dongarra, 2000 and Montagne and Ekambaram, 2004). A simple one consists in storing adjacency lists for each node. In this configuration, each node i is associated with a list of length equal to its outdegree d_i^+. Each element of the list corresponds to an arc (i,j) going out of i. The vector $f_{\mathcal{A}}(i,j) \in \mathbb{R}^q$ of values associated with the corresponding arc may also be stored in the list. The adjacency lists of the simple network represented in Figure 21.2, where each arc (i,j) is associated with a quantity $f_{\mathcal{A}}(i,j) = x_{ij}$, as illustrated in Figure 21.13. Each element in list i is associated with an arc (i,j) and contains: the number j of the downstream node of the corresponding arc, a vector of quantities associated with this arc (here, x_{ij}), and a pointer toward the next element in the list. In our example, node 1 has only one outgoing arc: $(1,2)$. Therefore, the list contains only one element, with three entries: the number 2 referring to node 2, the value x_{12} associated with the arc $(1,2)$, and a null pointer (illustrated by a dot). Node 2 has two outgoing arcs: $(2,3)$ and $(2,4)$. Therefore, the list associated with node 2 has two elements. The first one corresponds to the arc $(2,3)$ and contains the number 3 referring to node 3, the value x_{23}, and a pointer to the next element in the list. The second element corresponds to the arc $(2,4)$ and contains the number 4 referring to node 4, the value x_{24}, and a null pointer characterizing the last element of the list. As node 8 is not associated with any outgoing arc, the associated list is empty, and the corresponding pointer is null.

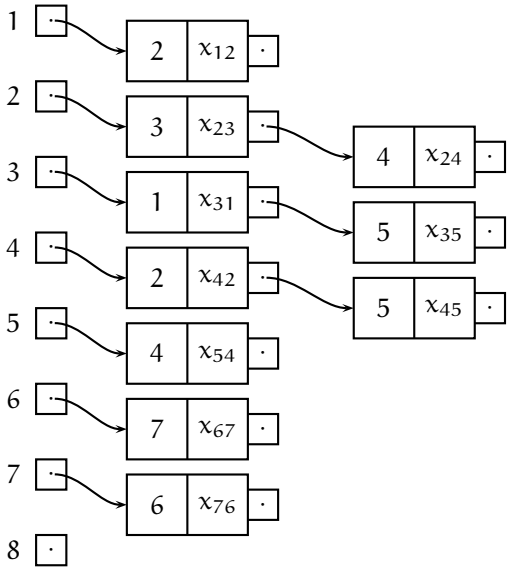


Figure 21.13: Representation of the network in Figure 21.2 using adjacency lists

21.6 Flow decomposition

Consider the network represented in Figure 21.2 and a path flow vector that sends flows along simple paths and cycles, as shown in Table 21.1.

Table 21.1: Flows on simple paths and cycles

Path number	Path	Flow
1	$1 \rightarrow 2 \rightarrow 4 \rightarrow 5$	1.5
2	$1 \rightarrow 2 \leftarrow 4 \rightarrow 5$	1.5
3	$1 \rightarrow 2 \leftarrow 4 \rightarrow 5 \leftarrow 3 \rightarrow 1$	1
4	$3 \rightarrow 5 \rightarrow 4$	1
5	$6 \rightarrow 7$	2

The procedure to calculate the resulting flow vector and associated divergences is called *network loading*. The flow vector is obtained by simply summing up for each arc the flow transported by paths containing the arc, and its divergence is defined by (21.11). It is represented in Figure 21.14. There are 1.5+1.5+1 units of flows leaving node 1 using paths 1, 2 and 3, and one unit of flow arriving at node 1 from path 3. So there are a total of 3 units of flows leaving node 1, which corresponds to its divergence. Note that the flow on arc $(3,5)$ is zero, as one unit of flow traverses the arc in the forward direction along path 4, and one unit of flow in the backward direction along path 4.

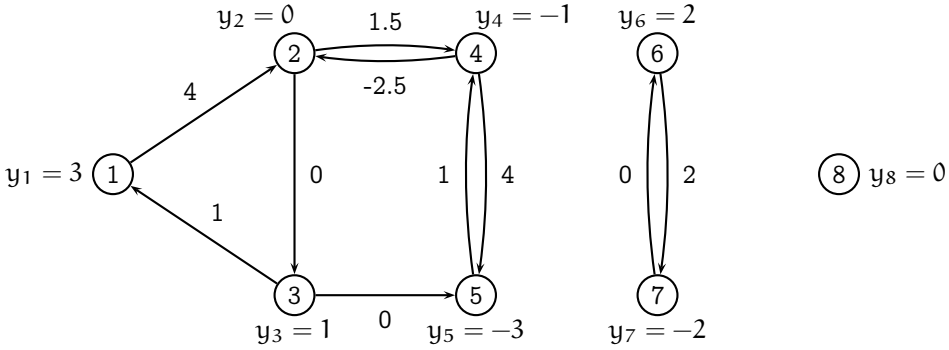


Figure 21.14: Assigning simple path flows on a network

The inverse procedure, consisting of reconstituting the path flows from the flow vector is called *flow decomposition*. It is particularly important in applications. For instance, consider the case where the flow represents trucks that are transporting goods on the network. As discussed in Chapter 22, we may want to transport these goods at minimum cost. The result of the optimization algorithm is a flow vector. However, the instructions to the drivers of the trucks should be expressed in terms of path flows. These are obtained from the flow decomposition procedure described

next. The procedure is more complex, and composed of three steps: (i) transform the flow vector into a circulation by adding artificial nodes and arcs, (ii) decompose the circulation into simple cycle flows, and (iii) remove the artificial nodes and obtain the simple path flows for the original network. We describe it first on the same example, using the flow vector and associated divergences obtained by applying the network loading procedure.

Step (i) First, we transform the flow vector into a circulation. We add an artificial node (call it a) and for each node i such that its divergence is non zero, we add an arc (a, i) with a flow equal to its divergence, as illustrated by Figure 21.15.

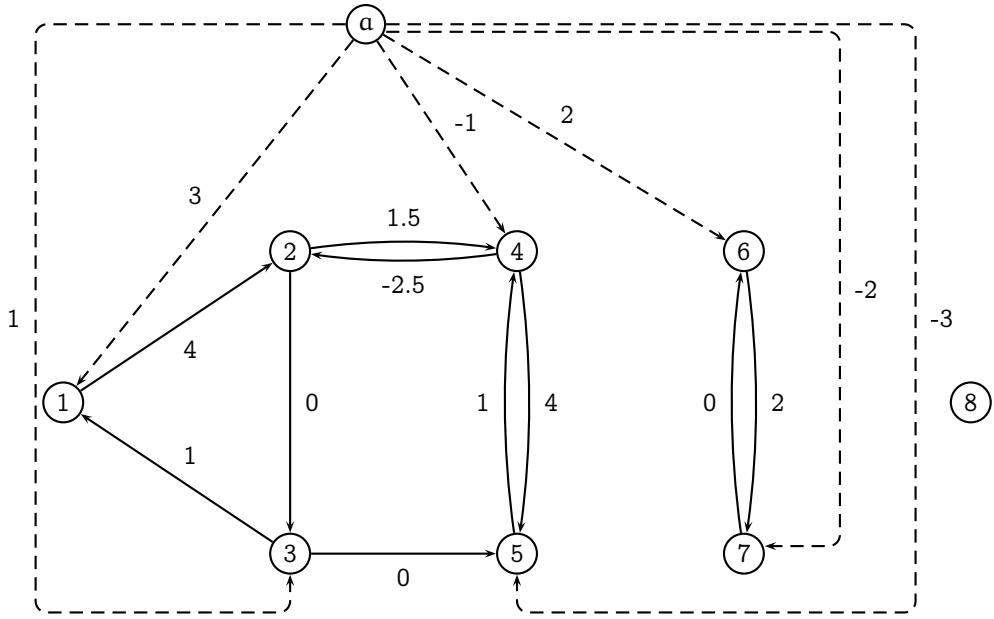


Figure 21.15: Adding an artificial node and arcs to transform a flow vector into a circulation

Step (ii) We generate a simple cycle flow z such that

$$\begin{aligned} z_{ij} &= x_{ij} \text{ for at least one arc } (i,j), \\ 0 \le z_{ij} &\le x_{ij} \text{ for each } (i,j) \text{ with } x_{ij} \ge 0, \\ 0 \ge z_{ij} &\ge x_{ij} \text{ for each } (i,j) \text{ with } x_{ij} \le 0, \end{aligned} \tag{21.21}$$

using Algorithm 21.1. It means that the simple cycle flow that we generate transports the entire flow on at least one arc and part of the total flow on each arc in a consistent way (that is, in the right direction).

Algorithm 21.1: Generation of a simple cycle flow from a circulation

Objective

Generate a simple cycle flow z such that

$$\begin{aligned} z_{ij} &= x_{ij} \text{ for at least one arc } (i,j), \\ 0 \le z_{ij} &\le x_{ij} \text{ for each } (i,j) \text{ with } x_{ij} \ge 0, \\ 0 \ge z_{ij} &\ge x_{ij} \text{ for each } (i,j) \text{ with } x_{ij} \le 0. \end{aligned} \tag{21.22}$$

Input

A network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs.
A circulation $x \in \mathbb{R}^n$.

Output

A simple cycle flow $z \in \mathbb{R}^n$ verifying (21.22).

Initialization

Select an arc (k,ℓ) such that $x_{k\ell} > 0$ or an arc (ℓ,k) such that $x_{\ell k} < 0$.
$\mathcal{S}_0 := \{\ell\}$, $\mathcal{C}_\ell := \mathcal{S}_0$, $t := 0$, $P = \{k\}$.

Repeat

$\mathcal{S}_t = \emptyset$.
for $i = 1, \dots, m$, $i \notin \mathcal{C}_\ell$ **do**
if $\exists (j,i)$ *such that* $j \in \mathcal{S}_{t-1}$ *and* $x_{ji} > 0$ *or* $\exists (i,j)$ *such that* $j \in \mathcal{S}_{t-1}$ *and* $x_{ij} < 0$ **then** $\mathcal{S}_t := \mathcal{S}_t \cup \{i\}$
$\mathcal{C}_\ell := \mathcal{C}_\ell \cup \mathcal{S}_t$.
$t := t + 1$.
Until $\mathcal{S}_t = \emptyset$.
T index such that $k \in \mathcal{S}_T$, $\gamma := k$, $f := +\infty$.
for $t = T-1, \dots, 0$ **do**
if $\exists i \in \mathcal{S}_t$ *such that* $x_{i\gamma} > 0$ **then**
$\mathcal{P} := \{i \rightarrow\} \cup \mathcal{P}$
if $x_{i\gamma} < f$ **then** $f := x_{i\gamma}$
else
Select $i \in \mathcal{S}_t$ such that $x_{\gamma i} < 0$
$\mathcal{P} := \{i \leftarrow\} \cup \mathcal{P}$
if $-x_{\gamma i} < f$ **then** $f := -x_{\gamma i}$
$\gamma = i$.
for $(i,j) \in \mathcal{A}$ **do**
if $(i,j) \in \mathcal{P}^{\rightarrow}$ **then** $z_{ij} := f$
else if $(i,j) \in \mathcal{P}^{\leftarrow}$ **then** $z_{ij} := -f$
else $z_{ij} := 0$

The algorithm works as follows. We first select an arc (k, ℓ) transporting a positive amount of flow, such as arc $(1, 2)$, for example. We group the nodes into layers using a recursive procedure. The first layer $\mathcal{S}_0 = \{\ell\}$ contains only node ℓ. Layer $\mathcal{S}_t$ is built from layer $\mathcal{S}_{t-1}$ in the following way: node i belongs to layer $\mathcal{S}_t$ if it does not belong to any previous layer $\mathcal{S}_0, \ldots, \mathcal{S}_{t-1}$, and there is an arc carrying flow between a node j in $\mathcal{S}_{t-1}$ and node i, that is, at least one of the two conditions is verified (one condition for forward flows, one for backward):

1. there is an arc (j, i) such that $j \in \mathcal{S}_{t-1}$ and $x_{ji} > 0$, or
2. there is an arc (i, j) such that $j \in \mathcal{S}_{t-1}$ and $x_{ij} < 0$.

Intuitively, the nodes in layer $\mathcal{S}_t$ are the next step for the flow going out of the nodes in layer $\mathcal{S}_{t-1}$. The recursive procedure is interrupted if $\mathcal{S}_t$ is empty. The set of nodes covered by the flow going out of node ℓ, that is $\mathcal{C}_\ell = \bigcup_t \mathcal{S}_t$ is "isolated" from the rest of the nodes $(\mathcal{N} \setminus \mathcal{C}_\ell)$, in the sense that there is no flow from one set to the other. Consider the cut $\Gamma = (\mathcal{C}_\ell, \mathcal{N} \setminus \mathcal{C}_\ell)$.
As we are dealing with a circulation, the flow through the cut is 0. Indeed, if some units of flow were transferred from set $\mathcal{C}_\ell$ to $\mathcal{N} \setminus \mathcal{C}_\ell$, there would be at least one arc (i, j) transporting positive flow such that $i \in \mathcal{C}_\ell$ and $j \notin \mathcal{C}_\ell$. It is not possible, as the procedure would have included j into one of the sets $\mathcal{S}_t$ and, therefore, it would belong to $\mathcal{C}_\ell$. If some units of flow were transferred from set $\mathcal{N} \setminus \mathcal{C}_\ell$ to set $\mathcal{C}_\ell$, as we have a circulation, the same amount of flow must also be transferred in the other direction, which is not possible according to the previous argument. Therefore, the flow through the cut Γ is zero.
Consequently, as the arc (k, ℓ) was selected such that it transports positive flow, it cannot be in the cut. This guarantees that node k belongs to $\mathcal{C}_\ell$, and more specifically, to one $\mathcal{S}_T$ such that $T \geq 1$, as node ℓ is the only node in $\mathcal{S}_0$.
In our example where arc $(1, 2)$ is selected, $\mathcal{S}_0 = \{2\}$. There are four arcs incident to node 2. Only two of them are transporting flow out of node 2: arc $(2, 4)$ transporting 1.5 units of flow (forward), and arc $(4, 2)$ transporting 2.5 units of flow (backward). Therefore, $\mathcal{S}_1 = \{4\}$. From node 4, arc $(a, 4)$ is transporting one unit of flow in the backward direction, and arc $(4, 5)$ is transporting 4 units of flow in the forward direction. Therefore, $\mathcal{S}_2 = \{a, 5\}$. From a, arcs $(a, 1)$, $(a, 3)$, and $(a, 6)$ are transporting positive quantities of flows. From node 5, only arc $(5, 4)$ is transporting a positive flow. However, as node 4 has already been included in a layer, it does not qualify for the next. Therefore, $\mathcal{S}_3 = \{1, 3, 6\}$. Finally, we obtain $\mathcal{S}_4 = \{7\}$ and $\mathcal{S}_5 = \emptyset$. It is seen that node 1 belongs to $\mathcal{S}_3$, and that node 8 has not been included in any set $\mathcal{S}_t$. Indeed, there is no path from node 2 to node 8.
Starting from node $k \in \mathcal{S}_T$, we select a sequence of nodes $i_{T-1} \in \mathcal{S}_{T-1}$, $i_{T-2} \in \mathcal{S}_{T-2}$, $\ldots i_0 \in \mathcal{S}_0$ such that there is an arc transporting positive flow (either forward or backward) between i_{t-1} and i_t (note that, by construction, such arcs always exist, and $i_0 = \ell$). Together with the arc (k, ℓ), the sequence of nodes and associated arcs form a simple cycle $\mathcal{P}$ such that all its forward arcs have positive flow, all its backward arcs have negative flow. Consider the minimum

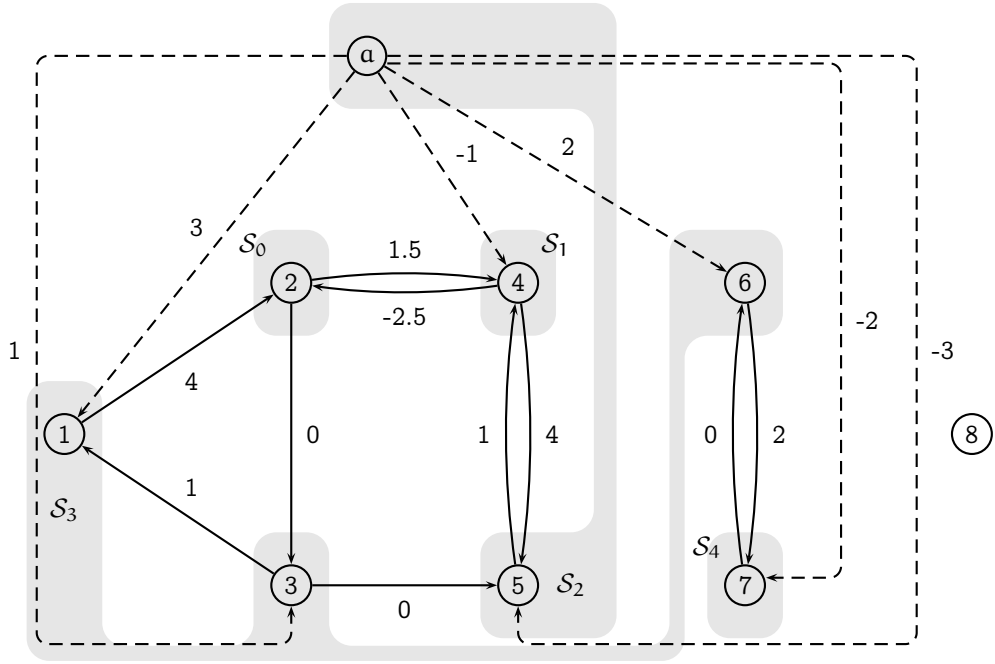


Figure 21.16: Set of nodes

flow f transported by all these arcs, that is

$$f = \min_{(i,j)\in\mathcal{P}} |x_{ij}| > 0. \tag{21.23}$$

We define a simple cycle flow z as follows

$$z_{ij} = \begin{cases} f & \text{if } (i,j) \in \mathcal{P}^{\rightarrow} \\ -f & \text{if } (i,j) \in \mathcal{P}^{\leftarrow} \\ 0 & \text{otherwise.} \end{cases} \tag{21.24}$$

By construction, the simple cycle flow verifies the properties (21.21).
In our example, the organization of the nodes into layers is illustrated in Figure 21.17, together with the arc (k,ℓ) that enables to create a simple cycle:

$$1 \rightarrow 2 \rightarrow 4 \leftarrow a \rightarrow 1. \tag{21.25}$$

The minimum amount of flow transported by one arc is 1 (arc $(a,4)$, backward), so that

$$z_{12} = z_{24} = -z_{a4} = z_{a1} = 1,$$

and all other arc flows are zero.

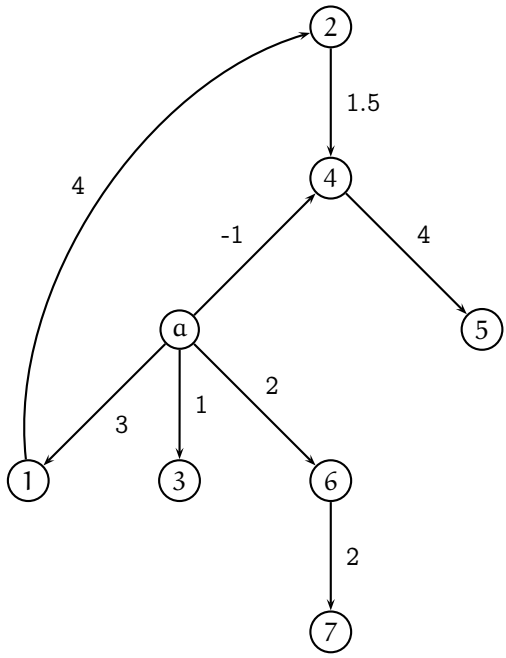


Figure 21.17: Construction of a simple cycle

We then subtract the obtained simple cycle flow from the original flow vector: $x^+ = x - z$. From properties (21.21), the flow on each arc carrying flow both for x and x^+, that is each arc (i,j) with $x_{ij}x^+_{ij} \neq 0$, has the same sign for x^+ and x. Moreover, there is at least one arc (i,j) such that $x_{ij} \neq 0$ and $x^+_{ij} = 0$. Note that this arc is not necessarily the arc (k,ℓ) that was chosen to initiate the procedure. The procedure is repeated until $x^+ = 0$. Note that it is guaranteed to happen, as each time the procedure is applied, at least one arc transporting positive flow before the identification of the simple cycle flow has zero flow after. So the maximum number of times that the procedure is applied equals the number of arcs with non zero flow in the original flow vector.
If we start the process again, we generate the following simple cycle flows:

Cycle	Flow
$1 \to 2 \to 4 \leftarrow a \to 1$	1
$1 \to 2 \to 4 \to 5 \leftarrow a \to 1$	0.5
$1 \to 2 \leftarrow 4 \to 5 \leftarrow a \to 1$	1.5
$1 \to 2 \leftarrow 4 \to 5 \leftarrow a \to 3 \to 1$	1
$4 \to 5 \to 4$	1
$6 \to 7 \leftarrow a \to 6$	2

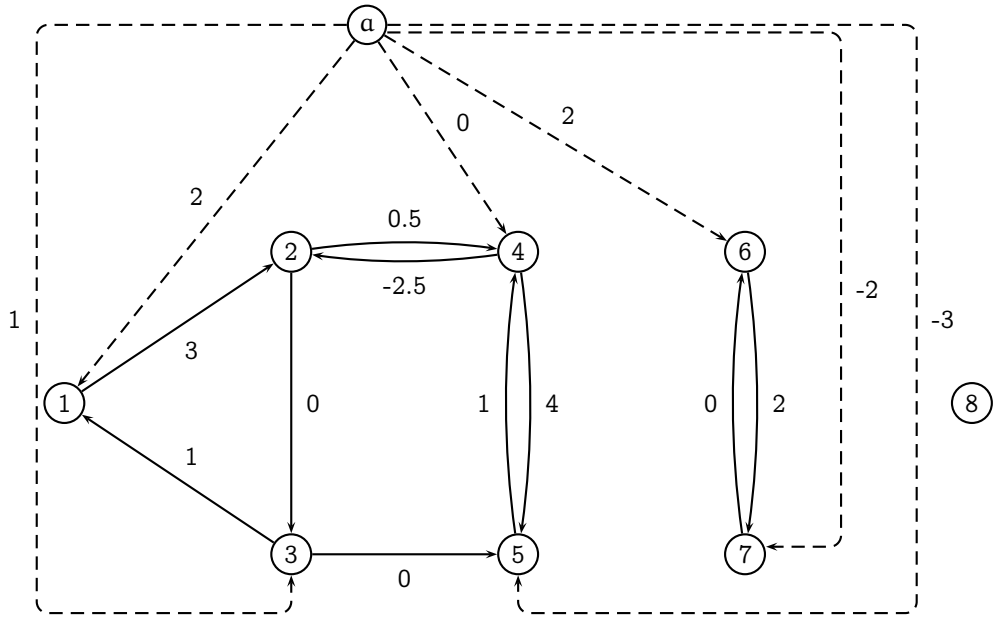


Figure 21.18: New flow vector after subtracting the simple cycle flow

Algorithm 21.2: Circulation decomposition

Objective

Decomposition of a circulation into consistent simple cycle flows.

Input

A network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs.

A circulation $x \in \mathbb{R}^n$.

Output

A list of simple cycle flows consistent with x.

Initialization

$w := x$, list $:= \emptyset$

Repeat

Obtain a simple cycle flow z using Algorithm 21.1.

list := list $\cup$ z.

$w = w - z$.

Until $w = 0$.

Step (iii) It is now sufficient to remove the artificial node a to obtain the paths from the original network:

Path	Flow
$1 \rightarrow 2 \rightarrow 4$	1
$1 \rightarrow 2 \rightarrow 4 \rightarrow 5$	0.5
$1 \rightarrow 2 \leftarrow 4 \rightarrow 5$	1.5
$3 \rightarrow 1 \rightarrow 2 \leftarrow 4 \rightarrow 5$	1
$4 \rightarrow 5 \rightarrow 4$	1
$6 \rightarrow 7$	2

Note that the decomposition is not unique. The above simple path flows are not the same as the ones described in Table 21.1:

Path	Flow
$1 \rightarrow 2 \rightarrow 4 \rightarrow 5$	1.5
$1 \rightarrow 2 \leftarrow 4 \rightarrow 5$	1.5
$1 \rightarrow 2 \leftarrow 4 \rightarrow 5 \leftarrow 3 \rightarrow 1$	1
$3 \rightarrow 5 \rightarrow 4$	1
$6 \rightarrow 7$	2

However, when each of them is loaded on the network, the same vector flow is generated.

Algorithm 21.3: Flow decomposition

Objective
  Decomposition of a flow into consistent simple path flows.
Input
  A network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs.
  A flow vector $x \in \mathbb{R}^n$.
Output
  A list of simple path flows consistent with x.
Initialization
  $\mathcal{N}^+ := \mathcal{N} \cup a$.
  $\mathcal{A}^+ := \mathcal{A} \bigcup_{i \in \mathcal{N}} (a, i)$.
  $y_{ij} := x_{ij}$ for each $(i, j) \in \mathcal{A}$.
  $y_{ai} = \text{div}(x)_i$ for each $i \in \mathcal{N}$.
Apply Algorithm 21.2 on network $(\mathcal{N}^+, \mathcal{A}^+)$ and circulation y.
For each generated cycle containing node a, remove a and the incident arcs.

We see that each of these simple path flows is either a simple cycle flow, or starts from a supply node and ends at a demand node. Also, the flows are transported in the same direction as the original flow on each arc. We say that they are *consistent* with the original flow x.

Definition 21.15 (Consistent simple path flow)**.** Let $(\mathcal{N}, \mathcal{A})$ be a network with m nodes and n arcs. Let $x \in \mathbb{R}^n$ be a flow vector, and $z \in \mathbb{R}^n$ be a simple path flow. z is said to be consistent with x if

- $x_{ij} > 0$ for each (i,j) such that $z_{ij} > 0$,
- $x_{ij} < 0$ for each (i,j) such that $z_{ij} < 0$,
- one of these two conditions holds
  - $\mathrm{div}(z) = 0$, that is, z is a cycle, or
  - $\mathrm{div}(x)_i \mathrm{div}(z)_i > 0$ for each i such that $\mathrm{div}(z)_i \neq 0$, that is the origin of the simple path is a supply node for x, and the destination of the simple path is a demand node.

We now provide a formal analysis of the above procedure.

Theorem 21.16 (Cycle flow decomposition)**.** *Let $(\mathcal{N}, \mathcal{A})$ be a network with m nodes and n arcs. Let $x \in \mathbb{R}^n$ be a circulation, that is $\mathrm{div}(x)_i = 0$, for $i = 1, \ldots, m$. Then the circulation can be decomposed into T simple cycle flows $z^1, \ldots, z^T$ consistent with x such that $x = \sum_{t=1}^{T} z^t$ and $T \leq n$.*

Proof. Let $0 < p_0 \leq n$ be the number of arcs transporting a non zero amount of flow. If we apply Algorithm 21.1, we obtain a simple cycle flow z consistent with x. Indeed, the conditions of Definition 21.15 are directly obtained from (21.22). Now, let us consider the flow vector $x^+ = x - z$, and let p_1 be the number of arcs transporting a non zero amount of flow. As $x_{ij} = z_{ij}$ for at least one arc, we have that $p_1 \leq p_0 - 1$. Moreover, conditions (21.22) guarantee that the non zero flows of x^+ have the exact same sign as the corresponding flow of x. If we repeat the same process several times, at each iteration t such that $p_t > 0$, we have $p_{t+1} \leq p_t - 1$. The process is stopped at iteration T when no arc is transporting a non zero amount of flow, that is when $p_T = 0$. We have

$$0 = p_T \leq p_{T-1} - 1 \leq p_{T-2} - 2 \leq \ldots \leq p_{T-k} - k \leq \ldots$$

for any $0 \leq k \leq T$. For $k = T$, we have

$$0 \leq p_0 - T \text{ that is } T \leq p_0 \leq n.$$

Therefore, there are at most T simple cycle flows generated from x. □

Corollary 21.17 (Flow decomposition)**.** *Let $(\mathcal{N}, \mathcal{A})$ be a network with m nodes and n arcs. Let $x \in \mathbb{R}^n$ be a flow vector. Then it can be decomposed into T simple path flows $z^1, \ldots, z^T$ consistent with x, such that $x = \sum_{t=1}^{T} z^t$ and $T \leq n + m$.*

Proof. Consider an extended network obtained from $(\mathcal{N}, \mathcal{A})$ by adding one node a. For each node i in the original network, add an arc (a, i). Now consider the flow vector

y such that $y_{ij} = x_{ij}$ if (i,j) is an arc from the original network, and $y_{ai} = \text{div}(x)_i$ for the newly added arcs (see Figure 21.15). This network has $n+1$ nodes and $n+m$ arcs. The flow vector y is a circulation. Indeed, for the nodes from the original network,

$$\begin{aligned}\text{div}(y)_i &= \sum_{j|(i,j)\in\mathcal{A}} y_{ij} - \sum_{k|(k,i)\in\mathcal{A}} y_{ki} - y_{ai} \\ &= \sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki} - \text{div}(x)_i \\ &= \text{div}(x)_i - \text{div}(x)_i \\ &= 0.\end{aligned}$$

For node a, as all arcs are going out of the node, we have

$$\text{div}(y)_a = \sum_{j\in\mathcal{N}} y_{aj} = \sum_{j\in\mathcal{N}} \text{div}(x)_i = 0,$$

where the last result is obtained from (21.12).

From Theorem 21.16, the circulation in the new network can be decomposed into $T \leq n + m$ simple cycle flows consistent with y. If such a simple cycle flow does not contain the node a, it is also a simple cycle flow consistent with x in the original network. Now, if node a belongs to a simple cycle transporting $f > 0$ units of flows, it must be in the following configuration: $\cdots d \leftarrow a \rightarrow o \cdots$, as a is the upstream node of all arcs incident to a in the modified network. If we remove node a from the cycle, we obtain a simple path flow starting from node o and reaching node d. The arc (a,o) is transporting $f > 0$ units of flow in the simple cycle. As it is consistent with y, it means that $y_{ao} = \text{div}(x)_o > 0$, and o is a supply node for x. Similarly, arc (a,d) is transporting $-f < 0$ units of flow in the simple cycle, as it appears backward. Again, from consistency of the cycle, we have that $y_{ad} = \text{div}(x)_d < 0$, and d is a demand node for x. Therefore, the simple path flow is consistent with x. □

Corollary 21.18 (Integer flow decomposition). *Let $(\mathcal{N},\mathcal{A})$ be a network with m nodes and n arcs. Let $x \in \mathbb{Z}^n$ be a flow vector containing only integer values. Then it can be decomposed into T simple path integer flows $z^1,\ldots,z^T$ consistent with x, such that $x = \sum_{t=1}^{T} z^t$ and $T \leq n+m$.*

Proof. In the decomposition of a circulation into simple cycle flows (Algorithm 21.1), the flow on each simple cycle flow is the smallest flow transported by an arc of the generated cycle. Therefore, if the flow on each arc is integer, the simple cycle flow is also integer. And when it is subtracted to the original flow to generate the next cycle, the flow obviously remains integer-valued. □

The above result has important practical implications. If the flow vector represents physical units (trucks, containers, etc.), it can be decomposed into simple paths transporting flows of these units.

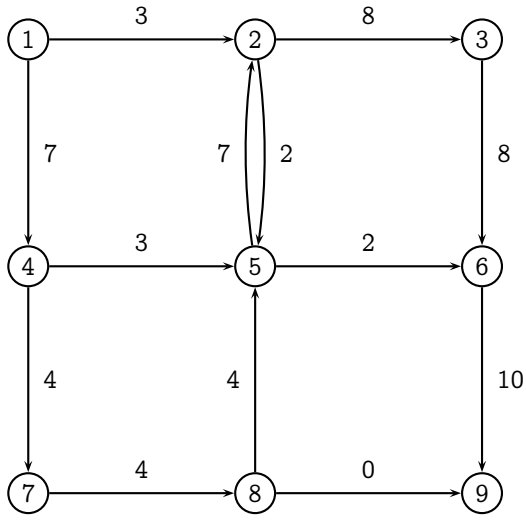


Figure 21.19: Example of an integer flow vector

Example 21.19 (Decomposition of an integer flow vector)**.** Consider the network presented in Figure 21.19, where the value on each arc represents the arc flow.

Applying Algorithm 21.3 produces the following simple path flows:

Path	Flow
$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9$	8
$1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \leftarrow 1$	3
$1 \rightarrow 4 \rightarrow 7 \rightarrow 8 \rightarrow 5 \rightarrow 2 \leftarrow 1$	4
$1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 9$	2

Again, the decomposition is not unique. The following simple path flows are also consistent with the flow vector:

Path	Flow
$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9$	1
$1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 9$	2
$1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9$	3
$1 \rightarrow 4 \rightarrow 7 \rightarrow 8 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9$	4

21.7 Minimum spanning trees

As characterized by condition 5 of Theorem 21.10, a spanning tree can be seen as the smallest structure that keeps all nodes of a network connected. Indeed, the removal of a single arc from a tree disconnects it. In this chapter, we analyze the problem of finding a spanning tree of a network that is associated with the smallest cost. As a

motivation, consider a telecommunication company who must install an optical fiber infrastructure to connect a set of cities. The company has identified for each pair of cities if it is feasible to build a connection, and, if so, at what cost. The cities (vertices) and these potential connections (edges) form a network with an underlying undirected graph, as the orientation of the arcs are ignored. The problem is to decide which connections have to be built in order for all cities to be connected, at minimal cost.

Example 21.20 (Network for the minimum spanning tree problem)**.** A telecommunication company must connect 7 cities with optical fiber. The potential connections to be built, together with the associated costs, are modeled by the network represented in Figure 21.20.

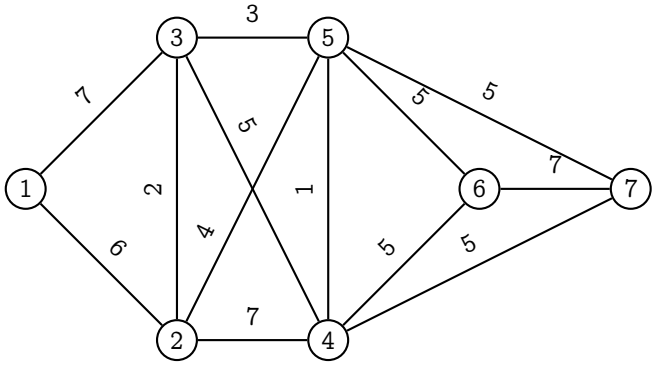


Figure 21.20: Network for the minimum spanning tree (Example 21.20)

A minimum spanning tree, with a total cost of 22, is represented in Figure 21.21, where arcs represented by a plain line are part of the tree. Note that including arc (5,7) instead of (4,7) would give another spanning tree with the same cost.

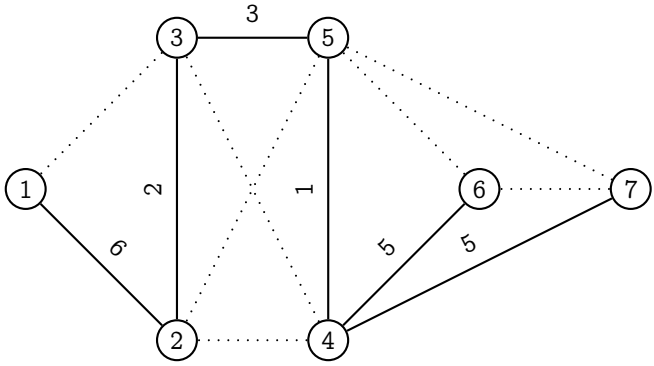


Figure 21.21: Minimum spanning tree for Example 21.20

The following theorem characterizes a minimum spanning tree.

Theorem 21.21 (Minimum spanning tree). *Let $(\mathcal{N}, \mathcal{E})$ be a network with m nodes and n edges (that is, undirected arcs). Let $c \in \mathbb{R}^n$ be the vector of edge costs. Let $\mathcal{E}' \subseteq \mathcal{E}$ be such that $\mathcal{T}^* = (\mathcal{N}, \mathcal{E}')$ is a spanning tree. If $(i,j) \in \mathcal{T}^*$, removing it disconnects the tree into two connected components. Let $\mathcal{N}_i$ be the set of nodes in the connected component containing node i, and $\mathcal{N}_j$ be the set of nodes in the other one. In the network $(\mathcal{N}, \mathcal{E})$, consider the cut $\Gamma_{ij} = (\mathcal{N}_i, \mathcal{N} \setminus \mathcal{N}_i) = (\mathcal{N}_i, \mathcal{N}_j)$. $\mathcal{T}^*$ is a minimum spanning tree if and only if $\mathcal{T}^*$ verifies the cut condition, that is $c_{ij} \leq c_{k\ell}$, for each $(i,j) \in \mathcal{T}^*$ and each $(k,\ell) \in \Gamma_{ij}$.*

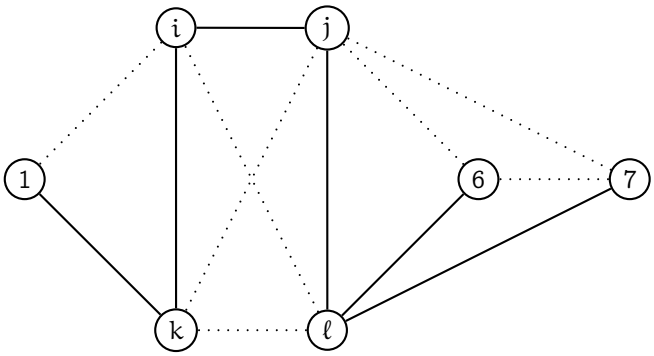


Figure 21.22: Illustration of Theorem 21.21

Proof. Figure 21.22 illustrates the edges involved in this theorem. Edge (i,j) belongs to the spanning tree. If removed, it defines the cut $\Gamma_{ij} = (\mathcal{N}_i, \mathcal{N}_j)$ with $\mathcal{N}_i = \{1, i, k\}$ and $\mathcal{N}_j = \{j, \ell, 6, 7\}$.

Necessary condition Assume first that $\mathcal{T}^*$ is a minimum spanning tree. Assume by contradiction that $c_{ij} > c_{k\ell}$. Then removing arc (i,j) from $\mathcal{T}^*$ disconnects the tree. As it belongs to the cut, adding arc (k,ℓ) reconnects it. And the total cost of the new tree is lower than the cost of $\mathcal{T}^*$, contradicting the fact that $\mathcal{T}^*$ is optimal.

Sufficient condition Assume that $\mathcal{T}^*$ verifies the cut condition. Consider a minimum spanning tree $\widehat{\mathcal{T}}$. From the necessary condition above, it also verifies the cut condition. If $\widehat{\mathcal{T}} = \mathcal{T}^*$, the proof is finished. If not, consider $(i,j) \in \mathcal{T}^*$ such that $(i,j) \notin \widehat{\mathcal{T}}$. Adding (i,j) to $\widehat{\mathcal{T}}$ creates a cycle (condition 4 of Theorem 21.10). This cycle must contain an edge $(k,\ell) \in \Gamma_{ij}$. Note that, by construction, the cut obtained by removing (i,j) from $\mathcal{T}^*$ is exactly the same as the cut obtained by removing (k,ℓ) from $\widehat{\mathcal{T}}$. As $\mathcal{T}^*$ verifies the cut condition, we have $c_{ij} \leq c_{k\ell}$. As $\widehat{\mathcal{T}}$ verifies the cut condition, we have that $c_{k\ell} \leq c_{ij}$. Consequently, $c_{ij} = c_{k\ell}$. Now replace edge (k,ℓ) by edge (i,j) in $\widehat{\mathcal{T}}$. The new tree is also optimal, as the cost has not changed. If this new tree is equal to $\mathcal{T}^*$, we are done. Otherwise, we start the process as many times as is needed to obtain $\mathcal{T}^*$, and show that it is optimal.

This can happen only a finite number of times, as each time a new arc from $\mathcal{T}^*$ is included in $\widehat{\mathcal{T}}$, and there are exactly $m-1$ of them.

□

The optimality condition provided by Theorem 21.21 suggests a simple algorithm that constructs step by step the spanning tree, making sure that the cut condition is always verified. At each iteration of this algorithm, we have a partial tree. We consider the set $\mathcal{M}$ of nodes connected by this partial tree, and the set of all other nodes $\mathcal{N}\setminus\mathcal{M}$. Among all edges belonging to the cut $\Gamma = (\mathcal{M}, \mathcal{N}\setminus\mathcal{M})$, we select the one with the minimum cost, and add it to the partial tree. Such a constructive algorithm, which considers a locally minimum strategy at each step without reconsidering any previous decision, is called a *greedy* algorithm.

Definition 21.22 (Greedy algorithm)**.** A greedy algorithm is an algorithm that always takes the best immediate, or local, move while finding an answer, without considering the possible impact of the immediate decisions on later ones.

The greedy algorithm for the minimum spanning tree problem described above is called the Jarnik-Prim algorithm, from the work of Jarník (1930) and Prim (1957). It is formally defined as Algorithm 21.4.

Algorithm 21.4: Jarnik-Prim algorithm for minimum spanning tree

Objective
  Calculate a minimum spanning tree.
Input
  A network $(\mathcal{N}, \mathcal{E})$ of m nodes and n edges.
  A vector $c \in \mathbb{R}^n$ with the cost of each edge.
Output
  The list $\mathcal{T}^*$ of edges belonging to the minimum spanning tree.
Initialization
  $\mathcal{M} = \{i\}$ where i is any node.
  $\mathcal{T}^* = \emptyset$.
Repeat
  $c := +\infty$.
  for $(i,j) \in \Gamma(\mathcal{M}, \mathcal{N}\setminus\mathcal{M})$ **do**
    if $c_{ij} < c$ **then**
      $c := c_{ij}$
      $(e_i, e_j) := (i,j)$
  $\mathcal{T}^* := \mathcal{T}^* \cup (e_i, e_j)$.
  $\mathcal{M} := \mathcal{M} \cup e_j$.
Until $\mathcal{M} = \mathcal{N}$.

The iterations of Algorithm 21.4 applied on Example 21.20 are reported in Table 21.2.

Table 21.2: Iterations of Algorithm 21.4 on Example 21.20

$\mathcal{M}$	$\mathcal{T}^*$	(i,j)	c_{ij}
1	$\emptyset$	(1,2)	6
1,2	(1,2)	(2,3)	2
1,2,3	(1,2),(2,3)	(3,5)	3
1,2,3,5	(1,2),(2,3),(3,5)	(4,5)	1
1,2,3,4,5	(1,2),(2,3),(3,5), (4,5)	(4,6)	5
1,2,3,4,5,6	(1,2),(2,3),(3,5), (4,5), (4,6)	(4,7)	5
1,2,3,4,5,6,7	(1,2),(2,3),(3,5), (4,5), (4,6), (4,7)	—	—

The Jarnik-Prim algorithm for the minimum spanning tree problem is an example where a greedy algorithm provides an optimal solution of a discrete optimization problem, as shown by Theorem 21.21. For other problems, greedy algorithms may not necessarily provide an optimal solution. Still, due to their simplicity, they can also be used as heuristics, as described in Section 27.1.

21.8 Exercises

Exercise 21.1. Consider the network represented in Figure 21.23, where the number associated with each arc represents the amount of flow traversing it.

1. What is the indegree, the outdegree, and the degree of each node?
2. Give the adjacency matrix of the network.
3. Represent the network using an adjacency list that also stores the flows.
4. Is the network connected?
5. Is the network strongly connected?
6. Enumerate all simple paths from node a to node g.
7. Enumerate all simple forward paths from node a to node g.
8. Give the divergence of the flow vector at each node. What are the supply nodes? What are the demand nodes?
9. Consider the cut $\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$, defined by the set $\mathcal{M} = \{a, b, c\}$.
   (a) What are the forward arcs of the cut?
   (b) What are the backward arcs of the cut?
   (c) What is the flow through the cut? Check that Theorem 21.14 is verified.
   (d) Assume that the capacities on each arc are -3 for the lower bound and 5 for the upper bound. What is the capacity of the cut?
10. Decompose the flow vector into consistent simple path/cycle flows.

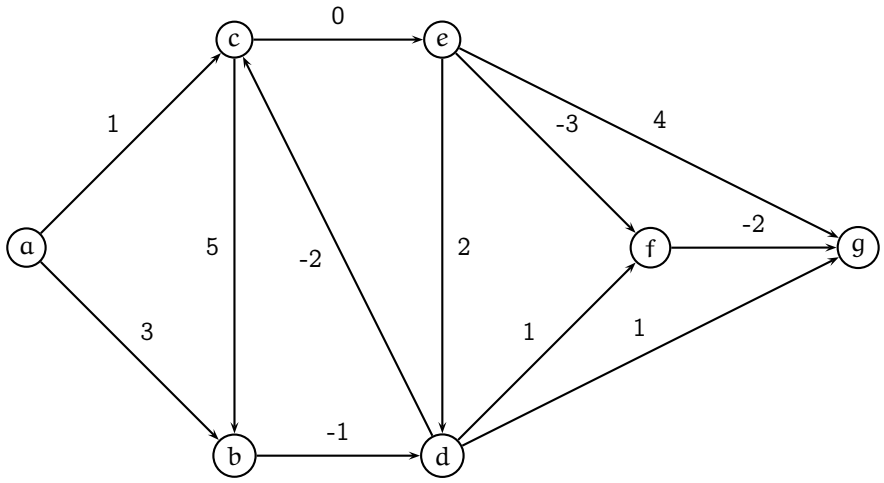


Figure 21.23: Network with a flow vector for Exercise 21.1

Exercise 21.2. Consider the network represented in Figure 21.24, where the number associated with each arc represents the amount of flow traversing it. Answer the same questions as Exercise 21.1.

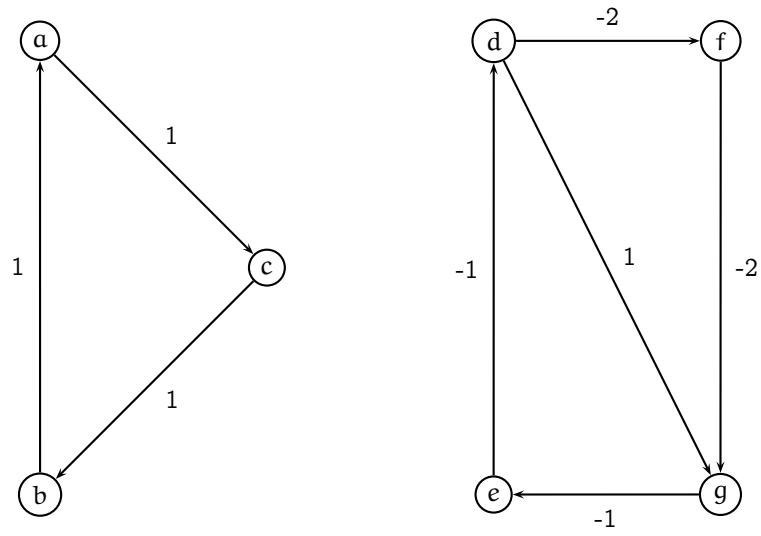


Figure 21.24: Network with a flow vector for Exercise 21.2

Exercise 21.3. Consider the network represented in Figure 21.25, where each arc (i, j) is associated with its lower bound ℓ_{ij}, its flow x_{ij} and its upper bound u_{ij} in the following way: $(\ell_{ij}, x_{ij}, u_{ij})$. Identify at least 4 cuts $\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$ separating o from d, such that $\Gamma^{\rightarrow}$ contains exactly 4 arcs. For each of them, give the flow through the cut and the capacity of the cut.

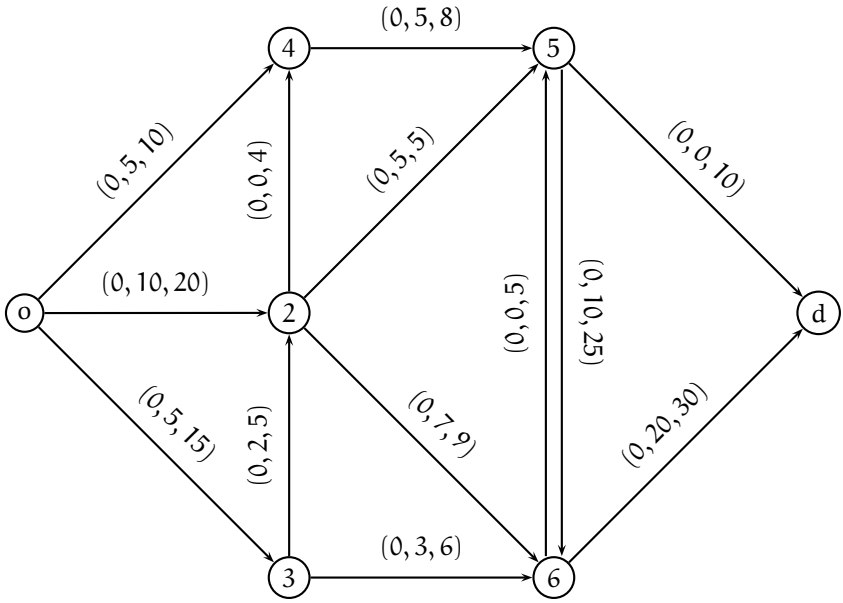


Figure 21.25: Network for Exercises 21.3, 22.5, and 24.2, where each arc (i, j) is associated with $(\ell_{ij}, x_{ij}, u_{ij})$, that is, lower bound, flow, and upper bound.

Exercise 21.4. Determine the minimum spanning tree for the network represented in Figure 21.26, where the value associated with each edge is its cost. Apply Algorithm 21.4 starting with $\mathcal{M} = \{a\}$.

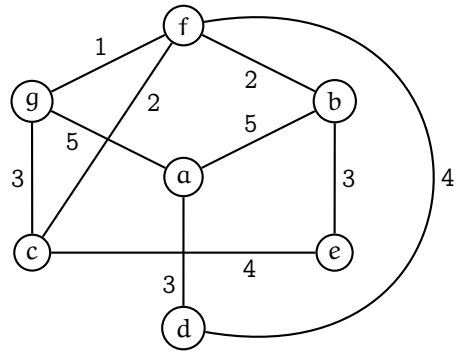


Figure 21.26: Network for Exercise 21.4, with cost associated with each edge

Exercise 21.5. A travel agent organizes hiking routes in the Alps for families. For each possible origin/destination pair, he wants to identify an itinerary that avoids high altitudes as much as possible. The network represented in Figure 21.27 represents various locations that serve as the origin or destination of the routes. Each edge represents a hiking trail between two of these locations. The value associated with each edge represents the highest altitude along the trail. Solve the problem for the travel agent.

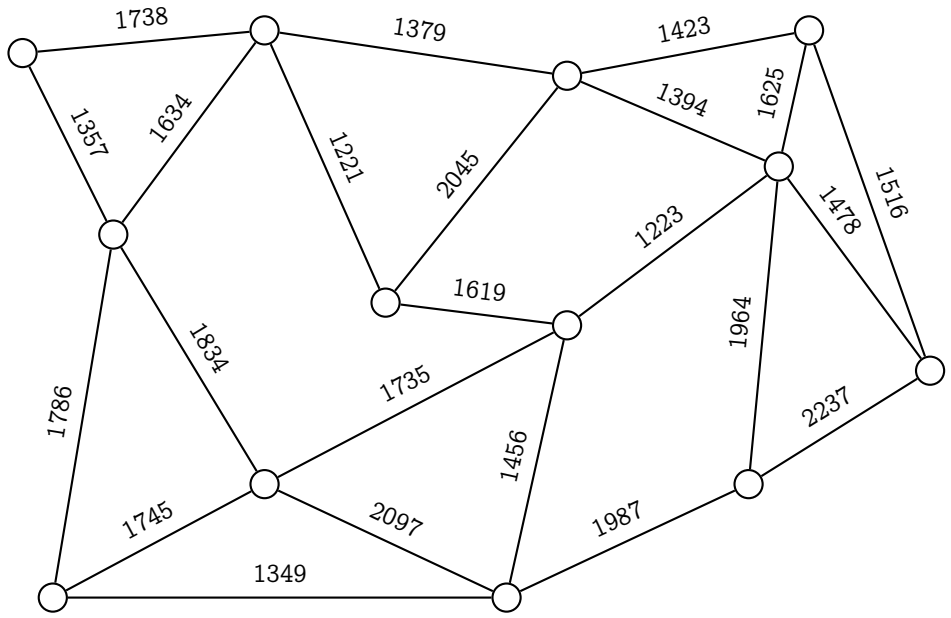


Figure 21.27: Network of hiking trails in the Alps. The value associated with each edge is the maximum altitude along the trail.

Chapter 22

The transhipment problem

Contents

As discussed in Chapter 21, networks are usually designed to transport flow. As transporting flow has a cost, we are looking at the least expensive way to transport the flow from the places where it is produced to the places where it is consumed. This optimization problem is called the *transhipment* problem or the *minimum cost flow* problem.

22.1 Formulation

We assume that we have a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs. The set of nodes is partitioned into three subsets:

- a set $\mathcal{N}^s$ of supply nodes representing the places where the flow is produced,
- a set $\mathcal{N}^d$ of demand nodes representing the places where the flow is consumed,
- a set $\mathcal{N}^t$ of transhipment nodes where the flow may just be transiting.

We also have access to the following data:

- a vector $s \in \mathbb{R}^m$ representing the supply/demand of flow, such that
  - $s_i > 0$ for each $i \in \mathcal{N}^s$ represents the quantity of flow produced at supply node i,

  - $s_i < 0$ for each $i \in \mathcal{N}^d$ is such that $-s_i$ represents the quantity of flow consumed at demand node i,
  - $s_i = 0$ for each $i \in \mathcal{N}^t$,
- a vector $c \in \mathbb{R}^n$ representing the cost of transporting a unit of flow on each arc,
- a vector $\ell \in \mathbb{R}^n$ representing the lower capacity of each arc,
- a vector $u \in \mathbb{R}^n$ representing the upper capacity of each arc.

Note that we assume that $\sum_{i=1}^{m} s_i = 0$, and $\ell_{ij} \leq u_{ij}$, for each $(i,j) \in \mathcal{A}$. Otherwise, there is no feasible flow vector verifying the constraints. Note also that these conditions are not sufficient to guarantee feasibility.

As described in Section 1.1, in order to obtain an optimization problem, we need to define the decision variables, the objective function, and the constraints.

Decision variables The decision variables are the flow on each arc, denoted by $x \in \mathbb{R}^n$. From Theorem 21.17, it is always possible to decompose the flow vector into simple path flows (possibly cycle flows), so that concrete instructions can be derived from the flow vector to transport the flow from the production sites to the consumption sites.

Objective function The objective is to minimize the total cost, that is

$$\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}.$$

Note that it is a linear function in the decision variables.

Constraints Two types of constraints have to be verified. First, the flow vector must be consistent with the given demand and supply. It means that the divergence on each node must correspond to the value of s_i, that is

$$\operatorname{div}(x)_i = s_i \ \forall i \in \mathcal{N},$$

or, from (21.11),

$$\sum_{j|(i,j) \in \mathcal{A}} x_{ij} - \sum_{k|(k,i) \in \mathcal{A}} x_{ki} = s_i, \ \forall i \in \mathcal{N}.$$

Second, the value of the flow on each arc must verify the capacity constraints, that is

$$\ell_{ij} \leq x_{ij} \leq u_{ij}, \ \forall (i,j) \in \mathcal{A}.$$

Therefore, we obtain the following optimization problem:

$$\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \tag{22.1}$$

subject to

$$\sum_{j|(i,j) \in \mathcal{A}} x_{ij} - \sum_{k|(k,i) \in \mathcal{A}} x_{ki} = s_i, \ \forall i \in \mathcal{N}, \tag{22.2}$$

$$\ell_{ij} \leq x_{ij} \leq u_{ij}, \ \forall (i,j) \in \mathcal{A}. \tag{22.3}$$

It is a linear optimization problem and can be solved using the simplex method described in Chapter 16. It is therefore appropriate to transform it into a linear problem in standard form (6.159)–(6.160). To do this, we apply the transformation techniques described in Section 1.2. First, in order to set the lower bounds to 0, we define new variables

$$x'_{ij} = x_{ij} - \ell_{ij}. \tag{22.4}$$

We therefore obtain the following formulation

$$\min_{x \in \mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij} x'_{ij} + \sum_{(i,j)\in\mathcal{A}} c_{ij}\ell_{ij}$$

subject to

$$\sum_{j|(i,j)\in\mathcal{A}} x'_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x'_{ki} = s_i + \sum_{k|(k,i)\in\mathcal{A}} \ell_{ki} - \sum_{j|(i,j)\in\mathcal{A}} \ell_{ij}, \ \forall i \in \mathcal{C},$$

and

$$0 \le x'_{ij} \le u_{ij} - \ell_{ij}, \ \forall (i,j) \in \mathcal{A}.$$

As $\sum_{(i,j)\in\mathcal{A}} c_{ij}\ell_{ij}$ is a constant, it can be omitted in the objective function. Defining

$$u'_{ij} = u_{ij} - \ell_{ij} \qquad \forall (i,j) \in \mathcal{A} \tag{22.5}$$

$$s'_i = s_i + \sum_{k|(k,i)\in\mathcal{A}} \ell_{ki} - \sum_{j|(i,j)\in\mathcal{A}} \ell_{ij} \qquad \forall i \in \mathcal{C}, \tag{22.6}$$

we obtain

$$\min_{x' \in \mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij} x'_{ij}$$

subject to

$$\sum_{j|(i,j)\in\mathcal{A}} x'_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x'_{ki} = s'_i, \ \forall i \in \mathcal{C},$$

and

$$0 \le x'_{ij} \le u'_{ij}, \ \forall (i,j) \in \mathcal{A}.$$

Next, we transform the upper bound constraints into equality constraints. For each arc (i,j) we include a slack variable y_{ij} (Definition 1.4) and the problem is written as

$$\min_{x' \in \mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij} x'_{ij} \tag{22.7}$$

subject to

$$\sum_{j|(i,j)\in\mathcal{A}} x'_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x'_{ki} = s'_i, \ \forall i \in \mathcal{N}, \tag{22.8}$$

$$x'_{ij} + y_{ij} = u'_{ij}, \ \forall (i,j) \in \mathcal{A}, \tag{22.9}$$

and

$$x'_{ij} \ge 0, \quad y_{ij} \ge 0. \tag{22.10}$$

The constraint (22.9) can actually be interpreted as a supply/demand constraint and included in the set of constraints (22.8). We illustrate this with Examples 22.1 and 22.2.

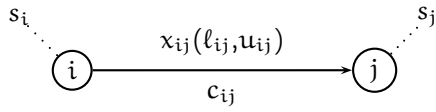
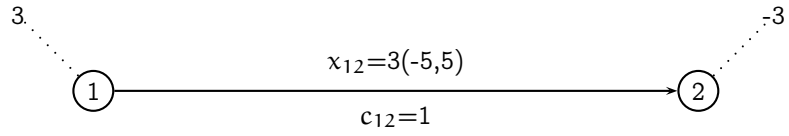
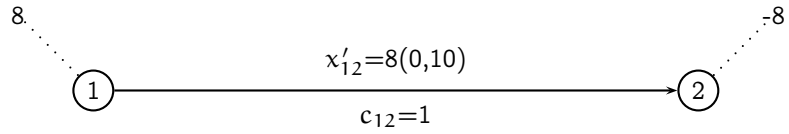


Figure 22.1: Convention for Example 22.1

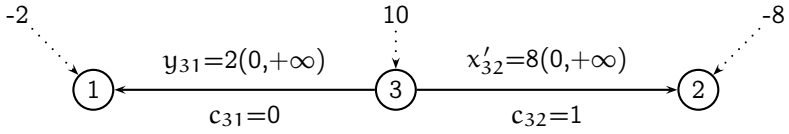
Example 22.1 (Transhipment problem in standard form – I)**.** Consider the simple example represented in Figure 22.2(a), where we adopt the convention depicted in Figure 22.1, and we report the flow and its bounds on the top of the arc, the cost on the bottom, and the supply of each node is represented by a dotted line. It is a network with two nodes and one arc. Node 1 supplies 3 units of flows that are consumed at node 2 (equivalently, node 2 supplies -3 units of flow). Arc $(1,2)$ has lower capacity -5, upper capacity 5, and cost 1. It transports 3 units of flow from node 1 to node 2. In order to set the lower bound to 0, we perform the change of variable (22.4), that is $x'_{12} = x_{12} - (-5) = 3 + 5 = 8$. From (22.5), the upper bound becomes now $5 - (-5) = 10$. From (22.6), we obtain that $s'_1 = 3 - (-5) = 8$ and $s'_2 = -3 - 5 = -8$, as illustrated in Figure 22.2(b). Figure 22.2(c) represents the modification to the network that accounts for the slack variable.



(a) Original formulation



(b) Change of variable to obtain zero lower bounds



(c) Slack variable to remove the upper bounds

Figure 22.2: Transforming a simple transhipment problem into standard form

Indeed, a new variable means a new arc transporting the corresponding flow. The arc $(1,2)$ is replaced by a node (node 3) and two arcs. The supply of the new node corresponds to the upper bound on the flow of the original arc. The new arc $(3,2)$ takes the role of the original arc and transports the flow from the original problem (here, 8) at the same cost. By design, as the supply of node 3 is equal to the upper bound, the flow on arc $(3,2)$ never exceeds that value. If it happens to transport less, the excess flow (that is, the slack) is transported by the new arc $(3,1)$, at zero cost and exits the network at node 1. Therefore, the supply of node 1 must be decreased by the amount of extra flow that has been injected into the network. In this example, as the upper bound is 10, the supply of node 3 is 10, and the new supply of node 1 is $8 - 10 = -2$.

Example 22.2 (Transhipment problem in standard form – II)**.** Consider the network represented in Figure 21.2, with the data presented in Table 22.1. The left part of the table contains the lower bound, the upper bound, the cost of each arc, and a feasible flow vector. The right part contains the supply for each node. Note that the values of the supply sum up to 0 and the bounds are compatible, that is, the lower bound is lower than the upper bound. The network representation of the transformed problem (22.11)–(22.13) is depicted in Figure 22.3 and the corresponding data in Table 22.2.

Table 22.1: Data for Example 22.2

	Arcs				Nodes	
(i,j)	ℓ_{ij}	u_{ij}	c_{ij}	x_{ij}	i	s_i
$(1,2)$	-1.1	2.5	1	2.2	1	-1.7
$(2,3)$	-2.2	2.5	1	-2.2	2	-8.4
$(2,4)$	-3.3	3.5	1	-3.3	3	5.0
$(3,1)$	-4.5	4.5	1	3.9	4	5.1
$(3,5)$	-5.5	5.5	1	-1.1	5	0.0
$(4,2)$	-6.6	6.5	1	0.7	6	0.5
$(4,5)$	-7.7	7.5	1	-7.7	7	-0.5
$(5,4)$	-8.8	8.5	1	-8.8	8	0.0
$(6,7)$	-9.9	9.5	1	-9.5		
$(7,6)$	-10.0	10.5	1	-10.0		

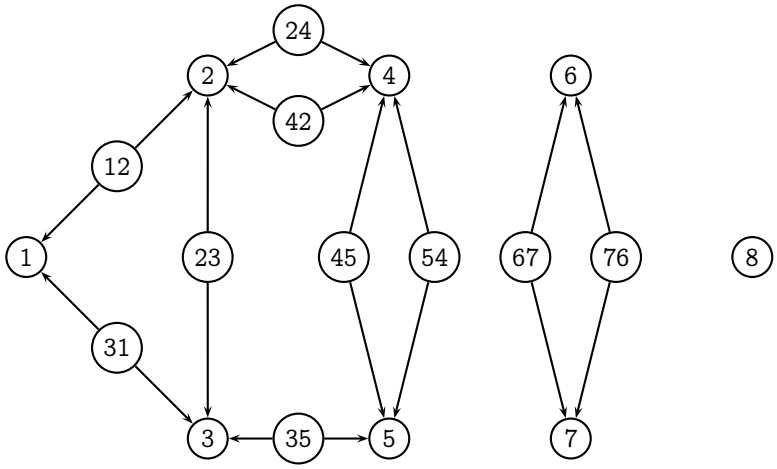


Figure 22.3: Transformed network for Example 22.2

Table 22.2: Data for the transformed network of Example 22.2

(i,j)	c_{ij}	x_{ij}	i	s_i
$(12,2)$	1	3.3	1	−8.7
$(12,1)$	0	0.0	2	−22.1
$(23,3)$	1	0.0	3	−7.2
$(23,2)$	0	8.4	4	−21.0
$(24,4)$	1	4.4	5	−21.7
$(24,2)$	0	7.3	6	−19.0
$(31,1)$	1	0.0	7	−20.9
$(31,3)$	0	0.0	8	0.0
$(35,5)$	1	0.4	12	3.6
$(35,3)$	0	0.0	23	4.7
$(42,2)$	1	0.3	24	6.8
$(42,4)$	0	4.7	31	9.0
$(45,5)$	1	6.8	35	11.0
$(45,4)$	0	0.6	42	13.1
$(54,4)$	1	6.6	45	15.2
$(54,5)$	0	5.8	54	17.3
$(67,7)$	1	15.2	67	19.4
$(67,6)$	0	17.3	76	20.5
$(76,6)$	1	19.0		
$(76,7)$	0	20.5		

We can assume without loss of generality that the transhipment problem is given in standard form, that is

$$\min_{x\in\mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \tag{22.11}$$

subject to

$$\sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki} = s_i,\ \forall i \in \mathcal{N}, \tag{22.12}$$

and

$$x_{ij} \geq 0,\ \forall (i,j) \in \mathcal{A}. \tag{22.13}$$

In matrix form, we have

$$\min_{x\in\mathbb{R}^n} c^T x \tag{22.14}$$

subject to

$$Ax = s \tag{22.15}$$

$$x \geq 0, \tag{22.16}$$

where $A \in \mathbb{R}^{m\times n}$ is the incidence matrix of the network. Its columns correspond to the arcs and the rows to the nodes of the network. The column corresponding to arc (i,j) contains only 0, except for the entry corresponding to node i that contains 1, and the entry corresponding to node j that contains -1.

Note that these transformations of the problem into a standard form are exactly the same as the transformations described in Section 1.2. We have simply given a concrete interpretation in the network context.

22.2 Optimality conditions

As discussed in Part II, optimality conditions provide a theoretical analysis of the optimization problem that is an important starting point for the design of algorithms. They characterize the optimal solution, and therefore provide a stopping criterion for the iterative methods. We investigate these conditions in the specific case of the transhipment problem.

The Karush-Kuhn-Tucker conditions (see Theorem 6.13) are significantly simpler for the transhipment problem (22.11)–(22.13). The Lagrangian is written as

$$L(x,\lambda,\mu) = \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \lambda_i \left(\sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki} - s_i \right) - \sum_{(i,j)\in\mathcal{A}} \mu_{ij}x_{ij}, \tag{22.17}$$

where $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^n$.

The derivative with respect to the flow variable x_{ij} is

$$\frac{\partial L}{\partial x_{ij}} = c_{ij} + \lambda_i - \lambda_j - \mu_{ij}. \tag{22.18}$$

Therefore, the necessary optimality conditions (6.55) and (6.56) write

$$c_{ij} + \lambda_i - \lambda_j \geq 0, \ \forall (i,j) \in \mathcal{A}. \tag{22.19}$$

Moreover, from condition (6.57), for each arc transporting flow, we have $\mu_{ij} = 0$ and, therefore,

$$c_{ij} + \lambda_i - \lambda_j = 0, \ \forall (i,j) \text{ such that } x_{ij} > 0. \tag{22.20}$$

Conditions (22.19) and (22.20) are the complementarity slackness conditions presented in Theorem 6.34. They are therefore sufficient and necessary optimality conditions for the transhipment problem. There is a dual variable λ_i associated with each node i. When the complementarity slackness conditions are verified, λ is also an optimal solution of the dual problem. Note that only differences of the dual variables are involved in these conditions. Therefore, if $\lambda \in \mathbb{R}^m$ verifies conditions (22.19) and (22.20), so does any vector such that all values of λ are shifted by a quantity σ, that is $\lambda + \sigma e$, where $e \in \mathbb{R}^m$ is a vector composed only of 1, for any $\sigma \in \mathbb{R}$.

22.3 Total unimodularity

As described in Section 22.1, the incidence matrix A involved in the constraints of the transhipment problem has a special structure. It has as many columns as arcs in the network and as many rows as nodes. For each arc (i,j), there are exactly two entries in the corresponding column: 1 at row i and -1 at row j. Therefore, the sum of all rows of the matrix is 0. It has an interesting property when the vector of supply/demand is integer.

Remember that the optimal solution of the linear optimization problem is a feasible basic solution (see Definition 3.38). It means that it has the form $x^* = (x_B^* \quad x_N^*)$, where $x_N^* = 0$ and

$$x_B^* = B^{-1} s \tag{22.21}$$

where B is a square invertible matrix. From Cramer's rule (C.26), we have

$$x_B^* = \frac{1}{\det(B)} C(B)^T s, \tag{22.22}$$

where $C(B)$ is the cofactor matrix of B (see Definition B.12). As each entry of $C(B)$ is a determinant of a matrix containing only 0, 1, and -1, they are all integers. Therefore, if the supply vector s is integer, the vector $C(B)^T s$ is also integer. Now, if the determinant of B happens to be either 1 or -1 (it cannot be 0, as B is invertible), we obtain the nice property that x_B^* (and, consequently, x^*) is integer. In this case, B is said to be a unimodular matrix (see Definition B.14).

It is particularly valuable to obtain integer solutions without including explicit integrality constraints in the optimization problem. Indeed, as discussed during the presentation of the example in Section 1.1.7, constraining the variables to have only integer values dramatically complicates the optimization problem, and methods such as the simplex algorithm cannot be used anymore. Therefore, the property described

above is particularly important, as it allows us to handle these constraints implicitly and not explicitly. Indeed, it suffices to verify that the property applies to the problem at hand, and to make sure that the data is integer to obtain an integer optimal solution from the simplex algorithm.

Definition 22.3 (Total unimodularity)**.** The matrix $A \in \mathbb{Z}^{m\times n}$ is totally unimodular if the determinant of each square submatrix of A is 0, -1 or $+1$. In particular, every entry of the matrix is 0, -1 or $+1$.

If the matrix A of the constraint is totally unimodular, each basis B has determinant -1 or 1. Indeed, as it is non singular, the determinant cannot be 0. Therefore, using the argument discussed above, any feasible basic solution (hence, any vertex of the constraint polyhedron) is integer, including the optimal basic solutions.

Theorem 22.4 (Integrality of the basic solutions)**.** *Consider the polyhedron represented in standard form* $\{x \in \mathbb{R}^n | Ax = b,\ x \geq 0\}$*, where* $A \in \mathbb{Z}^{m\times n}$ *and* $b \in \mathbb{Z}^m$*. If* A *is totally unimodular, then every basic solution is integer.*

Proof. According to Definition 3.38, every basic solution is decomposed into $x_N = 0$ and

$$x_B = B^{-1}b,$$

where the matrix B composed of m columns of the matrix A is non singular. Therefore $\det(B) \neq 0$. We use Cramer's rule to write

$$x_B = \frac{1}{\det(B)} C(B)^T b, \tag{22.23}$$

where $C(B)$ is the cofactor matrix of B, and is integer. As A is totally unimodular, $\det(B) = \pm 1$. As b is integer, so is x_B. □

Example 22.5 (Totally unimodular matrices)**.** Consider the incidence matrix of the network represented in Figure 22.2(c):

$$A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{pmatrix}.$$

Each square submatrix of size 1, that is each element of the matrix, is 0, -1 or $+1$. Each square submatrix of size 2 is unimodular:

$$\det\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = 1, \quad \det\begin{pmatrix} -1 & 0 \\ 1 & 1 \end{pmatrix} = -1, \quad \det\begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix} = 1.$$

Therefore, A is totally unimodular. Note that it is not sufficient to have entries 0, -1 or $+1$ to be totally unimodular. For example, the matrix

$$\begin{pmatrix} 1 & -1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix}$$

is not totally unimodular, as

$$\det \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = 2.$$

Theorem 22.6 (Total unimodularity of the incidence matrix)**.** *Let $(\mathcal{N}, \mathcal{A})$ be a network with m nodes and n arcs. Let $A \in \{-1, 0, 1\}^{m \times n}$ be the incidence matrix of the network, where each entry a_{ik} is defined as*

$$a_{ik} = \begin{cases} 1 & \text{if } i \text{ is the upstream node of arc } k, \\ -1 & \text{if } i \text{ is the downstream node of arc } k, \\ 0 & \text{otherwise.} \end{cases} \tag{22.24}$$

Then the matrix A is totally unimodular.

Proof. Assume by contradiction that A is not totally unimodular. There are, therefore, submatrices of A such that their determinant is not 0, 1, or -1. Among them, consider one submatrix with minimum size k and call it B, with $\det(B) \notin \{-1, 0, 1\}$.

As A has at most two non zero entries in each column, each column of B can have 0, 1, or 2 non zero entries.

Assume first that one column of B contains only 0. If it were the case, B would be singular, and its determinant would be 0, which is not possible. Consequently, each column of B contains at least one non zero entry.

Assume that one column of B contains exactly one non zero entry. Without loss of generality, assume this entry to be b_{11}. It means that B is of the form

$$B = \begin{pmatrix} b_{11} & b_{12} \cdots b_{1k} \\ 0 & \\ \vdots & B' \\ 0 & \end{pmatrix}.$$

Therefore,

$$\det(B) = b_{11} \det(B').$$

As b_{11} is either 1 or -1, then $\det(B')$ is the same as $\det(B)$, up to the sign. In particular, $\det(B')$ is not 0, 1, or -1. As B' is strictly smaller than B, this is not possible as B is the smallest submatrix with such a property. Consequently, each column of B contains exactly two non zero entries. As they are subvectors of the column of A, one of these entries is 1 and the other is -1.

Therefore, if we sum up all rows of the matrix B, we obtain 0, and the rows are linearly dependent. It means that B is singular and $\det(B) = 0$, which is not possible.

This contradicts the fact that A is not totally unimodular, and proves the result. □

Corollary 22.7 (Integer optimal solution of the transhipment problem)**.** *Consider the transhipment problem (22.11)–(22.13). If the supply vector* s *is integer, that is, if* $s \in \mathbb{Z}^m$*, and if the problem is bounded, then it has an optimal solution which is integer.*

Proof. Direct consequence of Theorems 22.4 and 22.6. □

The concept of totally unimodular matrices goes beyond the transhipment problem, and plays an important role in discrete optimization. We refer the reader to Nemhauser and Wolsey (1988), Wolsey (1998), or Bertsimas and Weismantel (2005) for a more comprehensive description of the topic.

22.4 Modeling

The transhipment model embeds a variety of other network problems. In general, these problems are associated with a dedicated algorithm that solves them more efficiently than the simplex method applied to the transhipment formulation. However, as they are instances of the transhipment problem, they also have its properties. In particular, the integrality of an optimal solution is guaranteed if the supply/demand is integer. In this section, we formulate these problems and show why they are transhipment problems. Two of them are treated in greater details in later chapters, where specific algorithms are presented.

22.4.1 The shortest path problem

The pervasiveness of GPS navigation systems and of online map services allows everybody to compute the fastest or the shortest itinerary between two points from a computer or a navigation device (see Figure 22.4). The problem of finding such itineraries is usually referred to as the *shortest path problem* and can be defined in different ways. The classical definition is as follows.

Definition 22.8 (Shortest path problem: single origin-single destination)**.** Consider a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs, and a vector $c \in \mathbb{R}^n$ representing the cost of traversing each arc. Consider a node o called the origin and a node d called the destination. The shortest path problem consists in finding a simple forward path with origin o and destination d, and with the smallest cost.

From an algorithmic point of view, it is convenient to solve the problem for all destinations at once.

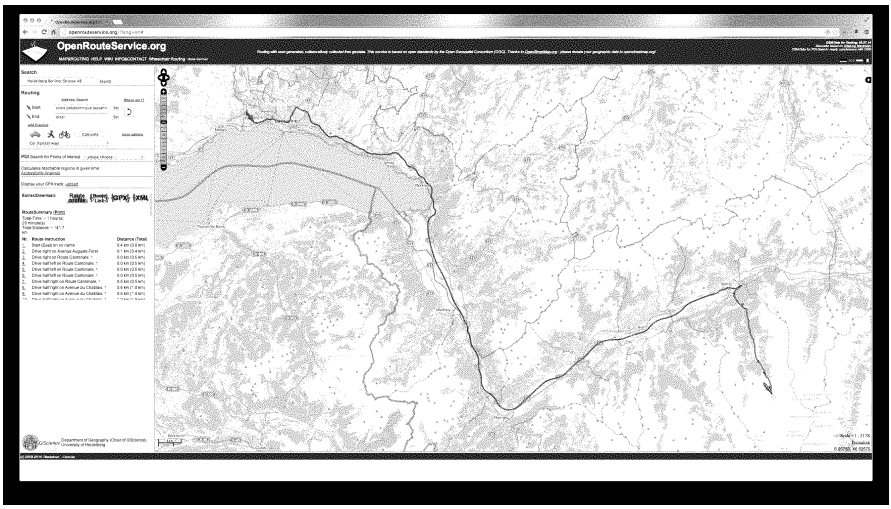


Figure 22.4: The fastest path between EPFL and Zinal computed by `OpenRouteService.org`

Definition 22.9 (Shortest path problem: single origin-multiple destinations)**.** Consider a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs, and a vector $c \in \mathbb{R}^n$ representing the cost of traversing each arc. Consider a node o called the origin. The shortest path problem consists in finding for each node $i \neq o$ in the network a simple forward path with origin o and destination i, and with the smallest cost.

The definition 21.17 of path cost simplifies here. Indeed, we are looking only at forward paths, and the set $P^{\leftarrow}$ is empty. Moreover, we assume that only one unit of flow is following the path, so that the cost of path P is

$$C(P) = \sum_{(i,j)\in P} c_{ij}. \tag{22.25}$$

As discussed in Section 21.5.4, the concept of cost is relatively general. Even if the name of the problem refers to the "shortest" path, the cost does not need to be the physical length of the arc. In the example of the online tools for itineraries, the cost can be the travel time to traverse each arc. In this case, the solution of the shortest path problem is actually the fastest path between o and d. As discussed in Section 21.3, the only requirement is that the cost of a path is the sum of the cost of its arcs.

The single origin-single destination shortest path problem is a transhipment problem where

- the cost on each arc is c_{ij},
- the supply for the origin is 1, that is $s_o = 1$,

- the demand for the destination is 1, that is $s_d = -1$,
- the supply for any other node is 0,
- the lower bound on each arc is 0,
- there is no upper bound.

The transhipment problem (22.11)–(22.13) becomes

$$\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \tag{22.26}$$

subject to

$$\begin{aligned}
\sum_{j|(o,j)\in\mathcal{A}} x_{oj} - \sum_{k|(k,o)\in\mathcal{A}} x_{ko} &= 1, \\
\sum_{j|(d,j)\in\mathcal{A}} x_{dj} - \sum_{k|(k,d)\in\mathcal{A}} x_{kd} &= -1, \\
\sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki} &= 0, \ \forall i \in \mathcal{N}, i \neq o, i \neq d, \\
x_{ij} &\geq 0, \ \forall (i,j) \in \mathcal{A}.
\end{aligned}$$

The properties of the shortest path problems, as well as specific algorithms to solve it, are discussed in Chapter 23.

22.4.2 The maximum flow problem

The maximum flow problem consists in pushing as much flow as possible through a network with given capacities on the arcs. It was first motivated by the analysis performed by the American army of the railway network operated by the Soviet Union across Eastern Europe during the cold war (Harris and Ross, 1955, Schrijver, 2002).

Definition 22.10 (Maximum flow problem)**.** Consider a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs, and a vector $u \in \mathbb{R}^n$ representing the capacity of each arc. Consider a node o called the origin (or the source), and a node d called the destination (or the sink). The maximum flow problem consists in identifying a feasible flow vector that transports as much flow as possible from o to d. More formally, it seeks a flow vector x such that

- $\operatorname{div}(x)_i = 0$ for all $i \neq o$, $i \neq d$,
- $\operatorname{div}(x)_o = -\operatorname{div}(x)_d$ is maximized.

Consider the network represented in Figure 22.5, where the value of the upper capacity of each arc is shown next to it. It may represent a railway network, with the maximum number of trains per hour that can proceed between two cities. Or it may represent a network of pipelines, where the capacity is the maximum number of megaliters of oil that the pipe can transport per hour.

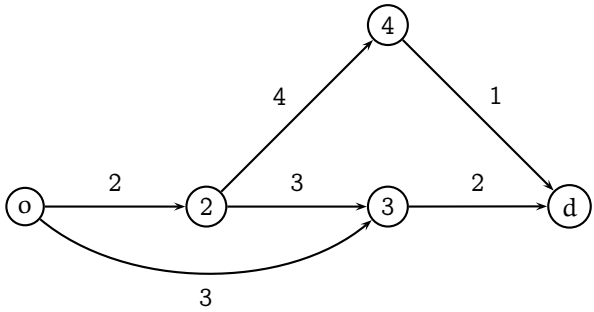


Figure 22.5: An example of a maximum flow problem. The value on each arc is its capacity.

In this simple example, there are only 3 paths that can be used to transport the flow:

- path 1: $o \to 2 \to 3 \to d$,
- path 2: $o \to 3 \to d$,
- path 3: $o \to 2 \to 4 \to d$.

Path 1 cannot transport more than 2 units of flow, which is the capacity of its first and last arcs. If we send 2 units along path 1, path 2 cannot be used. Indeed, path 2 includes arc $(3, d)$, which cannot accommodate more than the 2 units already transported along path 1. Similarly, path 3 cannot be used, as it includes arc $(o, 2)$, which is also at capacity. This strategy transports 2 units of flow from node o to node d. It is possible to do better with the following reasoning. Path 2 cannot transport more than 2 units of flow, which is the capacity of its last arc. If we send 2 units of flow along path 2, path 1 cannot be used, as they share arc $(3, d)$, which is at capacity. But no arc in path 3 has been used. We can send a maximum of 1 unit along path 3, which is the capacity of its last arc. With this strategy, a total of 3 units are sent from o to d. The associated flow vector is represented in Figure 22.6, where the flow is shown next to each arc, together with the arc capacity (in square brackets).

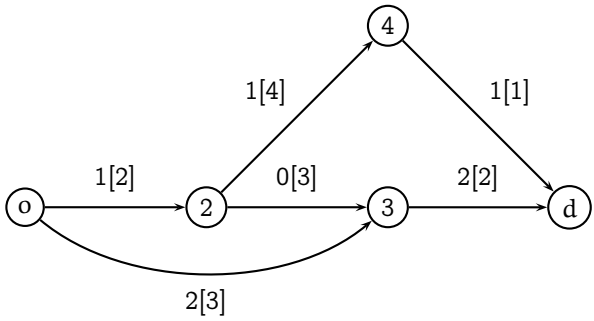


Figure 22.6: A solution of a maximum flow problem. The values on each arc are the flows x and the capacities u in format $x[u]$.

It happens to be the maximum possible. Indeed, all arcs arriving at node d are full and, whatever is possible upstream, no more flow can arrive there. Clearly, such enumerations cannot be done on real networks. Specialized algorithms are described in Chapter 24.

The maximum flow problem can be modeled as a transhipment problem. There is no cost associated with the arcs, and the quantity that is optimized is the divergence of a node. The idea is to include in the network an artificial arc that takes the role of a counter. This arc, connecting d to o, sends any unit of flow reaching the destination back to the origin, with a cost of -1. This creates a circulation (the divergence of each node is zero). As the real arcs have all zero costs, the total cost represents the number of units of flow that are able to reach d from o through the "real" network (with the opposite sign, as we need to maximize). This is illustrated in Figure 22.7, where the cost on each arc and the upper capacity are shown.

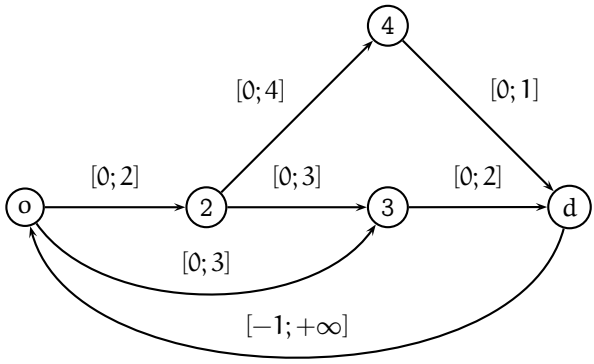


Figure 22.7: Modeling a maximum flow problem as a transhipment problem. The values on each arc are the cost c and the upper bound u in format $[c; u]$.

The maximum flow problem is therefore a transhipment problem where

- the cost on the artificial arc is -1,
- the cost on every other arc is 0,
- the supply/demand for each node is 0,
- the lower bound on each arc is 0,
- there is no upper bound on the artificial arc,
- the upper bound on every other arc is given by the problem definition.

The optimization problem can then be written as follows:

$$\min_{x \in \mathbb{R}^{n+1}} -x_{do} \tag{22.27}$$

subject to

$$\sum_{j|(i,j)\in\mathcal{A}\cup(d,o)} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}\cup(d,o)} x_{ki} = 0, \ \forall i \in \mathcal{N}, \tag{22.28}$$

$$x_{ij} \leq u_{ij}, \ \forall (i,j) \in \mathcal{A}, \tag{22.29}$$

$$x_{ij} \geq 0, \ \forall (i,j) \in \mathcal{A}, \tag{22.30}$$

$$x_{do} \geq 0, \tag{22.31}$$

where the arc (d,o) is the artificial arc added to create a circulation. In the example presented in Figure 22.6, this arc would transport 3 units of flow.

Note that the maximum flow problem can be used to model a wide variety of problems. An interesting example is the problem of locating n queens on a chessboard so that they are not attacking each other (Gardner and Nicolio, 2008).

22.4.3 The transportation problem

The transportation problem is a special case of the transhipment problem where the flow generated at the origin is directly sent to the destination, without transhipment.

Definition 22.11 (Transportation problem)**.** Consider m_o suppliers and m_d customers. Supplier i produces s_i units of flow, $i = 1, \ldots, m_o$, and customer j consumes t_j units of flow, $j = 1, \ldots, m_d$. Each supplier is associated with a list of customers that can be served. It costs c_{ij} to transport the flow from supplier i to customer j. The transportation problem consists in deciding how much flow each supplier must send to each customer to satisfy the demand at minimum cost.

Note that a necessary condition for the problem to be feasible is that the total supply $\sum_i s_i$ must equal the total demand $\sum_j t_j$. As it appears from the definition, this problem is not directly related to a physical network. The next example is about the distribution of electricity.

Example 22.12 (Provision of electricity in Switzerland)**.** Consider an electricity company which needs to serve 4 cities (Zürich, Geneva, Lausanne and Bern) using 3 nuclear plants: Mühleberg, producing 3,110 gigawatt-hours (GWh) per year, Beznau, with 3,198 GWh and Leibstadt, producing 10,205 GWh. The cities consume 8,961, 3,777, 2,517, and 1,258 GWh per year, respectively. Using an arbitrary unit, the cost of transporting 1 GWh from a given plant to a given city is as follows:

	Zürich	Geneva	Lausanne	Bern
Mühleberg	18	6	10	9
Beznau	9	16	13	7
Leibstadt	14	9	16	5

Note that, in this example, each supplier can potentially serve each client.

The optimal solution, for a total cost of 173,138, suggests serving Lausanne entirely from Mühleberg and serving Bern entirely from Leibstadt. The entire production of Beznau is dedicated to Zürich. The rest is distributed to match the demand and supply constraints, as described on the following table:

	Zürich	Geneva	Lausanne	Bern
Mühleberg	0	593	2,517	0
Beznau	3,198	0	0	0
Leibstadt	5,763	3,184	0	1,258

In order to obtain a transhipment problem formulation, it is necessary to model the problem using a mathematical network. This is done in the following way:

- a (supply) node is associated with each supplier,
- a (demand) node is associated with each customer,
- for each supplier i, an arc (i, j) connects the supplier with each customer j that can be served,
- the cost associated with each arc is the cost of the associated transportation.

The network for Example 22.12 is represented in Figure 22.8. The transportation problem is therefore a transhipment problem where

- the cost on each arc is c_{ij},
- the supply for the node corresponding to supplier i is s_i,
- the supply for the node corresponding to customer j is $-t_j$,
- the lower bound on each arc is 0,
- there is no upper bound.

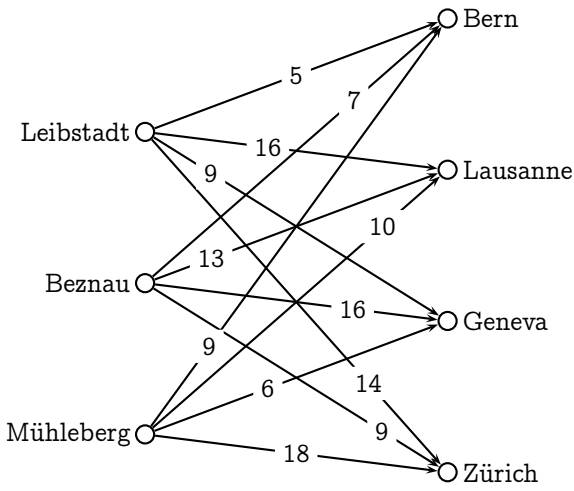


Figure 22.8: Modeling of Example 22.12 using a network representation

The optimization problem can then be written as follows:

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{m_o} \sum_{j=1}^{m_d} c_{ij} x_{ij}$$

subject to

$$\begin{aligned}
\sum_{j=1}^{m_d} x_{ij} &= s_i, && i = 1, \ldots, m_o, \\
\sum_{i=1}^{m_o} x_{ij} &= t_j, && j = 1, \ldots, m_d, \\
x_{ij} &\geq 0, && i = 1, \ldots, m_o,\ j = 1, \ldots, m_d, \\
x_{ij} &= 0, && \text{if } i \text{ does not serve } j.
\end{aligned}$$

22.4.4 The assignment problem

The assignment problem can be seen as a version of the transportation problem, where each supplier sends one unit of flow and each customer receives one. It is defined as follows.

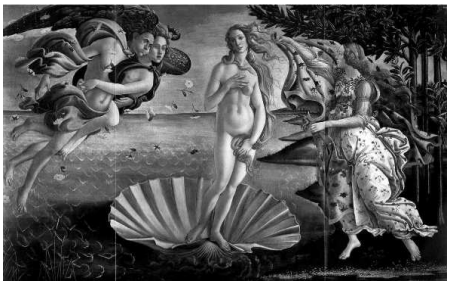
Definition 22.13 (Assignment problem)**.** Consider n resources and n tasks to be performed. The cost of assigning resource i to task j is c_{ij}. The problem consists in assigning the n resources to the n tasks at minimal total cost.

This is again an example of a problem that has no a priori relationship with a network. Still, it can be modeled as a transhipment problem.

Example 22.14 (Assignment)**.** After the death of her husband, my grandmother discovered four masterpieces in her attic (see Figure 22.9). She does not want to keep them and would like to sell each of them to one of her four children. Each child made an offer for the masterpieces of interest, in the following way (in kEuros):

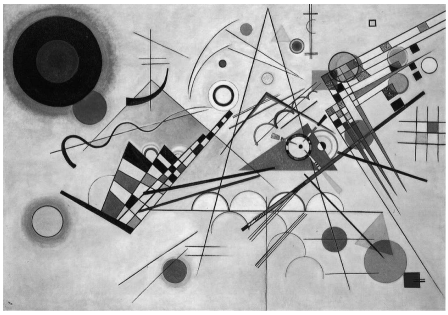
	Botticelli	Bruegel	Kandinsky	Bierlaire
Harry	8,000	11,000	—	—
Ron	9,000	13,000	12,000	—
Hermione	9,000	—	11,000	0.01
Ginny	—	14,000	12,000	—

Which masterpiece should she sell to which child? The best deal is obtained by selling the Botticelli to Harry, the Bruegel to Ginny, the Kandinsky to Ron, and the last one to Hermione, for a total of 34,000.01 kEuros. Note that Definition 22.13 mentions minimizing the costs and not maximizing the profit. Here, we have a maximization problem. Therefore, the cost for an assignment is the proposed price with the opposite sign.



(a) Botticelli, 1485

(b) Bruegel, 1558



(c) Kandinsky, 1923

(d) Bierlaire, 1971

Figure 22.9: Masterpieces

This problem is a typical example of a combinatorial optimization problem. Indeed, the objective is to identify the best combination of resources and tasks. Such problems are in general highly complicated, and require advanced techniques to be solved (some of them are described in Part VII of this book). However, in the case of the assignment problem, we can model it as a transhipment problem. The network model is built in a way similar to the transportation problem:

- a node is associated with each resource,
- a node is associated with each task,
- for each resource i, an arc (i, j) connects the resource with each task j if the assignment is feasible,
- the cost associated with each arc is the benefit of the associated assignment, with the opposite sign.

The network for Example 22.14 is represented in Figure 22.10. The assignment problem is therefore a transportation problem, that is a transhipment problem where

- the cost on each arc is c_{ij},
- the supply for each node corresponding to a resource is 1,

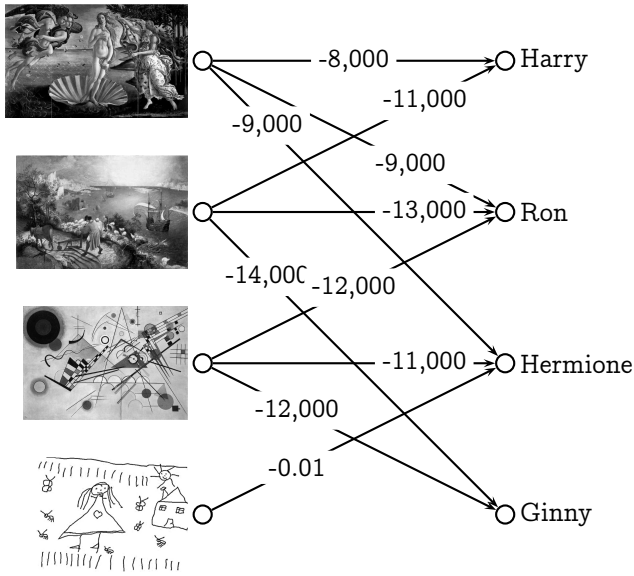


Figure 22.10: Modeling of Example 22.14 using a network representation

- the supply for each node corresponding to a task is -1,
- the lower bound on each arc is 0,
- the upper bound on each arc is 1.

The optimization problem can then be written as follows:

$$\min_{x \in \mathbb{R}^{n^2}} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n, \tag{22.32}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n, \tag{22.33}$$

$$x_{ij} \geq 0, \qquad i = 1, \ldots, n,\ j = 1, \ldots, n, \tag{22.34}$$

$$x_{ij} \leq 1, \qquad i = 1, \ldots, n,\ j = 1, \ldots, n. \tag{22.35}$$

The variable x_{ij} is to take on the value 1 if resource i is assigned to task j, and 0 otherwise. The last two constraints should be written as

$$x_{ij} \in \{0, 1\},\ i = 1, \ldots, n,\ j = 1, \ldots, n.$$

However, Corollary 22.7 guarantees that there is an optimal solution that is integer, as the supply vector is integer. Therefore, the entries of that optimal vector x are

guaranteed to be 0 or 1, without an explicit constraint imposing it. The constraints (22.32) guarantee that each resource is assigned to exactly one task. Similarly, the constraints (22.33) guarantee that each task is assigned to exactly one resource.

22.5 Exercises

Exercise 22.1. Consider a version of the assignment problem (Definition 22.13) where there are m resources and n tasks, where $m > n$. Write it as a transhipment problem.

Exercise 22.2. The coach of a swimming team needs to assign swimmers to a 200-meters medley relay team to send to the Junior Olympics. Since most of his best swimmers are very fast in more than one stroke, it is not clear which swimmer should be assigned to each of the four strokes. The five fastest swimmers and the best time (in seconds) they have achieved in each of the strokes (for 50 meters) are presented in Table 22.3.

Table 22.3: Best time for each swimmer and each stroke (Exercise 22.2)

Stroke	Anna	Eva	Marija	Shadi	Marianne
Backstroke	37.7	32.9	33.8	37.0	35.4
Breaststroke	43.4	33.1	42.2	34.7	41.8
Butterfly	33.3	28.5	38.9	30.4	33.6
Freestyle	29.2	26.4	29.6	28.5	31.1

Transform the problem into an assignment problem (Definition 22.13) and provide its mathematical formulation as a transhipment problem. Find an optimal solution using the simplex algorithm (Algorithm 16.5).

Exercise 22.3. During a wedding dinner gathering p families, the guests are invited to sit at q tables. Denote by a_i the number of members of family i, and by b_j the number of seats at table j. In order to encourage social exchanges, two members of the same family cannot sit at the same table. Moreover, the first and the second family do not talk to each other anymore, and do not want to be seated at the same table.

1. Formulate a network model that helps to seat all the guests and respect the above mentioned conditions.
2. Investigate the existence of a solution.
3. Solve the problem with $p = 6$, $q = 6$, $a_1 = 3$, $a_2 = 2$, $a_3 = 5$, $a_4 = 4$, $a_5 = 3$, $a_6 = 1$, $b_j = 3$, $j = 1, \ldots, 6$ using the simplex algorithm (Algorithm 16.5).

Exercise 22.4. After spending a weekend with friends, Aria has determined the amount of money that they owe each other, as presented in Table 22.4.

François requests a simple solution that minimizes the sum of transfers. Model that problem as a transhipment problem and solve it using the simplex algorithm

Table 22.4: Money owed by each friend (Exercise 22.4).

Who	How much	To whom
Aria	6.-	Monia
Aria	16.-	Gabriel
Monia	0.	-
Gabriel	8.-	Monia
François	10.-	Aria
François	16.-	Monia

(Algorithm 16.5).

Exercise 22.5. Consider the network represented in Figure 21.25, where each arc (i, j) is associated with its lower bound ℓ_{ij}, its flow x_{ij}, and its upper bound u_{ij} in the following way: $(\ell_{ij}, x_{ij}, u_{ij})$.

1. Write the mathematical formulation of the maximum flow problem, as a transhipment problem.
2. Solve it using the simplex algorithm (Algorithm 16.5).

Chapter 23

Shortest path

Contents

The shortest path problem is defined in Section 22.4.1. In this chapter, we focus on solving the problem.

When looking at a map, it may look relatively easy to identify the shortest path. But the shortest path problem for general networks does not inherit the intuitive properties of a map. First, an algorithm does not benefit from the bird's eye view of the network. As discussed in Section 21.5.5, data structures such as adjacency matrices or adjacency lists are used in general. They provide a myopic view of the network topology, similar to what would be the case in a labyrinth, in the sense that, given a node, we have direct access only to the adjacent nodes. Second, on geographical maps, the triangle inequality is verified, so that the distance as the crow flies between two nodes can be used as a reference to compare with the length of paths. The path that deviates the least from the straight line is likely to be the shortest, and the length of the detours can be roughly estimated. In general networks, the nodes may not necessarily correspond to geographical locations, and the arc cost may not represent the Euclidean distance. Consequently, the triangle inequality may not necessarily hold, and the intuition inspired by geographical maps cannot be used. For instance, consider the network represented in Figure 23.1, where the number next to each arc is its cost, and the number associated with each node is its identifier. The cost to go from node 9 to node 10 through nodes 13 and 14 is less then the cost of the direct arc, violating the triangle inequality.

Table 23.1 enumerates all simple paths connecting node 1 and node 16 in this network. There are 20 of them. The shortest one is represented in bold, both in Figure 23.1 and Table 23.1. Obviously, a straight line between node 1 and node 16 does not provide any intuition about the shortest path.

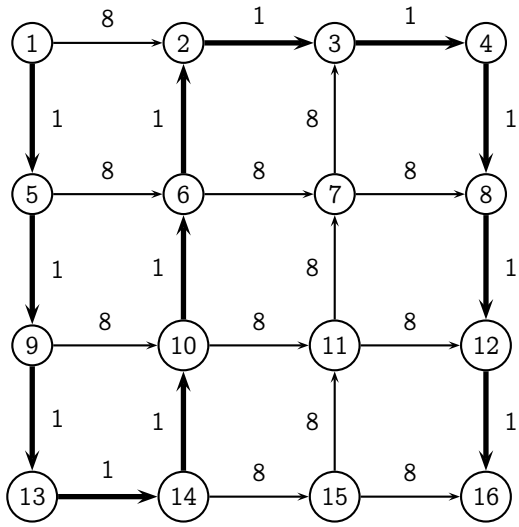


Figure 23.1: Triangle inequality does not hold

23.1 Properties

Clearly, the path enumeration is not appropriate in practice to identify the shortest path. The number of paths between two nodes can grow exponentially with the size of the network.

Instead, we exploit the fact that the shortest path problem is a transhipment problem (see Section 22.4.1). The optimality conditions (22.19)–(22.20) have a nice interpretation in that case. For each arc (i, j) in the network,

$$\lambda_j \leq c_{ij} + \lambda_i. \tag{23.1}$$

For each arc on the shortest path (that is, in terms of the transhipment problem, for each arc transporting flow), we have

$$\lambda_j = c_{ij} + \lambda_i. \tag{23.2}$$

As the optimality conditions of the shortest path problem are derived directly from the complementarity slackness conditions, the next theorem does not need a proof. However, we provide a proof to obtain some insight in the interpretation of the dual variables.

Table 23.1: List of simple forward paths between node 1 and node 16 in the network from Figure 23.1

Path	Cost	Sequence of nodes												
1	13	1	2	3	4	8	12	16						
2	15	1	5	6	2	3	4	8	12	16				
3	29	1	5	6	7	3	4	8	12	16				
4	27	1	5	6	7	8	12	16						
5	17	1	5	9	10	6	2	3	4	8	12	16		
6	31	1	5	9	10	6	7	3	4	8	12	16		
7	29	1	5	9	10	6	7	8	12	16				
8	38	1	5	9	10	11	7	3	4	8	12	16		
9	36	1	5	9	10	11	7	8	12	16				
10	27	1	5	9	10	11	12	16						
11	**12**	**1**	**5**	**9**	**13**	**14**	**10**	**6**	**2**	**3**	**4**	**8**	**12**	**16**
12	26	1	5	9	13	14	10	6	7	3	4	8	12	16
13	24	1	5	9	13	14	10	6	7	8	12	16		
14	33	1	5	9	13	14	10	11	7	3	4	8	12	16
15	31	1	5	9	13	14	10	11	7	8	12	16		
16	22	1	5	9	13	14	10	11	12	16				
17	40	1	5	9	13	14	15	11	7	3	4	8	12	16
18	38	1	5	9	13	14	15	11	7	8	12	16		
19	29	1	5	9	13	14	15	11	12	16				
20	20	1	5	9	13	14	15	16						

Theorem 23.1 (Optimality conditions for the shortest path problem)**.** *Consider a network* $(\mathcal{N}, \mathcal{A})$ *with* n *arcs and* m *nodes, and the cost vector* $c \in \mathbb{R}^n$*. Consider a vector* $\lambda \in \mathbb{R}^m$ *such that*

$$\lambda_j \leq \lambda_i + c_{ij}, \quad \forall (i,j) \in \mathcal{A}. \tag{23.3}$$

Consider a path P *between a node* o *and a node* d*. If*

$$\lambda_j = \lambda_i + c_{ij}, \quad \forall (i,j) \in P, \tag{23.4}$$

then P *is a shortest path between* o *and* d*.*

Proof. Consider any path Q between o and d, composed of $\ell + 1$ arcs, $\ell \geq 0$:

$$Q = o \to j_1 \to j_2 \ldots j_\ell \to d.$$

The total cost $C(Q)$ of Q is the sum of the cost of each arc on Q:

$$C(Q) = c_{oj_1} + c_{j_1 j_2} + \ldots + c_{j_\ell d}.$$

From (23.3), $c_{ij} \geq \lambda_j - \lambda_i$ and we obtain

$$C(Q) \geq (\lambda_{j_1} - \lambda_o) + (\lambda_{j_2} - \lambda_{j_1}) + \ldots + \lambda_d - \lambda_{j_\ell} = \lambda_d - \lambda_o, \tag{23.5}$$

as for each node i different from o and d, both λ_i and $-\lambda_i$ are involved in the sum, and cancel out.

Assume that (23.4) holds. The path P is composed of $k+1$ arcs, $k \geq 0$:

$$P = o \rightarrow i_1 \rightarrow i_2 \ldots i_k \rightarrow d.$$

The total cost $C(P)$ of P is the sum of the cost of each arc on P:

$$C(P) = c_{oi_1} + c_{i_1 i_2} + \ldots + c_{i_k d}.$$

From (23.4), $c_{ij} = \lambda_j - \lambda_i$ and we obtain

$$C(P) = (\lambda_{i_1} - \lambda_o) + (\lambda_{i_2} - \lambda_{i_1}) + \ldots + \lambda_d - \lambda_{i_k} = \lambda_d - \lambda_o. \tag{23.6}$$

From (23.5), we obtain for any path Q that $C(Q) \geq C(P)$, proving that P is the path with minimum cost. □

The dual variable λ_i is usually called the *label* of node i. Equation (23.6) shows that the length of the shortest path is the difference between the label of the destination and the label of the origin. As discussed in Section 22.2, the optimality conditions are not affected if all labels are shifted by the same constant. Therefore, it is always possible to impose $\lambda_o = 0$. In that case, the label λ_i can be interpreted as the cost of the shortest path from o to i.

Figure 23.2 represents the same network as Figure 23.1, where each node is associated with a label. It can easily be verified that condition (23.3) is verified for each arc in the network. Each arc (i, j) such that $\lambda_j = \lambda_i + c_{ij}$ is represented in bold, and each arc such that $\lambda_j < \lambda_i + c_{ij}$ is represented with a dotted line. The subnetwork consisting of all arcs in bold is a spanning tree (as discussed later, a solution of the single origin-multiple destination shortest path problem is always a spanning tree).

Therefore, from characterization 3 of Theorem 21.10, there is a single path with bold arcs between node o and any node i. As all arcs of this path verify the optimality conditions, it is the shortest path.

We now prove some properties of the shortest path. The first result states that no shortest path exists if the network contains a negative cost cycle. Intuitively, such a cycle could be followed as many times as needed to reach any arbitrarily low value for the cost of the path. From the point of view of the transhipment problem, the linear optimization problem is not bounded.

Theorem 23.2 (Negative cost cycle). *Consider a network $(\mathcal{N}, \mathcal{A})$ with n arcs and m nodes, the cost vector $c \in \mathbb{R}^n$, and two nodes o and d. Assume that there exists a forward path from o to d containing a cycle with negative cost. Therefore, no forward path is the shortest path between o and d.*

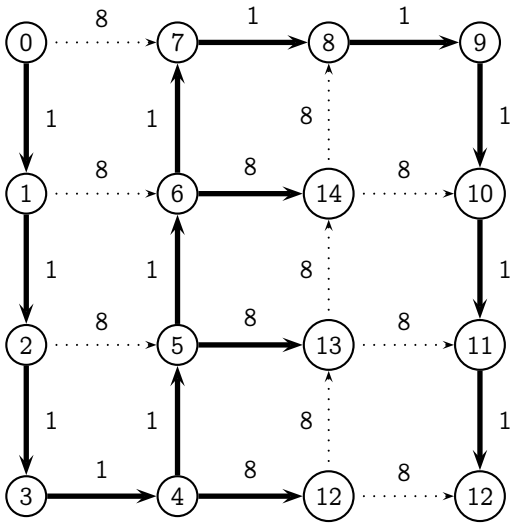


Figure 23.2: Shortest path tree, with node labels and arc costs

Proof. Consider the forward path P between o and d containing a negative cost cycle. Denote $C_c < 0$ the cost of the cycle, and $C_r = C(P) - C_c$ the cost of the rest of the path. A new path P_k between o and d can be created by including the cycle k times instead of only once. The cost of P_k is $C_r + kC_c$. Assume by contradiction that there exists a shortest path Q between o and d. Select

$$k > \frac{C(Q) - C_r}{C_c}.$$

Then, the cost of P_k is less than the cost of Q. Indeed, as $C_c < 0$, we have

$$C(P_k) = C_r + kC_c < C_r + \frac{C(Q) - C_r}{C_c} C_c = C(Q).$$

This contradicts the fact that Q is a shortest path. □

In principle, this should not be a problem as we are only considering simple paths, where cycles are not allowed. Unfortunately, the shortest simple path problem in the presence of negative cost cycles is much more difficult than in their absence. More advanced methods, such as those presented in Part VII of this book, must be considered. In this chapter, we assume that no negative cost cycle exists. It happens to be a sufficient condition for the existence of a shortest path. To show this, we start with a simple lemma.

Lemma 23.3. *[Shorter simple paths] Consider a network $(\mathcal{N}, \mathcal{A})$, and two nodes o and d. Consider a forward path P between o and d that does not contain a negative cost cycle. Then there exists a simple forward path Q from o to d such that $C(Q) \leq C(P)$.*

Proof. If the path P is simple, the result is immediate. Assume that path P contains node j several times, that is

$$P = o \to i_1 \to i_2 \dots i_k \to j \to i_{k+1} \dots i_\ell \to j \to i_{\ell+1} \dots \to i_m \to d,$$

where the first occurrence of node j is after node i_k, and the last occurrence before node $i_{\ell+1}$. The cost of P is

$$C(P) = C_1 + C_2 + C_3,$$

where

$$\begin{aligned} C_1 &= c_{oi_1} + c_{i_1 i_2} + \dots + c_{i_k j} \\ C_2 &= c_{j,i_{k+1}} + \dots + c_{i_\ell j} \\ C_3 &= c_{j,i_{\ell+1}} + \dots c_{i_m d}. \end{aligned}$$

By assumption, P does not contain a negative cost cycle, and the cost C_2 of the cycle is non negative, that is $C_2 \geq 0$. Therefore, the path obtained by removing the cycle is

$$P' = o \to i_1 \to i_2 \dots i_k \to j \to i_{\ell+1} \dots \to i_m \to d$$

and has lower cost

$$C(P') = C_1 + C_3 \leq C(P).$$

Note that node j appears exactly once in path P'. If P' is simple, the result is obtained. Otherwise, P' contains another cycle that can be removed in the same way. Cycles are removed until a simple path is obtained. □

The lemma motivates the use of simple paths when shortest paths are considered. Indeed, any cycle can be skipped to make the path shorter (until it becomes simple), except if one of those cycles has a negative cost. The existence of a shortest path can be deduced directly from this result.

Corollary 23.4 (Existence of a shortest path)**.** *Consider a network $(\mathcal{N}, \mathcal{A})$ with n arcs and m nodes, the cost vector $c \in \mathbb{R}^n$, and two nodes o and d. A shortest path between o and d exists if and only if there is at least one path between o and d, and no path between o and d contains a negative cost cycle.*

Proof. The necessary condition is Theorem 23.2. For the sufficient condition, assume that there is at least one path, and no path with a negative cycle. Consider all simple paths from o to d. From Lemma 21.6, there is a finite number of them. As there

is at least one path between o and d, Theorem 23.3 guarantees that there is also a simple path, so that the set of simple paths is not empty. Therefore, the simple path with minimum cost can be identified. Call it P. Take an arbitrary path Q from o to d. From Theorem 23.3, there is a simple path Q' such that $C(Q') \leq C(Q)$. By definition of P, we have also $C(P) \leq C(Q')$. Consequently, $C(P) \leq C(Q)$, proving that P is a shortest path. □

As an immediate corollary of Lemma 23.3, the shortest path can be found among the simple paths.

Corollary 23.5 (Simple shortest path). *Consider a network $(\mathcal{N}, \mathcal{A})$ with n arcs and m nodes, the cost vector $c \in \mathbb{R}^n$, and two nodes o and d. If there is a shortest path from o to d, there is one that is a simple path.*

Proof. Let P be a shortest path from o to d. From Theorem 23.2, the path does not contain a negative cost cycle. From Theorem 23.3, there exists a simple path Q such that $C(Q) \leq C(P)$. As P is a shortest path, we must have $C(Q) = C(P)$, proving the result. □

The next corollary gives a lower bound on the length of the shortest path. It is used in the algorithms to detect negative cost cycles.

Corollary 23.6 (Lower bound on the length of the shortest path). *Consider a network $(\mathcal{N}, \mathcal{A})$ with n arcs and m nodes, the cost vector $c \in \mathbb{R}^n$, two nodes o and d, and P a simple shortest path between o and d. If $c \geq 0$, $C(P) \geq 0$. If $c \not\geq 0$, then*

$$C(P) \geq (m-1) \min_{(i,j)\in\mathcal{A}} c_{ij}. \tag{23.7}$$

Proof. If $c \geq 0$, the result is obvious. If $c \not\geq 0$, then $c_{\min} = \min_{(i,j)\in\mathcal{A}} c_{ij} < 0$. Therefore,

$$C(P) = \sum_{(i,j)\in P} c_{ij} \geq \ell_P c_{\min},$$

where ℓ_P is the number of arcs in P. From Lemma 21.5, $\ell_P \leq m-1$, proving the result. □

The next property may look obvious. It happens to be an important property that allows us to decompose complex problems into simple ones. The optimization methodology known as *dynamic programming*[1] relies on the principle of optimality. We formulate it for the shortest path problem.

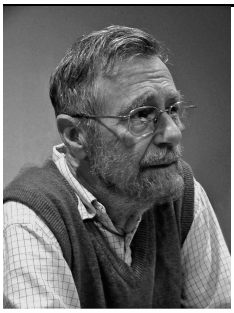
[1] Dynamic programming is not covered in this book. (See Bellman, 1957, or the more recent edition Bellman, 2010), among many references.

Theorem 23.7 (Principle of optimality). *Consider a network* $(\mathcal{N}, \mathcal{A})$*, and two nodes* o *and* d*. Let* $P = o \to i_1 \to i_2 \dots i_k \to d$ *be a shortest path from* o *to* d*. Then, for any* $\ell = 1, \dots, k$*, the subpath* $P_{o\ell} = o \to \dots \to i_\ell$ *is a shortest path from* o *to* i_ℓ *and* $P_{\ell d} = i_\ell \to \dots \to d$ *is a shortest path from* i_ℓ *to* d*.*

Proof. As there is a shortest path, there is no path with negative cost cycle. Assume by contradiction that there exists a path Q between o and i_ℓ such that $C(Q) < C(P_{o\ell})$. The path from o to d obtained by merging Q and $P_{\ell d}$ has cost

$$C(Q) + C(P_{\ell d}) < C(P_{o\ell}) + C(P_{\ell d}) = C(P),$$

which contradicts the fact that P is a shortest path. The proof that $P_{\ell d}$ is shortest is similar. □



Edsger Wybe Dijkstra [dɛikstra] was born on May 11, 1930, in Rotterdam, and died on August 6, 2002, in Nuenen (The Netherlands). The algorithm that bears his name (Algorithm 23.2) was published in 1959 in a two-page article (Dijkstra, 1959), together with an algorithm for the shortest spanning tree problem, while he was working at the Computation Department of the Mathematical Center in Amsterdam, where he was hired as a programmer. He was a member of the team that designed the computer language ALGOL-60, and designed a computer operating system while he was Professor of Mathematics at the Eindhoven University of Technology. He finished his career as the holder of the Schlumberger Centennial Chair in Computer Science at Austin, and retired in 1999.

Figure 23.3: Edsger Wybe Dijkstra

23.2 The shortest path algorithm

Although the shortest path problem can be modeled as a transhipment problem and, therefore, can be solved using the simplex method, this happens not to be efficient, if the structure of the problem is not properly exploited. We present here an algorithm that solves the single origin-multiple destination version of the shortest path problem. This algorithm can solve any problem that does not involve a negative costs cycle. In Section 23.3, we present a specific version of this algorithm for the common case where all the arc costs are non negative.

The main idea of the algorithm comes from the interpretation of the dual variables at the optimal solution. As discussed above, if the dual variable at the origin o is set to 0, the value of the dual variable λ_i at optimality is the length of the shortest path between o and i.

The algorithm maintains a vector of labels $\lambda \in \mathbb{R}^m$. Then, it analyzes each arc and checks if the optimality conditions of Theorem 23.1 are verified. If there is an arc (i, j) such that

$$\lambda_j > \lambda_i + c_{ij}, \tag{23.8}$$

then the value of λ_j is updated

$$\lambda_j = \lambda_i + c_{ij}.$$

The interpretation is as follows. At each point in time, the value of λ_i represents the length of a forward path between o and i. Therefore, if (23.8) is verified, that is, if the optimality condition is violated for arc (i, j), it means that the path with length λ_j, whatever it is, is strictly longer than the path from o to i associated with λ_i, followed by the arc (i, j), as illustrated in Figure 23.4. Therefore, this latter path is shorter, and its length $(\lambda_i + c_{ij})$ becomes the new label of the node.

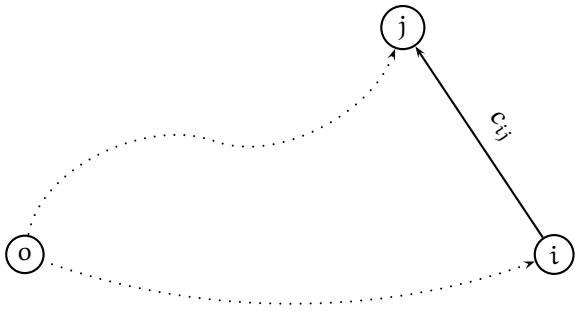


Figure 23.4: Interpretation of the labels as the length of a path from o

Using the labyrinth analogy suggested in Section 21.5.5, the algorithm is like an explorer systematically exploring all the corridors of this labyrinth (the arcs), while recording their length. The mileage counter is set to 0 at the starting point (the origin o). When the explorer reaches an intersection (a node), it records the current mileage, which is the mileage reached at the previous node (λ_i) plus the length of the corridor (c_{ij}). Now two things may happen. If it is the first time that node j is visited, the mileage is simply recorded and written on a wall of the intersection. Otherwise, there is already a mileage written on a wall, that is the length of a previous path that has been used to reach j. Interestingly, it does not matter what path it is. Its length is the only relevant information here. If the new mileage $(\lambda_i + c_{ij})$ is greater or equal to λ_j, nothing is done. The new path is not shorter. Otherwise, the new path is shorter, the value of λ_j is erased and replaced by $\lambda_i + c_{ij}$. It is also convenient to record that the predecessor of node j along the new path is node i.

The key difficulty here is to guarantee the systematic exploration of the network. The advantage of the algorithm compared to an explorer in a labyrinth is that it can be teleported to any location in the network. We need to identify the list of these locations that must be visited. We maintain a set $\mathcal{S}$ of nodes that must be

"treated," where the treatment consists in the label updating described above for each arc leaving that node. Once the node has been treated, it is removed from $\mathcal{S}$. During the treatment, the label of other nodes may be updated. Such nodes are included in the set $\mathcal{S}$ in order to be treated later. Before the algorithm starts, the set $\mathcal{S}$ contains only the origin. All labels are initialized to $+\infty$, except the label of the origin, that is set to 0.

The algorithm terminates if the set of nodes to be treated is empty. This may never happen if the network contains a negative cost cycle. If it were the case, the algorithm would follow this cycle forever, following paths with lower and lower cost, and the labels along the cycle would keep on decreasing. Therefore, a specific stopping criterion must be included to detect such cases.

This is described in Algorithm 23.1.

Algorithm 23.1: The shortest path algorithm

Objective
  Calculate a shortest path between a node o and all other nodes.
Input
  A network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs.
  A vector $c \in \mathbb{R}^n$ with the cost of each arc.
  The origin o.
Output
  A Boolean U that is true if the problem is unbounded.
  A vector $\lambda \in \mathbb{R}^m$ containing the optimal labels of the nodes.
  A vector $\pi \in \mathcal{N}^m$ such that π_i contains the node preceding node i in the shortest path if $\lambda_i \neq \infty$ and $i \neq o$.
Initialization
  $\lambda_o := 0$.
  $\lambda_i := +\infty \; \forall i \in \mathcal{N}, i \neq o$.
  $\mathcal{S} := \{o\}$.
  $U :=$ false.
Repeat
  Select node $i \in \mathcal{S}$.
  for *all* j *such that* $(i,j) \in \mathcal{A}$ **do**
    if $\lambda_j > \lambda_i + c_{ij}$ **then**
      $\lambda_j := \lambda_i + c_{ij}$
      if $\lambda_j < 0$ *and* $\lambda_j < (m-1)\min_{(i,j)\in\mathcal{A}} c_{ij}$ **then**
        $U :=$ true
        STOP: negative cost cycle detected (see Corollary 23.6)
      $\pi_j := i$
      $\mathcal{S} := \mathcal{S} \cup \{j\}$
  $\mathcal{S} := \mathcal{S} \setminus \{i\}$.
Until $\mathcal{S} = \emptyset$.

Example 23.8 (The shortest path algorithm). We apply Algorithm 23.1 on the network represented in Figure 23.1, where the origin is node 1. Table 23.2 reports the set $\mathcal{S}$, the treated node i and the value of all labels at each iteration. Table 23.3 reports the predecessors vector π at each iteration. During the first iteration, only node 1 is in set $\mathcal{S}$ and is therefore treated. There are only two arcs originating from node 1: $(1,2)$ and $(1,5)$. As the label of nodes 2 and 5 have been initialized to ∞, condition (23.8) is trivially verified, and the labels of nodes 2 and 5 are updated to $0+8$ and $0+1$ respectively. Meanwhile, π_2 and π_5 are set to 1, as node 1 is the predecessor in the current path that has been used to reach these two nodes.

During iteration 8, node 10 is treated. Its label is 10. The algorithm considers arc $(10,6)$ with cost 1. As $\lambda_6 = 9 \leq 10+1$, nothing is done. The same happens for arc $(10,11)$, as $\lambda_{11} = 18 \leq 10+8$. Therefore, no label is updated during that iteration, and node 10 is simply removed from $\mathcal{S}$.

At iteration 14, node 10 is treated again. Its label is now 5. The algorithm considers arc $(10,6)$ with cost 1. As $\lambda_6 = 9 > 5+1$, the label of node 6 is updated, π_6 is set to 10, and node 6 included into $\mathcal{S}$. The same happens for arc $(10,11)$, as $\lambda_{11} = 18 > 5+8$. Note that, by coincidence, the value of π_{11} used to be 10, and is therefore not updated.

The final labels correspond to those reported in Figure 23.2. The final value of π allows us to reconstruct the shortest paths using a backward procedure. We illustrate it for the shortest path from o to node 11. As $\pi_{11} = 10$, the predecessor of node 11 is node 10. As $\pi_{10} = 14$, the predecessor of node 10 is node 14, and so on. Following this scheme, one obtains the path

$$o \to 5 \to 9 \to 13 \to 14 \to 10 \to 11.$$

We now present some properties of Algorithm 23.1.

Theorem 23.9 (Invariants of the shortest path algorithm). *The following properties hold at the end of each iteration of Algorithm 23.1:*

1. *If $i \in \mathcal{S}$, then $\lambda_i \neq \infty$.*
2. *For each node i, the value of λ_i does not increase during the iteration.*
3. *If $\lambda_i \neq \infty$, it is the length of one path from o to i.*
4. *If $i \notin \mathcal{S}$, then $\lambda_i = \infty$ or $\lambda_j \leq \lambda_i + c_{ij}$, for all j such that $(i,j) \in \mathcal{A}$.*

Proof. 1. During the initialization, node o is included in $\mathcal{S}$ at step 14, and λ_o is set to 0 (step 12). Any other node in $\mathcal{S}$ has been included at step 25. Therefore, the condition at step 19 is verified, and $\lambda_i \neq \infty$. Therefore, the update at step 20 gives a finite label to j before including it in $\mathcal{S}$, proving the result.

2. It is a direct consequence of the condition at step 19 and the label update statement at step 20.

Table 23.2: Description of the iterations of the shortest path algorithm for Example 23.8

Iter.	$\mathcal{S}$	i	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}	λ_{13}	λ_{14}	λ_{15}	λ_{16}
0	{1}	1	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	{2 5}	2	0	8	∞	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	{5 3}	5	0	8	9	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	{3 6 9}	3	0	8	9	∞	1	9	∞	∞	2	∞	∞	∞	∞	∞	∞	∞
4	{6 9 4}	6	0	8	9	10	1	9	∞	∞	2	∞	∞	∞	∞	∞	∞	∞
5	{9 4 7}	9	0	8	9	10	1	9	17	∞	2	∞	∞	∞	∞	∞	∞	∞
6	{4 7 10 13}	4	0	8	9	10	1	9	17	∞	2	10	∞	∞	3	∞	∞	∞
7	{7 10 13 8}	7	0	8	9	10	1	9	17	11	2	10	∞	∞	3	∞	∞	∞
8	{10 13 8}	10	0	8	9	10	1	9	17	11	2	10	∞	∞	3	∞	∞	∞
9	{13 8 11}	13	0	8	9	10	1	9	17	11	2	10	18	∞	3	∞	∞	∞
10	{8 11 14}	8	0	8	9	10	1	9	17	11	2	10	18	∞	3	4	∞	∞
11	{11 14 12}	11	0	8	9	10	1	9	17	11	2	10	18	12	3	4	∞	∞
12	{14 12}	14	0	8	9	10	1	9	17	11	2	10	18	12	3	4	∞	∞
13	{12 10 15}	12	0	8	9	10	1	9	17	11	2	5	18	12	3	4	12	∞
14	{10 15 16}	10	0	8	9	10	1	9	17	11	2	5	18	12	3	4	12	13
15	{15 16 6 11}	15	0	8	9	10	1	6	17	11	2	5	13	12	3	4	12	13
16	{16 6 11}	16	0	8	9	10	1	6	17	11	2	5	13	12	3	4	12	13
17	{6 11}	6	0	8	9	10	1	6	17	11	2	5	13	12	3	4	12	13
18	{11 2 7}	11	0	7	9	10	1	6	14	11	2	5	13	12	3	4	12	13
19	{2 7}	2	0	7	9	10	1	6	14	11	2	5	13	12	3	4	12	13
20	{7 3}	7	0	7	8	10	1	6	14	11	2	5	13	12	3	4	12	13
21	{3}	3	0	7	8	10	1	6	14	11	2	5	13	12	3	4	12	13
22	{4}	4	0	7	8	9	1	6	14	11	2	5	13	12	3	4	12	13
23	{8}	8	0	7	8	9	1	6	14	10	2	5	13	12	3	4	12	13
24	{12}	12	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	13
25	{16}	16	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12
26	{}		0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12

Table 23.3: Value of π for each iteration of the shortest path algorithm for Example 23.8

Iter.	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}	π_{16}
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	1	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	1	2	-1	1	5	-1	-1	5	-1	-1	-1	-1	-1	-1	-1
4	-1	1	2	3	1	5	-1	-1	5	-1	-1	-1	-1	-1	-1	-1
5	-1	1	2	3	1	5	6	-1	5	-1	-1	-1	-1	-1	-1	-1
6	-1	1	2	3	1	5	6	-1	5	9	-1	-1	9	-1	-1	-1
7	-1	1	2	3	1	5	6	4	5	9	-1	-1	9	-1	-1	-1
8	-1	1	2	3	1	5	6	4	5	9	-1	-1	9	-1	-1	-1
9	-1	1	2	3	1	5	6	4	5	9	10	-1	9	-1	-1	-1
10	-1	1	2	3	1	5	6	4	5	9	10	-1	9	13	-1	-1
11	-1	1	2	3	1	5	6	4	5	9	10	8	9	13	-1	-1
12	-1	1	2	3	1	5	6	4	5	9	10	8	9	13	-1	-1
13	-1	1	2	3	1	5	6	4	5	14	10	8	9	13	14	-1
14	-1	1	2	3	1	5	6	4	5	14	10	8	9	13	14	12
15	-1	1	2	3	1	10	6	4	5	14	10	8	9	13	14	12
16	-1	1	2	3	1	10	6	4	5	14	10	8	9	13	14	12
17	-1	1	2	3	1	10	6	4	5	14	10	8	9	13	14	12
18	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
19	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
20	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
21	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
22	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
23	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
24	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
25	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
26	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12

3. Consider the first iteration. Before the iteration starts, all labels are set to ∞ (step 13), except $\lambda_o = 0$ (step 12). Note that the value 0 can be interpreted as the length of a path from o to o. During the first iteration, all nodes i such that the arc (o, i) exists have their label set to $\lambda_o + c_{oi} = c_{oi}$ (step 20). This is obviously the length of the direct path from o to i. All other labels remaining at ∞, the result is proved for the first iteration. Assume now by induction that the result is true at the beginning of an iteration where the treated node (selected at step 17) is i. From property 1, we have $\lambda_i \neq \infty$ and, by induction assumption, λ_i is the length of a path P between o and i. When the label of a node j is updated at step 20, its value is therefore set to the length of the path composed of path P extended by arc (i, j), proving the result for all labels updated during this iteration. As the other labels are untouched, the result holds for all nodes.
4. There are two reasons for i not to be in $\mathcal{S}$. First, if i has never been treated by the algorithm. In that case, its label has not been updated and remains at the initial value ∞. Second, i has been treated and removed from $\mathcal{S}$. Because of step 19, just after i has been treated, we have $\lambda_j \leq \lambda_i + c_{ij}$ for all $(i, j) \in \mathcal{A}$. Then the label of i is not touched anymore while it is out of $\mathcal{S}$. As soon as the label is updated at step 20, i comes back into $\mathcal{S}$ (step 25). In the meantime, any other label can only decrease (see property 2 above), so that the condition $\lambda_j \leq \lambda_i + c_{ij}$ is verified whenever i is not in $\mathcal{S}$.

□

Theorem 23.10 (Termination of the shortest path algorithm)**.** *Algorithm 23.1 terminates after a finite number of iterations.*

Proof. The algorithm terminates if $\mathcal{S} = \emptyset$. Suppose that it never happens. It means that some nodes are added to the set $\mathcal{S}$ an infinite number of times. Each time, their label is updated to a strictly lower value. Therefore, the label of these nodes goes to $-\infty$. It is not possible because of the condition at step 21 that interrupts the algorithm when a label becomes lower than a finite number. □

Theorem 23.11 (Bellman's equation)**.** *If Algorithm 23.1 terminates with $\mathcal{S} = \emptyset$, then $\lambda_j = \infty$ if and only if there is no path from o to j. If $\lambda_j \neq \infty$, then λ_j is the length of the shortest path from o and j, $\lambda_o = 0$ and, for all $j \neq o$,*

$$\lambda_j = \min_{(i,j)\in\mathcal{A}} (\lambda_i + c_{ij}). \tag{23.9}$$

Equation (23.9) is called Bellman's equation.

Proof. Assume that $\lambda_j = \infty$ and that there is a path from o to j. As $\mathcal{S} = \emptyset$, property 4 of Theorem 23.9 guarantees that for each arc, if the upstream node has a finite label, so does the downstream node. As $\lambda_o = 0$, the next node along the path has a finite label. This property propagates through the path until j, which must also have

a finite label. This contradicts the assumption, proving the sufficient condition. The necessary condition is shown by the contrapositive of property 3 of Theorem 23.9, which states that if there is no path from o to j, then $\lambda_j = \infty$.

Consider now the case where $\lambda_j \neq \infty$. By property 4 of Theorem 23.9, we have for each i such that $\lambda_i \neq \infty$,

$$\lambda_j \leq \lambda_i + c_{ij}, \forall (i,j) \in \mathcal{A}. \tag{23.10}$$

Consider a node ℓ. From property 3 of the same theorem, λ_ℓ is the length of a path from o to ℓ. Call it P_ℓ. Take any path Q from o to ℓ. Using the same argument as in the proof of Theorem 23.1, we have

$$C(Q) \geq \lambda_\ell - \lambda_o.$$

From property 2 of Theorem 23.9, as λ_o is initialized to 0, at the end of the algorithm we have $\lambda_o \leq 0$. Therefore,

$$C(Q) \geq \lambda_\ell = C(P_\ell),$$

showing that P_ℓ is a shortest path from o to ℓ.

From property 3 of the same theorem, $\lambda_o < 0$ would mean that there is a negative cost cycle from o to o. In that case, the algorithm would follow this cycle until the stopping criterion at step 21 is verified. This contradicts the assumption that the algorithm terminates with $\mathcal{S} = \emptyset$. Therefore, $\lambda_o = 0$.

Finally, Bellman's equation (23.9) is a direct consequence of the optimality conditions (23.3) and (23.4). Indeed, assume that

$$\lambda_j > \min_{(i,j)\in\mathcal{A}} (\lambda_i + c_{ij}).$$

It means that there exists $(i,j) \in \mathcal{A}$ such that $\lambda_j > \lambda_i + c_{ij}$, contradicting (23.3). Assume next that

$$\lambda_j < \min_{(i,j)\in\mathcal{A}} (\lambda_i + c_{ij}).$$

Then, $\lambda_j \neq \lambda_i + c_{ij}$ for all $(i,j) \in \mathcal{A}$, contradicting (23.4). □

Assume that Algorithm 23.1 terminates with $\mathcal{S} = \emptyset$. Consider the subnetwork $(\mathcal{N}, \mathcal{A}')$, where $\mathcal{N}$ is the set of all nodes of the network, and $\mathcal{A}'$ is generated as follows. For each node j different from the origin, select one arc (i,j) such that (23.9) is verified (if Bellman's equation is verified for several arcs, we arbitrarily select one of them). We call it *Bellman's subnetwork*. By construction, the number of arcs in the subnetwork is $m - 1$, where m is the number of nodes. Therefore, if the subnetwork does not contain any cycle, it is a spanning tree according to condition 7 of Theorem 21.10, and Definition 21.11. For any node d, there is only one path in the spanning tree connecting the origin o to d (condition 3 of Theorem 21.10). As each arc of the path verifies Bellman's equation, it verifies (23.4) and Theorem 23.1 guarantees that it is a shortest path from o to d. Therefore, the subnetwork is called a *shortest path spanning tree*.

Theorem 23.12 (Shortest path spanning tree). *Consider a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs, and a vector $c \in \mathbb{R}^n$ representing the cost of traversing each arc such that every cycle (if any) in the network has positive length. Then, Bellman's subnetwork is a shortest path spanning tree.*

Proof. As the network does not contain a cycle with negative length, Algorithm 23.1 terminates with $\mathcal{S} = \emptyset$. Assume by contradiction that Bellman's subnetwork contains a cycle $i_1, \ldots, i_\ell$. Its length is

$$c_{i_1 i_2} + c_{i_2 i_3} + \cdots + c_{i_{\ell-1} i_\ell} + c_{i_\ell i_1} = \lambda_{i_2} - \lambda_{i_1} + \lambda_{i_3} - \lambda_{i_2} + \cdots + \lambda_{i_\ell} - \lambda_{i_{\ell-1}} + \lambda_{i_1} - \lambda_\ell = 0,$$

which is not possible by assumption. Therefore, Bellman's subnetwork does not contain any cycle and is a spanning tree. As discussed above, the optimality conditions of Theorem 23.1 apply, and each path in the tree is a shortest path. □

23.3 Dijkstra's algorithm

The shortest path Algorithm 23.1 does not specify how the node to be treated during a given iteration has to be chosen within the set $\mathcal{S}$ (step 17). In Example 23.8, we have always selected the first node in the set, but any other choice would have produced a shortest path, too. While the selection strategy of the treated node does not affect the outcome of the algorithm, it can heavily affect its performance. For instance, if you run the algorithm on Example 23.8 and select the last node instead of the first one, it would require 31 iterations instead of 26.

In this section, we present a strategy that is a little bit more sophisticated, and particularly efficient, but restricted to the case when all costs are non negative. It consists in selecting the node in $\mathcal{S}$ associated with the smallest label. Therefore, in Algorithm 23.1, the statement 17 is replaced by "Select node $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$."

If we apply this version of the algorithm on Example 23.8, we obtain the results presented in Tables 23.4 and 23.5. We note that the algorithm has identified the same optimal solution as before, but in only 16 iterations, which means that each of the 16 nodes has only been treated once. As all the nodes are reachable from node 1, it is the minimum possible number of iterations. It happens that this version of the shortest path algorithm, called *Dijkstra's algorithm* and presented as Algorithm 23.2, never treats any node more than once, when all the arcs in the network have a non negative cost.

There are only a few differences between Algorithm 23.2 and Algorithm 23.1:

1. the cost vector must be non negative,
2. the node to be treated is the node in $\mathcal{S}$ with the smallest label,
3. the identification of a negative cost cycle is no longer necessary, as all arcs have non negative cost.

Table 23.4: Description of the iterations of the Dijkstra algorithm for Example 23.8

Iter.	$\mathcal{S}$	i	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}	λ_{13}	λ_{14}	λ_{15}	λ_{16}
0	{1}	1	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	{2 5}	5	0	8	∞	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	{2 6 9}	9	0	8	∞	∞	1	9	∞	∞	2	∞	∞	∞	∞	∞	∞	∞
3	{2 6 10 13}	13	0	8	∞	∞	1	9	∞	∞	2	10	∞	∞	3	∞	∞	∞
4	{2 6 10 14}	14	0	8	∞	∞	1	9	∞	∞	2	10	∞	∞	3	4	∞	∞
5	{2 6 10 15}	10	0	8	∞	∞	1	9	∞	∞	2	5	∞	∞	3	4	12	∞
6	{2 6 15 11}	6	0	8	∞	∞	1	6	∞	∞	2	5	13	∞	3	4	12	∞
7	{2 15 11 7}	2	0	7	∞	∞	1	6	14	∞	2	5	13	∞	3	4	12	∞
8	{15 11 7 3}	3	0	7	8	∞	1	6	14	∞	2	5	13	∞	3	4	12	∞
9	{15 11 7 4}	4	0	7	8	9	1	6	14	∞	2	5	13	∞	3	4	12	∞
10	{15 11 7 8}	8	0	7	8	9	1	6	14	10	2	5	13	∞	3	4	12	∞
11	{15 11 7 12}	12	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	∞
12	{15 11 7 16}	15	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12
13	{11 7 16}	16	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12
14	{11 7}	11	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12
15	{7}	7	0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12
16	{}		0	7	8	9	1	6	14	10	2	5	13	11	3	4	12	12

Table 23.5: Value of π for each iteration of the Dijkstra algorithm for Example 23.8

Iter.	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}	π_{16}
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	1	-1	-1	1	5	-1	-1	5	-1	-1	-1	-1	-1	-1	-1
3	-1	1	-1	-1	1	5	-1	-1	5	9	-1	-1	9	-1	-1	-1
4	-1	1	-1	-1	1	5	-1	-1	5	9	-1	-1	9	13	-1	-1
5	-1	1	-1	-1	1	5	-1	-1	5	14	-1	-1	9	13	14	-1
6	-1	1	-1	-1	1	10	-1	-1	5	14	10	-1	9	13	14	-1
7	-1	6	-1	-1	1	10	6	-1	5	14	10	-1	9	13	14	-1
8	-1	6	2	-1	1	10	6	-1	5	14	10	-1	9	13	14	-1
9	-1	6	2	3	1	10	6	-1	5	14	10	-1	9	13	14	-1
10	-1	6	2	3	1	10	6	4	5	14	10	-1	9	13	14	-1
11	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	-1
12	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
13	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
14	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
15	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12
16	-1	6	2	3	1	10	6	4	5	14	10	8	9	13	14	12

Algorithm 23.2: Dijkstra's algorithm

Objective
  Calculate a shortest path between a node o and all nodes.
Input
  A network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs.
  A vector $c \in \mathbb{R}^n$ with the cost of each arc, $c \geq 0$.
  The origin o.
Output
  A vector $\lambda \in \mathbb{R}^m$ containing the optimal labels of the nodes.
  A vector $\pi \in \mathcal{N}^m$ such that π_i contains the node preceding node i in the shortest path if $\lambda_i \neq \infty$ and $i \neq o$.
Initialization
  $\lambda_o := 0$.
  $\lambda_i := +\infty \; \forall i \in \mathcal{N}, i \neq o$.
  $\mathcal{S} := \{o\}$.
Repeat
  Select node $i \in \mathcal{S}$ such that $\lambda_i \leq \lambda_j$, for all $j \in \mathcal{S}$.
  for *all* j *such that* $(i, j) \in \mathcal{A}$ **do**
    if $\lambda_j > \lambda_i + c_{ij}$ **then**
      $\lambda_j := \lambda_i + c_{ij}$
      $\pi_j := i$
      $\mathcal{S} := \mathcal{S} \cup \{j\}$
  $\mathcal{S} := \mathcal{S} \setminus \{i\}$.
Until $\mathcal{S} = \emptyset$.

Dijkstra's algorithm has the same properties as the shortest path algorithm, described in Section 23.2. It has also the following properties.

Theorem 23.13 (Termination of Dijkstra's algorithm)**.** *Algorithm 23.2 terminates after a finite number of iterations.*

Proof. The algorithm terminates if $\mathcal{S} = \emptyset$. Suppose that it never happens. It means that some nodes are added to the set $\mathcal{S}$ an infinite number of times. Each time, their label is updated to a strictly lower value. Therefore, the label of these nodes goes to $-\infty$. From property 3 of Theorem 23.9, λ_i is the length of a path from node o to i. As all the costs are non negative, the length of any path is non negative, and λ_i cannot go below 0. □

Theorem 23.14 (Invariants of Dijkstra's algorithm). *Consider the set*

$$\mathcal{T} = \{i | \lambda_i \neq \infty \text{ and } i \notin \mathcal{S}\}. \tag{23.11}$$

The following properties hold at the end of each iteration of Algorithm 23.2:

1. *If $i \in \mathcal{T}$ and $j \notin \mathcal{T}$, then $\lambda_i \leq \lambda_j$.*
2. *If $i \in \mathcal{T}$ in the beginning of the iteration, then the label λ_i is not modified during the iteration.*
3. *If $i \in \mathcal{T}$ in the beginning of the iteration, then $i \notin \mathcal{S}$ at the end of the iteration.*
4. *If $i \in \mathcal{T}$, then λ_i is the length of the shortest path from o to i.*

Proof. 1. We prove properties 1 and 2 by induction. Property 1 holds for the first iteration, where node o is treated, with $\lambda_o = 0$. It is the only node in $\mathcal{T}$ at the end of the iteration. All other labels are equal to ∞, except for nodes j such that $(o, j) \in \mathcal{A}$. As $\lambda_j = c_{oj} \geq \lambda_o = 0$, the property holds for these labels. As $\mathcal{T} = \emptyset$ at the beginning of the iteration, property 2 trivially holds for the first iteration. Consider now another iteration, and assume that property 1 is true at the beginning of the iteration, that is $\lambda_i \leq \lambda_j$, $\forall i \in \mathcal{T}$, $\forall j \notin \mathcal{T}$. The iteration is treating node ℓ. According to the rule of node selection,

$$\lambda_\ell \leq \lambda_j, \quad \forall j \notin \mathcal{T}. \tag{23.12}$$

As $\ell \in \mathcal{S}$, $\ell \notin \mathcal{T}$ at the beginning of the iteration. When the iteration treats arc (ℓ, m), two cases must be considered: $m \in \mathcal{T}$ and $m \notin \mathcal{T}$.

$m \in \mathcal{T}$ Using the assumption of the induction, we have $\lambda_m \leq \lambda_\ell$. As $c_{\ell m} \geq 0$, we have also $\lambda_m \leq \lambda_\ell + c_{\ell m}$. It means that no node in $\mathcal{T}$ will see its label updated during the iteration. This proves property 2. As no label has been updated by the algorithm in this case, property 1 continues to hold.

$m \notin \mathcal{T}$ If the label of node m is not updated, nothing changes, and property 1 continues to hold. If the label is updated, we have, at the end of the iteration, $\lambda_m = \lambda_\ell + c_{\ell m}$. Take any node $i \in \mathcal{T}$. We have, at the end of the iteration, $\lambda_i \leq \lambda_\ell$ by the assumption of the induction and the fact that the label of i has not been updated by the iteration. As $c_{\ell m} \geq 0$, $\lambda_i \leq \lambda_\ell + c_{\ell m} = \lambda_m$, and the property holds after the iteration for all nodes that were in $\mathcal{T}$ at the beginning of the iteration.

Finally, as ℓ is in $\mathcal{T}$ at the end of the iteration, we need to show that $\lambda_\ell \leq \lambda_j$, $\forall j \notin \mathcal{T}$. This is guaranteed by the rule of node selection (23.12), and the fact that λ_ℓ has not been modified by the iteration.

2. Proved with the previous property.
3. This is an immediate corollary of the previous property. Indeed, if the label of i is not modified, i is not included in $\mathcal{S}$.

4. From property 2, the label of i will not be modified anymore by any iteration. Therefore, when the algorithm terminates (Theorem 23.13), the final value is λ_i. From Theorem 23.11, it is the length of the shortest path from o to i.

□

We conclude this section with some comments about the algorithm.

- The efficiency of Dijkstra's algorithm relies on an efficient procedure to identify the node with the smallest label in set $\mathcal{S}$. Most implementations rely on a data structure known as *heap* that maintains a partial order of the set (see Brassard and Bratley, 1996, Section 5.7).
- Theorem 23.14 is invoked when applying Dijkstra's algorithm to the single origin-single destination problem. Indeed, as soon as the destination node is treated, the shortest path has been identified and the algorithm can be interrupted.

23.4 The longest path problem

The longest path problem is defined in a similar way to the shortest path problem.

Definition 23.15 (Longest path problem: single origin-single destination)**.** Consider a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs, and a vector $c \in \mathbb{R}^n$ representing the cost of traversing each arc. Consider one node o called the origin and one node d called the destination. The longest path problem consists in finding a simple forward path with origin o and destination d, and with the largest cost.

It can also be written as a transhipment problem:

$$\max_{x \in \mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij} x_{ij} \tag{23.13}$$

subject to

$$\begin{aligned}
\sum_{j|(o,j)\in\mathcal{A}} x_{oj} - \sum_{k|(k,o)\in\mathcal{A}} x_{ko} &= 1, \\
\sum_{j|(d,j)\in\mathcal{A}} x_{dj} - \sum_{k|(k,d)\in\mathcal{A}} x_{kd} &= -1, \\
\sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{k|(k,i)\in\mathcal{A}} x_{ki} &= 0, \ \forall i \in \mathcal{N}, i \neq o, i \neq d, \\
x_{ij} &\geq 0, \ \forall (i,j) \in \mathcal{A}.
\end{aligned}$$

As discussed in Section 1.2.1 and as with any optimization problem, it can be transformed into a minimization problem by changing the sign of the objective function:

$$\max_{x \in \mathbb{R}^n} \sum_{(i,j)\in\mathcal{A}} c_{ij} x_{ij} \iff \min_{x \in \mathbb{R}^n} - \sum_{(i,j)\in\mathcal{A}} c_{ij} x_{ij} = \sum_{(i,j)\in\mathcal{A}} (-c_{ij}) x_{ij}. \tag{23.14}$$

Consequently, it is equivalent to the shortest path problem where the sign of each cost in the network has been changed. Note that, in many applications, this transformed problem is likely to contain negative cost cycles. In this case, the problem posed as a transhipment problem is unbounded, and the shortest path algorithm does not work. Other modeling frameworks related to combinatorial optimization (see Part VII) must be considered.

A concrete application of the longest path problem is the *program evaluation and review technique* (PERT) used in project management. Consider a project composed of m tasks of a given duration. Each task i is associated with a list of other tasks that must be completed before task i can start. They are the *predecessors* of task i. For example, consider a household that decides to renovate the bathroom in the house. The list of tasks, together with their duration (in days) and list of predecessors, is reported in Table 23.6.

Table 23.6: List of tasks to renovate a bathroom

	Tasks	Duration	Predecessors
1	Design the overall setup of the future bathroom	1	
2	Select the bathroom furniture	1	1
3	Quotation for the bathroom furniture	3	1,2
4	Order the bathroom furniture	3	3,6
5	Select the tiles	2	2
6	Quotation from the installer	5	3
7	Quotation from the tiler	5	4,5
8	Confirm the installer	6	6
9	Confirm the tiler	1	7
10	Remove existing furniture	2	8
11	Tiling	3	9,10
12	Installation of the furniture	2	11
13	Dispose of the old furniture	1	10

The relevant questions for project management are: what is the minimum possible duration of the project? What are the tasks that do not tolerate any delay without delaying the duration of the whole project? Such tasks are called *critical*.

We construct a network in the following way:

- a node is associated with each task;
- a node o represents the beginning of the project;
- a node d represents the end of the project;
- for each task j, insert an arc (i, j) for each predecessor i;
- for each task j without predecessor, insert an arc (o, j);
- for each task i without successor, that is, each task which is the predecessor of no other task, insert an arc (i, d);

- for each task i, the cost of each arc (i, j) is the duration of task i;
- the cost of each arc (o, j) is 0.

An arc (i, j) means that "task i must be completed before task j is started." Note that, thanks to the network structure, arcs are needed only for direct predecessors. In our example, task 11 (tiling) cannot start if task 5 (select the tiles) has not been completed. However, there is no need to insert an arc between these two tasks. By transitivity, the precedence is captured by the presence of a forward path between the two tasks. The network representation for the bathroom project is illustrated in Figure 23.5. To answer the two questions for the project management, a longest path problem must be solved. Note that such a network does not contain any forward cycle. Indeed, as each arc (i, j) means "task i must be completed before starting task j," a forward cycle would mean that, for each node i on the cycle, task i must be completed before starting task i, which is a non sense. If it appears to be the case, there must be a mistake in the problem definition. Consequently, when solving the longest path problem with the shortest path algorithm, although all costs are negative, no negative cost cycle exists, and the algorithm terminates with a valid solution. The optimal labels provide, after changing their sign, the earliest time when each task can be started. The label of node d is the maximum project duration. For our example, task 7, say, cannot start before 13 days, and the project cannot last less than 24 days (see Table 23.7). The arcs in the longest path tree are depicted in bold in Figure 23.5. It can be seen that the longest path is the sequence of tasks 1, 2, 3, 6, 4, 7, 9, 11, and 12, taking a total of 24 days. These tasks are critical tasks, and this path is called a *critical path*. If any of these tasks were delayed just a little, the whole project would be delayed as well. Consequently, the pressure should be put on the tiler. The installer can afford some delay (how much?) without affecting the end of the whole project. Note that, in the presence of several longest paths, the algorithm is not reporting all critical tasks.

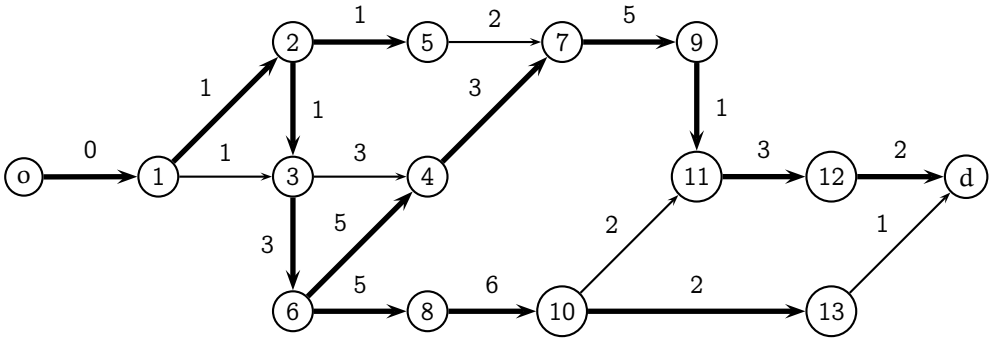


Figure 23.5: Network representation of the project organization

Table 23.7: Optimal labels of the critical paths

i	λ_i	i	λ_i	i	λ_i
o	0	5	-2	10	-16
1	0	6	-5	11	-19
2	-1	7	-13	12	-22
3	-2	8	-10	13	-18
4	-10	9	-18	d	-24

Many variants of the algorithms presented in this chapter have been proposed in the literature. Namely, some algorithms such as the A^* algorithm (Hart et al., 1968) are designed to solve the shortest path problem for road networks, where the Euclidean distance can be exploited as a proxy to the shortest path distance. Algorithms designed to be efficient for navigation systems have also been proposed (see, for instance, Abraham et al., 2011).

23.5 Exercises

Exercise 23.1. Consider the network represented in Figure 23.6, where the cost of each arc is shown next to it. Apply Dijkstra's algorithm (Algorithm 23.2) to identify the shortest path from node o to any other node.

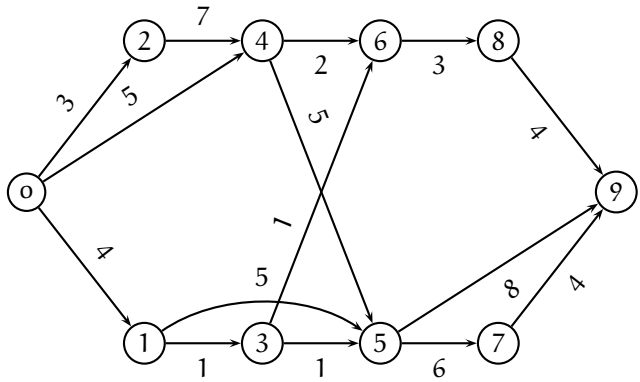


Figure 23.6: Network for Exercise 23.1. The cost of each arc is shown next to it

Exercise 23.2. Consider the network represented in Figure 23.7, where the value associated with each arc represents its length. Determine, when they exist, the shortest paths from node o to all other nodes in the network.

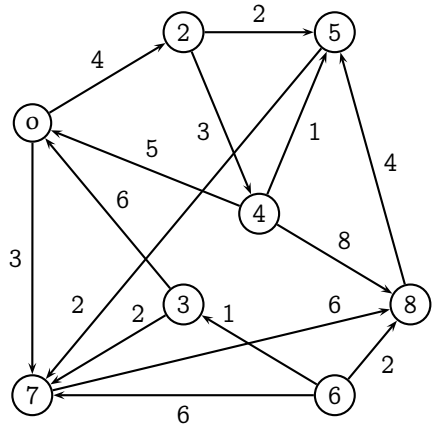


Figure 23.7: Network for Exercise 23.2, where the value associated with each arc represents its length

Exercise 23.3. Modify Dijkstra's algorithm to generate all shortest path trees. Hint: maintain several labels at each node.

Exercise 23.4. A museum hires attendants during opening hours, from 9:00 to 17:00. It has the possibility of hiring several people, each of them being available for several hours during the day, at a given cost, as reported in Table 23.8. Identify who the museum should hire so that there is at least one attendant present in the museum at any time, in order to minimize the total cost. Hint: model the problem as a shortest path problem.

Table 23.8: Availabilities and costs of the museum attendants for Exercise 23.4

Name	from	to	cost
Durin	9:00	13:00	30
Isumbras	9:00	11:00	18
Hamfast	12:00	15:00	14
Isengrim	12:00	17:00	38
Arathorn	14:00	17:00	16
Bilbo	13:00	16:00	22
Gelmir	16:00	17:00	9

Exercise 23.5. Consider the six Swiss cities represented in Figure 23.8. The travel time by car (C) and by public transportation (PT) is shown next to each arc connecting two cities.

1. Identify the fastest itinerary from Orbe to any other city by car.
2. Identify the fastest itinerary from Orbe to any other city by public transportation.

3. Thanks to the car sharing service, the traveler can change from car to public transportation, or the other way around, in any city. In this context, identify the fastest itinerary from Orbe to any other city.
4. Thanks to the car sharing service, the traveler can change from car to public transportation, or the other way around, but only in Lausanne, Bern, and Fribourg. In this context, identify the fastest itinerary from Orbe to any other city.

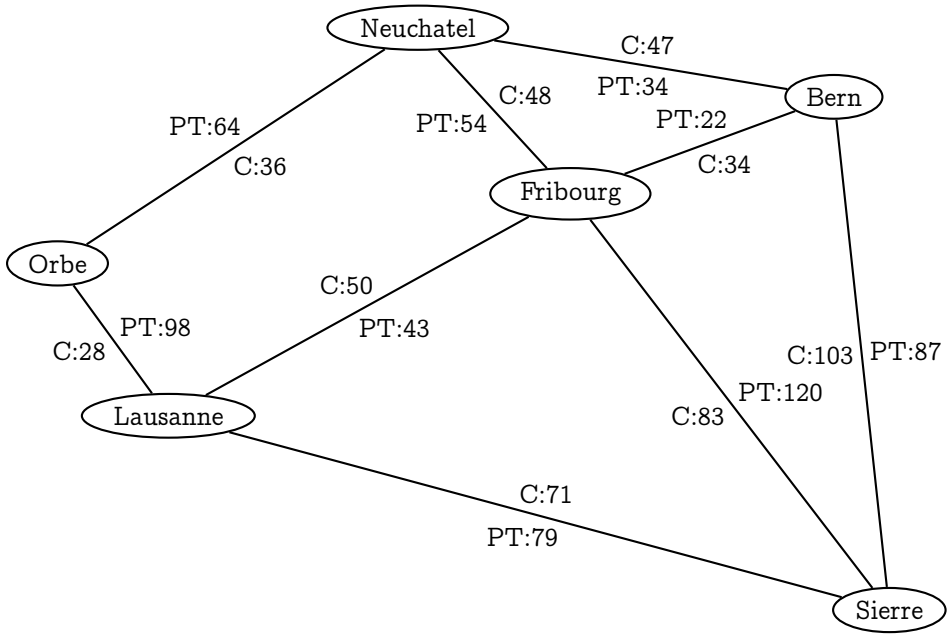


Figure 23.8: Six cities in Switzerland, with the travel time by car (C) and by public transportation (for Exercise 23.5)

Exercise 23.6. In the program evaluation and review technique (PERT) presented in Section 23.4, the longest path problem allows us to identify the earliest possible termination of each task. Design an algorithm to identify the latest possible termination of each task. Hint: start from node d and proceed backward.

Chapter 24

Maximum flow

Contents

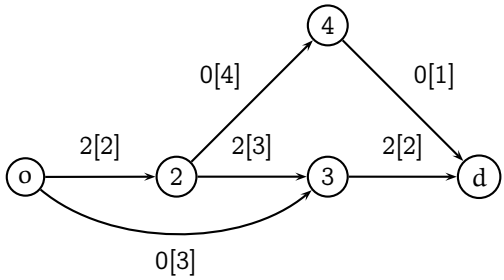
24.1 The Ford-Fulkerson algorithm

The maximum flow problem is defined in Section 22.4.2, where it is shown that it can be modeled as a transhipment problem. In this chapter, we present a dedicated algorithm originally proposed by Ford and Fulkerson (1956).

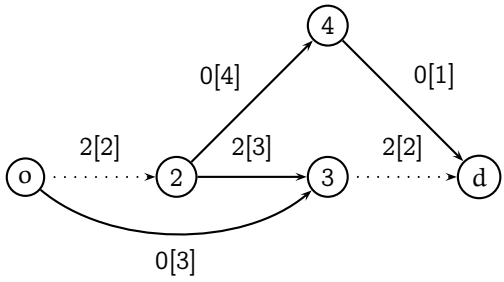
Delbert Ray Fulkerson was born on August 14, 1924, in Tamms, Illinois, USA. He obtained a Ph.D. in Mathematics from the University of Wisconsin in 1951. He then joined the Mathematics Department at the Rand Corporation, where George Dantzig and Richard Bellman were also working. He went to the Operations Research Department of Cornell University in 1971. He died on January 10, 1976, in Ithaca, New-York, USA. Together with Dantzig and Johnson, Fulkerson published on the traveling salesman problem applied to a salesman living in Washington DC and visiting the capitals of the 48 states of the USA. They write about the problem: "The origin of the problem is somewhat obscure. It appears to have been discussed informally among mathematicians and mathematics meetings for many years." They invented the concept of subtour elimination. But Fulkerson is best known for his work on network flows. The Ford-Fulkerson algorithm (Algorithm 24.2) was motivated by a military project aiming at assessing the capacity of the Eastern European rail network to support a conventional war.

Figure 24.1: Delbert Ray Fulkerson

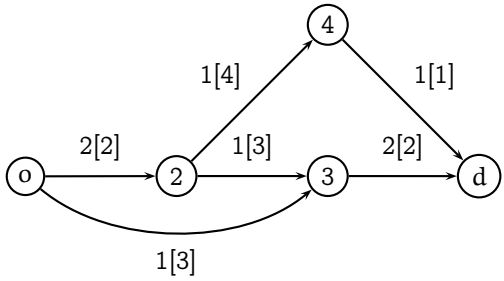
The algorithm is based on two concepts: unsaturated paths and saturated cuts. We illustrate these ideas using the small example presented in Section 22.4.2. As discussed there, we start by sending 2 units of flow on the path $o \rightarrow 2 \rightarrow 3 \rightarrow d$ (the maximum that can be transported) to obtain the flow pattern illustrated in Figure 24.2(a). No more flow can be sent on this path, which is said to be saturated.



(a) 2 units sent



(b) Unsaturated path



(c) 1 more unit sent

Figure 24.2: Finding the maximum flow. On each arc, the flow x_{ij} and the capacity c_{ij} are shown as $x_{ij}[c_{ij}]$

In Section 22.4.2, we discussed the fact that the paths $o \rightarrow 3 \rightarrow d$ and $o \rightarrow 2 \rightarrow 4 \rightarrow d$ are also saturated, as one of their arcs was at capacity. Note that the maximum flow problem imposes lower capacities of 0 so that arcs cannot be followed backward at the final solution. But during the course of the algorithm, nothing prevents us from

considering paths with backward arcs in order to update the current flow vector. In this example, the path $o \rightarrow 3 \leftarrow 2 \rightarrow 4 \rightarrow d$ represented in Figure 24.2(b) is unsaturated, in the sense that some flow can be sent along this path from o to d. Indeed, arc $(o,3)$ can transport up to 3 units of flow. Arc $(2,3)$ is traversed backward. It means that a unit of flow traversing it is *decreasing* the current value, which is 2. As the lower bound is 0, the arc is not at capacity and can transport up to 2 units of flow backward. Arc $(2,4)$ can transport 4 units of flow, and arc $(4,d)$ only 1. Therefore, 1 unit of flow can be sent along this path to obtain the flow pattern represented in Figure 24.2(c). If we decompose the flow (Algorithm 21.3, but it is easy to do it by hand here), we obtain that one unit of flow is sent along path $o \rightarrow 2 \rightarrow 3 \rightarrow d$, one unit along path $o \rightarrow 3 \rightarrow d$, and one unit along path $o \rightarrow 2 \rightarrow 4 \rightarrow d$. Note that it is not the same optimal solution as the one proposed in Section 22.4.2, but it achieves the same objective, that is, transporting 3 units of flow from o to d.

Consider now the cut

$$\Gamma = (\{o, 2, 3, 4\}, \{d\}).$$

We have $\Gamma^{\rightarrow} = \{(3,d), (4,d)\}$, and $\Gamma^{\leftarrow} = \emptyset$. Therefore, the flow through the cut is $X(\Gamma) = 1+2 = 3$, and its capacity is $U(\Gamma) = 1+2 = 3$. The cut is therefore saturated, and there is no way to send more flow from the first set of nodes to the second. As the cut separates o from d, there is therefore no way to send more flow from o to d, and the solution is optimal.

The above example provides the intuition of the concept of a saturated path. The formal definition follows.

Definition 24.1 (Saturated path)**.** Consider a network $(\mathcal{N}, \mathcal{A})$, with m nodes and n arcs, a vector of lower capacities $\ell \in \mathbb{R}^n$, a vector of upper capacities $u \in \mathbb{R}^n$, a feasible flow vector $x \in \mathbb{R}^n$, and a path P. The path P is *saturated* with respect to x if

$$\exists (i,j) \in P^{\rightarrow} \text{ with } x_{ij} = u_{ij}, \quad \text{or } \exists (i,j) \in P^{\leftarrow} \text{ with } x_{ij} = \ell_{ij}. \tag{24.1}$$

The path is said to be *unsaturated* if

$$x_{ij} < u_{ij}, \ \forall (i,j) \in P^{\rightarrow} \text{ and } x_{ij} > \ell_{ij}, \ \forall (i,j) \in P^{\leftarrow}. \tag{24.2}$$

In order to find an unsaturated path, we proceed by layers, similarly to the flow decomposition algorithm presented in Section 21.6. The first layer $\mathcal{S}_0 = \{o\}$ contains only the origin. Layer $\mathcal{S}_t$ is built from layer $\mathcal{S}_{t-1}$ in the following way: node j belongs to layer $\mathcal{S}_t$ if it does not belong to any previous layer $\mathcal{S}_0, \ldots, \mathcal{S}_{t-1}$, and at least one of the two conditions is verified (one condition for forward arcs, one for backward):

1. there is an arc (i,j) such that $i \in \mathcal{S}_{t-1}$ and $x_{ij} < u_{ij}$, or
2. there is an arc (j,i) such that $i \in \mathcal{S}_{t-1}$ and $x_{ji} > \ell_{ji}$.

The recursive procedure is interrupted if d is found or if $\mathcal{S}_t$ is empty. In Algorithm 24.1 when a node is included in a layer (steps 15 and 17), a label is associated, which

records the connected node in the previous layer and the direction of the connecting arc. The notation $j[i \rightarrow]$ means that node j has been reached by following arc (i,j) in the forward direction, while $j[\leftarrow i]$ means that node j has been reached by following arc (j,i) in the backward direction. This is useful to reconstruct a path from o to d, starting from d and following back the track across layers until node o is reached. Step 23 identifies the upstream node of the forward arcs thanks to the labels, and step 24 identifies the downstream node of the backward arcs.

Algorithm 24.1: Generation of an unsaturated path

1 **Objective**
2 Generate a simple path flow along an unsaturated path.
3 **Input**
4 A network $(\mathcal{N},\mathcal{A})$ of m nodes and n arcs.
5 Flow vector $x \in \mathbb{R}^n$, lower bounds $\ell \in \mathbb{R}^n$, upper bounds $u \in \mathbb{R}^n$.
6 Origin o, destination d.
7 **Output**
8 A simple path flow $z \in \mathbb{R}^n$ or a saturated cut.
9 **Initialization**
10 $\mathcal{S}_0 := \{o\}$, $\mathcal{M} := \mathcal{S}_0$, $t := 1$.
11 **Repeat**
12 $\mathcal{S}_t := \emptyset$.
13 **forall** $i \in \mathcal{S}_{t-1}$ **do**
14 **forall** j *such that* $(i,j) \in \mathcal{A}$, $j \notin \mathcal{M}$ *and* $x_{ij} < u_{ij}$ **do**
15 $\mathcal{S}_t := \mathcal{S}_t \cup \{j[i \rightarrow]\}$, $\mathcal{M} := \mathcal{M} \cup \{j\}$
16 **forall** j *such that* $(j,i) \in \mathcal{A}$, $j \notin \mathcal{M}$ *and* $x_{ji} > \ell_{ji}$ **do**
17 $\mathcal{S}_t := \mathcal{S}_t \cup \{j[\leftarrow i]\}$, $\mathcal{M} := \mathcal{M} \cup \{j\}$
18 $t := t+1$.
19 **Until** $d \in \mathcal{S}_{t-1}$ *or* $\mathcal{S}_{t-1} = \emptyset$.
20 **if** $\mathcal{S}_{t-1} = \emptyset$ **then return** $\mathcal{M}$ No unsaturated path.
21 $j := d$, $s := t-1$, $\mathcal{P} := \{d\}$.
22 **Repeat**
23 **if** $j[k \rightarrow]$ **then** $\mathcal{P} := \{k \rightarrow\} \cup \mathcal{P}$, $j := k$
24 **if** $j[\leftarrow k]$ **then** $\mathcal{P} := \{k \leftarrow\} \cup \mathcal{P}$, $j := k$
25 **Until** $j = o$
26 $f := \min\left(\min_{(i,j)\in\mathcal{P}^{\rightarrow}}(u_{ij}-x_{ij}), \min_{(i,j)\in\mathcal{P}^{\leftarrow}}(x_{ij}-\ell_{ij})\right)$.
27 **forall** $(i,j) \in \mathcal{A}$ **do**
28 **if** $(i,j) \in \mathcal{P}^{\rightarrow}$ **then** $z_{ij} = f$
29 **if** $(i,j) \in \mathcal{P}^{\leftarrow}$ **then** $z_{ij} = -f$
30 **if** $(i,j) \notin \mathcal{P}$ **then** $z_{ij} = 0$

If an unsaturated path $\mathcal{P}$ has been found, a flow can be sent along it. As we want to send as much flow as possible, we calculate, for each arc, the maximum additional flow that it can transport. For forward arcs, it is the difference $u_{ij} - x_{ij}$ between the upper bound and the current flow, as sending additional flow along the path increases the flow on this arc. For backward arcs, it is the difference $x_{ij} - \ell_{ij}$ between the current flow and the lower bound, as sending additional flow along the path decreases the flow on this arc. The quantity of flow that is feasible to send along the path is the minimum of these values across all arcs of the path, that is,

$$f := \min\left(\min_{(i,j)\in\mathcal{P}^{\rightarrow}}(u_{ij} - x_{ij}), \min_{(i,j)\in\mathcal{P}^{\leftarrow}}(x_{ij} - \ell_{ij})\right). \tag{24.3}$$

If node d has not been reached by the algorithm that generates the layers, it means that no unsaturated path exists. In this case, the algorithm is interrupted because no further arc with residual capacity can be found. The set $\mathcal{M}$ defines the cut $\Gamma = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$. By design of the algorithm, any arc with one node i in $\mathcal{M}$ and one node j not in $\mathcal{M}$ must be saturated. Indeed, if it were not the case, node j would have been included into a layer by the algorithm and would then be in $\mathcal{M}$. Therefore, the cut Γ is saturated. This is the Ford-Fulkerson algorithm, presented as Algorithm 24.2.

Algorithm 24.2: Ford-Fulkerson algorithm

Objective
  Identify the maximum flow through a network.
Input
  A network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs.
  Upper bounds $u \in \mathbb{R}^n$.
  Origin o, destination d.
Output
  A flow vector $x \in \mathbb{R}^n$ such that $\operatorname{div}(x)_o$ is maximum.
Initialization
  $x := 0$.
Repeat
  Use Algorithm 24.1 with $\ell = 0$ to find a feasible simple path flow $z \in \mathbb{R}^n$
  if *a saturated cut has not been found* **then**
    $x := x + z$
Until *a saturated cut has been found.*

Example 24.2 (Maximum flow)**.** Consider the network represented in Figure 24.3, where the capacities are shown in square brackets. The iterations of the Ford-Fulkerson algorithm are reported in Table 24.1. During each iteration, layers are built in order to identify an unsaturated path. They are described in Table 24.2.

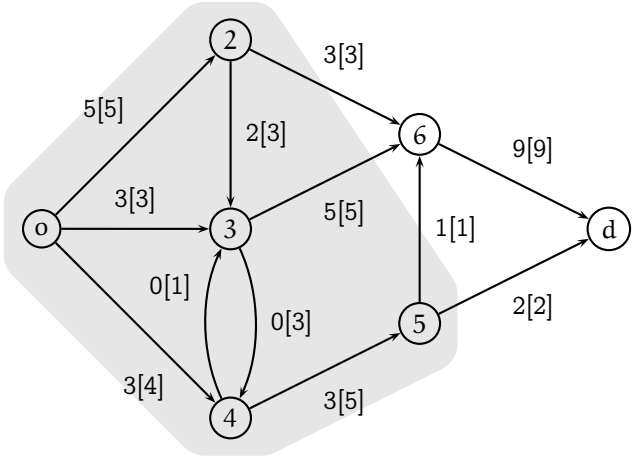


Figure 24.3: Network for Example 24.2, with flows and [capacities]

Table 24.1: Iterations of the Ford-Fulkerson algorithm for Example 24.2

Iter	(o,2)	(o,3)	(o,4)	(2,3)	(2,6)	(3,4)	(3,6)	(4,3)	(4,5)	(5,6)	(5,d)	(6,d)	Flow	
1	0	0	0	0	0	0	0	0	0	0	0	0	3	$o \to 2 \to 6 \to d$
2	3	0	0	0	3	0	0	0	0	0	0	3	3	$o \to 3 \to 6 \to d$
3	3	3	0	0	3	0	3	0	0	0	0	6	2	$o \to 4 \to 5 \to d$
4	3	3	2	0	3	0	3	0	2	0	2	6	2	$o \to 2 \to 3 \to 6 \to d$
5	5	3	2	2	3	0	5	0	2	0	2	8	1	$o \to 4 \to 5 \to 6 \to d$
6	5	3	3	2	3	0	5	0	3	1	2	9		

Note that Algorithm 24.1 identifies an unsaturated path with the minimum possible number of arcs. This is needed to guarantee that the Ford-Fulkerson algorithm converges. Indeed, if another strategy is used, it may fail to terminate when some capacities on the arc are irrational (see Zwick, 1995 for simple examples). The version of the Ford-Fulkerson algorithm presented in this text is not efficient. Indeed, at each iteration, the layers are reconstructed from the beginning. An algorithm proposed by Dinic (1970) is based on a more efficient implementation.

Table 24.2: Constructions of layers during the iterations of the Ford-Fulkerson algorithm for Example 24.2

Iteration 1		
$\mathcal{S}_1$	$=$	$\{1\}$
$\mathcal{S}_2$	$=$	$\{2[1 \rightarrow], 3[1 \rightarrow], 4[1 \rightarrow]\}$
$\mathcal{S}_3$	$=$	$\{6[2 \rightarrow], 5[4 \rightarrow]\}$
$\mathcal{S}_4$	$=$	$\{7[6 \rightarrow]\}$
Iteration 2		
$\mathcal{S}_1$	$=$	$\{1\}$
$\mathcal{S}_2$	$=$	$\{2[1 \rightarrow], 3[1 \rightarrow], 4[1 \rightarrow]\}$
$\mathcal{S}_3$	$=$	$\{6[3 \rightarrow], 5[4 \rightarrow]\}$
$\mathcal{S}_4$	$=$	$\{7[6 \rightarrow]\}$
Iteration 3		
$\mathcal{S}_1$	$=$	$\{1\}$
$\mathcal{S}_2$	$=$	$\{2[1 \rightarrow], 4[1 \rightarrow]\}$
$\mathcal{S}_3$	$=$	$\{3[2 \rightarrow], 5[4 \rightarrow]\}$
$\mathcal{S}_4$	$=$	$\{6[3 \rightarrow], 7[5 \rightarrow]\}$
Iteration 4		
$\mathcal{S}_1$	$=$	$\{1\}$
$\mathcal{S}_2$	$=$	$\{2[1 \rightarrow], 4[1 \rightarrow]\}$
$\mathcal{S}_3$	$=$	$\{3[2 \rightarrow], 5[4 \rightarrow]\}$
$\mathcal{S}_4$	$=$	$\{6[3 \rightarrow]\}$
$\mathcal{S}_5$	$=$	$\{7[6 \rightarrow]\}$
Iteration 5		
$\mathcal{S}_1$	$=$	$\{1\}$
$\mathcal{S}_2$	$=$	$\{4[1 \rightarrow]\}$
$\mathcal{S}_3$	$=$	$\{3[4 \rightarrow], 5[4 \rightarrow]\}$
$\mathcal{S}_4$	$=$	$\{2[\leftarrow 3], 6[5 \rightarrow]\}$
$\mathcal{S}_5$	$=$	$\{7[6 \rightarrow],\}$
Iteration 6		
$\mathcal{S}_1$	$=$	$\{1\}$
$\mathcal{S}_2$	$=$	$\{4[1 \rightarrow]\}$
$\mathcal{S}_3$	$=$	$\{3[4 \rightarrow], 5[4 \rightarrow]\}$
$\mathcal{S}_4$	$=$	$\{2[\leftarrow 3]\}$
$\mathcal{S}_5$	$=$	$\{\}$

24.2 The minimum cut problem

The Ford-Fulkerson algorithm (Algorithm 24.2) terminates when a saturated cut $\Gamma^* = (\mathcal{M}^*, \mathcal{N} \setminus \mathcal{M}^*)$ has been identified. This cut can be seen as the "bottleneck" of the network. Indeed, the arcs of the cut are the only way to connect nodes in $\mathcal{M}^*$ to nodes

not in $\mathcal{M}^*$, and they are all saturated. Using the analogy discussed in Section 21.2, the cut can be seen as a partition of the nodes into those on the left bank and those on the right bank of a river separating the city, and the arcs of the cut as the bridges from one bank to the other. If all bridges are saturated, there is no possibility to move more flow across the river. It happens that, among all the possible cuts in the network, Γ^* has the smallest capacity. It is the optimal solution of the *minimum cut problem*.

Definition 24.3 (Minimum cut problem)**.** Consider a network $(\mathcal{N}, \mathcal{A})$ of m nodes and n arcs, and a vector $u \in \mathbb{R}^n$ representing the capacity of each arc. Consider a node o called the origin (or the source), and a node d called the destination (or the sink). Consider any cut $\Gamma(o, d) = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$, where $o \in \mathcal{M}$ and $d \notin \mathcal{M}$, separating o from d. The minimum cut problem consists in finding, among these cuts, one with the minimum capacity, that is the cut $\Gamma^*(o, d)$ such that

$$U(\Gamma^*(o, d)) \leq U(\Gamma(o, d)) \; \forall \Gamma(o, d). \tag{24.4}$$

The minimum cut problem is intimately related to the maximum flow problem. Actually, both problems are dual to each other. In particular, their optimal value is the same.

Theorem 24.4 (Maximum flow/minimum cut theorem)**.** *Consider the maximum flow problem (Definition 22.10), and an optimal solution x^*. Consider the minimum cut problem (Definition 24.3) and an optimal solution $\Gamma^*(o, d)$. Then,*

$$\operatorname{div}(x*)_o = U(\Gamma^*(o, d)). \tag{24.5}$$

Proof. Consider any arbitrary cut $\Gamma(o, d) = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$ separating o from d. From Theorem 21.14, we have

$$X(\Gamma(o, d)) = \sum_{i \in \mathcal{M}} \operatorname{div}(x^*)_i = \operatorname{div}(x^*)_o,$$

as o and d are the only nodes with a non zero divergence, and d is not in $\mathcal{M}$. From (21.8), we also have

$$\operatorname{div}(x^*)_o = X(\Gamma(o, d)) \leq U(\Gamma(o, d)). \tag{24.6}$$

Now, if we apply Algorithm 24.1, as x^* is optimal, no unsaturated path can be found, and the algorithm stops with a saturated cut that separates o from d. Call it $\Gamma^*(o, d)$. Again, the flow through the cut is

$$X(\Gamma^*(o, d)) = \operatorname{div}(x^*)_o. \tag{24.7}$$

As the cut is saturated, we also have

$$X(\Gamma^*(o, d)) = U^*(\Gamma(o, d)). \tag{24.8}$$

Combining (24.6), (24.7), and (24.8), we obtain

$$U(\Gamma^*(o,d)) \leq U(\Gamma(o,d)).$$

Consequently, $\Gamma^*(o,d)$ is the cut with the minimum capacity, and its capacity is equal to the maximum flow from o to d. □

In order to investigate further the dual relationship between the two problems, we consider the formulation (22.27)–(22.31) of the maximum flow problem as a transhipment problem (see Section 22.4.2). To each supply constraint (22.28), that is to each node i, we associate a dual variable p_i. To each capacity constraint (22.29), that is to each arc (i,j), we associate a dual variable λ_{ij}. Using the techniques presented in Chapter 4, we write the dual problem as follows

$$\min \sum_{(i,j)\in\mathcal{A}} \lambda_{ij} u_{ij} \tag{24.9}$$

subject to

$$\lambda_{ij} - p_i + p_j \geq 0, \qquad \forall (i,j) \in \mathcal{A}, \tag{24.10}$$

$$p_o - p_d \geq 1, \tag{24.11}$$

$$\lambda_{ij} \geq 0, \qquad \forall (i,j) \in \mathcal{A}. \tag{24.12}$$

In the above formulation, the dual variables p are involved only with respect to their differences. Moreover, they do not appear in the objective function. Therefore, if p is feasible, $p + \alpha e$ is also feasible for any $\alpha \in \mathbb{R}$ (e is a vector with all entries equal to 1), and the objective function is not affected. Therefore, without loss of generality, one of the dual variables can be normalized to an arbitrary value. Later, we propose $p_o = 1$. Note that the matrix of the constraints of the dual problem is the transpose of the matrix from the primal, which is totally unimodular by Theorem 22.6. The determinant of each square submatrix is 0, -1, or $+1$. This holds as well for the transposed matrix, which is also totally unimodular. Therefore, Theorem 22.4 applies for the dual. As the right hand side of the constraints involves only 0 and 1, there is an integer optimal solution even if the capacities are not integer.

We now show the relationship between this formulation and the minimum cut problem. The following result shows how to generate the values of the dual variables given a cut.

Lemma 24.5. *Consider the minimum cut problem and an arbitrary cut* $\Gamma(o,d) = (\mathcal{M}, \mathcal{N} \setminus \mathcal{M})$ *separating* o *from* d. *Consider the vector* $p \in \mathbb{R}^m$ *defined as*

$$p_i = \begin{cases} 1 & \text{if } i \in \mathcal{M}, \\ 0 & \text{otherwise}, \end{cases} \tag{24.13}$$

and the vector $\lambda \in \mathbb{R}^n$ *defined as*

$$\lambda_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \Gamma^{\rightarrow}, \\ 0 & \text{otherwise.} \end{cases} \tag{24.14}$$

Then, p and λ verify the constraints (24.10)–(24.12), and the objective function (24.9) is the capacity of the cut.

Proof.
- The constraints (24.12) are verified by definition of λ.
- Consider an arc (i,j) where i and j are both in $\mathcal{M}$, or both not in $\mathcal{M}$. In this case, $p_i = p_j$ and the constraints (24.10) become $\lambda_{ij} \geq 0$, and are verified.
- Consider an arc (i,j) where i is in $\mathcal{M}$ and j not. Therefore $p_i = 1$ and $p_j = 0$. As (i,j) is an arc of the cut, we also have $\lambda_{ij} = 1$, so that (24.10) is written as $1 - 1 + 0 \geq 0$, and is verified.
- From (24.13), $p_o = 1$ and $p_d = 0$, and (24.11) is verified.
- From (21.7), $U(\Gamma) = \sum_{(i,j)\in\Gamma^\rightarrow} u_{ij}$. From the definition of λ, we have

$$U(\Gamma) = \sum_{(i,j)\in\mathcal{A}} \lambda_{ij} u_{ij}.$$

□

From a feasible dual solution, many cuts can be generated.

Lemma 24.6. *Consider the minimum cut problem. Consider also a vector $p \in \mathbb{R}^m$ and a vector $\lambda \in \mathbb{R}^n$ verifying the constraints (24.10)–(24.12). For any $0 \leq \gamma < 1$, consider the set*

$$\mathcal{M}_\gamma = \{j | p_o - p_j \leq \gamma\}. \tag{24.15}$$

Then, $\Gamma_\gamma = (\mathcal{M}_\gamma, \mathcal{N} \setminus \mathcal{M}_\gamma)$ is a cut separating o from d. Also, there exists γ^* such that

$$U(\Gamma_{\gamma^*}) \leq \sum_{(i,j)\in\mathcal{A}} \lambda_{ij} u_{ij}. \tag{24.16}$$

Proof. Node o is trivially in $\mathcal{M}_\gamma$ for any $0 \leq \gamma < 1$. From (24.10), $p_o - p_d \geq 1$, and d cannot be in $\mathcal{M}_\gamma$ for any $0 \leq \gamma < 1$. Therefore, the cut $\mathcal{M}_\gamma$ separates o from d, for any $0 \leq \gamma < 1$. For the second result, we need a probability argument.

Consider a random variable X uniformly distributed between 0 and 1. For each realization γ of X, we can generate a cut Γ_γ. The expected value of the capacity of this cut is

$$E[U(\Gamma_X)] = \sum_{(i,j)\in\mathcal{A}} u_{ij} \Pr\left((i,j) \in \Gamma_X^\rightarrow\right).$$

We show now that, for each (i,j), the probability that the arc is in the cut is bounded by the value of the dual variable, that is

$$\Pr\left((i,j) \in \Gamma_X^\rightarrow\right) \leq \lambda_{ij}. \tag{24.17}$$

The probability that (i,j) is in the cut is the probability that i is in $\mathcal{M}_X$ and j is not. By definition (24.15), we have

$$\Pr\left((i,j) \in \Gamma_X^{\rightarrow}\right) = \Pr(p_o - p_i \leq X < p_o - p_j).$$

If the inequalities are not compatible, that is if $p_o - p_j \leq p_o - p_i$, $p_o - p_1 > 1$, or $p_o - p_j \leq 0$, the probability is 0. Then (24.17) results from the fact that $\lambda_{ij} \geq 0$, that is from (24.12). If they are compatible, then, as X is uniformly distributed,

$$\Pr(p_o - p_j < X \leq p_o - p_i) = p_i - p_j,$$

and (24.17) holds from (24.10). Therefore,

$$\mathrm{E}[U(\Gamma_X)] \leq \sum_{(i,j)\in\mathcal{A}} u_{ij}\lambda_{ij}.$$

By definition of the expected value, there always exists a value $0 \leq \gamma^* < 1$ such that the realization is not greater than the expected value, that is $U(\Gamma_{\gamma^*}) \leq \mathrm{E}[U(\Gamma_X)]$, proving the result. □

We have now all the elements to show that the linear optimization problem (24.9)–(24.12) is the minimum cut problem.

Theorem 24.7 (Minimum cut problem as a linear optimization). *Consider the minimum cut problem and a cut $\Gamma^*(o,d) = (\mathcal{M}^*, \mathcal{N} \setminus \mathcal{M}^*)$ separating o from d. Consider the vector $p^* \in \mathbb{R}^m$ defined as*

$$p_i^* = \begin{cases} 1 \text{ if } i \in \mathcal{M}^*, \\ 0 \text{ otherwise,} \end{cases} \tag{24.18}$$

and the vector $\lambda^* \in \mathbb{R}^n$ defined as

$$\lambda_{ij}^* = \begin{cases} 1 \text{ if } (i,j) \in \Gamma^{\rightarrow}, \\ 0 \text{ otherwise.} \end{cases} \tag{24.19}$$

The cut $\Gamma^*(o,d)$ solves the minimum cut problem if and only if (p^*, λ^*) is the optimal solution of the linear optimization problem (24.9)–(24.12):

$$\min \sum_{(i,j)\in\mathcal{A}} \lambda_{ij} u_{ij}$$

subject to

$$\begin{aligned} \lambda_{ij} - p_i + p_j &\geq 0, && \forall (i,j) \in \mathcal{A}, \\ p_o - p_d &\geq 1, \\ \lambda_{ij} &\geq 0, && \forall (i,j) \in \mathcal{A}. \end{aligned}$$

Proof. If $\mathcal{M}^*$ is the optimal cut, we invoke Lemma 24.5, that states that (p^*, λ^*) is feasible, and that the objective function (that is, the capacity of the cut) is minimal.

For the other direction, we exploit the fact that there is an integer optimal solution to (24.9)–(24.12). Also, as discussed earlier, only the differences of the dual variables matter, so that one of them can be normalized to an arbitrary value. Therefore, we can assume without loss of generality that $p_o^* = 1$.

Consider now Lemma 24.6 and the definition of the set $\mathcal{M}_\gamma$. As p is integer, $p_o = 1$, and $\gamma < 1$, the condition $p_o - p_j \leq \gamma$ to be in the set becomes $p_j \geq 1$ for any value of γ. Similarly, as $\gamma \geq 0$, the condition $p_o - p_j > \gamma$ to be out of the set becomes $p_j \leq 0$ for any value of γ. Therefore, for any value of γ, the set $\mathcal{M}_\gamma$ of Lemma 24.6 is

$$\mathcal{M}_\gamma = \{i | p_i^* \geq 1\} = \{i | p_i^* = 1\},$$

which is exactly the set $\mathcal{M}^*$. Therefore, (24.16) is verified for $\mathcal{M}^*$ and

$$U(\Gamma^*) \leq \sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^* u_{ij}.$$

From the strong duality theorem (Theorems 4.17 and 6.32), we know that the optimal value of the dual problem is the same as the primal. Therefore, $\sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^* u_{ij}$ has the same value as the maximum flow problem which, from Theorem 24.4, is the optimal value of the minimum cut problem, so that

$$U(\Gamma^*) \geq \sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^* u_{ij}.$$

Consequently, we have

$$U(\Gamma^*) = \sum_{(i,j)\in\mathcal{A}} \lambda_{ij}^* u_{ij},$$

showing that Γ^* is a minimum cut. □

The material presented in Chapters 21 to 24 is based on lecture notes, inspired by Ahuja et al. (1993) and Bertsekas (1998).

24.3 Exercises

Exercise 24.1. Consider the network represented in Figure 24.4, where the lower capacity of each arc is 0, and the upper capacity is shown next to the arc.

1. Apply the Ford-Fulkerson algorithm (Algorithm 24.2) to obtain the maximum flow through the network from node o to node d, as a function of the parameter α, where $\alpha \geq 0$.
2. What are the values of α, $\alpha \geq 0$, such that arc (b, a) belongs to the minimum cut? And what is the minimum cut?

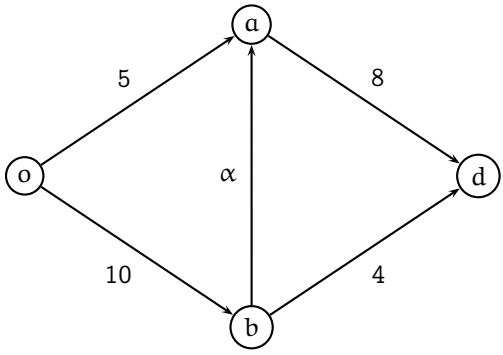


Figure 24.4: Network for Exercise 24.1, where the value of each arc represents its upper capacity

Exercise 24.2. Consider the network represented in Figure 21.25, where each arc (i, j) is associated with its lower bound ℓ_{ij}, its flow x_{ij}, and its upper bound u_{ij} in the following way: $(\ell_{ij}, x_{ij}, u_{ij})$.

1. Apply the Ford-Fulkerson algorithm (Algorithm 24.2) to obtain the maximum flow through the network from node o to node d.
2. Write the mathematical formulation of the minimum cut problem on this network. Solve it using the simplex algorithm (Algorithm 16.5), and verify that Theorem 24.7 applies in this case.

Part VII

Discrete optimization

The Good Lord made all the integers; the rest is man's doing.

Leopold Kronecker

To address continuous optimization problems, both from a theoretical and an algorithmic viewpoint, until now we have relied on the solid theories of analysis. But not every problem can be modeled using continuous and differentiable functions. In this part of the book, we focus on problems where the decision variables must take integer values, or where their values represent the decision to take or not some actions. We have already encountered such problems in the context of networks, as discussed in the previous part of the book. For instance, in the assignment problem discussed in Section 22.4.4, the actions consist in selling a masterpiece to potential buyers. In that case, the mathematical properties of the network optimization problem guarantee the integrality of the solution without any specific treatment. This does not always apply. We now address the category of problems where integrality must be enforced in order to generate meaningful results.

Chapter 25

Introduction to discrete optimization

Contents

25.1 Modeling

We focus now on optimization problems where the variables are discrete and take integer values. Such constraints are relevant in contexts where the decisions to be taken concern a number of items, or entities, that are indivisible. The example presented in Section 1.1.7 involves the production of toys, where it is not an option to produce parts of toys. It is also relevant in situations where the set of feasible solutions consists in all possible combinations of values of several discrete variables. In this case, it is referred to as "combinatorial optimization." Other examples are presented in Section 25.2.

A particularly useful type of variables in this context are binary variables, which can only take the value 0 or 1. A value 1 may refer to an action that is taken, a decision to do something, or a switch that is set to "on." The value 0 corresponds to an action not taken, a decision not to do something, a switch set to "off". Binary variables provide a powerful modeling tool to translate logical conditions into a mathematical formulation that can be used in an optimization framework. The constraints of the optimization problem correspond to logical conditions, and a solution is feasible if

all these conditions are verified. The possibilities are endless, and the modeling of optimization problems is more of an art than a science. We provide below a few examples of common modeling techniques using binary variables. In order to illustrate each of them, we consider the modeling of the following problem.

Example 25.1 (Locating plants for the supply of energy)**.** A company delivers gas and electricity, and has to decide on the location of its plants. n sites have been identified to locate the plants to cover the distribution to m cities that are clients of the company. For each potential plant i, there is a decision of the local authority if it is allowed to open the plant or not. Moreover, there is a fixed cost p_i to open it. There is also a cost of p_i^g for each unit of gas produced. The unitary production cost for electricity is p_i^e. For each city j, we have the following data:

- $g_j = 1$ if city j buys gas from the company, 0 otherwise,
- d_j^g is the quantity of gas needed by city j,
- $e_j = 1$ if city j buys electricity from the company, and 0 otherwise,
- d_j^e is the quantity of electricity needed by city j.

The cost of infrastructure (wires, pipes, etc.) to allow the transport of the gas (resp. the electricity) from plant i to city j is c_{ij}^g (resp. c_{ij}^e). Finally, each plant must be dedicated to either gas or electricity, but not both. The objective of the problem is to identify what plants must be opened, to assign each of them to either gas or electricity, and to decide the quantity to be produced and the city to be served.

More specifically, consider an instance with $n = 10$ potential sites (all approved by the authorities) and $m = 3$ cities. The data is reported in Table 25.1.

Table 25.1: Data for Example 25.1

Plants		1	2	3	4	5	6	7	8	9	10
p_i		1	1	1	1	1	1	1	1	1	1
p_i^g		1	1	1	2	2	1	1	1	2	2
p_i^e		2	2	2	1	1	2	1	2	2	2
c_{ij}^g	City 1	5	4	3	2	1	5	4	3	2	1
	City 2	2	2	2	2	2	1	2	2	2	2
	City 3	1	2	3	4	5	1	2	3	4	5
c_{ij}^e	City 1	5	4	3	8	1	5	4	3	2	1
	City 2	2	2	2	10	12	1	2	2	2	2
	City 3	1	2	3	1	7	1	2	3	4	5

Cities	1	2	3
g_j	1	0	1
e_j	1	1	1
d_j^g	50	0	30
d_j^e	30	20	10

A solution consists in opening plants 6 (30 units) and 8 (50 units) for gas, and plants 4 (10 units), 5 (30 units), and 7 (20 units) for electricity. City 1 receives gas from plant 8 and electricity from plant 5. City 2 receives electricity from plant 7. City 3 receives gas from plant 6 and electricity from plant 4. The total cost is 153. It happens to be an optimal solution.

We start by presenting how the common logical operations translate into binary variables or constraints for the optimization problem.

Logical identity As described above, a logical identity is characterized by a binary decision variable x that takes the value 1 if the proposition P is true, and 0 if it is false, as formalized in the following truth table:

P	x
True	1
False	0

Typically, this associates the logical propositions with binary parameters and variables. Considering Example 25.1, we may have the following:

- a_i is a binary parameter that is 1 if the local authorities allow to open a plant on site i, and 0 otherwise;
- x_i is a binary variable that is 1 if it is decided to open plant i, and 0 otherwise;
- y_i^g is a binary variable that is 1 if plant i is dedicated to gas, and 0 otherwise;
- y_i^e is a binary variable that is 1 if plant i is dedicated to electricity, and 0 otherwise;
- z_{ij}^g is a binary variable that is 1 if plant i serves gas to city j, and 0 otherwise;
- z_{ij}^e is a binary variable that is 1 if plant i serves electricity to city j, and 0 otherwise.

Logical negation If a proposition P is characterized by a binary variable x, the negation $\neg P$ is true if P is false, and false if P is true. It is characterized by the binary variable $z = 1 - x$, as shown in the following truth table:

P	$\neg P$	x	$1 - x$
True	False	1	0
False	True	0	1

In the above example, the quantity $1 - x_i$ is associated with the proposition "plant i is not opened."

Logical conjunction If we have two propositions P and Q characterized by two binary variables x and y, their conjunction $P \wedge Q$ is true if both P and Q are true, and false otherwise. It is characterized by the binary variable $z = xy$, as shown in the following truth table:

P	Q	$P \wedge Q$	x	y	xy
True	True	True	1	1	1
True	False	False	1	0	0
False	True	False	0	1	0
False	False	False	0	0	0

In our example, a plant is available on site i if the local authorities allow its construction (that is, $a_i = 1$) and if the company decides to construct it (that is, $x_i = 1$). The availability of the plant can therefore be modeled by a binary variable $x_i^a = a_i x_i$. Note that in general, this type of condition can be used to preprocess the problem and simplify its definition. In this example, it is easier simply to ignore all the sites that have not received the authorization and work only with the remaining ones. This is what we assume in the remaining of the presentation of the example.

When this formulation is used with two variables, it introduces a non linear relationship between the two variables: $xy = 1$, which is in general undesirable. In the context of optimization, it is more appropriate to model the conjunction as two constraints ($x = 1$ and $y = 1$), as all constraints have to be verified to achieve feasibility.

Therefore, most of the time it is not necessary to model explicitly the conjunction by a product. We have included it for the sake of completeness.

Logical disjunction If we have two propositions P and Q characterized by two binary variables x and y, their disjunction $P \vee Q$ is true if P or Q is true, and is false if both are false. It is characterized by the constraint $x + y \geq 1$, as shown in the following truth table:

P	Q	$P \vee Q$	x	y	$x + y \geq 1$
True	True	True	1	1	Yes
True	False	True	1	0	Yes
False	True	True	0	1	Yes
False	False	False	0	0	No

This can be generalized to several propositions $P_1, \ldots, P_r$, characterized by binary variables $x_1, \ldots, x_r$. The disjunction $P_1 \vee \ldots \vee P_r$ is true if at least one of the propositions $P_1, \ldots, P_r$ is true. It is characterized by the constraint

$$\sum_{i=1}^{r} x_i \geq 1. \tag{25.1}$$

Note that if the variable $z = x + y$ or the variable $z = \sum_{i=1}^{r} x_i$ is included in the model, they are not binary variables.

In our example, each city must receive its gas by plant 1, or plant 2, or ..., or plant n. This is modeled as

$$\sum_{i=1}^{n} z_{ij}^g \geq 1, \quad j = 1, \ldots, m. \tag{25.2}$$

Logical exclusive disjunction If we have two propositions P and Q characterized by two binary variables x and y, their exclusive disjunction $P \oplus Q$ is true if P or Q is true, but not both. It is characterized by the constraint $x + y = 1$, as shown in the following truth table:

P	Q	$P \oplus Q$	x	y	$x + y = 1$
True	True	False	1	1	No
True	False	True	1	0	Yes
False	True	True	0	1	Yes
False	False	False	0	0	No

Logical implication If we have two propositions P and Q characterized by two binary variables x and y, their implication $P \Rightarrow Q$ is true if Q is true or P is false. It is characterized by the constraint $x \leq y$, as shown in the following truth table:

P	Q	$P \Rightarrow Q$	x	y	$x \leq y$
True	True	True	1	1	Yes
True	False	False	1	0	No
False	True	True	0	1	Yes
False	False	True	0	0	Yes

In our example, the fact that plant i produces gas (characterized by y_i^g) implies that plant i is open (characterized by x_i). This is modeled as

$$y_i^g \leq x_i, \ \forall i. \tag{25.3}$$

Similarly, the fact that a plant serves gas to a city obviously implies that it produces gas. We obtain the constraints

$$z_{ij}^g \leq y_i^g, \ \forall i, j. \tag{25.4}$$

Logical equivalence If we have two propositions P and Q characterized by two binary variables x and y, their equivalence $P \Leftrightarrow Q$ is true if both propositions have the same truth value. It is therefore characterized by the constraint $x = y$, as shown in the following truth table:

P	Q	$P \Leftrightarrow Q$	x	y	$x = y$
True	True	True	1	1	Yes
True	False	False	1	0	No
False	True	False	0	1	No
False	False	True	0	0	Yes

Optional constraints (I) Consider an optimization problem with the constraint $f(x) \geq a$, where $a > 0$. We need to model the fact that the constraint must sometimes be verified, sometimes not. More specifically, there is a binary variable z in the problem such that the constraint $f(x) \geq a$ must be verified if $z = 1$. If $z = 0$, it does not matter if it is verified or not. In order to model this, we need a lower bound on the value of $f(x)$. Without loss of generality,[1] we can assume that

[1] If the lower bound is $\ell < 0$, consider the function $\tilde{f}(x) = f(x) - \ell$ which is such that $\tilde{f}(x) \geq 0$.

$f(x) \geq 0$ for each feasible x. In this case, we write

$$f(x) \geq az. \tag{25.5}$$

Indeed, if $z = 1$, (25.5) becomes $f(x) \geq a$, and the original constraint applies. If $z = 0$, a does not play a role anymore, and (25.5) is written $f(x) \geq 0$, which is always verified.

In our example, if city j buys gas from the company (that is, if $g_j = 1$), it must be served by at least one plant, that is

$$\sum_i z_{ij}^g \geq 1. \tag{25.6}$$

As $\sum_i z_{ij}^g$ is always non negative, it plays the role of $f(x)$ in the above discussion. Setting $a = 1$ and $z = g_j$, (25.5) is written as

$$\sum_i z_{ij}^g \geq g_j. \tag{25.7}$$

This technique is also illustrated in Section 25.2.3.

Optional constraints (II) The previous modeling technique can be used for lower-than inequality constraints, too. In this case, an upper bound is needed. Suppose that we have a function g such that $g(x) \leq M$ for each x. We have a binary variable z, and we want to model the fact that, if $z = 1$, the constraint $g(x) \leq b$, where $b < M$, must be verified. If $z = 0$, it does not matter if the constraint is verified or not. It can be modeled using the constraint

$$g(x) \leq bz + (1 - z)M. \tag{25.8}$$

Indeed, if we define $f(x) = M - g(x)$ and $a = M - b$, we obtain the same configuration as in the previous case, and (25.5) is equivalent to (25.8). This technique is sometimes called the "big-M" model.

In our example, if a plant is not dedicated to gas, then the quantity of gas produced must be equal to zero. Denote $q_i^g \geq 0$ the variables characterizing the quantity of gas produced by plant i. As the variable q_i^g must be constrained to be non negative in any case, the optional constraint $q_i^g = 0$ can be written as $q_i^g \leq 0$. We need now to find a value M such that $q_i^g \leq M$ in any circumstance. As there is no point producing more than needed, the quantity of gas produced by any plant never exceeds the total demand of gas. Therefore, we can define

$$M = \sum_{j=1}^{m} g_j d_j^g. \tag{25.9}$$

The constraint $q_i^g \leq 0$ must be verified if plant i is not dedicated to gas, that is if $y_i^g = 0$. Therefore, we apply (25.8) with $g(x) = q_i^g$, $b = 0$, $z = 1 - y_i^g$ and obtain

$$q_i^g \leq y_i^g \sum_{j=1}^{m} g_j d_j^g. \tag{25.10}$$

Disjunctive constraints The modeling techniques seen above can be generalized to model disjunctive constraints. Suppose that we have two functions f and g, such that $f(x) \geq 0$ and $g(x) \geq 0$ for each x. We need to model that one of the two constraints

$$f(x) \geq a \text{ or } g(x) \geq b \tag{25.11}$$

must be verified, but not necessarily both. We introduce a variable z that is 1 if the first constraint is enforced, and 0 if it is the second one. In that case, (25.11) can be replaced by

$$f(x) \geq az \text{ and } g(x) \geq b(1-z). \tag{25.12}$$

Indeed, if $z = 1$, (25.12) is written as

$$f(x) \geq a \text{ and } g(x) \geq 0,$$

which is verified if $f(x) \geq a$, as the second term is always verified. Similarly, if $z = 0$, (25.12) is written as

$$f(x) \geq 0 \text{ and } g(x) \geq b,$$

which is verified if $g(x) \geq b$, as the first term is always verified.

Linearization As we discuss later, it is highly desirable to have a specification of the model that is linear in the variables. A non linear specification that happens often in practice is

$$xy = z, \tag{25.13}$$

where x, y, and z are binary variables. This non linear constraint is equivalent to the following set of linear constraints:

$$\begin{aligned} x + y &\leq 1 + z \\ z &\leq x \\ z &\leq y, \end{aligned} \tag{25.14}$$

as can be seen from the following truth table:

x	y	z	$x+y \leq 1+z$	$z \leq x$	$z \leq y$	$xy = z$
1	1	1	Yes	Yes	Yes	Yes
1	1	0	No	Yes	Yes	No
1	0	1	Yes	Yes	No	No
1	0	0	Yes	Yes	Yes	Yes
0	1	1	Yes	No	Yes	No
0	1	0	Yes	Yes	Yes	Yes
0	0	1	Yes	No	No	No
0	0	0	Yes	Yes	Yes	Yes

One way to model Example 25.1 is as follows.

Decision variables:

- x_i is a binary variable that is 1 if it is decided to open plant i, and 0 otherwise;
- y_i^g is a binary variable that is 1 if plant i is dedicated to gas, and 0 otherwise;
- y_i^e is a binary variable that is 1 if plant i is dedicated to electricity, and 0 otherwise;
- z_{ij}^g is a binary variable that is 1 if plant i serves gas to city j, and 0 otherwise;
- z_{ij}^e is a binary variable that is 1 if plant i serves electricity to city j, and 0 otherwise;
- $q_i^g \in \mathbb{R}$ represents the quantity of gas to be produced by plant i;
- $q_i^e \in \mathbb{R}$ represents the quantity of electricity to be produced by plant i.

Objective function: the costs involved in this problem are

- the fixed costs associated with the opening of the plants: $\sum_{i=1}^{n} p_i x_i$,
- the production costs of gas: $\sum_{i=1}^{n} p_i^g q_i^g$,
- the production costs of electricity: $\sum_{i=1}^{n} p_i^e q_i^e$,
- the cost of the transportation infrastructure for gas

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}^g z_{ij}^g, \tag{25.15}$$

  as z_{ij}^g is 1 if gas has to be delivered to city j from plant i, and
- the cost of the transportation infrastructure for electricity

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}^e z_{ij}^e. \tag{25.16}$$

Therefore, the objective function is

$$\sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n} p_i^g q_i^g + \sum_{i=1}^{n} p_i^e q_i^e + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}^g z_{ij}^g + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}^e z_{ij}^e. \tag{25.17}$$

Constraints:

- If plant i produces gas, then plant i is open:

$$y_i^g \leq x_i \quad \forall i. \tag{25.18}$$

- If plant i produces electricity, then plant i is open:

$$y_i^e \leq x_i \quad \forall i. \tag{25.19}$$

- If plant i produces gas, it cannot produce electricity. Using the logical implication and the logical negation, we obtain $y_i^g \leq 1 - y_i^e$. Similarly, if plant i produces electricity, it cannot produce gas, which gives $y_i^e \leq 1 - y_i^g$. Both constraints are equivalent and can be written as

$$y_i^g + y_i^e \leq 1, \quad \forall i. \tag{25.20}$$

- Plant i must produce gas in sufficient quantity to satisfy the total demand associated with it. The demand for city j is $d_j^g g_j$, that is d_g if city j buys gas from the company:
$$q_i^g \geq \sum_j d_j^g g_j z_{ij}^g. \tag{25.21}$$
- Plant i must produce electricity in sufficient quantity to satisfy the total demand associated with it. The demand for city j is $d_j^e e_j$, that is d_e if city j buys electricity from the company:
$$q_i^e \geq \sum_j d_j^e e_j z_{ij}^e. \tag{25.22}$$
- If plant i is not dedicated to gas, then the quantity of gas produced must be equal to zero. From the discussion above, the constraint is (25.10):
$$q_i^g \leq y_i^g \sum_{j=1}^{m} g_j d_j^g, \quad \forall i. \tag{25.23}$$
- If plant i is not dedicated to electricity, then the quantity of electricity produced must be equal to zero:
$$q_i^e \leq y_i^e \sum_{j=1}^{m} e_j d_j^e, \quad \forall i. \tag{25.24}$$
- If city j buys gas from the company (that is, if $g_j = 1$), it must be served by at least one plant. As discussed above, the constraint is (25.7):
$$\sum_i z_{ij}^g \geq g_j, \quad \forall j. \tag{25.25}$$
- If city j buys electricity from the company (that is, if $e_j = 1$), it must be served by at least one plant:
$$\sum_i z_{ij}^e \geq e_j, \quad \forall j. \tag{25.26}$$
- If city j receives gas from plant i, it implies that plant i produces gas:
$$z_{ij}^g \leq y_i^g, \quad \forall i, j. \tag{25.27}$$
- If city j receives electricity from plant i, it implies that plant i produces electricity:
$$z_{ij}^e \leq y_i^e, \quad \forall i, j. \tag{25.28}$$
- If city j receives gas from plant i, it implies that city j buys gas from the company:
$$z_{ij}^g \leq g_j, \quad \forall i, j. \tag{25.29}$$
- If city j receives electricity from plant i, it implies that city j buys electricity from the company:
$$z_{ij}^e \leq e_j, \quad \forall i, j. \tag{25.30}$$

- Variables x_i, y_i^g, y_i^e, z_{ij}^g, and z_{ij}^e take the value 0 or 1.
- Variables q_i^g and q_i^e are non negative real numbers.

Putting everything together, the optimization problem is written as follows.

$$\min \sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n} p_i^g q_i^g + \sum_{i=1}^{n} p_i^e q_i^e + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}^g z_{ij}^g + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}^e z_{ij}^e.$$

subject to

$$\begin{aligned}
y_i^g &\le x_i \quad \forall i,\\
y_i^e &\le x_i \quad \forall i,\\
y_i^g + y_i^e &\le 1, \quad \forall i,\\
q_i^g &\ge \sum_j d_j^g g_j z_{ij}^g,\\
q_i^e &\ge \sum_j d_j^e e_j z_{ij}^e,\\
q_i^g &\le y_i^g \sum_{j=1}^{m} d_j^g, \quad \forall i,\\
q_i^e &\le y_i^e \sum_{j=1}^{m} d_j^e, \quad \forall i,\\
\sum_i z_{ij}^g &\ge g_j, \quad \forall j,\\
\sum_i z_{ij}^e &\ge e_j, \quad \forall j,\\
z_{ij}^g &\le y_i^g, \quad \forall i,j,\\
z_{ij}^e &\le y_i^e, \quad \forall i,j,\\
z_{ij}^g &\le g_j, \quad \forall i,j,\\
z_{ij}^e &\le e_j, \quad \forall i,j,\\
x_i, y_i^g, y_i^e, z_{ij}^g, z_{ij}^e &\in \{0,1\}, \quad \forall i,j,\\
q_i^g, q_i^e &\ge 0, \quad \forall i.
\end{aligned}$$

Note that the above formulation is certainly not the only possible way to model the problem, and probably not the best one. Several simplifications can be done (for instance, constraints (25.29) and (25.30) can be used to reduce the number of variables). Still it illustrates various aspects of modeling that appear in many applications.

Once the modeling step has been finalized, we obtain a mixed integer optimization problem.

Definition 25.2 (Integer optimization problem)**.** An optimization problem is an *integer* optimization problem if all of its variables are restricted to take integer values. If some variables are allowed to take non integer values, the optimization problem is called a *mixed integer optimization problem*.

In this book, we focus only on integer linear optimization problems.

Definition 25.3 (Integer linear optimization problem)**.** An optimization problem is an *integer linear* optimization problem if the objective function and the constraints are linear functions of the decision variables, and if all of its variables are restricted to take integer values. If some variables are allowed to take non integer values, the optimization problem is called a *mixed integer linear optimization problem*. Using the techniques described in Section 1.2, such a problem can always be written as

$$\min_{x \in \mathbb{R}^{n_x}, z \in \mathbb{N}^{n_z}} c_x^T x + c_z^T z \tag{25.31}$$

subject to

$$\begin{aligned} A_x x + A_z z &= b \\ x &\geq 0 \\ z &\in \mathbb{N}^{n_z}, \end{aligned} \tag{25.32}$$

where $A_x \in \mathbb{R}^{m \times n_x}$, $A_z \in \mathbb{R}^{m \times n_z}$ and $b \in \mathbb{R}^m$.

The special case where all variables are binary is called a *binary linear optimization problem*.

Definition 25.4 (Binary linear optimization problem)**.** An optimization problem is a *binary linear* optimization problem if the objective function and the constraints are linear functions of the decision variables, and if all the variables are restricted to take the values 0 or 1.

Such a problem can be written as

$$\min_{x \in \mathbb{N}^n} c^T x \tag{25.33}$$

subject to

$$\begin{aligned} Ax &= b \\ x &\in \{0,1\}^n, \end{aligned} \tag{25.34}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Note that it is possible to transform an integer variable into several binary variables, if the variable is bounded and can take only a finite number of values. Indeed, any number can be converted into binary notation. This is exactly what happens in a computer anyway. More specifically, consider an integer variable x that can take

any value up to u, that is $x \in \{0, 1, \ldots, u\}$. We define K binary variables z_i, where K is the smallest integer such that

$$u \leq 2^K - 1, \tag{25.35}$$

that is

$$K = \lceil \log_2(u+1) \rceil. \tag{25.36}$$

Then, we replace x by

$$\sum_{i=0}^{K-1} 2^i z_i. \tag{25.37}$$

Note that this transformation may allow x to exceed u. Indeed, suppose that u is 8. Then $K = 4$ and 4 variables are used. If they are all set to 1, the corresponding value is 15, which is above 8. Therefore, the constraint

$$\sum_{i=0}^{K-1} 2^i z_i \leq u \tag{25.38}$$

must also be included.

Another possible way to transform an integer variable into binary variables is as follows. Assume that x is an integer variable such that $\ell \leq x \leq u$. We introduce $u - \ell$ binary variables z_i, $i = 1, \ldots, u - \ell$, and define

$$x = \ell + \sum_{i=1}^{u-\ell} z_i. \tag{25.39}$$

If all variables z_i are 0, the value of x is ℓ. If all variables z_i are 1, the value of x is $\ell + (u - \ell) = u$. Any other combination of 0 and 1 for the variables z_i corresponds to an integer value of x between ℓ and u. There is a major shortcoming to this approach, though. Indeed, the same value of x may correspond to several combinations of the variables z_i. Consider an example where $\ell = 3$ and $u = 6$. We introduce 3 binary variables z_1, z_2, and z_3, and we associate each combination with a value of x as represented in Table 25.2.

The values 1 and 2 can be represented in 3 different ways each. It artificially increases the size of the feasible set. In order to avoid that, additional constraints must be introduced. For instance, we may impose the use of binary variables in the order that they appear. It means that we may set a binary variable z_k to 1 only if the previous variable z_{k-1} is 1. Using the modeling of logical implications ($z_k = 1$ implies $z_{k-1} = 1$), we obtain the constraints

$$z_k \leq z_{k-1}, \quad k = 2, \ldots, u - \ell. \tag{25.40}$$

In our example, these constraints exclude all combinations represented in italic in Table 25.2, and there is now a bijection between the feasible combinations of z_k and the feasible values of x. Constraints (25.40) are called *symmetry breaking* constraints.

We conclude this section by mentioning that integer optimization is strongly related to combinatorial optimization.

x	z_1	z_2	z_3
0	0	0	0
1	*0*	*0*	*1*
1	*0*	*1*	*0*
2	*0*	*1*	*1*
1	1	0	0
2	*1*	*0*	*1*
2	1	1	0
3	1	1	1

Table 25.2: Coding an integer with binary variables

Definition 25.5 (Combinatorial optimization)**.** A combinatorial optimization problem consists in identifying the optimal element of a large finite set.

It is named as such because large finite sets are often generated by the list of combinations of given elements. For instance, there are 26^5 possible words with 5 letters. Even if integer optimization problems may happen to have an infinite feasible set, the use of upper bounds on the objective function allows to transform them into problems with finite sets.

25.2 Classical problems

We describe in this section some classical combinatorial optimization problems and discuss their formulation as integer linear optimization problems, to illustrate the strong link between the two types of problems.

25.2.1 The knapsack problem

Example 25.6 (The knapsack problem)**.** Patricia is about to spend several days on a long hike in the mountain to climb the Bishorn. She is now preparing her knapsack and thinking about which items to take. The alpine guides strongly recommends carrying no more than W kg (say, $W = 15$). Therefore, Patricia has to decide which items to carry and which items to leave at home. Each item has a different level of importance. While it is critical to carry water and food, it is less critical to carry a laptop. For each item i considered by Patricia, she knows its weight $w_i \geq 0$ and its level of importance or utility, $u_i \in \mathbb{R}$. The problem that Patricia has to solve consists in deciding what are the items to include in her knapsack in order to maximize the total utility while satisfying the maximum weight constraint.

The name of the knapsack problem comes from Example 25.6. Definition 25.7 provides a more general definition of the problem. Definition 25.8 describes the $0-1$ knapsack problem, another version of the problem that forbids multiple selection of the same item.

Definition 25.7 (The knapsack problem)**.** Consider a set of n items. Each item i is associated with a value characterizing its utility $u_i \in \mathbb{R}$ and a weight $w_i \geq 0$. The *knapsack problem* consists in deciding the number of times that each item must be selected so that the total weight of the selected items does not exceed an upper bound W and the total utility is maximized.

Definition 25.8 (The $0-1$ knapsack problem)**.** Consider a set of n items. Each item i is associated with a value $u_i \in \mathbb{R}$ and a weight $w_i \geq 0$. The $0-1$ knapsack problem consists in selecting a subset of the items so that the total weight of the selected items does not exceed an upper bound W and the total utility is maximized.

This problem can be modeled as an integer linear optimization problem. As described in Section 1.1, we model the problem in three steps:

1. Decision variables: for each item i that can potentially be carried, we define a variable $x_i \in \mathbb{N}$ that represents the number of items of type i that are carried in the knapsack.
2. Objective function: for each item i, the contribution to the utility of the knapsack is $u_i x_i$. Therefore, the total utility of the knapsack as a function of the decision variables is
$$\sum_{i=1}^{n} u_i x_i, \tag{25.41}$$
where n is the total number of items.
3. Constraints: similarly, for each item i, its contribution to the weight is $w_i x_i$. Therefore, as the maximum weight is W, the constraint is written as
$$\sum_{i=1}^{n} w_i x_i \leq W. \tag{25.42}$$
Moreover, by definition of the variables, the following constraints must also be verified:
$$x_i \in \mathbb{N},\ i = 1, \ldots, n. \tag{25.43}$$

Note that both the objective function and the constraints are linear functions of the decision variables.

If you consider the $0-1$ version of the problem (Definition 25.8), the variable x_i is binary and corresponds to a yes/no decision, where $x_i = 1$ means that item i is included in the knapsack, while $x_i = 0$ means that it is not. Constraints (25.43) are then replaced by
$$x_i \in \{0, 1\},\ i = 1, \ldots, n. \tag{25.44}$$

The knapsack problem has many applications and variants. Consider a portfolio of stocks, where each stock has a price w_i and a potential return on investment u_i. The question how to invest a total budget of W in order to maximize the expected profit is also a "knapsack problem."

25.2.2 Set covering

The set covering problem is illustrated by Example 25.9. Definition 25.10 provides a more general definition of the problem.

Example 25.9 (Set covering)**.** After the FIFA World Cup, Camille wants to complete her collection of stickers representing the players of each competing team. She has the possibility of buying collections of stickers from her schoolmates. In each collection, there are stickers that she needs, but also stickers that she does not need. However, schoolmates do not agree to sell stickers individually. The whole collection has to be purchased. Camille must decide which collections to purchase, in order to complete her own album, at a minimum price.

Definition 25.10 (Set covering)**.** Consider a set U of elements and a list of n subsets of U, denoted by S_i, $i = 1, \ldots, n$, each associated with a cost c_i, such that

$$\bigcup_{j=1}^{n} S_i = U. \tag{25.45}$$

The set covering problem consists in selecting a sublist S_{i_j}, $j = 1, \ldots, J$ such that their union includes all the elements in U, that is

$$\bigcup_{j=1}^{J} S_{i_j} = U, \tag{25.46}$$

and the total cost $\sum_{j=1}^{J} c_{i_j}$ is minimal.

Note that if condition (25.45), that requires the union of all S_i to be equal to U, is not verified, the problem is not feasible. Note also that the problem is often presented in the literature with $c_i = 1$ for each subset i, so that the covering has to involve the minimum number of subsets.

In the above example, U is the set of stickers missing by Camille, and each subset S_i corresponds to the collection of one of her schoolmates. We denote $j = 1, \ldots, m$ the missing stickers, and $i = 1, \ldots, n$ the available collections. For each i, the price of collection i is c_i. For each i and j, the parameter a_{ij} is equal to 1 if sticker j belongs to collection i, and to 0 otherwise. These parameters characterize the subsets S_i.

Decision variables: for each i we define a binary variable x_i with value 1 if it is decided to purchase collection i, and 0 otherwise.

Objective function: for each i, the associated cost is $c_i x_i$, that is c_i if collection i is purchased, and 0 otherwise. Therefore, the total cost associated with the decisions characterized by the vector x is

$$\sum_{i=1}^{n} c_i x_i. \tag{25.47}$$

Constraints: we have to guarantee that each missing sticker is available in at least one purchased collection. For a sticker j and a collection i, the quantity $a_{ij}x_i$ is equal to 1 if Camille obtains sticker j, as collection i is purchased and sticker j belongs to collection i. Therefore, the constraint is written as

$$\sum_{i=1}^{n} a_{ij}x_i \geq 1,\ j = 1, \ldots, m. \tag{25.48}$$

Indeed, for each sticker j, at least one of the terms $a_{ij}x_i$ must be 1, so that the sum over all collections must be at least 1. Moreover, by definition of the variables, the following constraints must also be verified:

$$x_i \in \{0, 1\},\ i = 1, \ldots, n. \tag{25.49}$$

Putting everything together, the optimization problem is written as

$$\min_{x \in \mathbb{N}^n} \sum_{i=1}^{n} c_i x_i,$$

subject to

$$\sum_{i=1}^{n} a_{ij}x_i \geq 1,\ j = 1, \ldots, m,$$

$$x_i \in \{0, 1\},\ i = 1, \ldots, n.$$

25.2.3 The traveling salesman problem

The traveling salesman problem is probably the most famous problem in combinatorial optimization. Easy to state, it is particularly difficult to solve.

Definition 25.11 (The traveling salesman problem)**.** A salesman must visit $n - 1$ customers. Starting from home, he has to plan a tour, that is, a sequence of customers to visit, in order to minimize the travel distance.

The problem is modeled using a network with n nodes, corresponding to the home of the salesman and the location of the $n - 1$ customers. Each pair of nodes is connected by an arc, and the cost of the arc represents the travel cost (the distance, or the travel time, for example).

Example 25.12 (The traveling salesman problem: 4 cities)**.** A salesman living in Lausanne (L) must visit 3 customers during the day: one in Geneva (G), one in Bern (B), and one in Zürich (Z). Starting from home, he has to plan a tour, that is a sequence of customers to visit, in order to minimize the travel distance. We model the problem with a graph represented in Figure 25.1 (see Chapter 21 for the definition of a graph).

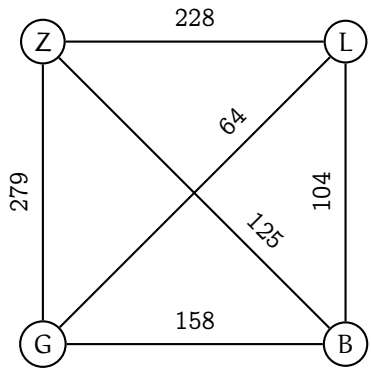


Figure 25.1: Traveling salesman problem: 4 cities (distances are in kilometers)

There are several ways to model this problem as an integer optimization problem. Here is one of them.

Decision variables: for each pair of nodes we define the binary variable x_{ij}, which is 1 if the salesman visits node j just after node i, and 0 otherwise.

Objective function: the total length of the tour must be minimized. For n cities, it is

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} = \min \sum_{i=1}^{n} \sum_{j\neq i} c_{ij}x_{ij}, \tag{25.50}$$

which for our example is

$$\begin{aligned} &64x_{LG} + 104x_{LB} + 228x_{LZ} \\ &+64x_{GL} + 158x_{GB} + 279x_{GZ} \\ &+104x_{BL} + 158x_{BG} + 125x_{BZ} \\ &+228x_{ZL} + 279x_{ZG} + 125x_{ZB}. \end{aligned}$$

Constraints:

- Each city must have exactly one successor in the tour:

$$\sum_{j|(i,j)\in\mathcal{A}} x_{ij} = 1, \quad \forall i \in \mathcal{N}, \tag{25.51}$$

that is,

$$\begin{aligned} x_{LG} + x_{LB} + x_{LZ} &= 1, \\ x_{GL} + x_{GB} + x_{GZ} &= 1, \\ x_{BL} + x_{BG} + x_{BZ} &= 1, \\ x_{ZL} + x_{ZG} + x_{BZ} &= 1. \end{aligned}$$

- Each city must have exactly one predecessor in the tour:

$$\sum_{i|(i,j)\in\mathcal{A}} x_{ij} = 1, \quad \forall j \in \mathcal{N}, \tag{25.52}$$

that is,

$$\begin{aligned} x_{GL} + x_{BL} + x_{ZL} &= 1, \\ x_{LG} + x_{BG} + x_{ZG} &= 1, \\ x_{LB} + x_{GB} + x_{ZB} &= 1, \\ x_{LZ} + x_{GZ} + x_{ZB} &= 1. \end{aligned}$$

Unfortunately, these constraints are not sufficient. Indeed, the optimal solution of the above problem is $x_{LG} = x_{GL} = x_{BZ} = x_{ZB} = 1$, and all other variables set to 0. The interpretation is to travel from Lausanne to Geneva and back, and to travel from Bern to Zürich and back. Instead of one tour, the optimization problem proposes two subtours that verify the constraints (each city has exactly one predecessor and one successor) and minimize the distance. Additional constraints must be included to eliminate these subtours.

One possible idea is to explicitly keep track of the order of the customers along the tour, starting from home. We introduce new variables representing the position of each customer in the tour. If customer j is visited after customer i in the tour, the position of customer j must be strictly larger then the position of customer i. It is sufficient to impose that constraint for each pair of successive customers, so that it is verified for all customers in the tour. Denoting y_i the position of customer i, we impose that

$$x_{ij} = 1 \Longrightarrow y_j \geq y_i + 1, \tag{25.53}$$

where i and j are customers, that is any node except home. It is important to exclude home, because the ordering must end at the last customer, and the above constraint does not hold for the last leg, from the last customer back to home. It could have been included for the first leg, from home to the first customer, but it is not necessary as the objective is to remove subtours that do not include home. This is actually the target of this constraint. If there is a tour that does not include home, there is no way to position its nodes. Without a reference node, it is not possible to decide if node i comes before or after node j in the loop. In the above example, one subtour includes the arcs ZB and BZ. The constraints $y_B \geq y_Z + 1$ and $y_Z \geq y_B + 1$ are incompatible, as the first imposes that Bern comes after Zürich, while the second imposes exactly the opposite.

We now need to transform the condition (25.53) into a constraint for the optimization problem. We apply the technique for optional constraints described in Section 25.1. It requires finding a version of the constraint that is always valid. In our case, the value $y_j - y_i$ is always greater or equal to $2 - n$. Indeed, as the variables represent the position of the customer in the tour, the largest difference happens when customer j is visited just after home ($y_j = 2$) and customer i is visited last ($y_j = n$). Therefore, the technique of optional constraints can be applied with $f(y) = y_j - y_i + n - 2$. Indeed, $f(y)$ is greater than zero for any y. The constraint $y_j \geq y_i + 1$ is equivalent to $y_j - y_i + n - 2 \geq n - 1$, that is, $f(y) \geq n - 1$. Therefore, we use $a = n - 1$ in (25.5) to obtain

$$y_j - y_i + n - 2 \geq x_{ij}(n-1), \tag{25.54}$$

or, equivalently,

$$x_{ij}(n-1) + y_i - y_j \leq n - 2. \tag{25.55}$$

Therefore, if $x_{ij} = 1$, we have $y_j - y_i + n - 2 \geq n - 1$, that is $y_j - y_i \geq 1$, which is the required constraint. If $x_{ij} = 0$, we have $y_i - y_j \leq n - 2$, which is always verified.

In our example, we must add the constraints

$$\begin{aligned}
x_{GB}(n-1) + y_G - y_B &\leq n - 2,\\
x_{BG}(n-1) + y_B - y_G &\leq n - 2,\\
x_{GZ}(n-1) + y_G - y_Z &\leq n - 2,\\
x_{ZG}(n-1) + y_Z - y_G &\leq n - 2,\\
x_{BZ}(n-1) + y_B - y_Z &\leq n - 2,\\
x_{ZB}(n-1) + y_Z - y_B &\leq n - 2.
\end{aligned}$$

Note that the solution proposed above, where both x_{BZ} and x_{ZB} are equal to 1 is not feasible anymore, as there is no value of y_B and y_Z such that both constraints $(n-1) + y_B - y_Z \leq n - 2$ and $(n-1) + y_Z - y_B \leq n - 2$ are verified.

Therefore, the optimization problem for the traveling salesman problem with n cities is written as:

$$\min_{x \in \mathbb{Z}^{n(n-1)}, y \in \mathbb{Z}^{(n-1)(n-2)}} \sum_{i=1}^{n} \sum_{j \neq i} c_{ij} x_{ij} \tag{25.56}$$

subject to

$$\begin{aligned}
\sum_{j \neq i} x_{ij} &= 1 && \forall i = 1, \ldots, n,\\
\sum_{i \neq j} x_{ij} &= 1 && \forall j = 1, \ldots, n,\\
x_{ij}(n-1) + y_i - y_j &\leq n - 2, && \forall i = 2, \ldots, n, j = 2, \ldots, n, i \neq j,\\
x_{ij} &\in \{0, 1\} && \forall i = 1, \ldots, n, j = 1, \ldots, n, i \neq j,\\
y_i &\geq 0 && \forall i = 1, \ldots, n.
\end{aligned} \tag{25.57}$$

There are many other combinatorial optimization problems that are not described in this book, such as vehicle routing problems, scheduling problems, bin packing problems, or facility location problems, to cite the most classical. We refer the reader to Papadimitriou and Steiglitz (1998) and Pardalos et al. (2013), among other references.

25.3 The curse of dimensionality

We have seen in Section 22.3 that some problems have the property that the integrality constraints can be safely ignored, as the optimal solution of the problem is guaranteed to have only integer values. Unfortunately, the family of such problems is rather restricted, and it does not correspond to the most general case.

The key difficulty for discrete optimization is that there is no optimality condition for the global optimum of the problem. For all other optimization problems analyzed so far in this book, the optimality conditions are the starting point for the development of algorithms. We do not have this opportunity for discrete optimization.

Example 25.13 (A simple integer optimization problem)**.** Consider the following integer optimization problem:

$$\min_{x \in \mathbb{N}^2} -3x_1 - 13x_2 \tag{25.58}$$

subject to

$$\begin{aligned} 2x_1 + 9x_2 &\leq 29 \\ 11x_1 - 8x_2 &\leq 79. \end{aligned} \tag{25.59}$$

The feasible set is defined as the intersection between the polyhedron

$$\{x \in \mathbb{R}^2 | 2x_1 + 9x_2 \leq 29, 11x_1 - 8x_2 \leq 79, x \geq 0\} \tag{25.60}$$

and the set $\mathbb{N}^2$. The polyhedron is represented in Figure 25.2, together with the points of $\mathbb{N}^2$ with x_1 ranging from 0 to 9 and x_2 ranging from 0 to 4. The feasible points are the 24 points of this lattice that are inside the polyhedron. The list of these points is reported in Table 25.3. The level curves of the objective function are represented by dotted lines, and the arrow identifies the direction of descent. In order to identify the optimal solution, we can enumerate the feasible points, calculate the value of the objective function for each of them, and select the point corresponding to the smallest value. From the values of the objective function reported in Table 25.3, the optimal solution is $x^* = (1, 3)$, corresponding to the value -42.

The enumeration method suggested in Example 25.13 is in general not appropriate. Remember that we have already proposed an enumeration method in Algorithm 16.1, where the optimal solution of a linear optimization problem is one of the vertices of the constraint polyhedron. Those vertices are enumerated to identify the best one. The combinatorially large number of such vertices was the motivation

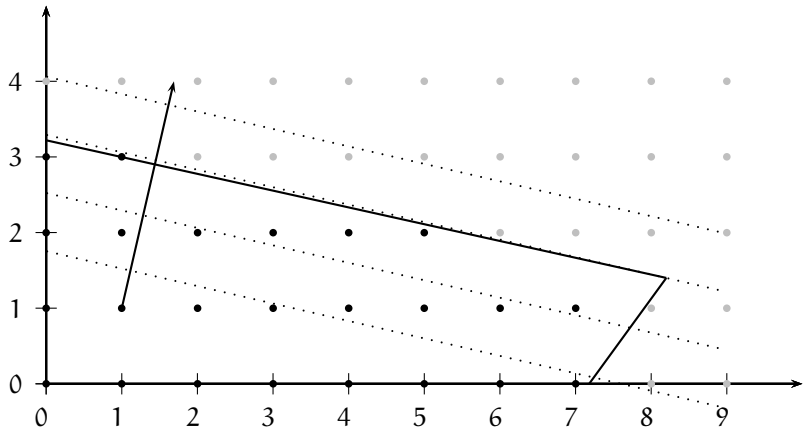


Figure 25.2: The feasible set and the level curves for Example 25.13

Table 25.3: Enumeration of the feasible solutions of Example 25.13 with the corresponding value of the objective function

x_1	x_2	$c^T x$	x_1	x_2	$c^T x$	x_1	x_2	$c^T x$
0	0	0	2	0	-6	4	2	-38
0	1	-13	2	1	-19	5	0	-15
0	2	-26	2	2	-32	5	1	-28
0	3	-39	3	0	-9	5	2	-41
1	0	-3	3	1	-22	6	0	-18
1	1	-16	3	2	-35	6	1	-31
1	2	-29	4	0	-12	7	0	-21
1	3	-42	4	1	-25	7	1	-34

for abandoning the vertex enumeration method and for developing the simplex algorithm. Integer optimization problems suffer from the same issue, which precludes applying an enumeration technique.

Consider the knapsack problem introduced in Example 25.6, where all variables are binary, that is, each item potentially carried is available only once. In order to solve the problem by enumeration, each configuration of the variables x has to be analyzed, that is, the corresponding weight $\sum_i w_i x_i$ must be calculated and, if smaller than the capacity W, the objective function $\sum_i u_i x_i$ must be recorded. If there are n items to be considered, there are 2^n combinations of values for x. Each of them needs about $2n$ floating point operations to compute the weight and the objective function. Assume that we are using a processor with 1 Teraflops, that is a processor able to perform 10^{12} floating point operations per second.

- If $n = 34$, it takes about 1 second to solve the problem by complete enumeration.
- If $n = 40$, it takes about 1 minute.
- If $n = 45$, it takes about 1 hour.

- If $n = 50$, it takes about 1 day.
- If $n = 58$, it takes about 1 year.
- If $n = 69$, it takes about 2,583 years, that is, more than the duration of the Christian Era.
- If $n = 78$, it takes about 1,500,000 years, that is, about the time that has elapsed since "homo erectus" appeared on earth.
- If $n = 91$, it takes about 10^{10} years, that is, about the age of the universe.

For all practical purposes, this method is not applicable for problems of size larger than $n = 50$.

Now, assume that we have a processor that is 1,000 times more powerful, that is, a processor able to perform 10^{15} floating point operations per second. In that case, a full enumeration could be applied to problems of size up to $n = 59$, which would take about a day to be solved. This is only 9 more variables. For the problem with $n = 69$, the time would decrease to about 3 years. If $n = 78$, it would be about 1,500 years, and for $n = 91$, about 14 million years.

Clearly, performing a full enumeration is not a viable option, except maybe for problems of small size. With a careful implementation of the method, as well as the availability of faster processors, the size of problems that may be solved by enumeration increases a little bit. But the fact that the running time increases exponentially with the size of the problem does not allow us to solve problems of realistic size. The huge explosion of the running time as a function of the dimension of the problem is sometimes referred to as the *curse of dimensionality*.

25.4 Relaxation

As complete enumeration is not an operational algorithm, other methods have to be investigated. In particular, it would be convenient to transform the problem into a continuous optimization problem by ignoring the integrality constraints and using a relevant algorithm to solve the continuous optimization problem. This problem is called the *relaxation* of the integer optimization problem.

Definition 25.14 (Relaxation)**.** Consider the mixed integer optimization problem P:

$$\min_{x \in \mathbb{R}^{n_x}, y \in \mathbb{Z}^{n_y}, z \in \mathbb{N}^{n_z}} f(x, y, z) \tag{25.61}$$

subject to

$$\begin{aligned} g(x, y, z) &\leq 0 \\ h(x, y, z) &= 0 \\ y &\in \mathbb{Z}^{n_y} \\ z &\in \{0, 1\}^{n_z}, \end{aligned} \tag{25.62}$$

where $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_z} \to \mathbb{R}$, $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_z} \to \mathbb{R}^m$, $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_z} \to \mathbb{R}^p$.

The optimization problem

$$\min_{x\in\mathbb{R}^{n_x},y\in\mathbb{R}^{n_y},z\in\mathbb{R}^{n_z}} f(x,y,z) \tag{25.63}$$

subject to

$$\begin{aligned} g(x,y,z) &\leq 0 \\ h(x,y,z) &= 0 \\ 0 \leq z &\leq 1 \end{aligned} \tag{25.64}$$

is called the relaxation of P and denoted by R(P).

By definition, the feasible set of the original problem is included in the feasible set of its relaxation. In other words, if (x,y,z) is feasible for P, it is also feasible for R(P). Therefore, the optimal value of R(P) is a lower bound on the optimal value of P.

Theorem 25.15 (Lower bound from the relaxation)**.** *Let* P *be a mixed integer optimization problem and* R(P) *its relaxation. Denote* (x^*,y^*,z^*) *the optimal solution of problem* P *and* (x_R^*,y_R^*,z_R^*) *the optimal solution of problem* R(P)*. Then,*

$$f(x_R^*,y_R^*,z_R^*) \leq f(x^*,y^*,z^*). \tag{25.65}$$

Proof. It is an immediate consequence of the feasibility of (x^*,y^*,z^*) for problem R(P). □

The above result is useful only if the global minimum of the relaxation can be identified. It is the case when the objective function and the feasible set are convex. And in particular, it is the case when the objective function and the constraints are linear.

Definition 25.16 (Linear relaxation)**.** Consider the mixed integer linear optimization problem P:

$$\min_{x\in\mathbb{R}^{n_x},y\in\mathbb{N}^{n_y},z\in\mathbb{N}^{n_z}} c_x^Tx + c_y^Ty + c_z^Tz \tag{25.66}$$

subject to

$$\begin{aligned} A_xx + A_yy + A_zz &= b \\ x,y,z &\geq 0 \\ y &\in \mathbb{N}^{n_y} \\ z &\in \{0,1\}^{n_z}, \end{aligned} \tag{25.67}$$

where $c_x \in \mathbb{R}^{n_x}$, $c_y \in \mathbb{R}^{n_y}$, $c_z \in \mathbb{R}^{n_z}$, $A_x \in \mathbb{R}^{m\times n_x}$, $A_y \in \mathbb{R}^{m\times n_y}$, $A_z \in \mathbb{R}^{m\times n_z}$, and $b \in \mathbb{R}^m$. The linear optimization problem

$$\min_{x\in\mathbb{R}^{n_x},y\in\mathbb{R}^{n_y},z\in\mathbb{R}^{n_z}} c_x^Tx + c_y^Ty + c_z^Tz \tag{25.68}$$

subject to

$$\begin{aligned} A_x x + A_y y + A_z z &= b \\ x, y, z &\geq 0 \\ z &\leq 1 \end{aligned} \tag{25.69}$$

is called the relaxation of P and denoted by $R(P)$.

The relaxation is a linear optimization problem and can be solved using any appropriate algorithm, such as the simplex method. Except if the matrix of the constraints is totally unimodular (see Definition 22.3), the optimal solution of the relaxation may not provide an integer solution.

Unfortunately, it is not sufficient to round up or round down the fractional solution of the relaxation in order to identify the optimal solution of the integer optimization problem. Consider again Example 25.13. The relaxation of the problem is

$$\min_{x \in \mathbb{R}^2} -3x_1 - 13x_2 \tag{25.70}$$

subject to

$$\begin{aligned} 2x_1 + 9x_2 &\leq 29 \\ 11x_1 - 8x_2 &\leq 79 \\ x_1, x_2 &\geq 0. \end{aligned} \tag{25.71}$$

The optimal solution of the relaxation is $(8.2, 1.4)$, as illustrated in Figure 25.3, where the level curves of the objective function are represented by dotted lines, and the arrow identifies the direction of descent.

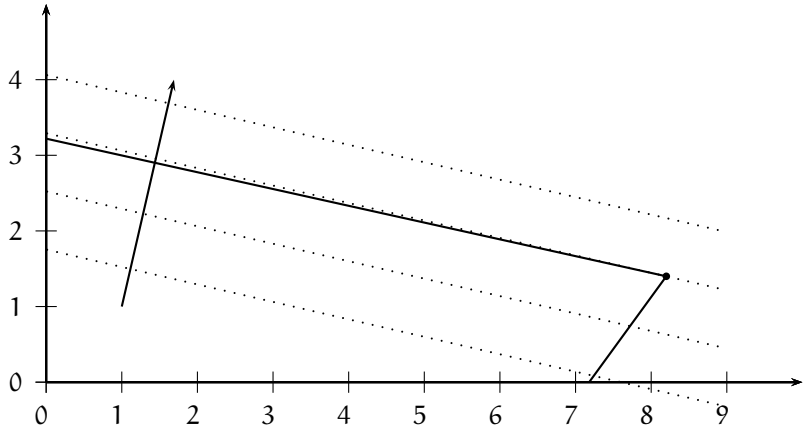


Figure 25.3: The feasible set of the relaxation and the level curves for Example 25.13

There are two ways to round a real number to an integer: round down or round up. As there are two real numbers to round, there are four ways to round the optimal solution of the relaxation: $(8, 1)$, $(8, 2)$, $(9, 1)$, and $(9, 2)$ as depicted in Figure 25.4.

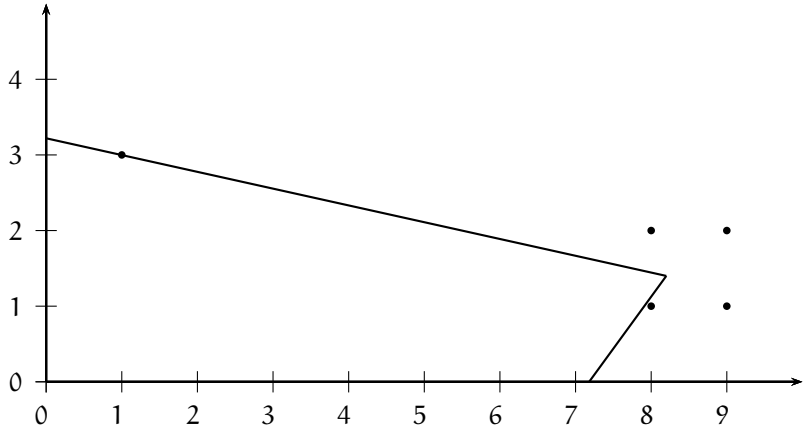


Figure 25.4: Example 25.13: rounding the optimal solution of the relaxation

In this example, none of these integer solutions obtained by rounding the fractional solution of the relaxation in one way or the other is feasible. Moreover, even if one of them was, it appears that the fractional solution of the relaxation does not lie in the vicinity of the optimal solution of the integer optimization problem, which is $(1, 3)$.

The problem relaxation plays an important role in the algorithms to solve integer optimization problems. But as it appears from the example above, there is more to it than simply rounding the optimal solution. In Chapter 26, we present methods that use the bound provided by the relaxation (see Theorem 25.15) to avoid an explicit complete enumeration of the feasible space.

25.5 Exercises

Exercise 25.1. In order to participate in a cooking competition called "Top Chef," Benoît must build a team of excellent chefs. He has access to a pool of candidates (see Table 25.4). For each of them, he knows the age, the restaurant where they come from, and their cooking specialties (some candidates have only one, some two). He also knows the number of points that each of them has been able to collect in previous similar competitions. And he wants to build a team with the maximum number of such points. But several constraints apply.

1. The team must contain at least 3 chefs who are skilled in preparing appetizers.
2. The team must contain at least 4 chefs who are skilled in preparing fish.
3. The team must contain at least 4 chefs who are skilled in preparing meat.
4. The team must contain at least 3 chefs who are skilled in preparing dessert.
5. To promote young people, the team must contain at least 2 chefs who are 20 or less.
6. Yoda and Obi-Wan cannot stand each other, and it is not possible to have both of them in the team.

Table 25.4: Potential candidates for the Top Chef competition for Exercise 25.1

Name	Age	Restaurant	Points	First specialty	Second specialty
Anakin	49	Lausanne	124	Fish	Meat
Chewbacca	39	Noirmont	105	Fish	Meat
Dooku	49	Crissier	118	Meat	Dessert
Gial	33	Mézières	134	Dessert	—
Han	35	Crissier	160	Appetizer	—
Jabba	47	Crissier	184	Fish	—
Jubnuk	28	Neuchatel	101	Fish	—
Lando	17	Lucens	116	Dessert	—
Leia	18	Sierre	187	Appetizer	Fish
Lorth	41	Baulmes	132	Dessert	—
Luke	37	Vufflens	181	Appetizer	—
Obi-Wan	22	Vufflens	199	Fish	Meat
Padme	38	Cossonay	192	Appetizer	—
Qui-Gon	19	Cossonay	136	Fish	—
Sebulba	43	Valeyres	123	Meat	—
Sheev	21	Crissier	127	Meat	—
Teebo	32	Sierre	132	Meat	—
Watto	20	Sierre	131	Meat	—
Yoda	39	Noirmont	102	Appetizer	Fish
Zuckuss	42	Vufflens	123	Dessert	—

7. Han and Sheev have so much experience in working together that if one of the two is included in the team, the other one must be too.
8. For the sake of fairness, not more than 3 chefs from the same restaurant should be hired in the team.

   Write an integer optimization problem to help Benoît build his team.

Exercise 25.2 (Scheduling)**.** The journal *Millenium* needs to schedule the staff for the printing workshop for the five days of the week. During each day, there are eight one-hour time slots. Four employees are available for the tasks. Each employee has reported his or her preference for each time slot and each day on a scale from 0 to 10, where 10 corresponds to the highest preference and 0 corresponds to unavailability (see Table 25.5). The following constraints must be verified.

1. Each of the 40 slots must be covered by exactly one employee.
2. An employee cannot be assigned to a time slot if she/he is not available.
3. Every person must take a lunch break either between 12:00 and 13:00, or between 13:00 and 14:00.

4. Because of the noisy work environment, every person can work only two consecutive time slots. A break of at least one hour must be taken after that.
5. No one can work more than 20 hours per week.

Write an integer optimization problem to help *Millenium* schedule the workshop employees in order to maximize their satisfaction according to the stated preferences, while verifying the constraints.

Table 25.5: Preference of each worker for each time slot (Exercise 25.2)

Name	Slot	Mo	Tu	We	Th	Fr
Mikael	9–10	10	10	10	10	10
Mikael	10–11	9	9	9	9	9
Mikael	11–12	8	8	8	8	8
Mikael	12–13	1	1	1	1	1
Mikael	13–14	1	1	1	1	1
Mikael	14–15	1	1	1	1	1
Mikael	15–16	1	1	1	1	1
Mikael	16–17	1	1	1	1	1
Lisbeth	9–10	10	9	8	7	6
Lisbeth	10–11	10	9	8	7	6
Lisbeth	11–12	10	9	8	7	6
Lisbeth	12–13	10	3	3	3	3
Lisbeth	13–14	1	1	1	1	1
Lisbeth	14–15	1	2	3	4	5
Lisbeth	15–16	1	2	3	4	5
Lisbeth	16–17	1	2	3	4	5
Harriet	9–10	10	10	10	10	10
Harriet	10–11	9	9	9	9	9
Harriet	11–12	8	8	8	8	8
Harriet	12–13	0	0	0	0	0
Harriet	13–14	1	1	1	1	1
Harriet	14–15	1	1	1	1	1
Harriet	15–16	1	1	1	1	1
Harriet	16–17	1	1	1	1	1
Alexander	9–10	10	9	8	7	6
Alexander	10–11	10	9	8	7	6
Alexander	11–12	10	9	8	7	6
Alexander	12–13	10	3	3	3	3
Alexander	13–14	1	1	1	1	1
Alexander	14–15	1	2	3	4	5
Alexander	15–16	1	2	3	4	5
Alexander	16–17	1	2	3	4	5

Exercise 25.3 (Graph coloring)**.** Consider the map of Belgium (Figure 25.5). We want to color each province of the map so that two provinces with a common border have different colors. What is the minimum number of colors that must be used? Write an integer optimization problem to answer this question and to provide a valid coloring.

Figure 25.5: The provinces of Belgium (for Exercise 25.3)

Exercise 25.4 (Bin packing)**.** You have 10 objects of different heights (in cm): 80, 70, 60, 50, 40, 40, 20, 20, 10, and 10. As you are moving, you need to stack them in boxes of 1 meter high. Assuming that the sections of the objects do not allow to store them side by side, but only one over the others, how do you arrange the objects into the boxes to use a minimum number of them? Write an integer optimization problem to answer this question.

Exercise 25.5 (Vehicle routing)**.** The Nespresso factory in Orbe must deliver 18,000 capsules to Lausanne, 26,000 to Neuchatel, 11,000 to Fribourg, 30,000 to Berne and 21,000 to Sierre. The travel time between each city is reported in Table 25.6. The company has vehicles that can transport at most Q capsules each. How does the company have to organize the delivery of the capsules to satisfy the demand of each city, while minimizing the total time traveled? Write an integer optimization problem to answer this question.

Table 25.6: Travel time between each pair of cities for Exercise 25.5.

	Orbe	Lausanne	Neuchatel	Fribourg	Berne	Sierre
Orbe		28	39	50	57	83
Lausanne			52	49	74	72
Neuchatel				48	48	111
Fribourg					35	85
Bern						105

Chapter 26

Exact methods for discrete optimization

Contents

In the context of discrete optimization, the absence of optimality conditions complicates the design of algorithms. Even proving that a given solution is optimal is difficult. The enumeration of feasible solutions would provide such a proof, but is often not feasible due to the curse of dimensionality discussed earlier.

The *exact methods* presented in this chapter guarantee that an optimal solution is found if the algorithm terminates in a reasonable time. We present here two such methods. The first performs an implicit enumeration of the feasible set, by ruling out large sets of feasible points using mathematical properties of the optimization problem. This is achieved by partitioning the feasible set into smaller subsets, using a so-called "branching" strategy, and by using bounds to identify subsets that are guaranteed not to contain an optimal solution. This "branch and bound" method is presented in Section 26.1. The second method modifies the formulation of the problem so that one of its optimal solutions coincides with the solution of its relaxation. It can therefore be identified using an algorithm for continuous optimization. The cutting planes method presented in Section 26.2 consists in cutting the constraint polyhedron using appropriate hyperplanes.

26.1 Branch and bound

Consider the combinatorial optimization problem P defined as

$$\min_{x} f(x) \tag{26.1}$$

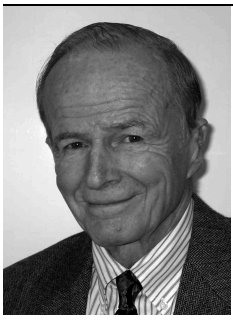
subject to

$$x \in \mathcal{F}, \tag{26.2}$$

where $\mathcal{F}$ represents the feasible set. The idea is to partition it into smaller subsets:

$$\mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_K.$$

In the context of integer optimization, a common branching strategy consists in dividing the feasible set into two parts, by selecting one integer variable x_i and a threshold integer value c for this variable. The feasible set is then partitioned into a subset containing all feasible solutions such that $x_i \leq c$, and another one containing all feasible solutions such that $x_i \geq c + 1$.



Born on February 1, 1928, in Boston, Massachusetts, USA, John D. C. Little was the first American to obtain a Ph.D. in Operations Research (OR), under the supervision of Philip M. Morse (MIT, 1955). Among his many contributions to OR, Little provided the first proof of what is now known as Little's law in queueing theory. It states that the time-average number of customers in the system L is equal to the average arrival rate of customers accepted into the system λ multiplied by the average time that each spends in the system W: $L = \lambda W$. It is so famous that funny T-shirts were printed for an OR conference with the following statement: "It may be Little, but It's the Law." With his co-authors Murty, Sweeney, and Karel (Little et al., 1963), he introduced the name "branch and bound" in the OR literature. He has been Institute Professor at the MIT Sloan School of Management since 1989.

Figure 26.1: John D. C. Little

Call P_k the optimization problem associated with the subset $\mathcal{F}_k$, that is

$$\min_{x} f(x) \tag{26.3}$$

subject to

$$x \in \mathcal{F}_k, \tag{26.4}$$

and call x_k^* a (global) optimal solution of P_k. As $\mathcal{F}_k \subseteq \mathcal{F}$, x_k^* is feasible for P. The key idea is that each optimal solution of P is one of the optimal solutions x_k^*.

Theorem 26.1 (Optimal solution of a partitioned problem). *Consider the optimization problem* P *defined by (26.1)–(26.2), and consider a partition* $\mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_K$ *of the feasible set into* K *subsets. For each* $k = 1, \ldots, K$, *let* x_k^* *be an optimal solution of the optimization problem* P_k *defined as* $\min_x f(x)$ *subject to* $x \in \mathcal{F}_k$. *Let* i *be such that*

$$f(x_i^*) \leq f(x_k^*), \quad k = 1, \ldots, K. \tag{26.5}$$

Then, x_i^* *is an optimal solution of the optimization problem* P.

Proof. Let y^* be an optimal solution of P, meaning that $f(y^*) \leq f(x), \quad \forall x \in \mathcal{F}$, and in particular

$$f(y^*) \leq f(x_i^*). \tag{26.6}$$

As we consider a partition, y^* belongs to one of the subsets, say $\mathcal{F}_k$. As x_k^* is an optimal solution of problem P_k, we have $f(x_k^*) \leq f(x)$, for each $x \in \mathcal{F}_k$. In particular

$$f(x_k^*) \leq f(y^*). \tag{26.7}$$

Combining (26.5), (26.6), and (26.7), we have

$$f(y^*) \leq f(x_i^*) \leq f(x_k^*) \leq f(y^*).$$

Consequently, $f(y^*) = f(x_i^*)$ and x_i^* is indeed an optimal solution of the problem. □

Corollary 26.2 (Optimal solution of a partitioned problem). *Consider the optimization problem* P *defined by (26.1)–(26.2), and consider a partition* $\mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_m \cup \mathcal{F}_{m+1} \cup \ldots \cup \mathcal{F}_K$ *of the feasible set into* K *subsets. For each* $k = 1, \ldots, m$, *let* x_k^* *be an optimal solution of the optimization problem* P_k *defined as* $\min_x f(x)$ *subject to* $x \in \mathcal{F}_k$. *For each* $k = m+1, \ldots, K$ *let* $\ell(P_k)$ *be a lower bound on* P_K, *that is*

$$\ell(P_k) \leq f(x_k) \; \forall x_k \in \mathcal{F}_k. \tag{26.8}$$

Let i *be such that*

$$f(x_i^*) \leq f(x_k^*), \quad k = 1, \ldots, m, \tag{26.9}$$

and

$$f(x_i^*) \leq \ell(P_k), \quad k = m+1, \ldots, K. \tag{26.10}$$

Then, x_i^* *is an optimal solution of the optimization problem* P.

Proof. As $\ell(P_k) \leq f(x_k^*)$ for each k, Theorem 26.1 applies. □

Corollary 26.2 is the motivation for the implicit enumeration. Indeed, it is not necessary to solve each subproblem in order to find an optimal solution of P. For several problems, the availability of lower bounds is sufficient to exclude them as candidates to produce an optimal solution. In practice, it is much easier to calculate lower bounds than to solve the subproblem to optimality.

The fact that problem P_k is simpler than problem P does not mean that it is simple. To solve P_k, it may again be necessary to partition its feasible set into smaller subsets. And they can be partitioned themselves, if needed. This decomposition can be depicted as a tree, as represented in Figure 26.2. The "root" of the tree corresponds to the original problem P. Each subproblem corresponds to a "branch," which can be divided into other branches. Clearly, the number of nodes in this tree, that is, the number of subproblems, increases rather quickly with the size of the problem, and the curse of dimensionality is again in the way. To deal with this, we need to avoid constructing all possible branches of the tree.

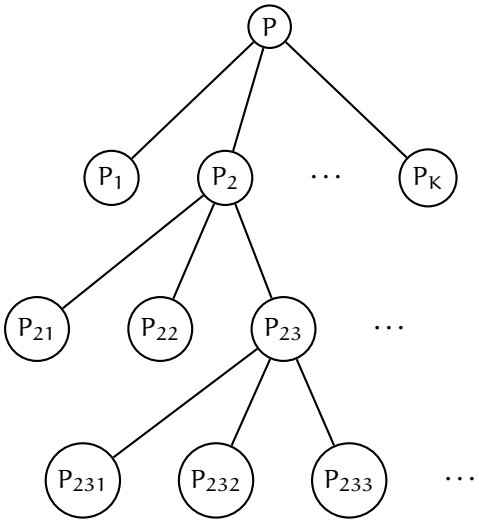


Figure 26.2: Decomposition of the optimization problem

Consider the optimization problem P defined by (26.1)–(26.2), and suppose that we have a feasible solution $x_0 \in \mathcal{F}$. Using a branching strategy, the problem is partitioned into K problems $P_1, \ldots, P_K$. Consider one subproblem P_k, and suppose that we can obtain a lower bound $\ell(P_k)$ on the optimal value, that is, we have $\ell(P_k)$ such that $\ell(P_k) \leq f(x)$, $\forall x \in \mathcal{F}_k$. Whatever the optimal value of P_k is, it cannot be better than $\ell(P_k)$. Now, if it happens that $f(x_0) \leq \ell(P_k)$, it means that the optimal value of P_k, whatever it is, is not better than $f(x_0)$. Consequently, there is no need to solve P_k. As P_k may be a difficult problem to solve, discarding it may save a lot of effort. This is the main idea of the branch and bound algorithm, described as Algorithm 26.1.

Note also that a branching strategy may generate empty subsets $\mathcal{F}_k$. This is why it is necessary to verify that the problem is feasible (step 17) before trying to solve it. Finally, the exact procedure to calculate a good lower bound (step 20) is problem

dependent.

Algorithm 26.1: Branch and bound

Objective
  Find a global minimum of the problem P: $\min_x f(x)$ subject to $x \in \mathcal{F}$.
Input
  The objective function $f : \mathbb{R}^n \to \mathbb{R}$.
  The feasible set $\mathcal{F}$.
  If available, an initial feasible solution $x_0 \in \mathcal{F}$.
Output
  A global minimum x^*.
Initialization
  $\mathcal{P} := \{P\}$.
  if x_0 *is available* **then**
    $f^* := f(x_0)$, $x^* := x_0$
  else
    $f^* := +\infty$
Repeat
  Select a problem P_k in $\mathcal{P}$.
  if P_k *is infeasible* **then**
    Discard P_k from $\mathcal{P}$
  else
    Calculate a lower bound $\ell(P_k)$
    if $f^* > \ell(P_k)$ **then**
      if P_k *is easy to solve* **then**
        Calculate x_k^* a global optimal solution of P_k
        if $f(x_k^*) < f^*$ **then**
          $x^* := x_k^*, f^* := f(x_k^*)$
      else
        Create a list of subproblems $\{P_{k1}, \ldots, P_{kK}\}$
        $\mathcal{P} := \mathcal{P} \cup \{P_{k1}, \ldots, P_{kK}\}$
  Discard from $\mathcal{P}$ each P_i such that $f^* \leq \ell(P_i)$.
Until $\mathcal{P} = \emptyset$.

Example 26.3 (Tasks assignment for Geppetto)**.** Geppetto has hired four workers with skills in carpentry and finishing to take care of the production of his toys (trains and soldiers, see Section 1.1.7). Each worker is able to work on any task, with the same efficiency. The number of hours that each worker needs to perform each task is reported in the following table.

	Tasks			
Workers	Carpentry trains	Finishing trains	Carpentry soldiers	Finishing soldiers
Pinocchio	9	2	7	8
Jiminy	6	4	3	7
Lampwick	5	8	1	8
Figaro	7	6	9	4

Geppetto wants to assign each task to a worker, in such a way that the total time is minimized.

We illustrate Algorithm 26.1 on Example 26.3. In this case, a lower bound on the optimal value can easily be derived from the data:

- the minimum number of hours that Pinocchio would work is 2,
- the minimum number of hours that Jiminy would work is 3,
- the minimum number of hours that Lampwick would work is 1,
- the minimum number of hours that Figaro would work is 4.

Consequently, the total number of hours cannot be lower than $2+3+1+4=10$. As an initial feasible solution, we consider an arbitrary assignment:

Worker	Task	Time
Pinocchio	Carpentry trains	9
Jiminy	Finishing trains	4
Lampwick	Carpentry soldiers	1
Figaro	Finishing soldiers	4
		18

We partition the set of feasible solutions by assigning the task to Pinocchio. Problem P_1 consists in assuming that Pinocchio takes care of the carpentry for trains, and in assigning the three other tasks to the three other workers. Problems P_2, P_3, and P_4 are defined similarly, where Pinocchio is assigned to the finishing of trains, the carpentry of soldiers, and the finishing of soldiers, respectively.

A lower bound is derived for each subproblem, in the same way as above:

- the minimum number of hours that Jiminy would work is 3,
- the minimum number of hours that Lampwick would work is 1,
- the minimum number of hours that Figaro would work is 4.

For the three of them, the total time cannot be lower than $3+1+4=8$ hours.

- For P_1, Pinocchio is working 9 hours so that $\ell(P_1)=9+8=17$.
- For P_2, Pinocchio is working 2 hours so that $\ell(P_2)=2+8=10$.
- For P_3, Pinocchio is working 7 hours so that $\ell(P_3)=7+8=15$.
- For P_4, Pinocchio is working 8 hours so that $\ell(P_4)=8+8=16$.

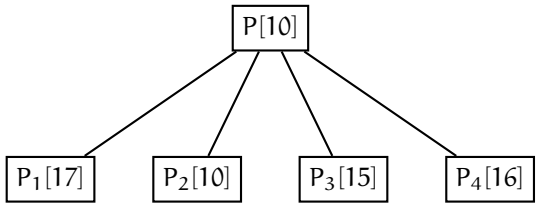


Figure 26.3: First branching for Example 26.3, with bounds

The branching and the calculation of the bound are summarized in Figure 26.3, where the number into square brackets is the bound for the corresponding problem.

The initial feasible solution corresponds to a total of 18 hours. Therefore, no subproblem can be discarded, as each lower bound is better than this value.

We now solve one of the subproblems. We select P_2, as it has the smallest lower bound. It corresponds to the decision that Pinocchio is assigned to finishing trains. We branch now based on the assignment of Jiminy and create the following problems:

- P_{21}, where Jiminy is assigned to carpentry for trains,
- P_{22}, where Jiminy is assigned to finishing for trains,
- P_{23}, where Jiminy is assigned to carpentry for soldiers,
- P_{24}, where Jiminy is assigned to finishing for soldiers.

We immediately note that problem P_{22} is not feasible, as Pinocchio and Jiminy are not allowed to perform the same task. It is therefore discarded. A lower bound is derived for each remaining subproblem:

- the minimum number of hours that Lampwick would work is 1,
- the minimum number of hours that Figaro would work is 4.

For the two of them, the total time cannot be lower than $1 + 4 = 5$ hours.

- For P_{21}, Jiminy is working 6 hours so that $\ell(P_1) = 2 + 6 + 5 = 13$.
- For P_{23}, Jiminy is working 3 hours so that $\ell(P_3) = 2 + 3 + 5 = 10$.
- For P_{24}, Jiminy is working 7 hours so that $\ell(P_4) = 2 + 7 + 5 = 14$.

The branching and the calculation of the bound is summarized in Figure 26.4.

Now we select problem P_{23} that corresponds to the smallest lower bound. As Pinocchio and Jiminy have already been assigned for a total of $2 + 3 = 5$ hours, the problem can be solved by complete enumeration.

- Lampwick is assigned to carpentry for trains: $5 + 5 + 4 = 14$, or
- Lampwick is assigned to finishing for soldiers: $5 + 8 + 7 = 20$.

The optimal solution of P_{23} assigns Pinocchio to finishing for trains, Jiminy to carpentry for soldiers, Lampwick to carpentry for trains, and Figaro to finishing for soldiers. The total number of hours is 14. It is better than the initial feasible solution that has a value of 18. Therefore, it becomes the candidate to be the optimal solution. We write $x^* := [P(FT), J(CS), L(CT), F(FS)]$, and $f^* := 14$.

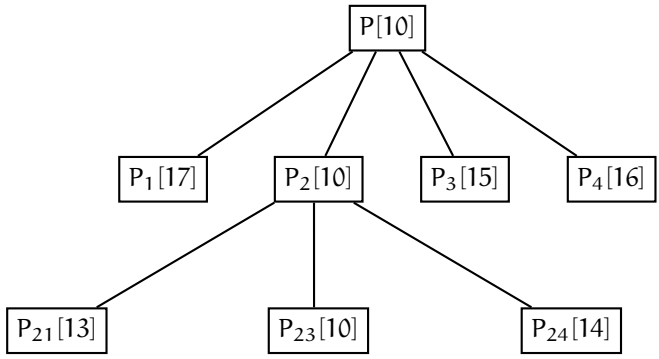


Figure 26.4: Second branching for Example 26.3, with bounds

We can now remove all problems with a lower bound equal or higher to $f^* = 14$, discarding P_1, P_3, P_4, and P_{24}. As P_1, P_3, and P_4 are discarded, we know that each optimal solution of P is an optimal solution of P_2. In particular, we know that Pinocchio is assigned to the finishing of trains in any optimal solution, as each optimal solution of problem P_2 is also an optimal solution of problem P. The current status of the algorithm is illustrated in Figure 26.5, where the round shape corresponds to a solved problem, and the value in parentheses is its optimal value.

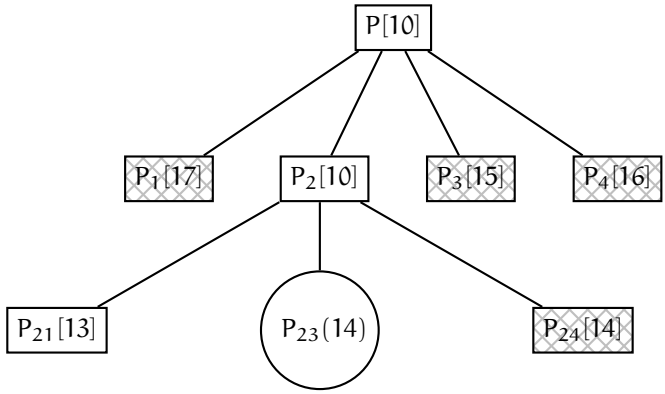


Figure 26.5: Branch & bound for Example 26.3: one subproblem is solved

The next problem to solve is problem P_{21}. As Pinocchio and Jiminy have already been assigned for a total of $2 + 6 = 8$ hours, the problem can be solved by complete enumeration.

- Lampwick is assigned to carpentry for soldiers: $8 + 1 + 4 = 13$, or
- Lampwick is assigned to finishing for soldiers: $8 + 8 + 9 = 25$.

The optimal solution to P_{21} is therefore $x^*_{21} = [P(FT), J(CT), L(CS), F(FS)]$. As its value is lower than the current value of f^*, it becomes the new candidate for the optimal solution: $x^* := x^*_{21}$ and $f^* := 13$. All the subproblems of problem P_2 have

been either discarded or solved to optimality. Therefore, x^*_{21} is also an optimal solution of P_2, that is $x^*_2 = x^*_{21}$. The same reasoning applies to the original problem P. Therefore, we have found an optimal solution of the problem. Note that even though we did not enumerate all possible combinations of the assignments, we guarantee the optimality of the solution. The final status of the algorithm is illustrated by Figure 26.3.

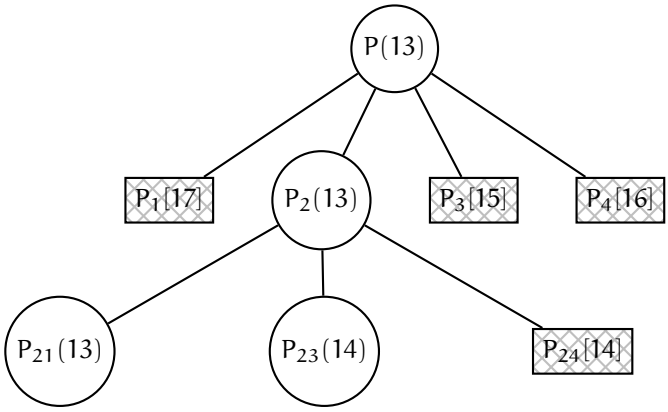


Figure 26.6: Branch & bound for Example 26.3: final tree with all subproblems solved or discarded

We now discuss Algorithm 26.1 in the specific case of an integer optimization problem P. For the calculation of bounds, Theorem 25.15 suggests using the relaxation $R(P_k)$ (see Definition 25.14) of the subproblem. Also, if an optimal solution x^*_R of the relaxation happens to be integer, it is also an optimal solution of the subproblem and, consequently, a feasible solution of P, candidate to be an optimal solution. If it is not integer, we use the following branching mechanism:

- select one index i such that $(x^*_R)_i$ is not integer,
- define an integer threshold c as the largest integer strictly lower than $(x^*_R)_i$, that is $c = \lfloor (x^*_R)_i \rfloor$, obtained by rounding down the value of $(x^*_R)_i$,
- partition the feasible set into a subset containing all feasible solutions such that $x_i \leq c = \lfloor (x^*_R)_i \rfloor$, and another one containing all feasible solutions such that $x_i \geq c+1$. Note that, in this case, $c+1$ is the smallest integer strictly larger than $(x^*_R)_i$, that is,

$$c + 1 = \lceil (x^*_R)_i \rceil,$$

obtained by rounding up the value of $(x^*_R)_i$.

For instance, if $(x^*_R)_i = 3.4$, the two constraints are $x_i \leq 3$ and $x_i \geq 4$. An interesting property of this branching scheme is that x^*_R no longer belongs to the feasible set of any subproblem, as it violates both $x_i \leq c$ and $x_i \geq c+1$. Therefore, we have the guarantee that future fractional solutions generated by the algorithm will be different from x^*_R.

Using these ingredients, the branch and bound algorithm for integer optimization is described in Algorithm 26.2.

Algorithm 26.2: Branch and bound for integer optimization

Objective
  Find a global minimum of the problem P_0: $\min_x f(x)$ subject to $g(x) \leq 0$, $h(x) = 0$, $x \in \mathbb{Z}^n$.
Input
  The functions $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^p$.
Output
  A global minimum x^*.
Initialization
  $\mathcal{P} := \{P_0\}$.
  $f^* := +\infty$.
Repeat
  Select a problem P_k in $\mathcal{P}$.
  if P_k *is infeasible* **then**
    discard P_k from $\mathcal{P}$
  else
    Calculate the global minimum x_R^* of the relaxation $R(P_k)$
    $\ell(P_k) := f(x_R^*)$
    if $f^* > \ell(P_k)$ **then**
      if x_R^* *is integer* **then**
        if $f(x_R^*) < f^*$ **then**
          $x^* := x_R^*, f^* := f(x_R^*)$
      else
        Select i such that $(x_R^*)_i$ is not integer
        Create subproblem P_k^ℓ by adding the constraint $x_i \leq \lfloor (x_R^*)_i \rfloor$ to P_k
        Create subproblem P_k^r by adding the constraint $x_i \geq \lceil (x_R^*)_i \rceil$ to P_k
        $\mathcal{P} := \mathcal{P} \cup \{P_k^\ell, P_k^r\} \setminus P_k$
  Discard from $\mathcal{P}$ each P_i such that $f^* \leq \ell(P_i)$.
Until $\mathcal{P} = \emptyset$.

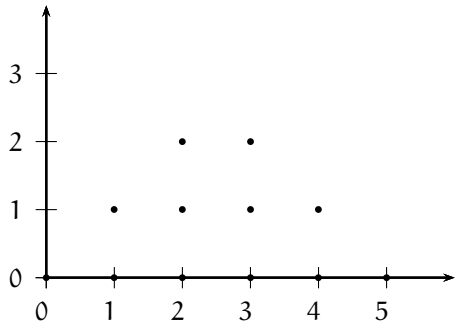
Example 26.4 (Simple integer linear optimization problem)**.** Define problem P_0 as

$$\min x_1 - 2x_2 \tag{26.11}$$

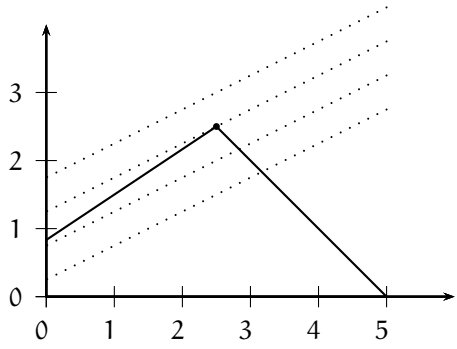
subject to

$$\begin{aligned} -4x_1 + 6x_2 &\leq 5 \\ x_1 + x_2 &\leq 5 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

The feasible set of P_0 is represented in Figure 26.7(a).



(a) Feasible set of P_0



(b) Feasible set and level curves of $R(P_0)$

Figure 26.7: Example 26.4 and its relaxation

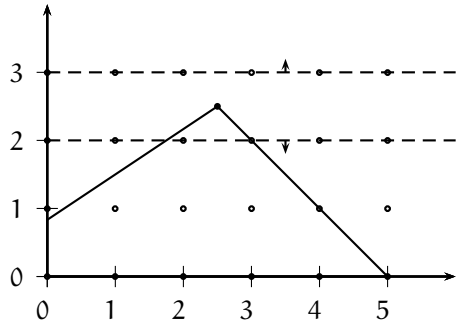
Example inspired by Bertsimas and Weismantel (2005)

We now illustrate Algorithm 26.2 on Example 26.4. Call the original problem P_0 and initialize $f^* = +\infty$ and $\mathcal{P} := \{P_0\}$. The optimal solution of the relaxation $R(P_0)$ is $x_0^* = (2.5, 2.5)$, with value $f(x_0^*) = -2.5$. The feasible set of the relaxation $R(P_0)$, some level curves of the objective function, and the optimal solution are represented in Figure 26.7(b). As the optimal solution is not integer, we decide to branch on x_2. In the first subproblem $P_1 = P_0^\ell$, we include the constraint $x_2 \leq \lfloor 2.5 \rfloor = 2$. In the second one $P_2 = P_0^r$, we include the constraint $x_2 \geq \lceil 2.5 \rceil = 3$.

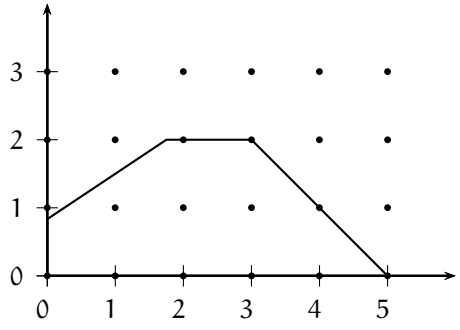
We now have $\mathcal{P} = \{P_1, P_2\}$

$P_1 = P_0^\ell$			$P_2 = P_0^r$		
$\min x_1 - 2x_2$			$\min x_1 - 2x_2$		
subject to			subject to		
$-4x_1 + 6x_2$	$\leq$	5	$-4x_1 + 6x_2$	$\leq$	5
$x_1 + x_2$	$\leq$	5	$x_1 + x_2$	$\leq$	5
x_1, x_2	$\geq$	0	x_1, x_2	$\geq$	0
x_1, x_2	$\in$	$\mathbb{N}$	x_1, x_2	$\in$	$\mathbb{N}$
x_2	$\leq$	2	x_2	$\geq$	3

These two additional constraints are depicted in Figure 26.8(a). It appears clearly that problem P_2 is infeasible as no feasible solution of the original problem is such that x_2 is 3 or larger. Remember that the simplex algorithm is designed to detect infeasible problems (see Section 16.3). It is immediately discarded and $\mathcal{P} = \{P_1\}$. The feasible set of problem P_1 is represented in Figure 26.8(b).



(a) Additional branching constraints



(b) Feasible set of problem P_1

Figure 26.8: First branching for Example 26.4

It is seen that all feasible (that is, integer) points of the original problem are also feasible for P_1. Therefore, the optimal solution of P_1 is also the optimal solution of P_0. What has changed is the feasible set of the relaxation. The polygon has shrunk. Moreover, the optimal solution of the relaxation, $(2.5, 2.5)$ is now excluded from the feasible set of $R(P_1)$. Note that one more vertex of the polygon now has integer coordinates: $(3, 2)$. As the optimal solution is to be found at one vertex, we increase our chances to find an integer optimal solution by solving the relaxation.

However, the optimal solution of $R(P_1)$ is not integer: $x_1^* = (1.75, 2)$, with value $f(x_1^*) = -2.25$. Here, we branch on x_1, which is the only fractional variable. In the first subproblem $P_{11} = P_1^\ell$, we include the constraint $x_1 \leq \lfloor 1.75 \rfloor = 1$. In the second one $P_{12} = P_1^r$, we include the constraint $x_1 \geq \lceil 1.75 \rceil = 2$. We now have $\mathcal{P} = \{P_{11}, P_{12}\}$.

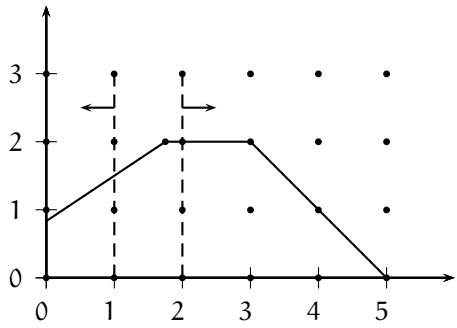
P_{11}			P_{12}		
$\min x_1 - 2x_2$			$\min x_1 - 2x_2$		
subject to			subject to		
$-4x_1 + 6x_2$	$\leq$	5	$-4x_1 + 6x_2$	$\leq$	5
$x_1 + x_2$	$\leq$	5	$x_1 + x_2$	$\leq$	5
x_1, x_2	$\geq$	0	x_1, x_2	$\geq$	0
x_1, x_2	$\in$	$\mathbb{N}$	x_1, x_2	$\in$	$\mathbb{N}$
x_2	$\leq$	2	x_2	$\leq$	2
x_1	$\leq$	1	x_1	$\geq$	2

These additional constraints are illustrated in Figure 26.9(a). The feasible set of $R(P_{12})$, which is the polygon on the right of Figure 26.9(b), appears to be such that each of its vertices corresponds to an integer solution. Therefore, it is guaranteed that one of its optimal solutions is integer and, therefore, also an optimal solution of P_{12}. In this case, the optimal solution is unique and is $x_{12}^* = (2, 2)$, with value $f(x_{12}^*) = -2$. It becomes therefore the current (and first) candidate to be the optimal solution of P: $x^* = (2, 2)$ and $f^* = -2$. Hence, P_{12} is discarded from $\mathcal{P}$ that now comprises $\{P_0, P_1, P_{11}\}$.

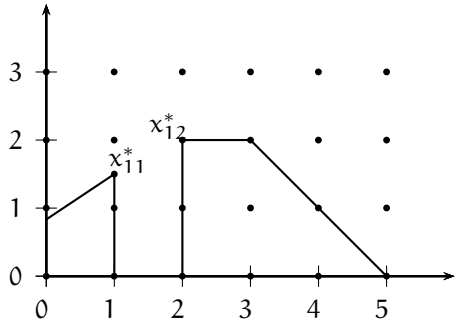
We now treat P_{11}. The optimal solution of $R(P_{11})$ is $x_{11}^* = (1, 1.5)$, with value $\ell(P_{11}) = f(x_{11}^*) = -2$. It is not an integer solution and, therefore, not a feasible solution of P_{11}. Because $f^* \leq \ell(P_{11})$, the problem can be discarded without being solved. Indeed, its optimal value cannot be better than -2. Therefore, the optimal solution of P_1 (and consequently of P_0) is the optimal solution of P_{12}, so that P_1 and P_0 can also be removed from $\mathcal{P}$, as they have been solved. We now have $\mathcal{P} = \emptyset$, and the algorithm terminates. An optimal solution of P is $x^* = (2, 2)$ with value $f^* = -2$.

26.2 Cutting planes

As illustrated by Example 26.4, it is possible to shrink the constraints polyhedron, without modifying the feasible set of the integer optimization problem. It is due to the fact that there are infinitely many polyhedra that characterize the same feasible set of integer solutions. Figure 26.10 represents four different polyhedra corresponding



(a) Additional branching constraints



(b) Feasible set of problem P_{11} and P_{12}

Figure 26.9: Second branching for Example 26.4

to exactly the same feasible set of integer solutions. Clearly, the last of them is the most appealing. Indeed, each vertex is integer, so that an optimal solution of the relaxation is also an optimal solution of the original problem. This polyhedron is the convex hull of the feasible solutions (see Definition B.3).

The idea of cutting planes methods is to start from the original formulation, include additional constraints that shrink the polyhedron without modifying the feasible set, and force some vertices of the new polyhedron to be integer. These additional constraints are called *valid inequalities*.

Definition 26.5 (Valid inequality)**.** Let $\mathcal{F} \subseteq \mathbb{R}^n$ be a set. The inequality $a^T x \geq b$ is a valid inequality for $\mathcal{F}$ if it is satisfied by all $x \in \mathcal{F}$.

It is good practice to include valid inequalities in the original formulation, by exploiting the properties of the problem. The modeling step should be designed to obtain a formulation that is as tight as possible. But it is usually not possible to identify the convex hull of the feasible set in that way. Cutting planes methods derive valid inequalities from the optimal solution of the relaxation. The most pop-

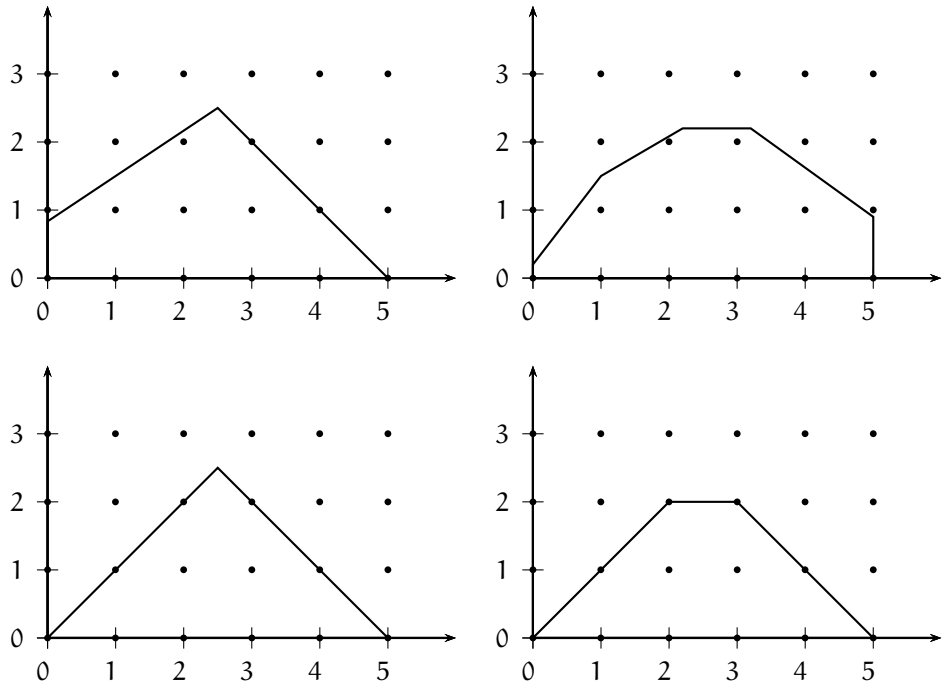


Figure 26.10: The same feasible set characterized by different polyhedra

ular method has been proposed by Gomory (1958) and exploits the simplex tableau introduced in Section 16.2.

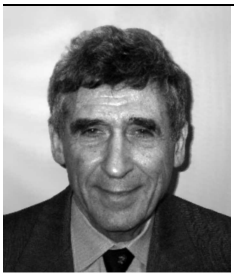
Consider the (mixed) integer linear optimization problem P. We solve its relaxation P using the simplex algorithm in two phases (Algorithm 16.5), and we obtain the optimal tableau

$B^{-1}A$	$B^{-1}b$
$c^T - c_B^T B^{-1} A$	$-c_B^T B^{-1} b$

where B contains the columns of A corresponding to the basic variables. The top part of the tableau contains a transformed version of the equality constraints. Separating the variables into basic variables x_B and non basic variables x_N, and denoting N the columns corresponding to non basic variables (see Section 3.4), it is written as

$$\begin{aligned} B^{-1}Ax &= B^{-1}b, \\ B^{-1}Bx_B + B^{-1}Nx_N &= B^{-1}b, \\ x_B + B^{-1}Nx_N &= B^{-1}b. \end{aligned} \tag{26.12}$$

To simplify the following equations, let us assume that the variables are numbered in such a way that the m first variables are basic, so that $x_i = (x_B)_i$. Denote α_{ij} the entry in row i and column j of the matrix $B^{-1}A$, which is obtained directly from the



Ralph Gomory was born on May 7, 1929, in Brooklyn Heights, New York, USA. He received his Ph.D. in mathematics from Princeton University in 1954. From 1957 to 1959, he was Assistant Professor at Princeton, where he interacted with Kuhn and Tucker. The cutting plane algorithm (Algorithm 26.3) was created during a project for the Navy, who insisted on obtaining integer solutions, while linear optimization provided fractional solutions. Gomory was responsible for IBM's Research Division between 1970 and 1986, when he became IBM Senior Vice President for Science and Technology. Since 1989, he has been the president of the Alfred P. Sloan Foundation. He is currently a Research Professor at New York University.

Figure 26.11: Ralph E. Gomory

tableau, remembering that $B^{-1}b = x_B^*$. Therefore, the i^{th} constraint is written as

$$x_i + \sum_{j \text{ non basic}} \alpha_{ij} x_j = (x_B^*)_i. \tag{26.13}$$

As x is feasible, it is non negative. Therefore, if we round down all coefficients α_{ij} in the left hand side of (26.13), its value cannot increase and we obtain a valid inequality for all feasible solutions of the relaxation:

$$x_i + \sum_{j \text{ non basic}} \lfloor \alpha_{ij} \rfloor x_j \leq x_i + \sum_{j \text{ non basic}} \alpha_{ij} x_j = (x_B^*)_i. \tag{26.14}$$

Rounding down the two sides of this inequality, we obtain another valid inequality for the feasible solutions of the relaxation:

$$\left\lfloor x_i + \sum_{j \text{ non basic}} \lfloor \alpha_{ij} \rfloor x_j \right\rfloor \leq \lfloor (x_B^*)_i \rfloor. \tag{26.15}$$

Now, consider only the x that are integer. Therefore,

$$\left\lfloor x_i + \sum_{j \text{ non basic}} \lfloor \alpha_{ij} \rfloor x_j \right\rfloor = x_i + \sum_{j \text{ non basic}} \lfloor \alpha_{ij} \rfloor x_j \tag{26.16}$$

and (26.15) is written as

$$x_i + \sum_{j \text{ non basic}} \lfloor \alpha_{ij} \rfloor x_j \leq \lfloor (x_B^*)_i \rfloor, \tag{26.17}$$

which is a valid inequality for all feasible solutions of the integer optimization problem. Equation (26.17) is called a *Gomory cut*.

We show now that the optimal solution x^* of the relaxation does not verify (26.17). Indeed, all non basic components of x^* are zero, and (26.17) is written as

$$x_i^* \leq \lfloor (x_B^*)_i \rfloor = \lfloor x_i^* \rfloor, \tag{26.18}$$

where $(x_B^*)_i = x_i^*$ because of our numbering convention. Inequality (26.18) is satisfied by x^* only if x_i^* is integer. Therefore, in order to generate a valid inequality that excludes x^* from the relaxation polyhedron, the index i must be chosen such that x_i^* is fractional. Algorithm 26.3 describes how Gomory cuts are used to solve an integer linear optimization problem.

We illustrate the method in Example 26.4. Note that in order for the relaxation to be solved by Algorithm 16.5, the problem must first be transformed into standard form by adding two slack variables:

$$\min x_1 - 2x_2$$

subject to

$$\begin{aligned} -4x_1 + 6x_2 + x_3 &= 5 \\ x_1 + x_2 + x_4 &= 5 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

The optimal tableau is

x_1	x_2	x_3	x_4		
0	1	0.1	0.4	2.5	x_2
1	0	−0.1	0.6	2.5	x_1
0	0	0.3	0.3	2.5	

The first constraint of the tableau corresponds to a fractional value of x_2. It is written as

$$x_2 + 0.1x_3 + 0.4x_4 = 2.5, \tag{26.19}$$

and the valid inequality (26.17) is written as

$$x_2 + \lfloor 0.1 \rfloor x_3 + \lfloor 0.4 \rfloor x_4 \leq \lfloor 2.5 \rfloor, \tag{26.20}$$

that is

$$x_2 \leq 2. \tag{26.21}$$

The second constraint of the tableau corresponds to a fractional value of x_1. It is written as

$$x_1 - 0.1x_3 + 0.6x_4 = 2.5, \tag{26.22}$$

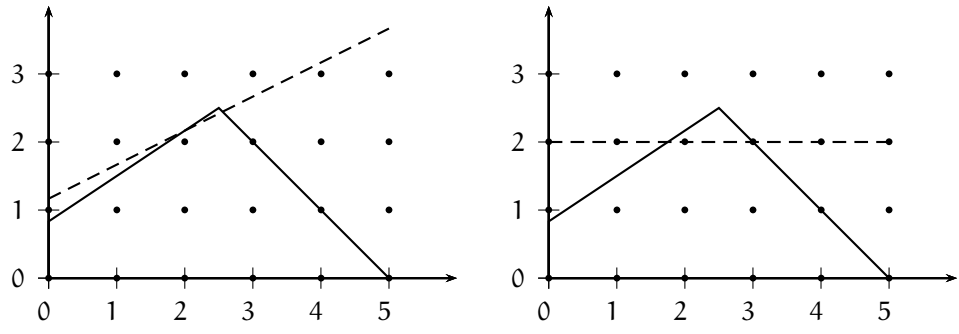
and generates the valid inequality

$$x_1 - x_3 \leq 2, \tag{26.23}$$

as $\lfloor -0.1 \rfloor = -1$, $\lfloor 0.6 \rfloor = 0$, and $\lfloor 2.5 \rfloor = 2$. As $x_3 = 5 + 4x_1 - 6x_2$, the valid inequality in the original variables is

$$-3x_1 + 6x_2 \leq 7. \tag{26.24}$$

These valid inequalities are illustrated in Figure 26.12.



(a) Gomory cut on x_1 (b) Gomory cut on x_2

Figure 26.12: Gomory cuts for Example 26.4

We now introduce the cut on x_2 in the formulation to obtain, after including another slack variable:

$$\min x_1 - 2x_2$$

subject to

$$\begin{aligned} -4x_1 + 6x_2 + x_3 &= 5 \\ x_1 + x_2 + x_4 &= 5 \\ x_2 + x_5 &= 2 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

x_1	x_2	x_3	x_4	x_5		
0	1	0	0	1	2	x_2
0	0	0.25	1	−2.5	1.25	x_4
1	0	−0.25	0	1.5	1.75	x_1
0	0	0.25	0	0.5	2.25	

Variables x_1 and x_4 are fractional. Therefore, the following valid inequalities can be generated:

$$x_4 - 3x_5 \leq 1 \tag{26.25}$$

for x_4 and

$$x_1 - x_3 + x_5 \leq 1 \tag{26.26}$$

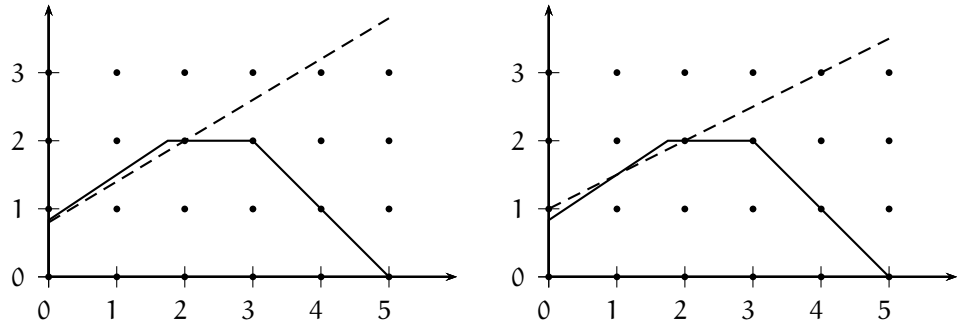
for x_1. In the original variables, these are

$$-x_1 + 2x_2 \leq 2 \tag{26.27}$$

and

$$-3x_1 + 5x_2 \leq 4. \tag{26.28}$$

These cuts are illustrated in Figure 26.13. They both have the property that they cut the polygon at $(2, 2)$, which becomes a vertex. Actually, the optimal solution of the relaxation of any of these two problems is $(2, 2)$, which is also the optimal solution of the integer optimization problem. No more cuts are necessary.



(a) Gomory cut on x_1 (b) Gomory cut on x_4

Figure 26.13: More Gomory cuts for Example 26.4

Algorithm 26.3: Gomory cuts for integer linear optimization

Objective

Find a global minimum of the integer linear optimization problem P: $\min_x c^T x$ subject to $Ax = b$, $x \geq 0$, $x \in \mathbb{Z}^n$.

Input

The matrix $A \in \mathbb{R}^{m \times n}$.

The vector $b \in \mathbb{R}^m$.

The vector $c \in \mathbb{R}^n$.

Output

A global minimum x^*.

Repeat

Solve the relaxation using Algorithm 16.5 with A, b and c.

Call x_R^* the optimal solution and T^* the optimal tableau.

if x_R^* *is integer* **then**

$x^* = x_R^*$

else

Let $i \leq m$ be such that $T^*(i, n+1)$ is fractional

Let $\gamma := (\lfloor T^*(i,1) \rfloor, \ldots, \lfloor T^*(i,n) \rfloor)$ be the first n elements of row i of T^* rounded down

Let $A := \begin{pmatrix} A & 0 \\ \gamma & 1 \end{pmatrix}$, $b := \begin{pmatrix} b \\ \lfloor T^*(i, n+1) \rfloor \end{pmatrix}$ and $c := \begin{pmatrix} c \\ 0 \end{pmatrix}$

$m := m + 1$, $n := n + 1$.

Until x^* *is integer.*

We provide another illustration of the Gomory cuts with Example 25.13. The feasible set is represented in Figure 25.2. After the introduction of the slack variables, we obtain the relaxation:

$$\min_{x \in \mathbb{N}^2} -3x_1 - 13x_2 \tag{26.29}$$

subject to

$$\begin{aligned} 2x_1 + 9x_2 + x_3 &= 29 \\ 11x_1 - 8x_2 + x_4 &= 79 \\ x_1, x_2, x_3, x_4 &\geq 0. \end{aligned} \tag{26.30}$$

The optimal tableau of the relaxation is

x_1	x_2	x_3	x_4		
0	1	0.10	−0.02	1.4	x_2
1	0	0.07	0.08	8.2	x_1
0	0	1.45	0.01	42.8	

The two variables are fractional, so two valid inequalities can be generated:

$$x_2 - x_4 \leq 1 \tag{26.31}$$

and

$$x_1 \leq 8. \tag{26.32}$$

As $x_4 = 79 - 11x_1 + 8x_2$, the first inequality in the original variables is written as

$$11x_1 - 7x_2 \leq 80. \tag{26.33}$$

These two cuts are illustrated in Figure 26.14. It appears in this example that the part of the polyhedron that is cut can be small. We expect, in this case, the algorithm to take a while to converge. Depending on the cut selected for inclusion at each iteration, the algorithm may use more than 50 iterations to find the optimal solution of this problem.

In practice, the cutting plane method is usually combined with the branch and bound algorithm described in Section 26.1. In addition to the additional constraints generated by the branching strategy, valid inequalities such as Gomory cuts are also included. Such a method is called *branch and cut*.

26.3 Exercises

Exercise 26.1. Find better bounds for Example 26.3.

Exercise 26.2. Consider the assignment problem presented as Exercise 22.2.

1. Solve it using the branch and bound algorithm (Algorithm 26.1).
2. Consider its mathematical formulation as a transhipment problem (see Exercise 22.2). Solve it with the simplex algorithm (Algorithm 16.5).

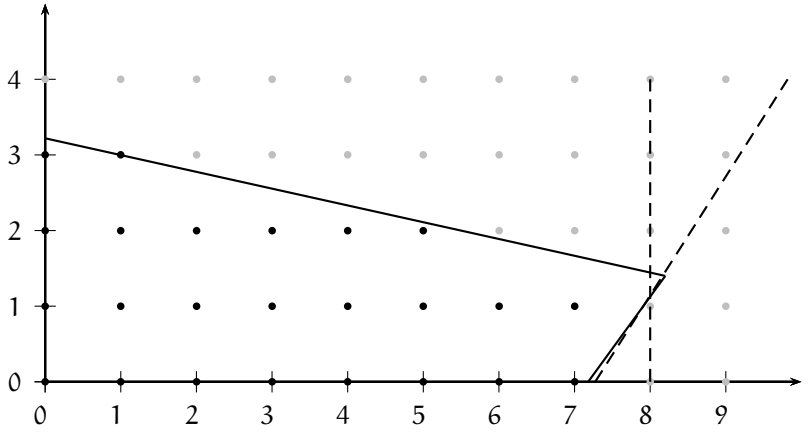


Figure 26.14: The feasible set for Example 25.13 with Gomory cuts

26.4 Project

The general organization of the projects is described in Appendix D.

Objective

The objective of the project is to analyze how different exact methods handle different optimization problems.

Approach

1. For each problem,
   - apply the branch and bound algorithm (Algorithm 26.2) using different strategies to select the next problem to solve (step 11 of the algorithm):
     (a) select the problem associated with the best bound,
     (b) select the last problem that has been introduced in $\mathcal{P}$ (called last-in-first-out or depth-first strategy),
     (c) select the first problem that has been introduced in $\mathcal{P}$ (called first-in-first-out or breadth-first strategy);
   - apply the Gomory cut algorithm (Algorithm 26.3).
2. Report, for each run, the number of problems that have been processed and the running time.

Algorithms

Algorithms 26.2 and 26.3.

Problems

Exercise 26.3. Solve the instance of the problem of locating plants for the supply of energy described in Example 25.1, with 10 sites and 3 cities, using the data in Table 25.1.

Exercise 26.4. Solve the knapsack problem presented in Example 27.2.

Exercise 26.5. Write and solve the traveling salesman problem with 16 cities presented in Example 27.3 as an optimization problem using the method described in Section 25.2.3.

Exercise 26.6. Solve the task assignment problem of Exercise 25.1.

Exercise 26.7. Solve the scheduling problem of Exercise 25.2.

Exercise 26.8. Solve the graph coloring problem of Exercise 25.3. Write also the version of the problem where you must color the cantons in Switzerland.

Exercise 26.9. Solve the bin packing problem of Exercise 25.4.

Exercise 26.10. Solve the vehicle routing problem of Exercise 25.5 with $Q = 100{,}000$, $Q = 150{,}000$, and $Q = 200{,}000$.

Chapter 27

Heuristics

Contents

Due to the combinatorial nature of the integer optimization problems, the methods described in Chapter 26 may fail to identify the optimal solution in a reasonable amount of time. Also, the number of subproblems to consider in a branch and bound tree may be so high that the tree may sometimes not fit into the memory of the computer. Under such circumstances, it is hopeless to find the optimal solution of the problem. Instead, for all practical purposes, we are interested in efficient techniques that identify a "good" solution, meaning a solution that is feasible and (hopefully) significantly better than a solution that would have been designed "by hand" by a human being, expert in the problem. Such techniques are called *heuristic algorithms*.

Definition 27.1 (Heuristic algorithm)**.** A heuristic algorithm is a method that explores the set of feasible solutions of an optimization problem, exploiting the structure of the problem to identify quickly good feasible solutions.

The above definition is relatively vague, as it involves vague adjectives such as "intuitive," "quickly" and "good." Designing heuristics is more of an art than a science. Indeed, the success of a heuristic depends on the level of intuition that can be developed about a problem, on the time that is available to obtain a practical solution, and on the exact concrete characterization of what "good" means in the given context.

For these reasons, the heuristics introduced in this chapter are described for specific problems, in order to illustrate the concepts. We refer the reader to the large literature for additional examples (see, among others, Gendreau and Potvin, 2010).

27.1 Greedy heuristics

A *greedy* heuristic usually refers to a method that constructs a feasible solution step by step in a way that each step is locally optimal (see Definition 21.22). We illustrate the idea on the knapsack problem and the traveling salesman problem.

27.1.1 The knapsack problem

An example of a greedy algorithm to solve the knapsack problem consists in sorting the items by decreasing order of their relative value, that is, by decreasing value of the ratio u_i/w_i. The items are considered in that sequence and, for each of them, if there is enough capacity left in the knapsack, it is added.

Example 27.2 (A larger knapsack problem)**.** Consider a knapsack problem with 12 items. The utilities and the weights are as follows:

i	1	2	3	4	5	6	7	8	9	10	11	12
u	80	31	48	17	27	84	34	39	46	58	23	67
w	84	27	47	22	21	96	42	46	54	53	32	78

The capacity of the knapsack is 300.

Consider Example 27.2. We sort the items according to their relative value:

i	5	2	10	3	1	6	12	9	8	7	4	11
u_i	27	31	58	48	80	84	67	46	39	34	17	23
w_i	21	27	53	47	84	96	78	54	46	42	22	32
$\frac{u_i}{w_i}$	1.29	1.15	1.09	1.02	0.95	0.88	0.86	0.85	0.85	0.81	0.77	0.72

We proceed through the list and include items 5, 2, 10, 3, and 1, for a total weight of 232 and the total utility to 244. Items 6 and 12 cannot be included, as it would violate the capacity constraint. Item 9 can be included, increasing the weight to 286 and the utility to 290. None of the remaining items can be included anymore.

27.1.2 The traveling salesman problem

An example of a greedy algorithm for the traveling salesman problem is to start from home and, at each step, select the closest city as the next one, as described in Algorithm 27.1.

Algorithm 27.1: Nearest neighbor greedy algorithm

Objective
  Find a good tour for the traveling salesman problem.
Input
  The number of cities n.
  The distance $d(i,j)$, $i = 1,\dots,n$, $j = 1,\dots,n$, $i \neq j$.
Output
  A sequence T of cities.
Initialization
  $T := \{1\}$.
  $S := \{2,\dots,n\}$.
  $c := 1$.
Repeat
  Select $i \in \operatorname{argmin}_{j \in S} d(c,j)$.
  $c := i$.
  $T := T \cup \{i\}$.
  $S := S \setminus \{i\}$.
Until $S = \emptyset$.

To illustrate the algorithm, we consider an instance of the TSP with 16 cities, as described in Example 27.3.

Example 27.3 (The traveling salesman problem: 16 cities)**.** A salesman must visit 15 customers during the day. Starting from home, he has to plan a tour, that is, a sequence of customers to visit, in order to minimize the travel distance. We model it with a graph, where each vertex represents either the home place or a customer. There is an edge between each pair of vertices, and the cost of the edge is the Euclidean straight distance between two vertices. The location of the home place (vertex 1) and of the 15 customers are reported in Table 27.1 and illustrated in Figure 27.1.

Table 27.1: Traveling salesman problem: position of the home (vertex 1) and of 15 customers to visit

Vertex	x-coord	y-coord	Vertex	x-coord	y-coord
1	9.14	3.92	9	3.41	27.54
2	18.46	1.17	10	13.63	22.11
3	28.35	9.72	11	24.41	22.10
4	39.41	7.39	12	39.90	22.64
5	8.87	10.33	13	3.37	30.48
6	12.56	17.55	14	14.52	39.34
7	27.29	13.97	15	21.75	37.64
8	32.31	10.78	16	32.48	30.06

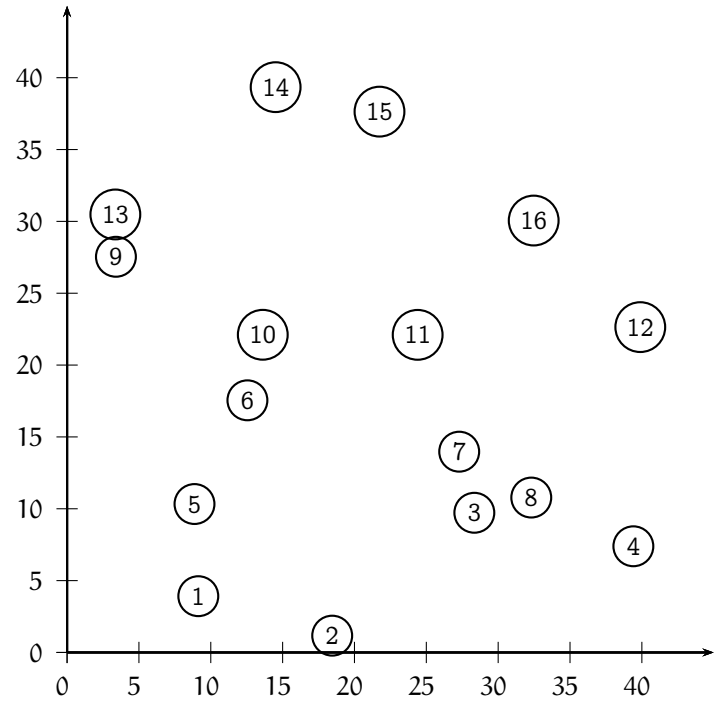


Figure 27.1: Traveling salesman problem: position of the home (vertex 1) and of 15 customers to visit

We obtain an itinerary of length 158.5. illustrated in Figure 27.2.

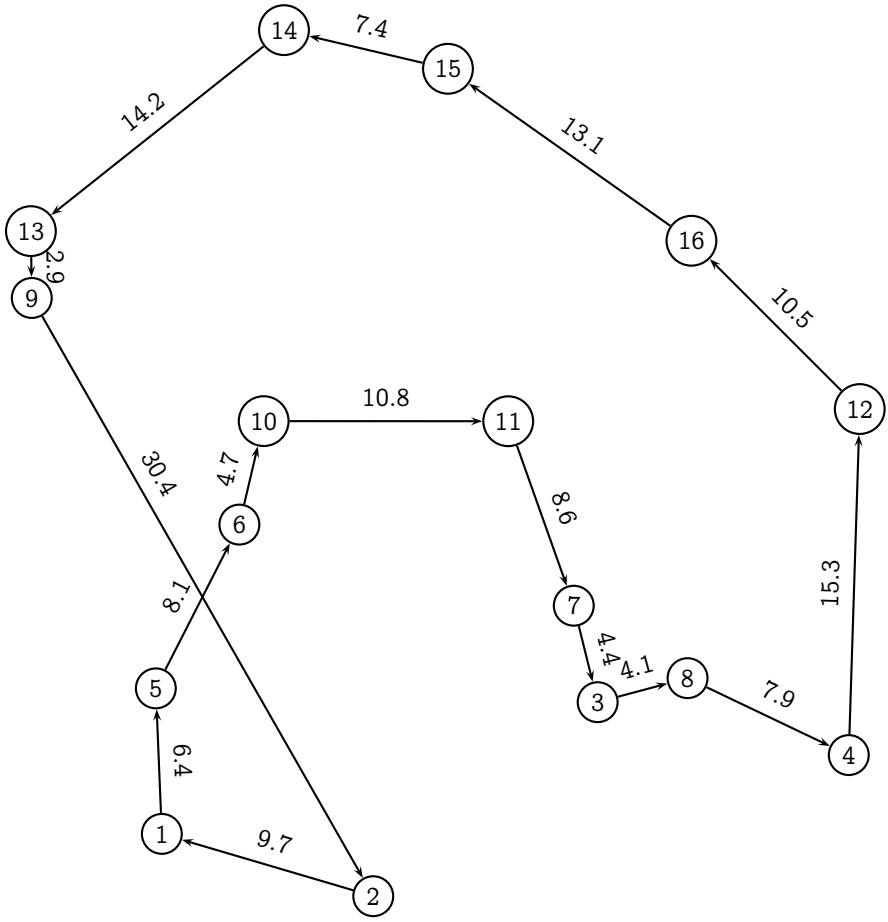


Figure 27.2: Feasible solution provided by the nearest neighbor greedy algorithm for Example 27.3 (length: 158.5)

Another greedy heuristic for the traveling salesman problem consists in improving an existing subtour by inserting a vertex. An example of insertion is illustrated in Figure 27.3, where a subtour of length 134.4 is constructed by inserting city 14 into a subtour of 12 cities (of length 108.1). Note that it is not the best possible insertion, which would consist in inserting city 15 after city 16 in the tour and obtaining a subtour of length 125.6. The greedy algorithm consists in selecting the best possible insertion at each step, as described in Algorithm 27.2.

The iterations of the insertion greedy algorithm (Algorithm 27.2) on Example 27.3 are reported in Table 27.2, and the final tour illustrated in Figure 27.4.

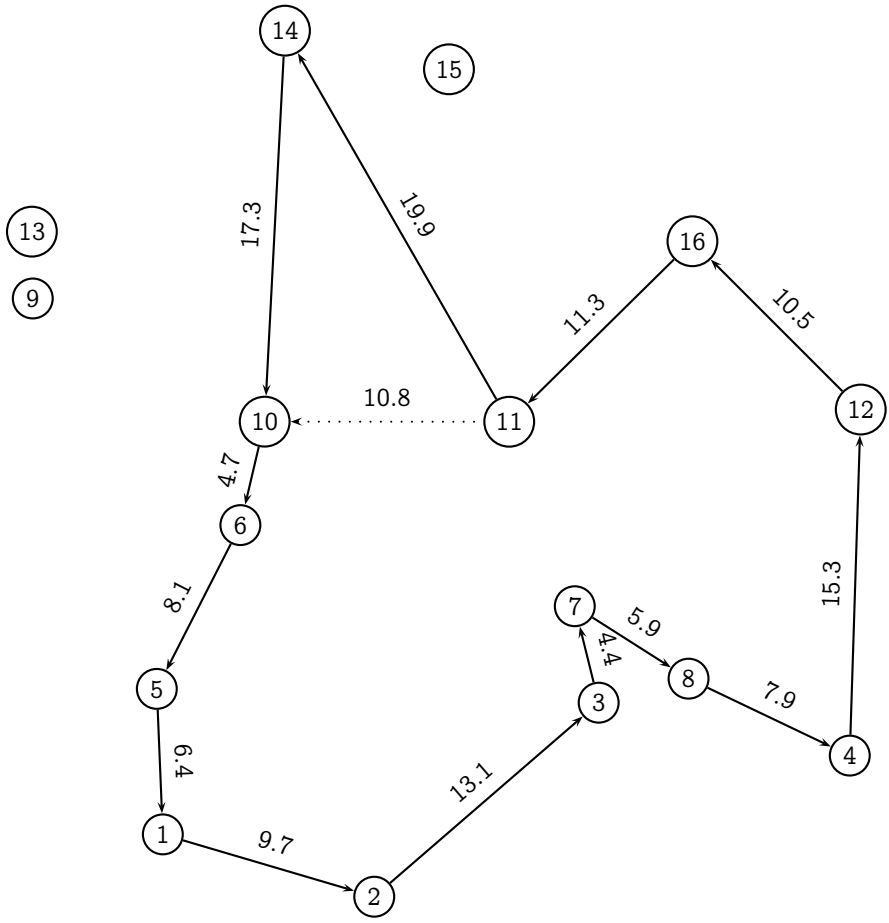


Figure 27.3: Example of a subtour involving 12 cities, of length 108.1, with insertion of city 14

Algorithm 27.2: Insertion greedy algorithm

```
1  Objective
2      Find a good tour for the traveling salesman problem.
3  Input
4      The number of cities n.
5      The distance d(i, j), i = 1, ..., n, j = 1, ..., n, i ≠ j.
6      ℓ(T, i, j) calculates the length of the tour obtained by inserting city i after j
         in T.
7      Initial subtour: sequence of cities T, including home (vertex 1).
8  Output
9      A sequence T of cities.
10 Initialization
11     S = {1, ..., n} \ T.
12 Repeat
13     L = ∞.
14     for i ∈ S do
15         for j ∈ T do
16             if ℓ(T, i, j) < L then
17                 i* := i, j* := j
18                 L := ℓ(T, i, j)
19     Insert vertex i* after vertex j* in T
20     S := S \ {i*}.
21 Until S = ∅.
```

Table 27.2: Iterations of the insertion greedy algorithm (Algorithm 27.2) on Example 27.3

Length	Subtour															
12.8	1	5	1													
28.6	1	**6**	5	1												
37.9	1	6	**10**	5	1											
50.9	1	**2**	6	10	5	1										
64.2	1	2	**3**	6	10	5	1									
66.1	1	2	3	**7**	6	10	5	1								
71.8	1	2	3	**8**	7	6	10	5	1							
78.0	1	2	3	8	7	**11**	6	10	5	1						
93.1	1	2	3	**4**	8	7	11	6	10	5	1					
110.0	1	2	3	4	8	7	11	6	10	**9**	5	1				
114.6	1	2	3	4	8	7	11	6	10	**13**	9	5	1			
132.9	1	2	3	4	8	7	11	6	10	**14**	13	9	5	1		
140.5	1	2	3	4	8	7	11	6	10	**15**	14	13	9	5	1	
156.6	1	2	3	4	8	7	11	6	10	**16**	15	14	13	9	5	1
172.9	1	2	3	4	8	7	11	6	10	**12**	16	15	14	13	9	5	1

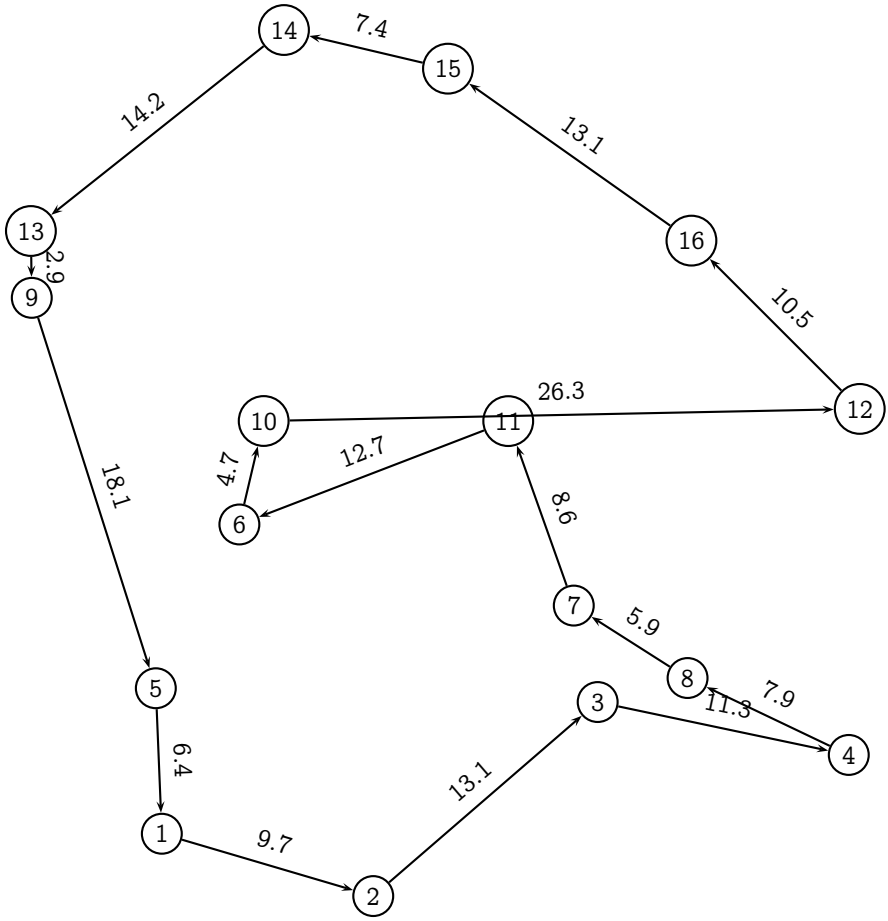


Figure 27.4: Result of the insertion greedy algorithm (Algorithm 27.2) on Example 27.3 (length: 172.9)

27.2 Neighborhood and local search

As emphasized in Definition 27.1, heuristic algorithms should explore the large set of feasible solutions. This is not really achieved by the greedy algorithms, which generate only one (hopefully good) feasible solution. Such an exploration requires a proper exploration tool that generates a sequence of feasible solutions. Potentially, it should be able to generate *all* feasible solutions. Such a tool is called a *neighborhood structure*.

Definition 27.4 (Neighborhood structure)**.** A neighborhood structure N is a function that associates a solution x of the optimization problem with a set $N(x)$ of other solutions (not necessarily feasible). Each element of $N(x)$ is called a *neighbor* of x.

In general, the neighborhood of a feasible solution x is defined by a set of simple modifications of x, each of them generating a neighbor.

For instance, consider an integer optimization problem and $x \in \mathbb{Z}^n$ a feasible solution. For each index k, we generate two neighbors by increasing and decreasing the value of x_k by 1. The two neighbors, denoted by y^{k+} are y^{k-}, are defined as

$$y_i^{k+} = y_i^{k-} = x_i, \forall i \neq k, \quad y_k^{k+} = x_k + 1, \quad y_k^{k-} = x_k - 1. \tag{27.1}$$

For example, if $k = 2$,

$$x = (3, \mathbf{5}, 2, 8) \quad y^{2+} = (3, \mathbf{6}, 2, 8) \quad y^{2-} = (3, \mathbf{4}, 2, 8).$$

The simple modifications of x characterize a neighborhood of size $2n$. It is illustrated, for $n = 2$, by Figure 27.5.

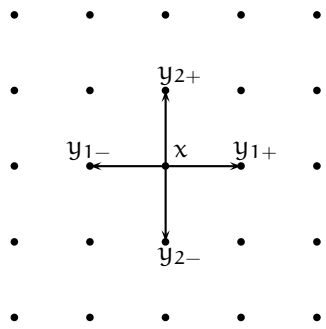


Figure 27.5: A simple neighborhood structure

There is an infinite number of possible neighborhood structures, and the choice of a specific structure should be motivated by the properties of the optimization problem. Creativity is required here. Another example of a neighborhood for an

integer optimization problem is illustrated in Figure 27.6, where the neighbors are generated using moves inspired by the knights of a chess game. Other examples of neighborhood structures are provided later in this chapter. When dealing with practical applications, it is good practice to get inspiration from experts from the field when defining a neighborhood structure. In particular, a good way to define a neighborhood is to mimic how an expert would modify an existing feasible solution in order to try to improve it.

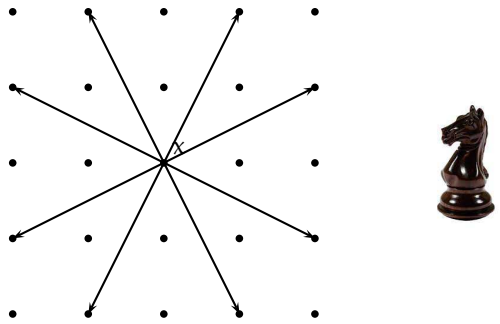


Figure 27.6: Another neighborhood structure

An important property of a good neighborhood structure is that solving an optimization problem within the neighborhood of a feasible solution x should be an easy task. Typically, it should be feasible to perform an exhaustive enumeration of its elements. Algorithm 27.3, called *local search*, directly exploits this feature.

Algorithm 27.3: Local search

Objective

Find a good feasible solution of the optimization problem $\min_x f(x)$ subject to $x \in \mathcal{F}$.

Input

The objective function $f : \mathbb{R}^n \to \mathbb{R}$.

The feasible set $\mathcal{F}$.

A neighborhood structure N.

An initial feasible solution $x_0 \in \mathcal{F}$ such that $N(x_0) \cap \mathcal{F} \neq \emptyset$.

Output

A feasible solution x^*.

Repeat

Select $x_c \in \operatorname{argmin}_{x \in N(x_k) \cap \mathcal{F}} f(x)$

if $f(x_c) < f(x_k)$ **then**

$x_{k+1} := x_c$

$k := k + 1$

Until $f(x_c) = f(x_k)$ *or* $N(x_k) \cap \mathcal{F} = \emptyset$.

$x^* := x_k$.

The idea is simple. At each iteration, the current iterate is replaced by its best feasible neighbor, until the current feasible solution is the best in the neighborhood. Therefore, Algorithm 27.3 generates a local minimum with respect to the given neighborhood structure. Note that this concept of local minimum is a generalization of the one introduced for continuous optimization (Definition 1.5), where the neighborhood was defined as all points within a ball around x.

The main issue with heuristic methods is that there is no theoretical support for their validity. Only empirical evidences can be derived. However, we can suggest some properties that a neighborhood should carry in order to be used in Algorithm 27.3.

- An element x belongs to its own neighborhood: for all x, $x \in N(x)$. This property is important only to characterize a local minimum as $x^* \in \operatorname{argmin}_{x \in N(x) \cap \mathcal{F}} f(x)$.
- The neighborhood structure is symmetric, that is, for all x and y, $x \in N(y)$ if and only if $y \in N(x)$.
- The neighborhood structure should allow any feasible point to be reached from any other feasible point in a finite number of steps.
- The size of the neighborhood structure should not grow too fast with the size of the problem. The optimization problem solved at each iteration within the neighborhood structure must be tractable.

There are several variants of the local search method. One of them consists in evaluating the neighbors in a given order, and to accept as next iterate the first one that is better than the current iterate (Algorithm 27.4). This may allow computational time to be saved in early iterations, when many neighbors are better than the current feasible solution. It means that the neighborhood structure should be associated with an order of its elements.

For large neighborhoods, some variants propose to randomly select a fixed number of candidates in the neighborhood. If none of these candidates is better than the current iterate, the algorithm is interrupted. The advantage of this approach is that the computational complexity of the algorithm can be controlled independently of the size of the neighborhood.

Consider Example 25.13, together with the neighborhood structure defined by (27.1) and illustrated in Figure 27.5. We apply Algorithm 27.3 with $x_0 = (6, 0)$. The iterations are described in Table 27.3 and illustrated in Figure 27.7. The starting point $x_0 = (6, 0)$ has four neighbors. The point $(6, -1)$ is infeasible. Among the others, $(6, 1)$ is associated with the lowest value of the objective function and is selected as the next iterate. Among its four neighbors, $(6, 2)$ is infeasible and $(7, 1)$ is selected as the next iterate. Among its neighbors, two are infeasible and two have a higher value of the objective function. Therefore, $(7, 1)$ is a local minimum for this neighborhood structure.

Algorithm 27.4: Local search: a variant

Objective
  Find a good feasible solution of the optimization problem $\min_x f(x)$ subject to $x \in \mathcal{F}$.
Input
  The objective function $f : \mathbb{R}^n \to \mathbb{R}$.
  The feasible set $\mathcal{F}$.
  A neighborhood structure N such that for each x, $N(x)$ is an ordered sequence of solutions.
  An initial feasible solution $x_0 \in \mathcal{F}$ such that $N(x_0) \cap \mathcal{F} \neq \emptyset$.
Output
  A feasible solution x^*.
Repeat
  improvement :=FALSE .
  for $x_c \in N(x_k) \cap \mathcal{F}$ **do**
    if $f(x_c) < f(x_k)$ **then**
      improvement :=TRUE
      $x_{k+1} := x_c$
      $k := k + 1$
      break `// The "for" loop is interrupted`
Until *improvement = FALSE or* $N(x_k) \cap \mathcal{F} = \emptyset$.
$x^* := x_k$.

Table 27.3: Local search on Example 25.13 with $x_0 = (6, 0)$

	x_k	Neighbors			
x	(6,0)	(7,0)	(5,0)	(6,1)	(6,-1)
f(x)	-18	-21	-15	-31	—
x	(6,1)	(7,1)	(5,1)	(6,2)	(6,0)
f(x)	-31	-34	-28	—	-18
x	(7,1)	(8,1)	(6,1)	(7,2)	(7,0)
f(x)	-34	—	-31	—	-21

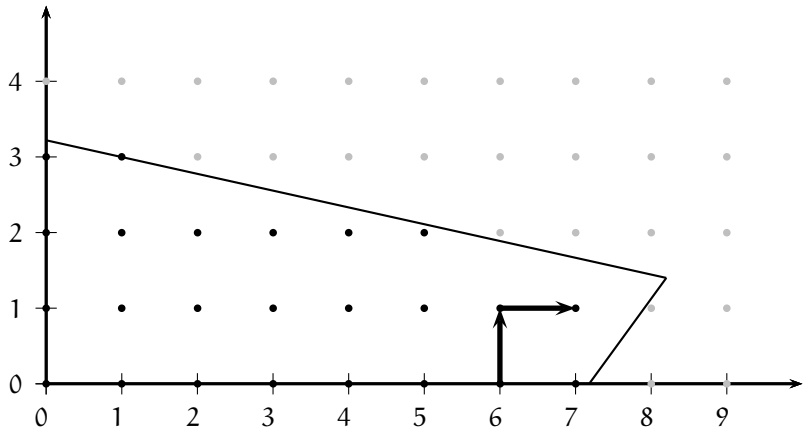


Figure 27.7: Local search on Example 25.13 with $x_0 = (6, 0)$

If another starting point is selected, a different local minimum can be reached. The iterations starting from $(2, 0)$ are reported in Table 27.4 and illustrated in Figure 27.8.

Table 27.4: Local search on Example 25.13 with $x_0 = (2, 0)$

	x_k	Neighbors			
x	(2,0)	(3,0)	(1,0)	(2,1)	(2,-1)
f(x)	-6	-9	-3	-19	—
x	(2,1)	(3,1)	(1,1)	(2,2)	(2,0)
f(x)	-19	-22	-16	-32	-6
x	(2,2)	(3,2)	(1,2)	(2,3)	(2,1)
f(x)	-32	-35	-29	—	-19
x	(3,2)	(4,2)	(2,2)	(3,3)	(3,1)
f(x)	-35	-38	-32	—	-22
x	(4,2)	(5,2)	(3,2)	(4,3)	(4,1)
f(x)	-38	-41	-35	—	-25
x	(5,2)	(6,2)	(4,2)	(5,3)	(5,1)
f(x)	-41	—	-38	—	-28

Similarly, if a different neighborhood structure is selected, the algorithm may also end up at a different local minimum. The iterations starting from $(2, 0)$, using the neighborhood inspired by the chess knights (see Figure 27.6), are reported in Table 27.5 and illustrated in Figure 27.9.

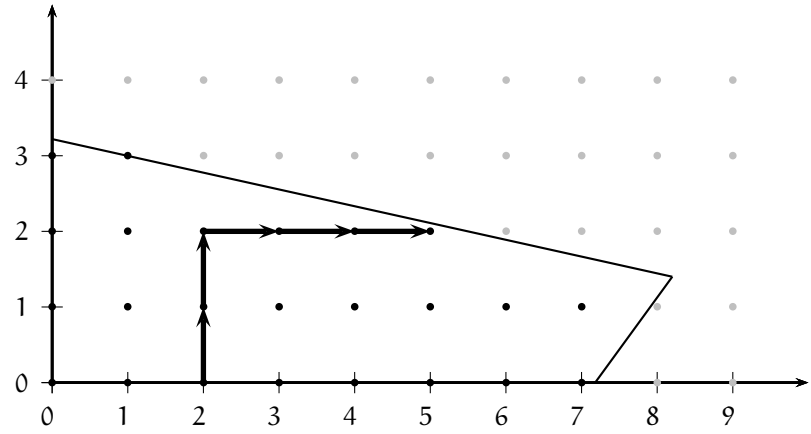


Figure 27.8: Local search on Example 25.13 with $x_0 = (2, 0)$

Table 27.5: Local search on Example 25.13 with the "knight" neighborhood and $x_0 = (2, 0)$

	x_k	Neighbors							
x	(2,0)	(4,1)	(4,-1)	(0,1)	(0,-1)	(3,2)	(3,-2)	(1,2)	(1,-2)
f(x)	-6	-25	—	-13	—	-35	—	-29	—
x	(3,2)	(5,3)	(5,1)	(1,3)	(1,1)	(4,4)	(4,0)	(2,4)	(2,0)
f(x)	-35	—	-28	-42	-16	—	-12	—	-6
x	(1,3)	(3,4)	(3,2)	(-1,4)	(-1,2)	(2,5)	(2,1)	(0,5)	(0,1)
f(x)	-42	—	-35	—	—	—	-19	—	-13

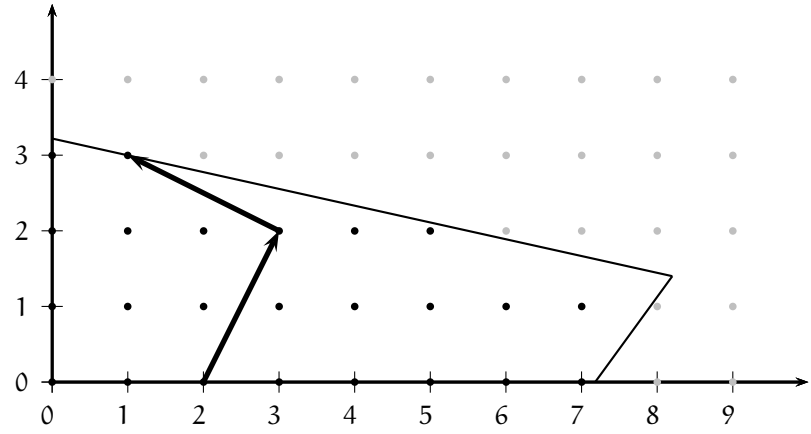


Figure 27.9: Local search on Example 25.13 with the "knight" neighborhood and $x_0 = (2, 0)$

27.2.1 The knapsack problem

We illustrate the local search method described by Algorithm 27.4 on the 0–1 knapsack problem. To do so, we need to define a feasible starting point and a neighborhood structure. There is an obvious feasible solution that can be considered as a starting point: the empty sack. It consists in carrying no item, that is $x_i = 0$, $i = 1, \ldots, n$. Consider now a configuration of the knapsack characterized by the vector $x \in \{0,1\}^n$. A neighbor of x is obtained by selecting one item i and by changing the decision with respect to it. If the item belongs to the knapsack, we remove it. Otherwise, we include it. Therefore, we define $N(x) = \{x, y^i,\ i = 1, \ldots, n\}$, where y^i is defined as

$$\begin{aligned} y^i_j &= x_j \qquad \forall j \neq i, \\ y^i_i &= 1 - x_i. \end{aligned}$$

The iterations of the local search algorithm (Algorithm 27.4) for Example 27.2 are reported in Table 27.6. Each block represents an iteration with the list of neighbors that have been considered, the last one being selected for the next iteration. In the last block, all neighbors are rejected, so that the current iterate is a local minimum. The interpretation of these iterations is simple: each item is included one by one into the knapsack until the next one does not fit. A total of 6 items can fit, for a total weight of 297 and a total utility of 203. Note that the greedy algorithm presented in Section 27.1.1 found a feasible solution with weight 286 and utility 290.

The simple neighborhood structure presented above can be generalized. We define a neighborhood of size k by selecting k items, and modify the decision about them. In particular, if $k = 1$, there are n neighbors, and the neighborhood structure is the one used earlier. If $k = n$, there is only 1 neighbor, obtained by changing the status of all the items. The size of this neighborhood is

$$\frac{n!}{k!(n-k)!} \approx \frac{2^{n+\frac{1}{2}}}{\sqrt{\pi n}}, \tag{27.2}$$

where the approximation holds when n is large and $k = n/2$. Therefore, the size of this neighborhood grows exponentially with the size of the problem, which is not desirable. In order to avoid that, the neighborhood is defined by the random selection of a fixed number of neighbors, as defined in Algorithm 27.5. In this case, the size of the neighborhood is bounded above by M, irrespectively of the values of k and n. Note the randomization of the procedure, which prevents items being considered in the same order. Note also that only feasible solutions are considered in the neighborhood. This illustrates the flexibility of the neighborhood definition.

This procedure may generate the same neighbor several times, as the random draws are made with replacement. The exact size of the neighborhood varies from run to run, as infeasible combinations are discarded. It may also generate an empty sequence, if the feasibility test at step 19 fails for each selected item. The use of this neighborhood is illustrated in Section 27.3.1.

Algorithm 27.5: Neighborhood of size k for the knapsack problem

Objective

Generate an ordered list of feasible neighbors for the knapsack problem.

Input

The number of items n.
For each item $i = 1, \ldots, n$, its status $x_i \in \{0, 1\}$.
The vector of weights w.
The capacity W.
The size of the neighborhood k.
The maximum number of trials M.

Output

An ordered list of feasible neighbors.

Initialization

$\mathcal{N} = \emptyset$.
$m = 1$.

Repeat

$x_c := x$.
Select randomly k items $i_1, \ldots, i_k$.
$(x_c)_{i_j} := 1 - (x_c)_{i_j}$, $j = 1, \ldots, k$.
if $x_c^T w \leq W$ **then**
  $\mathcal{N} := \mathcal{N} \cup x_c$
$m := m+1$.

Until $m = M$.

Table 27.6: Iterations of Algorithm 27.4 on Example 27.2

k	1	2	3	4	5	6	7	8	9	10	11	12	$w^T x_c$	$u^T x_c$	$u^T x^*$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	84	80	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80
2	1	1	0	0	0	0	0	0	0	0	0	0	111	111	80
	0	1	0	0	0	0	0	0	0	0	0	0	27	31	111
	1	0	0	0	0	0	0	0	0	0	0	0	84	80	111
3	1	1	1	0	0	0	0	0	0	0	0	0	158	159	111
	0	1	1	0	0	0	0	0	0	0	0	0	74	79	159
	1	0	1	0	0	0	0	0	0	0	0	0	131	128	159
	1	1	0	0	0	0	0	0	0	0	0	0	111	111	159
4	1	1	1	1	0	0	0	0	0	0	0	0	180	176	159
	0	1	1	1	0	0	0	0	0	0	0	0	96	96	176
	1	0	1	1	0	0	0	0	0	0	0	0	153	145	176
	1	1	0	1	0	0	0	0	0	0	0	0	133	128	176
	1	1	1	0	0	0	0	0	0	0	0	0	158	159	176
5	1	1	1	1	1	0	0	0	0	0	0	0	201	203	176
	0	1	1	1	1	0	0	0	0	0	0	0	117	123	203
	1	0	1	1	1	0	0	0	0	0	0	0	174	172	203
	1	1	0	1	1	0	0	0	0	0	0	0	154	155	203
	1	1	1	0	1	0	0	0	0	0	0	0	179	186	203
	1	1	1	1	0	0	0	0	0	0	0	0	180	176	203
6	1	1	1	1	1	1	0	0	0	0	0	0	297	287	203
	0	1	1	1	1	1	0	0	0	0	0	0	213	207	287
	1	0	1	1	1	1	0	0	0	0	0	0	270	256	287
	1	1	0	1	1	1	0	0	0	0	0	0	250	239	287
	1	1	1	0	1	1	0	0	0	0	0	0	275	270	287
	1	1	1	1	0	1	0	0	0	0	0	0	276	260	287
	1	1	1	1	1	0	0	0	0	0	0	0	201	203	287
	1	1	1	1	1	1	1	0	0	0	0	0	339	321	287
	1	1	1	1	1	1	0	1	0	0	0	0	343	326	287
	1	1	1	1	1	1	0	0	1	0	0	0	351	333	287
	1	1	1	1	1	1	0	0	0	1	0	0	350	345	287
	1	1	1	1	1	1	0	0	0	0	1	0	329	310	287
7	1	1	1	1	1	1	0	0	0	0	0	1	375	354	287

27.2.2 The traveling salesman problem

We illustrate the local search method in Example 27.3 involving 16 cities. In order to initiate the iterations, we generate a random permutation of the customers and obtain

$$1, 7, 10, 3, 16, 15, 11, 4, 6, 14, 8, 2, 12, 13, 5, 9,$$

illustrated in Figure 27.10, with length 358.6.

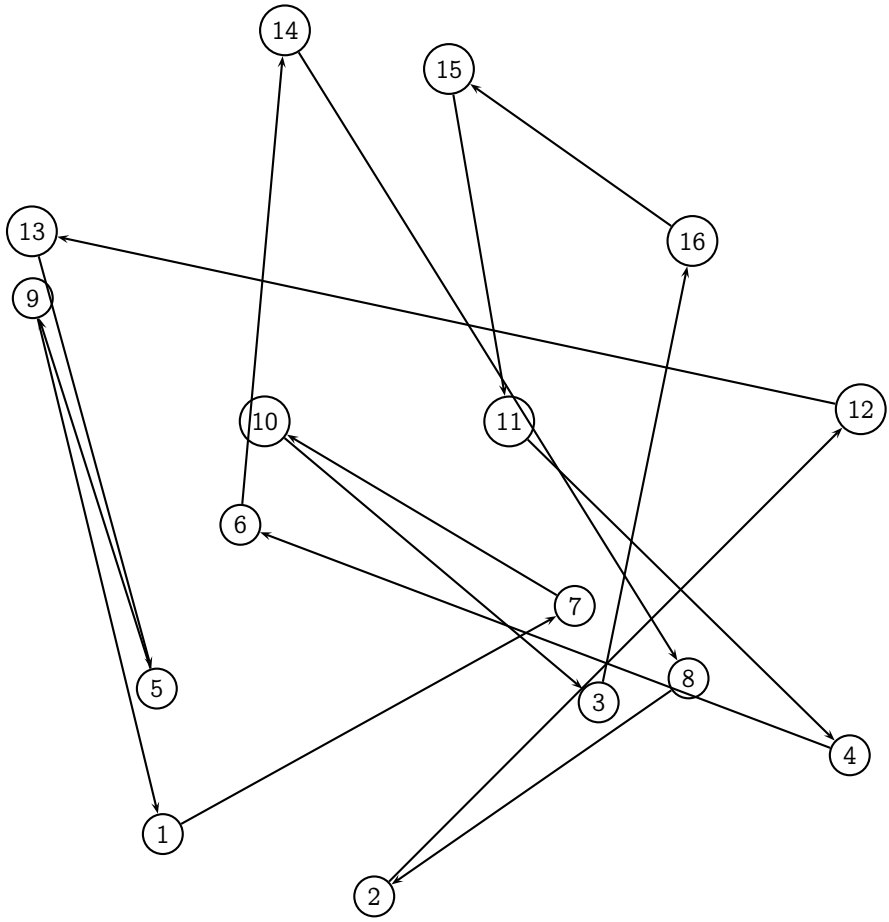


Figure 27.10: Randomly generated initial feasible solution (length: 358.6)

The neighborhood structure that we consider for the local search algorithm is called 2-OPT. It consists in selecting two customers and deciding to swap their position in the tour, inverting the sequence of visits between the two. If a and b are the two customers selected to be swapped, the 2-OPT neighbor of

$$h, i_1, \ldots i_m, a, j_1, j_2, \ldots j_n, b, k_1, \ldots, k_p$$

Table 27.7: Iterations of Algorithm 27.3 on Example 27.3

Tour																Length	2-OPT	
1	7	10	3	16	15	11	4	6	14	8	2	12	13	5	9	358.63		
1	7	10	3	16	15	11	4	6	14	8	2	12	13	9	5	322.80	5	9
1	7	10	3	16	15	11	4	12	2	8	14	6	13	9	5	287.84	6	12
1	7	10	3	8	2	12	4	11	15	16	14	6	13	9	5	257.76	16	8
1	2	8	3	10	7	12	4	11	15	16	14	6	13	9	5	231.67	7	2
1	2	8	3	4	12	7	10	11	15	16	14	6	13	9	5	213.51	10	4
1	2	8	3	4	12	7	10	11	16	15	14	6	13	9	5	196.29	15	16
1	2	8	3	4	12	7	10	11	16	15	14	9	13	6	5	180.68	6	9
1	2	3	8	4	12	7	10	11	16	15	14	9	13	6	5	173.45	8	3
1	2	3	8	4	12	7	10	11	16	15	14	13	9	6	5	169.16	9	13
1	2	3	8	4	12	7	10	9	13	14	15	16	11	6	5	169.11	11	9
1	2	3	8	4	12	7	11	16	15	14	13	9	10	6	5	153.82	10	11

is

$$h, i_1, \ldots i_m, b, j_n, \ldots j_2, j_1, a, k_1, \ldots, k_p.$$

In our example, the 2-OPT neighbor based on cities 13 and 16 of

$$1, 7, 10, 3, \mathbf{16}, 15, 11, 4, 6, 14, 8, 2, 12, \mathbf{13}, 5, 9$$

is

$$1, 7, 10, 3, \mathbf{13}, 12, 2, 8, 14, 6, 4, 11, 15, \mathbf{16}, 5, 9.$$

The neighborhood therefore consists of a set of tours generated using this procedure for any pair of cities. We apply now Algorithm 27.3 using this neighborhood structure. The iterations are reported in Table 27.7. Each row corresponds to an iteration. For each iteration, the current tour and its length are reported, as well as the two cities involved in the 2-OPT neighbor.

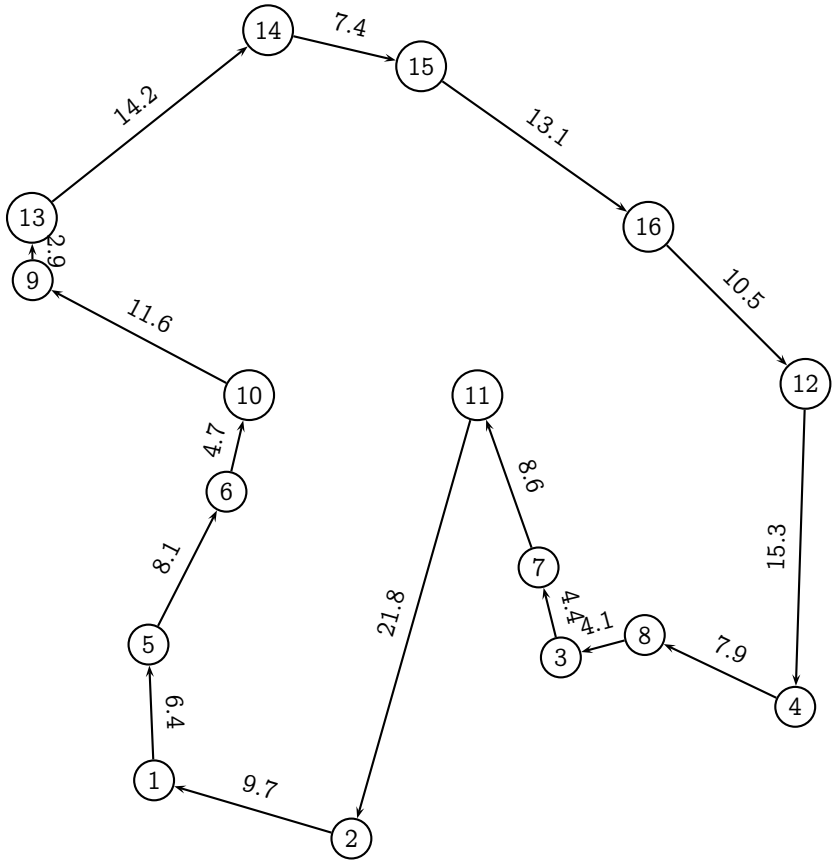


Figure 27.11: Feasible solution provided by the local search algorithm, started from the feasible solution provided by the greedy algorithm for Example 27.3 (length: 150.7)

This feasible solution is a little bit better than the feasible solution provided by the greedy algorithm presented in Section 27.1.2 (which is 158.5, see Figure 27.2), but it involves a significantly higher computational effort. Therefore, it may make sense to initiate the local search with the feasible solution provided by the greedy algorithm as a starting point, instead of a randomly generated initial tour. For our example, it performs only one iteration, by applying the 2-OPT operator on cities 9 and 11, to obtain the tour represented in Figure 27.11, with length 150.7.

As local search methods get trapped in local minima, it is good practice to apply them from different starting points, in order to increase the chance of finding different local minima. The methods presented in the next sections are also designed to escape from local minima.

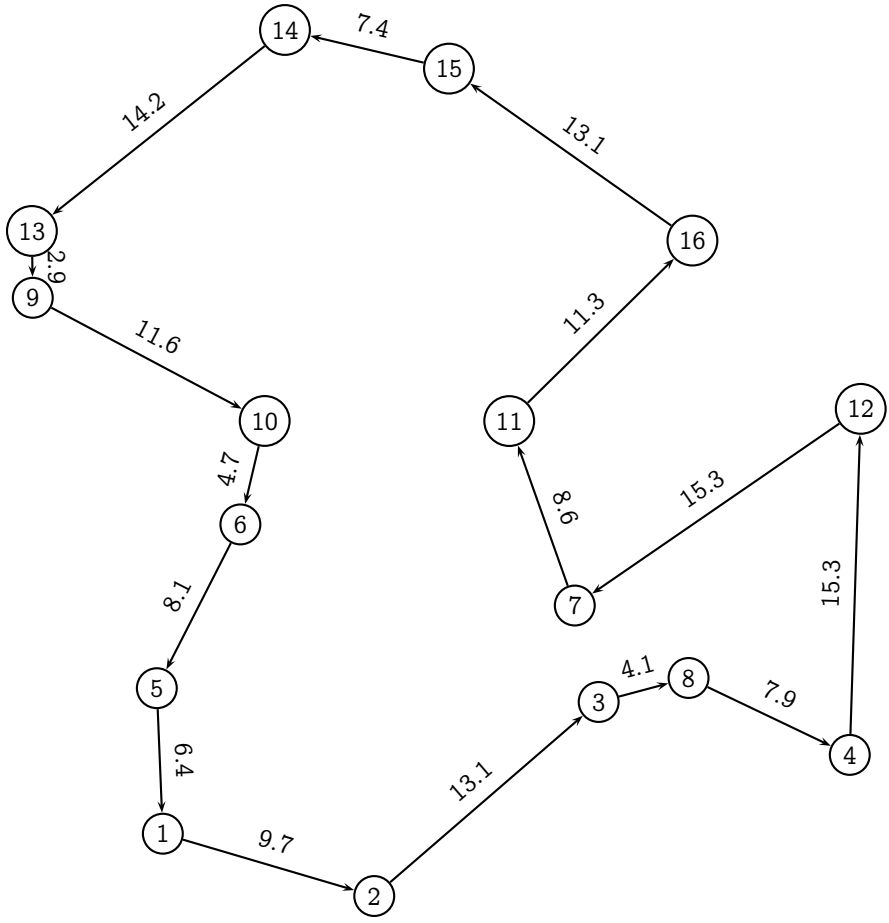


Figure 27.12: Feasible solution provided by Algorithm 27.3 on Example 27.3 (length: 153.8)

The feasible solution provided by the local search algorithm (Algorithm 27.3) is

$$1, 2, 3, 8, 4, 12, 7, 11, 16, 15, 14, 13, 9, 10, 6, 5.$$

Its length is 153.8, and it is illustrated in Figure 27.12.

27.3 Variable neighborhood search

Introduced by Mladenović and Hansen (1997), the heuristic called "Variable Neighborhood Search" (VNS) is a simple but powerful extension of the local search that is designed to escape from local optima. The idea is to consider a collection of different neighborhood structures. Once a local minimum has been reached for a neighborhood structure, use it as a starting point for another local search using another neighborhood structure. If a better feasible solution is found, restart the process from the first neighborhood. Otherwise, try the next structure until all neighborhood structures have been considered. The procedure is described as Algorithm 27.6.

Algorithm 27.6: Variable Neighborhood Search

Objective
  Find a good feasible solution of the optimization problem $\min_x f(x)$ subject to $x \in \mathcal{F}$.
Input
  The objective function $f : \mathbb{R}^n \to \mathbb{R}$.
  The feasible set $\mathcal{F}$.
  An initial feasible solution $x_0 \in \mathcal{F}$.
  K neighborhood structures $N_1, N_2, \ldots, N_K$.
Output
  A feasible solution x^*.
Initialization
  $x^* := x_0$.
  $k := 1$.
Repeat
  Apply a local search from x^* using neighborhood N_k:

$$x^+ := LS(f, \mathcal{F}, x^*, N_k)$$

  if $f(x^+) < f(x^*)$ **then**
    $x^* := x^+$
    $k := 1$
  else
    $k := k + 1$
Until $k = K$.

27.3.1 The knapsack problem

We illustrate the method on the knapsack problem in Example 27.2. In order to apply the VNS method (Algorithm 27.6), we need to define a series of neighborhood structures. We use Algorithm 27.5 to generate a neighborhood of size k.

The iterations of the VNS algorithm on Example 27.2 using the neighborhood structures defined by Algorithm 27.5 (with $M = 1{,}000$) are illustrated in Figure 27.13.

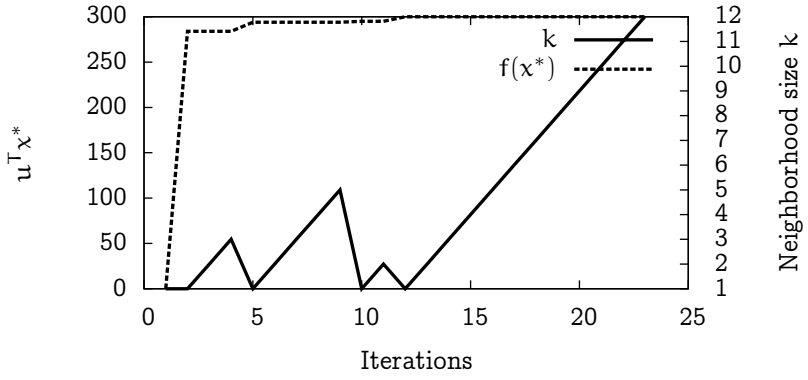


Figure 27.13: Iterations of run 1 of Algorithm 27.6 on Example 27.2 using the neighborhood structures defined by Algorithm 27.5.

The solid line shows how the size k of the neighborhood changes from iteration to iteration. Note that it is reset to $k = 1$ each time a better feasible solution is found. The dashed line provides the value of the objective function for the best feasible solution found so far at each iteration. The first local search allows a value of 284 to be reached. The mechanism of the VNS allows us to escape from this local minimum, and reach an objective value of 300, obtained by the selection of items 1, 2, 3, 4, 5, 8, and 10. There is no guarantee that it is the optimal solution. As it is a randomized algorithm, it may be appropriate to run the same algorithm several times. In this case, the algorithm was run 10 times. The final feasible solution was always the same. The iterations of runs 2 and 3 of this experiment are reported in Figures 27.14 and 27.15.

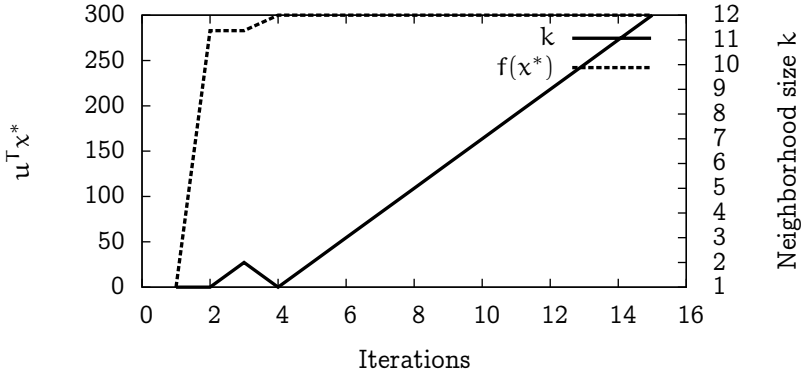


Figure 27.14: Iterations of run 2 of Algorithm 27.6 on Example 27.2 using the neighborhood structures defined by Algorithm 27.5

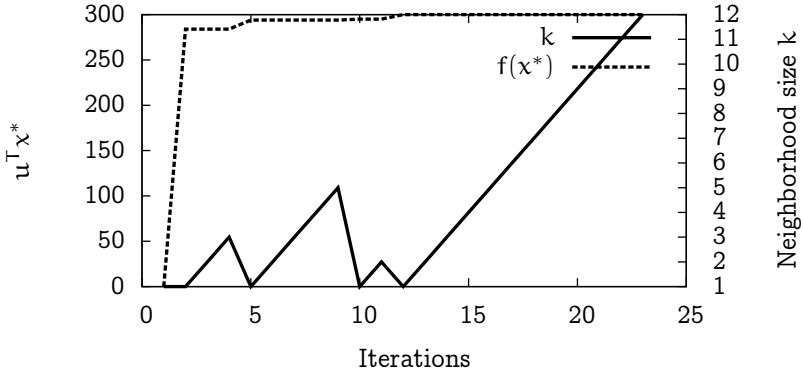


Figure 27.15: Iterations of run 3 of Algorithm 27.6 on Example 27.2 using the neighborhood structures defined by Algorithm 27.5

27.3.2 The traveling salesman problem

In order to illustrate the VNS method on the traveling salesman problem, we define a family of neighborhoods. We propose a neighborhood structure based on the insertion greedy algorithm (Algorithm 27.2). Given a tour t, a member of the neighborhood of size k is obtained as follows:

- consider the subtour s consisting of the first k cities of t,
- generate a 2-OPT neighbor s^+ of s,
- generate a complete tour t^+ using the insertion greedy algorithm (Algorithm 27.2) starting from s^+.

Figure 27.16 illustrates a tour t of length 156.2 obtained using the insertion greedy algorithm (Algorithm 27.2) starting with subtour s=1–2–3–7–8–4–12–16–11–1.

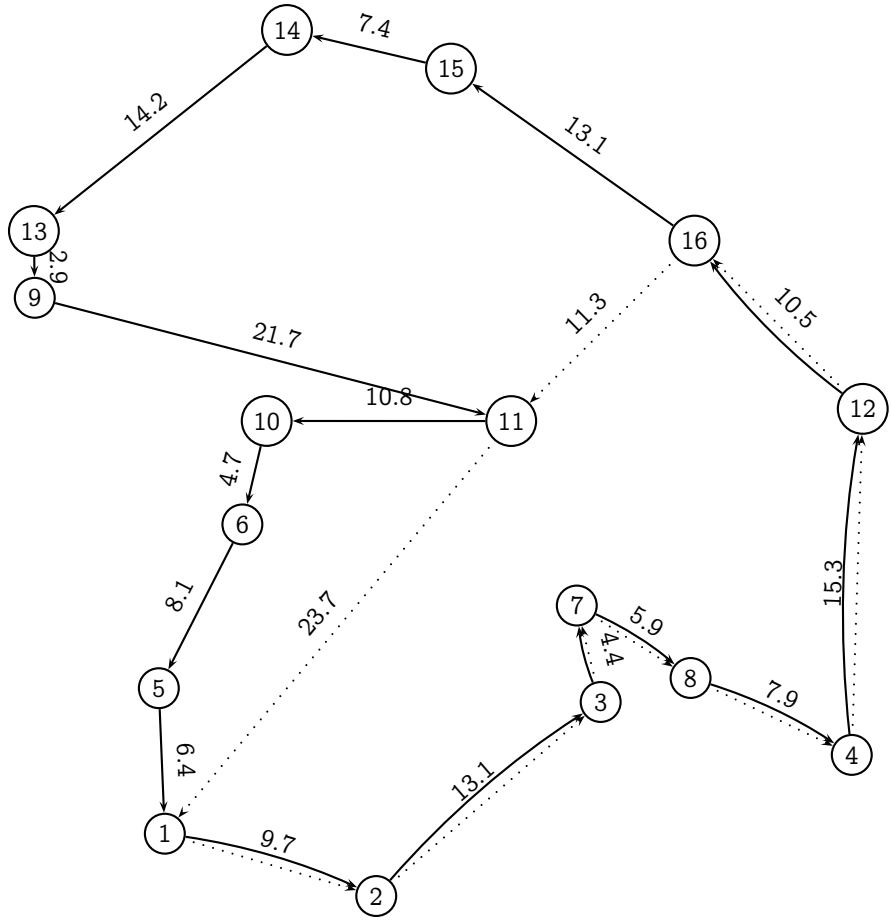


Figure 27.16: Tour (length: 156.2) obtained using the insertion greedy algorithm (Algorithm 27.2) starting with subtour 1–2–3–7–8–4–12–16–11–1

To obtain a neighbor of t, we apply a 2-OPT to s. For instance, we swap cities 16 and 11 to obtain the subtour s^+=1–2–3–7–8–4–12–11–16–1. Applying the insertion greedy algorithm (Algorithm 27.2) to s^+, we obtain a tour t^+ of length 151.6, illustrated in Figure 27.17. Note that t^+ is shorter than t, although s^+ is longer than s (118.1 instead of 101.8). Such a neighborhood is defined for $2 \leq k \leq n$.

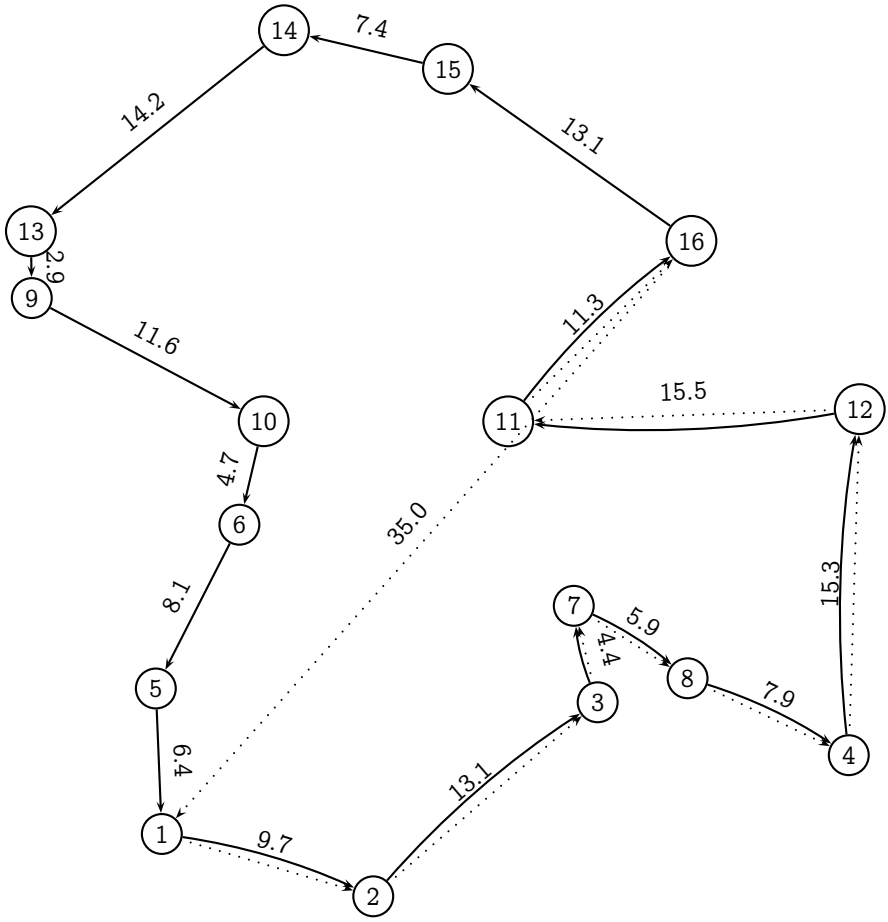


Figure 27.17: Tour (length: 151.6) obtained using the insertion greedy algorithm (Algorithm 27.2) starting with subtour 1–2–3–7–8–4–12–11–16–1

This neighborhood structure can be used in Algorithm 27.6 (where the neighborhood of size 1 is undefined and therefore skipped by the algorithm). The iterations in the case of Example 27.3 are illustrated in Figure 27.18. The algorithm succeeds in improving the objective function with a neighborhood of size 2 in the first iteration. The next improvement is obtained with a neighborhood of size 6, and the last one with a neighborhood of size 9.

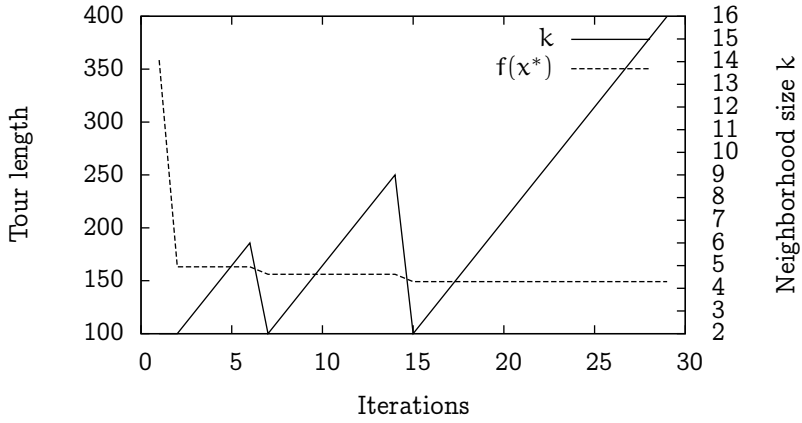


Figure 27.18: Iterations of Algorithm 27.6 on Example 27.3 using the neighborhood structures introduced in Section 27.3.2 (best feasible solution: 149.2)

The above heuristic is certainly not efficient and has been designed to illustrate the concepts introduced in this book. Many other neighborhood structures are possible. In particular, a natural structure for the VNS method applied to the traveling salesman problem would be a generalization of the 2-OPT neighborhood, such as the k-OPT neighborhood, where k is the number of cities that are re-organized in the tour (see Helsgaun, 2009). In their paper, Mladenović and Hansen (1997) illustrate the efficiency of the VNS heuristic on the traveling salesman problem. They combine the VNS idea with the GENIUS heuristic proposed by Gendreau et al. (1992).

27.4 Simulated annealing

In metallurgy, annealing is a technique consisting in heating a metal and then in cooling it down slowly in order to improve its properties. By analogy, *simulated annealing* is an extension of local search that may accept iterates with a higher value of the objective function. The "temperature" is a parameter that controls the probability to accept such iterates. The higher the temperature, the higher the probability of accepting them.

More specifically, consider the current iterate x_k and $y \in N(x_k)$ is a neighbor of x_k. If $f(y) \leq f(x_k)$, than y is immediately accepted as the next iterate. If $f(y) > f(x_k)$, the algorithm accepts y as the next iterate with probability

$$\exp\left(-\frac{f(y) - f(x_k)}{T}\right), \tag{27.3}$$

where $T > 0$ is a parameter. Note that the probability is close to 1 if $f(y)$ is close to $f(x_k)$ and decreases if the difference between the two values increases. Intuitively, the algorithm accepts from time to time to proceed uphill to escape from local optima, still

being discouraged by steep paths. The parameter T is the "temperature" parameter mentioned above. This mechanism is illustrated in Figure 27.19, where the acceptance probability of various neighbors as a function of the temperature T is represented, assuming that the value of the objective function at the current iterate is $f(x_k) = 3$.

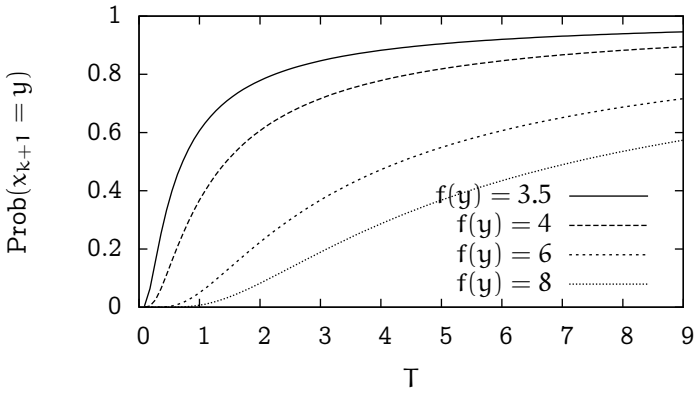


Figure 27.19: Simulated annealing: probability for a neighbor y of x_k to be accepted, with $f(x_k) = 3$

Note that this extension may be applied to any version of the local search algorithm. Algorithm 27.7 proposes a version of the simulated annealing method where the candidate neighbor is selected randomly in the neighborhood.

The performance of the algorithm varies with the value of its parameters K, that is, the number of trials for each level of temperature, and T_0, T_f and the sequence $(T_\ell)_\ell$, controlling the temperature reduction.

With respect to the temperature reduction mechanism, it is easier to interpret it in terms of probability of acceptance. Let δ_t be a typical increase of the objective function in the neighborhood structure N. In the beginning of the algorithm, we would like such an increase to be accepted with high probability $p_0 = 0.999$, say. At the end, it should be accepted with low probability $p_f = 0.00001$, say. If we allow the algorithm to modify the temperature M times, then for $\ell = 0, \ldots, M$ we have

$$T_\ell = -\frac{\delta_t}{\ln(p_0 + \frac{p_f - p_0}{M}\ell)}. \tag{27.4}$$

As $p_f - p_0 < 0$, an increase of m corresponds to a strict decrease of the denominator and, consequently, a strict decrease of the temperature. Moreover, as $0 \leq \ell \leq M$, the logarithm at the denominator is always negative, and the temperature always positive, as requested by the algorithm. Here, T_f is defined as $-\delta_t / \ln p_f$. In this way, the decrease of the acceptance probability is linear in ℓ for a given level δ_t, as illustrated in Figure 27.20.

Algorithm 27.7: Simulated annealing

Objective
  Find a good feasible solution of the optimization problem $\min_x f(x)$ subject to $x \in \mathcal{F}$.
Input
  The objective function $f : \mathbb{R}^n \to \mathbb{R}$.
  The feasible set $\mathcal{F}$.
  An initial feasible solution $x_0 \in \mathcal{F}$.
  A neighborhood structure N.
  Initial temperature $T_0 > 0$, final temperature $0 < T_f < T_0$.
  A sequence of temperatures T_ℓ, such that $0 < T_{\ell+1} < T_\ell$, for each ℓ and there exists L such that $T_\ell \leq T_f$, for each $\ell \geq L$.
  Number of iterations per level of temperature K.
Output
  A feasible solution x^*.
Initialization
  $x_c := x_0$, $x^* := x_0$, $\ell = 0$.
Repeat
  for $k := 1$ **to** K **do**
    Select randomly $y \in N(x_c) \cap \mathcal{F}$
    $\delta := f(y) - f(x_c)$
    if $\delta < 0$ **then**
      $x_c := y$
    else
      Select randomly $r \in \mathbb{R}$ between 0 and 1
      if $r < \exp(-\delta/T_\ell)$ **then**
        $x_c := y$
    if $f(x_c) < f(x^*)$ **then**
      $x^* := x_c$
  Reduce the temperature: $\ell := \ell + 1$
Until $T_\ell \leq T_f$.

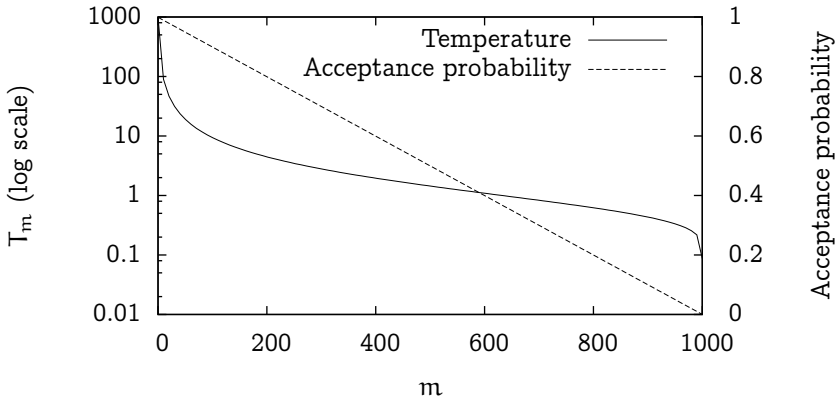


Figure 27.20: Illustration of the temperature reduction function (27.4) with $\delta_t = 1$, $M = 1000$, $p_0 = 0.999$ and $p_f = 0.00001$

27.4.1 The knapsack problem

We illustrate the simulated annealing algorithm on the knapsack problem of Example 27.2. We consider the neighborhood structure generated by Algorithm 27.5 with $k = 1$, which consists in randomly selecting one item, changing its status from chosen to unchosen, or the other way around. Algorithm 27.7 is applied with $K = 50$, and the temperature defined by (27.4), where $\delta_t = 20$, $M = 100$, $p_0 = 0.999$, and $p_f = 0.00001$. Note that the algorithm is written for a minimization problem, while the knapsack problem is a maximization problem. The number of iterations is $MK = 5{,}000$. The feasible solution provided by the algorithm consists in including items 1, 2, 3, 4, 5, 8, and 10, for a total utility of 300 and a total weight of 300. The iterations of the algorithm are illustrated on Figure 27.21, where the plain line represents the value of the objective function at each iteration, the dashed line is the temperature T, and the dotted line is the best value of the objective function identified so far at each iteration. It appears clearly that the method frequently accepts states with a low utility when the value of the temperature is high, while it happens less frequently when T is low. Note also that the best value is reached at iteration 1,421.

The algorithm stays there for 3 iterations and then goes downhill to escape from that local maximum. It returns to that feasible solution later on, during iteration 3,493. It escapes again from that local maximum and does not find any better feasible solution afterwards. Eventually, the algorithm converges to a feasible solution with value 273 and stops there.

For the sake of comparison, the algorithm is run with $K = 1{,}000$ and $M = 5$. That is, the temperature is modified only 5 times, and each time a total number of 1,000 candidates are tested. The iterations are illustrated in Figure 27.22, using the same convention as before.

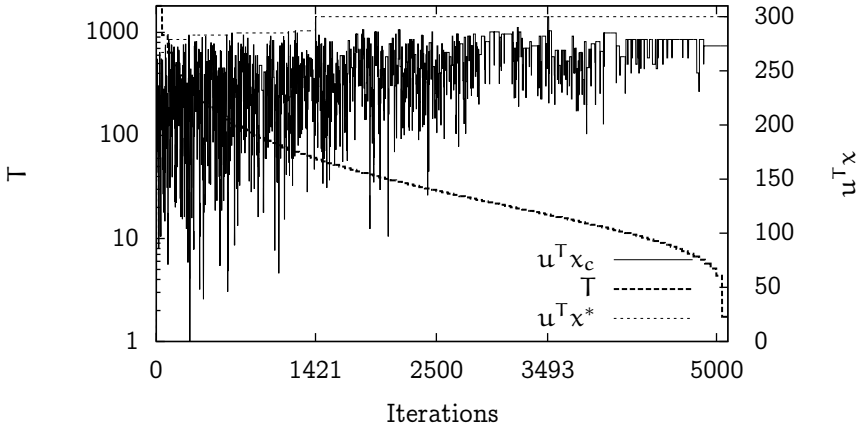


Figure 27.21: Iterations of the simulated annealing algorithm on the knapsack problem ($K = 50$, $M = 100$)

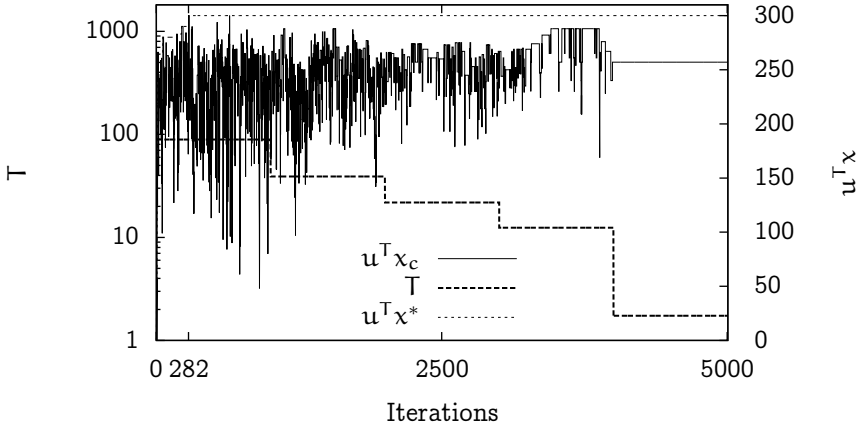


Figure 27.22: Iterations of the simulated annealing algorithm on the knapsack problem $K = 1{,}000$, $M = 5$

The same feasible solution is obtained, but is reached earlier (iteration 282), when the temperature is equal to $T = 89.22$. When the temperature has reached 1.74, the algorithm cannot escape from the local maximum anymore, at the value 257. It is in general not a good idea to drop the temperature too quickly.

Finally, the algorithm is run with $K = 5$ and $M = 1{,}000$. That is, the temperature is modified 1,000 times, and each time 5 candidates are tested. The iterations are illustrated in Figure 27.23, using the same convention as before. The same feasible solution is obtained and reached at iteration 4,206, when the temperature is equal to $T = 16.6$.

It is good practice to test different values of the parameters on small instances of a problem before running it on large instances.

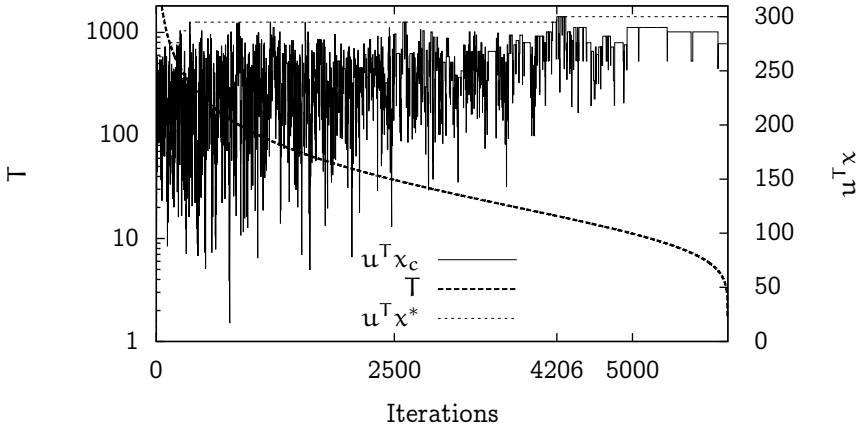


Figure 27.23: Iterations of the simulated annealing algorithm on the knapsack problem $K = 5$, $M = 1{,}000$

27.4.2 The traveling salesman problem

We illustrate the simulated annealing algorithm on the traveling salesman problem of Example 27.3. We consider the 2-OPT neighborhood structure described in Section 27.2.2. Algorithm 27.7 is applied with $K = 50$, and the temperature defined by (27.4), where $\delta_t = 5$, $M = 100$, $p_0 = 0.999$ and $p_f = 0.00001$. The number of iterations is $MK = 5{,}000$. The evolution of the value of the objective function at the current iterate and the best iterate, as well as the temperature, are represented in Figure 27.24.

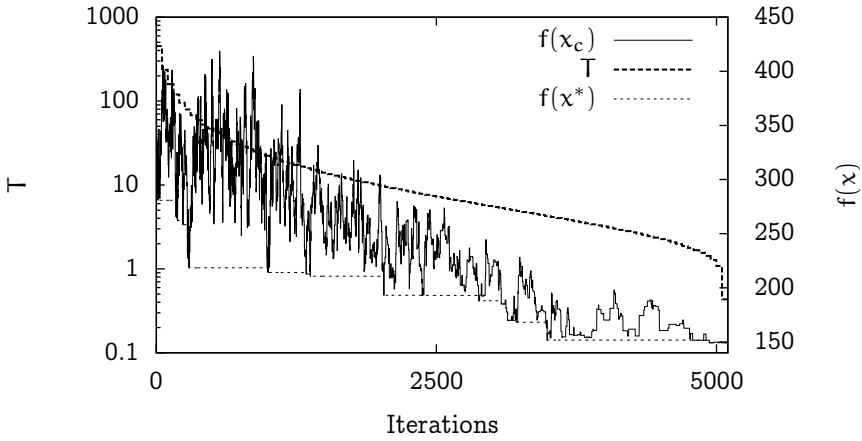


Figure 27.24: Iterations of the simulated annealing algorithm on the traveling salesman problem ($K = 50$, $M = 100$)

The feasible solution provided by the algorithm has length 149.2 and is represented in Figure 27.25.

The algorithm has also been run with K = 1,000 and M = 5 (Figure 27.26) and with K = 5 and M = 1,000 (Figure 27.27). In both cases, the feasible solution reached length 150.7.

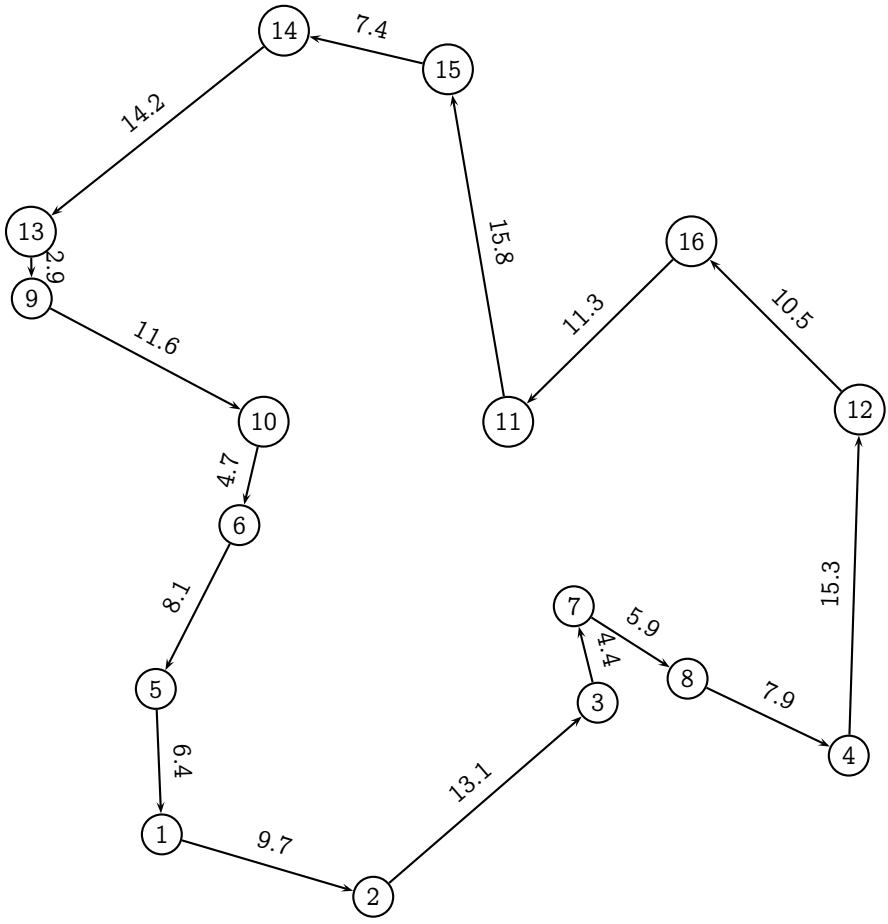


Figure 27.25: A feasible solution provided by the simulated annealing algorithm on Example 27.3 (length: 149.2)

Note that these results are based on only one execution of the algorithm. As the algorithm is randomized, its outcome varies from run to run. Applying the algorithm 100 times with K = 50, M = 100, we obtain 53 times the value 149.2, and 26 times the value 150.7. The complete results are reported in Table 27.8.

Simulated annealing can of course be applied to any type of neighborhood. For instance, Alfa et al. (1991) use a 3-OPT neighborhood in this context.

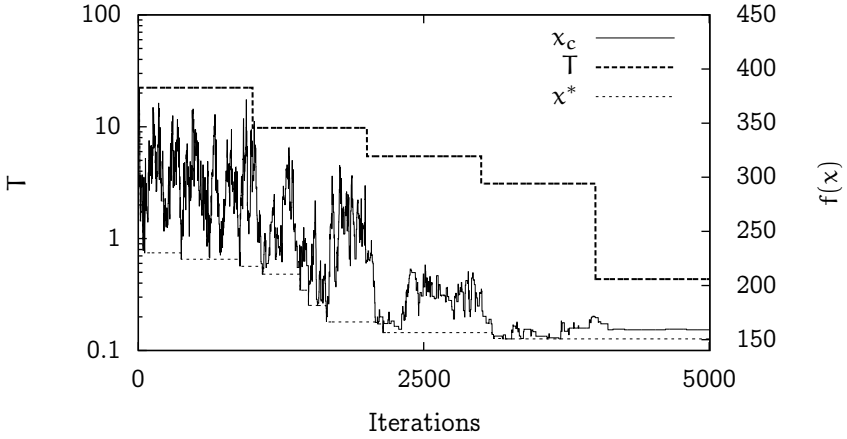


Figure 27.26: Iterations of the simulated annealing algorithm on the traveling salesman problem (K = 1,000, M = 5)

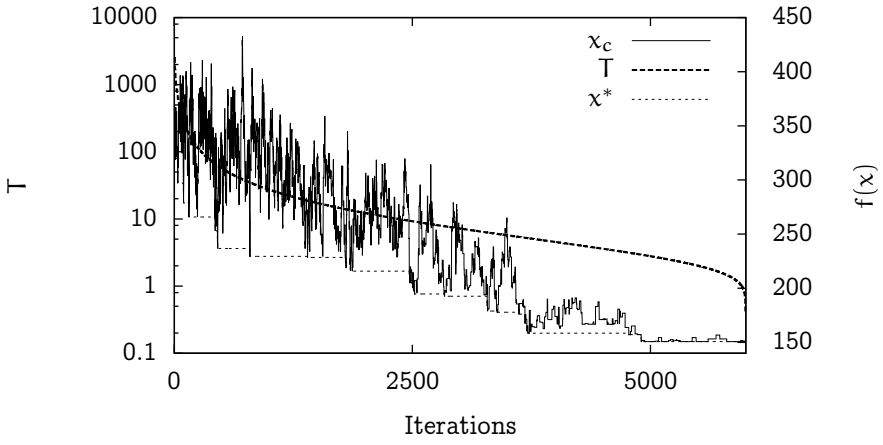


Figure 27.27: Iterations of the simulated annealing algorithm on the traveling salesman problem (K = 5, M = 1,000)

$f(x^*)$	No. of runs	$f(x^*)$	No. of runs
149.2	53	151.7	1
149.9	5	152.3	1
150.7	26	154.1	3
151.1	7	154.4	1
151.1	3		

Table 27.8: Value of the objective function for 100 runs of the simulated annealing algorithm

27.5 Conclusion

Heuristics play an important role in optimization because, for some problems, they are the only possible way of tackling them. This is the only family of methods presented in this book that is not supported by a rigorous theoretical framework. Instead, we have presented various examples of problems and algorithms to illustrate the main concepts. Creativity, as well as intense experimentation, are necessary to obtain methods that are useful to practitioners.

27.6 Project

The general organization of the projects is described in Appendix D.

Objective

The objective of the project is to analyze how different heuristics handle different optimization problems.

Approach

1. For each problem,
   - design several neighborhood structures (at least 3),
   - identify a feasible solution as starting point,
   - apply two versions of the local search algorithm (Algorithms 27.3 and 27.4),
   - apply the variable neighborhood search method with the neighborhood structures defined above (Algorithm 27.6),
   - apply the simulated annealing method with each of the neighborhood structures defined above (Algorithm 27.7).
2. Report, for each run, the evolution of the objective function with the iterations, and the best value found.
3. Compare it to the optimal solution (when available).
4. Select the algorithm that has found the worse solution, and propose some variants to improve it.

Algorithms

Algorithms 27.3, 27.4, 27.6 and 27.7.

Problems

Exercise 27.1. Solve the instance of the problem of locating plants for the supply of energy described in Example 25.1, with 10 sites and 3 cities, using the data in Table 25.1.

Exercise 27.2. Solve the knapsack problem presented in Example 27.2.

Exercise 27.3. Solve the traveling salesman problem with 16 cities presented in Example 27.3.

Exercise 27.4. Solve the task assignment problem of Exercise 25.1.

Exercise 27.5. Solve the scheduling problem of Exercise 25.2.

Exercise 27.6. Solve the graph coloring problem of Exercise 25.3. Write also the version of the problem where you must color the cantons in Switzerland.

Exercise 27.7. Solve the bin packing problem of Exercise 25.4.

Exercise 27.8. Solve the vehicle routing problem of Exercise 25.5 with $Q = 100{,}000$, $Q = 150{,}000$, and $Q = 200{,}000$.

Part VIII

Appendices

Appendix A

Notations

The book uses standard notations in linear algebra and analysis. We provide here some further details that the reader may find useful.

Positive / negative A number x is positive if $x > 0$. A number x is negative if $x < 0$. Zero is neither positive or negative. We refer to a non negative number if $x \geq 0$, and to a non positive number if $x \leq 0$.

Vectors Vectors of $\mathbb{R}^n$ are column vectors, represented with lowercase letters, such as

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

The notation x_k refers to the kth entry of vector x. When included in the core of the text, the notation $x = (x_1 \dots x_n)^T$ is used, where superscript T refers to "transposed." The inner product of $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ is denoted by

$$x^T y.$$

Matrices Matrices of $\mathbb{R}^{m \times n}$ have m rows and n columns and are represented by uppercase letters:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}.$$

The notation a_{ij} refers to the entry at row i and column j. The multiplication of two matrices $A \in \mathbb{R}^{m \times p}$ and $B \in \mathbb{R}^{p \times n}$ is denoted by $AB \in \mathbb{R}^{m \times n}$, and is such that

$$(AB)_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj},\ i = 1, \dots, m, j = 1, \dots, n.$$

As a vector $x \in \mathbb{R}^n$ is a column vector, it is considered as a matrix of dimension $n \times 1$.

The inverse of matrix A is denoted by A^{-1}, the transpose of matrix A is denoted by A^T, and the inverse of the transpose of matrix A is denoted by A^{-T}.

min and argmin Consider the function $f : \mathbb{R}^n \to \mathbb{R}$ and the set of constraints $X \subseteq \mathbb{R}^n$. The equation

$$f^* = \min_{x \in X} f(x) \tag{A.1}$$

means that

$$f^* \leq f(x), \quad \forall x \in X, \tag{A.2}$$

and $f^* \in \mathbb{R}$ represents the value of the objective function at a minimum. The equation

$$x^* \in \operatorname{argmin}_{x \in X} f(x) \tag{A.3}$$

means that

$$x^* \in \{x \in X | f(x) \leq f(y) \quad \forall y \in Y\} \tag{A.4}$$

and x^* belongs to the set of minima. In an algorithmic context, x^* usually refers to the minimum returned by the algorithm under consideration.
When there is a unique minimum, the equation can be written

$$x^* = \operatorname{argmin}_{x \in X} f(x). \tag{A.5}$$

Iterations Most algorithms presented in the book are iterative algorithms. In this context, the notation $x_k \in \mathbb{R}^n$ is used to refer to the value of the iterate at iteration k. It is a vector of dimension n. In general, there is no ambiguity about the meaning of the notation x_k as representing the kth entry of x or iterate k of the algorithm. In the former case, it is a real number, in the latter, a vector of $\mathbb{R}^n$. In the event of a possible ambiguity, the exact meaning is made explicit.

Appendix B

Definitions

Definition B.1 (Vector norms)**.** A *vector norm* on $\mathbb{R}^n$ is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ satisfying the following conditions.

1. $\|x\| \geq 0$ for all $x \in \mathbb{R}^n$.
2. $\|x\| = 0$ if and only if $x = 0$.
3. $\|x + y\| \leq \|x\| + \|y\|$, for all $x, y \in \mathbb{R}^n$.
4. $\|\alpha x\| = |\alpha| \, \|x\|$, for all $\alpha \in \mathbb{R}$, $x \in \mathbb{R}^n$.

Consider $x \in \mathbb{R}^n$ and $p \geq 1$. The p-norm of x is defined by

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \, .$$

When p tends toward infinity, we get

$$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i| \, .$$

When $p = 2$, the norm $\|\cdot\|_2$ is called the *Euclidean norm*.

Definition B.2 (Convex set)**.** A set $X \subseteq \mathbb{R}^n$ is called convex if for all $x \in X$ and for all $y \in X$, we have

$$\lambda x + (1 - \lambda) y \in X,$$

for all $0 \leq \lambda \leq 1$.

Definition B.3 (Convex combination)**.** Consider the vectors $y_1, \ldots, y_k$ of $\mathbb{R}^n$. We say that a vector x of $\mathbb{R}^n$ is a *convex combination* of $y_1, \ldots, y_k$ if there exist $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ such that

$$x = \sum_{j=1}^{k} \lambda_j y_j \, , \tag{B.1}$$

with $\lambda_j \geq 0$, $j = 1, \ldots, k$, and

$$\sum_{j=1}^{k} \lambda_j = 1 \, . \tag{B.2}$$

The set of all convex combinations of $y_1, \ldots, y_k$ is called the *convex hull* of the vectors $y_1, \ldots, y_k$.

Definition B.4 (Convex cone)**.** The set $C \subseteq \mathbb{R}^n$ is a convex cone if, for any subset $x, y \in C$, and any $\alpha_x, \alpha_y > 0$, then $\alpha_x x + \alpha_y y \in C$.

Definition B.5 (Continuous function)**.** Consider $f : X \subseteq \mathbb{R}^n \to \mathbb{R}^m$ and $x_0 \in X$. The function f is continuous at x_0 if and only if

$$\lim_{x \to x_0} f(x) = f(x_0)\,, \tag{B.3}$$

i.e., if and only if, for all $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$, there exists $\eta > 0$ such that

$$\big\|x - x_0\big\| < \eta \text{ and } x \in X \quad \Longrightarrow \quad \big\|f(x) - f(x_0)\big\| < \varepsilon\,. \tag{B.4}$$

Definition B.6 (Strictly unimodal function)**.** Consider $f : [0, T] \to \mathbb{R}$. The function f is strictly unimodal on the interval $[0, T]$ if it has a unique global minimum x^* in $[0, T]$, and if the following conditions are verified:

1. for each x_1, x_2 such that $x_1 < x_2 < x^*$, we have $f(x_1) > f(x_2) > f(x^*)$,
2. for each x_1, x_2 such that $x^* < x_1 < x_2$, we have $f(x^*) < f(x_1) < f(x_2)$,

that is, the function decreases on the left of x^* and increases on its right.

Definition B.7 (Eigenvalues and eigenvectors)**.** Consider a square matrix $A \in \mathbb{R}^{n \times n}$. The *eigenvalues* of A are the roots of its characteristic polynomial

$$p(z) = \det(zI - A)\,,$$

where I is the identity matrix of dimension n. If λ is an eigenvalue of A, the non zero vectors $x \in \mathbb{R}^n$ such that

$$Ax = \lambda x$$

are called *eigenvectors*.

Definition B.8 (Positive semidefinite matrix)**.** The square matrix $A \in \mathbb{R}^{n \times n}$ is called *positive semidefinite* when

$$x^T A x \geq 0\,, \qquad \forall x \in \mathbb{R}^n\,. \tag{B.5}$$

If, moreover, A is symmetric, then none of its eigenvalues is negative.

Definition B.9 (Positive definite matrix)**.** The square matrix $A \in \mathbb{R}^{n \times n}$ is called *positive definite* when

$$x^T A x > 0\,, \qquad \forall x \in \mathbb{R}^n\,,\ x \neq 0\,. \tag{B.6}$$

If, moreover, A is symmetric, all its eigenvalues are positive.

Definition B.10 (Orthogonal matrix)**.** The square matrix $A \in \mathbb{R}^{n \times n}$ is *orthogonal* if and only if

$$A^T A = A A^T = I. \tag{B.7}$$

Equivalently, its transpose is equal to its inverse

$$A^T = A^{-1}. \tag{B.8}$$

Definition B.11 (Minor)**.** Consider the element a_{ij} of a square matrix $A \in \mathbb{R}^{n\times n}$. The *minor* of a_{ij} is the matrix obtained by removing row i and column j from A.

Definition B.12 (Cofactor matrix)**.** Consider the element a_{ij} of a square matrix $A \in \mathbb{R}^{n\times n}$. The *cofactor* of a_{ij} is the determinant of the minor of a_{ij} multiplied by $(-1)^{i+j}$. The cofactor matrix is the matrix such that its element (i, j) is the cofactor of a_{ij}.

Definition B.13 (Determinant)**.** The *determinant* of a square matrix $A \in \mathbb{R}^{n\times n}$ is defined as

$$\det(A) = \sum_{\sigma\in P_n} \operatorname{sgn}(\sigma) \prod_{k=1}^{n} a_{i\sigma_i}, \tag{B.9}$$

where P_n is the set of all permutations of the set $\{1, \ldots, n\}$, and the sign of a permutation σ is

$$\operatorname{sgn}(\sigma) = (-1)^M \tag{B.10}$$

where M is the number of pairwise swaps that are required to obtain σ from $\{1, \ldots, n\}$. The following recursive definition is more adequate to calculate the determinant. If $A \in \mathbb{R}^{1\times 1}$, that is, if A contains only one element a_{11}, then $\det(A) = a_{11}$. If $A \in \mathbb{R}^{n\times n}$, then

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}) = \sum_{j=1}^{n} a_{1j} C_{1j}, \tag{B.11}$$

where A_{1j} is the minor of a_{1j}, and C_{1j} is the cofactor of a_{1j} (see Definition B.12).

Definition B.14 (Unimodular matrix)**.** A unimodular matrix is a square integer matrix with determinant equal to 1 or -1.

Rudolf Otto Sigismund Lipschitz was born on May 14, 1832, in Bönkein, close to Königsberg, now Kaliningrad (Russia), and died in Bonn (Germany) on October 7, 1903. Lipschitz was a student of Dirichlet in Berlin. He contributed significantly to the progress of knowledge in fields as diverse as number theory, the theory of Bessel functions and Fourier series, ordinary and partial differential equations, analytical mechanics, and the theory of harmonic functions. He is particularly known for the condition that bears his name (Definition B.15).

Figure B.1: Rudolf Otto Sigismund Lipschitz

Definition B.15 (Lipschitz condition)**.** In a metric space E, a function f satisfies the Lipschitz condition of order $\alpha > 0$, with a constant $k > 0$, if for all (x, y)

$$d\big(f(x), f(y)\big) \leq k\big(d(x, y)\big)^{\alpha},$$

where $d(x, y)$ is the distance between x and y. When $a = 1$, the function is called a Lipschitz function. If, moreover, $k < 1$, the function is called contracting. A Lipschitz function is uniformly continuous on E.

Definition B.16 (Lipschitz continuity)**.** Consider $f : X \subseteq \mathbb{R}^n \to \mathbb{R}^m$. The function f is Lipschitz continuous on X if there exists a constant $M > 0$ such that, for all $x, y \in X$, we have

$$\big\|f(x) - f(y)\big\|_m \leq M\big\|x - y\big\|_n\,, \tag{B.12}$$

where $\|\cdot\|_m$ is a norm on $\mathbb{R}^m$ and $\|\cdot\|_n$ is a norm on $\mathbb{R}^n$. If $M = 0$, then f is constant on X, i.e., there exists c such that $f(x) = c$, $\forall x \in X$.

Definition B.17 (Landau notation $o(\cdot)$)**.** Let f and g be two functions of $\mathbb{R} \to \mathbb{R}$, with $f(x) \neq 0$, $\forall x$. The Landau notation $g(x) = o\big(f(x)\big)$ signifies that

$$\lim_{x \to 0} \frac{g(x)}{f(x)} = 0\,. \tag{B.13}$$

By abuse of language, we say that $g(x)$ tends toward zero faster than $f(x)$.

Definition B.18 (Cholesky decomposition)**.** Let $A \in \mathbb{R}^{n \times n}$ be a positive definite symmetric matrix. The Cholesky decomposition of A is

$$A = LL^T\,, \tag{B.14}$$

where $L \in \mathbb{R}^{n \times n}$ is a lower triangular matrix.

Definition B.19 (Convergence of a sequence)**.** Let $\big(x_k\big)_k$ be a sequence of points of $\mathbb{R}^n$. We say that the sequence $\big(x_k\big)_k$ converges toward x if for all $\varepsilon > 0$, there exists an index $\widehat{k}$ such that

$$\|x_k - x\| \leq \varepsilon\,, \qquad \forall k \geq \widehat{k}\,. \tag{B.15}$$

We thus write

$$\lim_{k \to +\infty} x_k = x\,. \tag{B.16}$$

Definition B.20 (Limit point of a sequence)**.** Let $\big(x_k\big)_k$ be a sequence of points of $\mathbb{R}^n$. We say that x is an *accumulation point* or a *limit point* of the sequence if there exists a subsequence $\big(x_{k_i}\big)_i$ that converges toward x.

Definition B.21 (Semi-continuity)**.** Consider $X \subseteq \mathbb{R}^n$ and let $f : X \to \mathbb{R}$ be a function of real values. f is called *lower semi-continuous* in $x \in X$ if for all sequences $\big(x_k\big)_k$ of elements of X converging toward x, we have

$$f(x) \leq \liminf_{k \to \infty} f(x_k)\,. \tag{B.17}$$

Definition B.22 (Coercive function)**.** Consider $X \subseteq \mathbb{R}^n$ and let $f : X \to \mathbb{R}$ be a function of real values. f is called *coercive* if for all sequences $\big(x_k\big)_k$ of elements of X such that $\|x_k\| \to +\infty$ for any norm, we have

$$\lim_{k \to \infty} f(x_k) = +\infty\,. \tag{B.18}$$

Definition B.23 (Compact set)**.** Let S be a subset of a metric set. S is compact if for all sequences $(x_k)_k$ of elements of S, there exists a subsequence converging toward an element of S. If the metric set is of finite dimension (as $\mathbb{R}^n$), S is compact if and only if S is closed and bounded.

Definition B.24 (Equivalence relation)**.** Let X be a set and R a relation, that is, a collection of ordered pairs of elements of X. The relation R is an equivalence relation if the following properties are satisfied.

1. Reflexivity: $(x, x) \in R$, for all $x \in X$.
2. Symmetry: $(x, y) \in R \Longrightarrow (y, x) \in R$, for all $x, y \in X$.
3. Transitivity: $(x, y) \in R$ and $(y, z) \in R \Longrightarrow (x, z) \in R$, for all $x, y, z \in X$.

If R is an equivalence relation, the notation $x \equiv y$ means that $(x, y) \in R$.

Definition B.25 (Equivalence class)**.** Let X be a set and R be an equivalence relation on X. An equivalence class is a subset of the form $\{x \in X | (x, r) \in R\}$, where r is a representative element of the class. Note that any element of the class can be its representative, by symmetry of the equivalence relation.

Definition B.26 (Frobenius norm)**.** Consider $A \in \mathbb{R}^{m\times n}$. The *Frobenius norm* of A is

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}\,.$$

Definition B.27 (Induced norm)**.** Let $\|\cdot\|$ be a vector norm on $\mathbb{R}^n$. The matrix norm $\|\cdot\|_{m\times n}$ on $\mathbb{R}^{m\times n}$ defined by

$$\|A\|_{m\times n} = \max_{x\in\mathbb{R}^n,\, x\neq 0} \frac{\|Ax\|}{\|x\|} \tag{B.19}$$

is the matrix norm induced by the vector norm.

Definition B.28 (Singular values)**.** Let $A \in \mathbb{R}^{m\times n}$. There exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ such that

$$U^T A V = \operatorname{diag}(\sigma_1, \ldots, \sigma_p)\,,$$

where $p = \min(m, n)$. The σ_i are called *singular values* of A.

Definition B.29 (Rank of a matrix)**.** Let $A \in \mathbb{R}^{m\times n}$ be a matrix and

$$\operatorname{Im}(A) = \left\{y \in \mathbb{R}^m \mid \exists x \in \mathbb{R}^n \text{ t.q. } y = Ax\right\}$$

the subspace generated by the matrix A. The rank of A is the dimension of this subspace. It is equal to the number of singular values of A that are non zero.

Appendix C

Theorems

Theorem C.1 (First-order Taylor theorem). *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a continuously differentiable function on an open sphere* S *centered in* x*. Then,*

- *for all* d *such that* $x + d \in S$*, we have*

$$f(x + d) = f(x) + d^T \nabla f(x) + o\big(\|d\|\big)\,, \tag{C.1}$$

- *for all* d *such that* $x + d \in S$*, there exists* $\alpha \in [0, 1]$ *such that*

$$f(x + d) = f(x) + d^T \nabla f(x + \alpha d)\,. \tag{C.2}$$

The result (C.2) is also called the mean value theorem*.*

Theorem C.2 (Second-order Taylor theorem). *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a twice differentiable function on an open sphere* S *centered in* x*. Then,*

- *for all* d *such that* $x + d \in S$*, we have*

$$f(x + d) = f(x) + d^T \nabla f(x) + \frac{1}{2}\, d^T \nabla^2 f(x) d + o\big(\|d\|^2\big)\,, \tag{C.3}$$

- *for all* d *such that* $x + d \in S$*, there exists* $\alpha \in [0, 1]$ *such that*

$$f(x + d) = f(x) + d^T \nabla f(x) + \frac{1}{2}\, d^T \nabla^2 f(x + \alpha d) d\,. \tag{C.4}$$

Theorem C.3 (Chain rule differentiation). *Consider* $f : \mathbb{R}^m \to \mathbb{R}^n$*,* $g : \mathbb{R}^n \to \mathbb{R}^p$ *and* $h : \mathbb{R}^m \to \mathbb{R}^p$ *such that* $h(x) = g\big(f(x)\big)$*. Then,*

$$\nabla h(x) = \nabla f(x) \nabla g\big(f(x)\big)\,, \qquad \forall x \in \mathbb{R}^m\,, \tag{C.5}$$

where $\nabla f : \mathbb{R}^m \to \mathbb{R}^{m \times n}$*,* $\nabla g : \mathbb{R}^n \to \mathbb{R}^{n \times p}$ *and* $\nabla h : \mathbb{R}^m \to \mathbb{R}^{m \times p}$*. When* f *is linear, i.e., when* $f(x) = Ax$ *with* $A \in \mathbb{R}^{n \times m}$*, we have*

$$\nabla h(x) = A^T \nabla g(Ax)\,. \tag{C.6}$$

Theorem C.4 (Rayleigh-Ritz theorem). *Let $A \in \mathbb{R}^{n\times n}$ be a real symmetric matrix. Let λ_1 be the largest eigenvalue of A and λ_n the smallest. Then,*

$$\lambda_1 = \max_{x\neq 0} \frac{x^T A x}{x^T x} \tag{C.7}$$

and

$$\lambda_n = \min_{x\neq 0} \frac{x^T A x}{x^T x}\,. \tag{C.8}$$

Theorem C.5 (Symmetric Schur decomposition). *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Then there exists an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ such that*

$$Q^T A Q = \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n). \tag{C.9}$$

Also, for each column Q_k of Q,

$$AQ_k = \lambda_k Q_k, \tag{C.10}$$

so that λ_k is an eigenvalue of A, and Q_k the corresponding eigenvector.

Theorem C.6 (Implicit functions). *Let $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ be a continuous function. Consider $x^+ \in \mathbb{R}^n$ and $y^+ \in \mathbb{R}^m$ such that*

$$f(x^+, y^+) = 0 \tag{C.11}$$

and such that the gradient matrix $\nabla_y f(x, y)$ is continuous and non singular in a neighborhood of (x^+, y^+). Then, there exist neighborhoods $\mathcal{V}_{x^+}$ and $\mathcal{V}_{y^+}$ of x^+ and y^+, respectively, as well as a continuous function

$$\phi : \mathcal{V}_{x^+} \longrightarrow \mathcal{V}_{y^+} \tag{C.12}$$

such that

$$y^+ = \phi(x^+) \tag{C.13}$$

and

$$f\big(x, \phi(x)\big) = 0\,, \qquad \forall x \in V_{x^+}\,. \tag{C.14}$$

The function ϕ is unique in the sense that any $(x, y) \in V_{x^+} \times V_{y^+}$ such that $f(x, y) = 0$ also satisfies $y = \phi(x)$. If, moreover, f is differentiable, then so is ϕ and

$$\nabla\phi(x) = -\nabla_x f\big(x, \phi(x)\big)\left(\nabla_y f\big(x, \phi(x)\big)\right)^{-1}, \qquad \forall x \in V_{x^+}\,. \tag{C.15}$$

Theorem C.7 (Projection on the kernel of a matrix). *Let $A \in \mathbb{R}^{m\times n}$ be a matrix of full rank. Then, the matrix*

$$P = I - A^T\big(AA^T\big)^{-1}A \tag{C.16}$$

is the projection operator on the kernel of A, i.e., we have $APy = 0$ for all $y \in \mathbb{R}^n$.

Theorem C.8 (Convexity of polyhedra). *All polyhedra are convex sets.*

Lemma C.9 (Farkas' lemma). *Consider* $A \in \mathbb{R}^{m\times n}$ *and* $b \in \mathbb{R}^m$. *Then, exactly one of the following two statements holds:*

1. *There exists* $x \in \mathbb{R}^n$, $x \geq 0$, *such that* $Ax = b$.
2. *There exists* $p \in \mathbb{R}^m$ *such that* $A^T p \geq 0$ *and* $p^T b < 0$.

Lemma C.10 (Farkas' lemma (equivalent formulation)). *Consider the linear system of inequalities* $Ax \leq b$, *where* $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ *and* $b \in \mathbb{R}^m$. *The system has a solution if and only if* $\lambda^T b \geq 0$ *for all* $\lambda \in \mathbb{R}^m$ *such that* $\lambda \geq 0$ *and* $\lambda^T A = 0$.

Theorem C.11 (Newton's theorem). *Let* $f : \mathbb{R}^n \to \mathbb{R}^m$ *be a continuously differentiable function on an open convex* $X \subset \mathbb{R}^n$. *For all* $\widehat{x}$, $x^+ \in X$,

$$f(x^+) - f(\widehat{x}) = \int_0^1 \nabla f\big(\widehat{x} + t(x^+ - \widehat{x})\big)^T (x^+ - \widehat{x})\, dt = \int_{\widehat{x}}^{x^+} \nabla f(z)\, dz\,. \tag{C.17}$$

Theorem C.12 (Bound on an integral). *Let* $f : X \subset \mathbb{R}^n \to \mathbb{R}^{m\times n}$, *where* X *is an open convex set, and consider* x *and* $x + d$ *in* X. *Then, if* f *is integrable on* $[x, x + d]$,

$$\left\| \int_0^1 f(x + td)d\, dt \right\| \leq \int_0^1 \|f(x + td)d\|\, dt \leq \int_0^1 \|f(x + td)\|\, \|d\|\, dt\,, \tag{C.18}$$

where $\|\cdot\|$ *is a norm on* $\mathbb{R}^{m\times n}$.

Theorem C.13 (Cauchy-Schwarz inequalities). *Consider* x *and* $y \in \mathbb{R}^n$. *Then,*

$$|x^T y| \leq \|x\|_2\, \|y\|_2, \tag{C.19}$$

$$|x^T y| \leq \|x\|_1\, \|y\|_\infty\,. \tag{C.20}$$

Theorem C.14 (Matrix norms). *The matrix norms* $\|\cdot\|_2$ *(Definition B.27) and* $\|\cdot\|_F$ *(Definition B.26) satisfy the following properties:*

1. *Consider* A *and* $B \in \mathbb{R}^{n\times n}$. *Then*

$$\|AB\|_F \leq \|A\|_F\, \|B\|_F\,. \tag{C.21}$$

2. *Consider* A *and* $B \in \mathbb{R}^{n\times n}$. *Then*

$$\|AB\|_2 \leq \|A\|_2 \|B\|_2\,. \tag{C.22}$$

3. *Consider* A *and* $B \in \mathbb{R}^{n\times n}$. *Then*

$$\|AB\|_F \leq \min\big(\|A\|_2\, \|B\|_F\,,\, \|A\|_F\, \|B\|_2\big)\,. \tag{C.23}$$

4. *Consider* $A \in \mathbb{R}^{n\times n}$ *and* $x \in \mathbb{R}^n$. *Then*

$$\|Ax\|_2 \leq \|A\|_F\, \|x\|_2\,. \tag{C.24}$$

5. Consider x *and* $y \in \mathbb{R}^n$*. Then*

$$\left\|xy^T\right\|_F = \left\|xy^T\right\|_2 = \|x\|_2 \|y\|_2 . \tag{C.25}$$

Theorem C.15 (Cramer's rule)**.** *Consider an invertible square matrix* $A \in \mathbb{R}^{n\times n}$*. Then,*

$$A^{-1} = \frac{1}{\det(A)} C(A)^T, \tag{C.26}$$

where $C(A)^T$ *is the cofactor matrix of* A *(see Definition B.12).*

Theorem C.16 (Inverse of a perturbed matrix)**.** *Let* $\|\cdot\|$ *be a norm on* $\mathbb{R}^{n\times n}$ *satisfying the conditions*

$$\begin{aligned} \|AB\| &\leq \|A\| \|B\| \\ \|I\| &= 1 . \end{aligned} \tag{C.27}$$

Let A *be a non singular matrix and let us take* B *such that*

$$\left\|A^{-1}(B-A)\right\| < 1 . \tag{C.28}$$

Then, B *is non singular and*

$$\|B^{-1}\| \leq \frac{\|A^{-1}\|}{1 - \left\|A^{-1}(B-A)\right\|} . \tag{C.29}$$

Theorem C.17 (Sherman-Morrison-Woodbury formula)**.** *Let* $A \in \mathbb{R}^{n\times n}$ *be a square non singular matrix, and let us take* U *and* $V \in \mathbb{R}^{n\times p}$*, with* $1 \leq p \leq n$*. Then, the matrix*

$$B = A + UV^T \tag{C.30}$$

is invertible and

$$B^{-1} = A^{-1} - A^{-1}U\left(I + V^T A^{-1} U\right)^{-1} V^T A^{-1} . \tag{C.31}$$

Theorem C.18 (Positive definite matrix)**.** *Let* A *and* B *be two symmetric matrices such that* B *is positive semidefinite and* A *is positive definite in the kernel of* B*, i.e., that* $x^T Ax > 0$ *for all non zero* x *such that* $x^T Bx = 0$*. Then, there exists* $\bar{c} \in \mathbb{R}$ *such that* $A + cB$ *is positive definite for all* $c > \bar{c}$*.*

Appendix D

Projects

D.1 General instructions

The aim of the projects is to implement the different algorithms described in the book and test them on various problems. It is advisable to use a mathematical programming language. The Octave language (Eaton, 1997) has been used for all examples presented in this book. If a language such as C, C++, or Fortran is preferred, it is useful to have a library to manage the linear algebra, such as LAPACK (Anderson et al., 1999).

Here is some general advice for preparing these projects.

- It is important to define the interface conventions (format of the data, transfer of vectors and matrices, etc.) between algorithms and problems, and to follow them strictly during the projects. The implementation of these interfaces depends on the programming language.
- Each description of an algorithm explicitly identifies the input and the output. It is wise to be guided by this information for the implementation.
- Certain problems to solve and certain algorithms appear in several projects. It is therefore recommended to isolate the different modules of the programs in order to be able to reuse them.
- It is recommended to first perform a test on the examples described in the book to debug your programs. The details of the iterations are given for each example.
- It is often instructive to vary the parameters of the different algorithms in order to understand their role. During the implementation, it is inadvisable to define the value of these parameters in the code. It is better to read these values in a file, which can easily be modified during testing. The extra time devoted to the programming is largely outweighed by the time saved by these tests.
- Whether it is to identify a programming error or to analyze the behavior of the algorithm, it is important to keep note in a file of the information related to each iteration of the algorithms (current iterate, value of the objective function, gradient norm, constraint norm, etc.) It is advisable to keep the presentation of

this file neat so that it can be easily read or easily imported into another software (spreadsheet, database, visualization software, etc.)

- The use of a visualization software for the functions and level curves is recommended. The freeware Gnuplot (`www.gnuplot.info`) has been used in this book.
- One must be attentive when it comes to numeric problems. Computers operate with what is called *finite* arithmetic, in the sense that only a finite set of real numbers can be represented. One of the consequences is that the result of the operation $1+\varepsilon$ can be 1, even if $\varepsilon > 0$. The smallest value of ε such that $1+\varepsilon \neq 1$ is called the ε-machine and depends on the representation of real numbers in the employed processor. Typically, for a representation in double precision, the ε-machine is on the order of 10^{-16} (see Algorithm 7.1 and the discussions about it).
- Error handling is important. For example, if the algorithm tries to invert a singular matrix, it must be properly detected and an adequate error message should be displayed.

D.2 Performance analysis

It is instructive to compare the performances of various algorithms on several different problems. However, the analysis and the synthesis of the results can be tedious. Here we present an analysis method proposed by Dolan and Moré (2002), which enables a large amount of results to be synthesized.

The first thing to do is to choose which performance measure to consider. In general, this includes the calculation time, the number of iterations, or the number of evaluations of the objective function. It is essential to choose a measure that is comparable from one algorithm to another and from one problem to another. For instance, if two algorithms converge toward different solutions, comparing the number of iterations makes little sense. Moreover, if one algorithm utilizes derivatives and the other doesn't, comparing the number of function evaluations is not representative of the performance of the algorithm.

Let $\tau_{p,a}$ be the performance of the algorithm a for solving the problem p. Without loss of generality, we take as a convention here that $\tau_{p,a} < \tau_{p,b}$ signifies that algorithm a is better than algorithm b for solving problem p. Define $\tau_{p,a} = +\infty$ if algorithm a is not able to solve problem p. For each problem, we can identify the best performance, i.e.,

$$T_p = \min_a \tau_{p,a} .$$

If $T_p = +\infty$, no algorithm can solve this problem. We then normalize the performance indices by defining

$$\rho_{p,a} = \begin{cases} \dfrac{\tau_{p,a}}{T_p} & \text{if } \tau_{p,a} \neq +\infty \\ R & \text{otherwise,} \end{cases}$$

where R is sufficiently large, in the sense that $R > \rho_{p,a}$ for any p and a such that $\tau_{p,a} \neq +\infty$. The quantity $\rho_{p,a}$ represents the performance of algorithm a on problem

p, compared to the best algorithm among those tested. For each algorithm, we consider the performance function, defined by

$$P_a : [1, \infty[\rightarrow [0, 1] : \pi \rightsquigarrow Pr(\tau_{p,a} \leq \pi),$$

where $Pr(\tau_{p,a} \leq \pi)$ is the proportion of problems for which $\rho_{p,a} \leq \pi$. If $\pi = 1$, it is the proportion of problems for which algorithm a is the best, which enables us to measure the pure performance. If $\pi \geq R$, it is the proportion of problems that have been solved by algorithm a, independently of the performance, which enables us to measure the robustness of the method. The intermediary values of π enable us to analyze the compromise between efficiency and robustness. The faster this function increases, the better the algorithm.

We illustrate these concepts with an example where two algorithms are tested on 10 problems. The performances $(\tau_{p,a})$ are listed in Table D.1.

Table D.1: Example of performances for two algorithms: $\tau_{p,a}$

	Problems									
Algorithms	1	2	3	4	5	6	7	8	9	10
Algo A	20	10	∞	10	∞	20	10	15	25	∞
Algo B	10	30	70	60	70	80	60	75	∞	∞
T_p	10	10	70	10	70	20	10	15	25	∞

After normalization, the relative performances $(\rho_{p,a})$ are given in Table D.2. Note that $R = 10$ in this case. We could have chosen any value such that $R > 6$. Finally, the function P_a for each algorithm is shown in Figure D.1. In this example, algorithm A turns out to be more efficient than algorithm B, but the latter is slightly more robust.

Table D.2: Example of performances for two algorithms: $\rho_{p,a}$

	Problems									
Algorithms	1	2	3	4	5	6	7	8	9	10
Algo A	2	1	10	1	10	1	1	1	1	10
Algo B	1	3	1	6	1	4	6	5	10	10

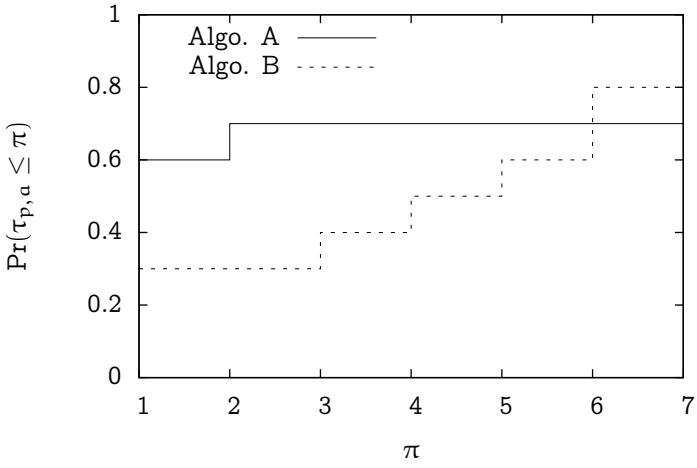


Figure D.1: Example of a performance profile

Bibliography

Abadie, J. (1967). On the Kuhn-Tucker Theorem, *in* J. Abadie (ed.), *Nonlinear Programming*.

Abraham, I., Delling, D., Goldberg, A. V. and Werneck, R. F. (2011). A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks, *Experimental Algorithms*, Springer, pp. 230–241.

Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993). *Network Flows. Theory, Algorithms and Applications*, Prentice-Hall Inc.

Alfa, A. S., Heragu, S. S. and Chen, M. (1991). A 3-OPT Based Simulated Annealing Algorithm for Vehicle Routing Problems, *Computers & Industrial Engineering* **21**(1–4): 635–639.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D. (1999). *LAPACK Users' Guide*, 3rd edn, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Armijo, L. (1966). Minimization of Functions Having Continuous Partial Derivatives, *Pacific J. Math.* **16**: 1–3.

Avis, D. and Fukuda, K. (1992). A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra, *Discrete & Computational Geometry* **8**(1): 295–313.

Axelsson, O. (1994). *Iterative Solution Methods*, Cambridge University Press, Cambridge, UK.

Axhausen, K., Hess, S., Koenig, A., Abay, G., Bates, J. and Bierlaire, M. (2008). Income and Distance Elasticities of Values of Travel Time Savings: New Swiss Results, *Transport Policy* **15**(3): 173–185.

Bartels, R. H. and Golub, G. H. (1969). The Simplex Method of Linear Programming Using LU Decomposition, *Commun. ACM* **12**(5): 266–268.

Beck, A. (2014). *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB*, MPS-SIAM Series on Optimization, SIAM, Philadelphia, PA.

Bellman, R. E. (1957). *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Bellman, R. E. (2010). *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Ben-Tal, A., El Ghaoui, L. and Nemirovski, A. (2009). *Robust Optimization*, Princeton Series in Applied Mathematics, Princeton University Press, Princeton, NJ.

Ben-Tal, A. and Nemirovski, A. (2001). *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, MPS Series on Optimization, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Bertsekas, D. P. (1976). On the Goldstein-Levitin-Polyak Gradient Projection Method, *IEEE Transactions on Automatic Control* **21**: 174–184.

Bertsekas, D. P. (1982). *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, London.

Bertsekas, D. P. (1998). *Network Optimization – Continuous and Discrete Models*, Athena Scientific, Belmont, MA.

Bertsekas, D. P. (1999). *Nonlinear Programming*, 2nd edn, Athena Scientific, Belmont, MA.

Bertsimas, D. and Tsitsiklis, J. N. (1997). *Introduction to Linear Optimization*, Athena Scientific, Belmont, MA.

Bertsimas, D. and Weismantel, R. (2005). *Optimization over Integers*, Athena Scientific, Belmont, MA.

Bierlaire, M. (2006). *Introduction à l'optimisation différentiable*, Presses polytechniques et universitaires romandes, Lausanne, Switzerland. In french.

Birge, J. R. and Louveaux, F. (1997). *Introduction to Stochastic Programming*, Springer.

Bland, R. G. (1977). New Finite Pivoting Rules for the Simplex Method, *Mathematics of Operations Research* **2**(2): 103–107.

Bland, R. G. and Orlin, J. B. (2005). IFORS' Operational Research Hall of Fame: Delbert Ray Fulkerson, *International Transactions in Operational Research* **12**: 367–372.

Bonnans, J. F., Gilbert, J.-C., Lemaréchal, C. and Sagastizábal, C. (1997). *Optimisation numérique – Aspects théoriques et pratiques*, number 27 in *Mathématiques et applications*, Springer Verlag, Berlin.

Bonnans, J. F., Gilbert, J.-C., Lemaréchal, C. and Sagastizábal, C. (2003). *Numerical Optimization: Theoretical and Numerical Aspects*, Springer Verlag, Berlin.

Bonnans, J., Gilbert, J., Lemarechal, C. and Sagastizábal, C. (2006). *Numerical Optimization: Theoretical and Practical Aspects*, Universitext, Springer.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*, Cambridge University Press, Cambridge, UK.

Brassard, G. and Bratley, P. (1996). *Fundamentals of Algorithmics*, Prentice Hall, Englewood Cliffs, NJ.

Breton, M. and Haurie, A. (1999). *Initiation aux techniques classiques de l'optimisation*, Modulo, Montréal, CA.

Broyden, C. G. (1965). A class of Methods for Solving Nonlinear Simultaneous Equations, *Mathematics of Computation* **19**: 577–593.

Byrd, R., Nocedal, J. and Schnabel, R. B. (1994). Representation of Quasi-Newton Matrices and Their Use in Limited Memory Methods, *Mathematical Programming* **63**: 129–136.

Calafiore, G. and El Ghaoui, L. (2014). *Optimization Models*, Control Systems and Optimization, Cambridge University Press, Cambridge, UK.

Cherruault, Y. (1999). *Optimisation – Méthodes locales et globales*, Presses Universitaires de France, Paris, FR.

Coleman, T. F. (1984). *Large Sparse Numerical Optimization*, Springer Verlag, Berlin. Lecture Notes in Computer Sciences 165.

Conn, A. R., Gould, N. I. M. and Toint, P. L. (1992). LANCELOT*: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, number 17 in *Springer Series in Computational Mathematics*, Springer Verlag, Heidelberg, DE.

Conn, A. R., Gould, N. I. M. and Toint, P. L. (2000). *Trust Region Methods*, MPS–SIAM Series on Optimization, SIAM, Philadelphia, PA.

Conn, A. R., Scheinberg, K. and Vicente, L. N. (2009). *Introduction to Derivative-Free Optimization*, Vol. 8 of *MPS-SIAM Series on Optimization*, SIAM.

Dantzig, G. B. (1949). Programming of Interdependent Activities. II. Mathematical Model, *Econometrica* **17**: 200–211.

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.

Davidon, W. C. (1959). Variable Metric Method for Minimization, *Report ANL-5990(Rev.)*, Argonne National Laboratory, Research and Development.

Davidon, W. C. (1991). Variable Metric Method for Minimization, *SIAM Journal on Optimization* **1**: 1–17.

de Werra, D., Liebling, T. M. and Hêche, J.-F. (2003). *Recherche opérationnelle pour ingénieurs I*, Presses polytechniques et universitaires romandes, Lausanne, CH.

Dem'Yanov, V., Vasil'Ev, L. and Sasagawa, T. (2012). *Nondifferentiable Optimization*, Translations Series in Mathematics and Engineering, Springer, London.

Dennis, J. E. and Schnabel, R. B. (1996). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Deuflhard, P. (2012). A Short History of Newton's Method, *Documenta Math* **Extra Volume: Optimization Stories**: 25–30.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik* **1**: 269–271.

Dikin, I. I. (1967). Iterative Solution of Problems of Linear and Quadratic Programming, *Soviet Math. Doklady* **8**: 674–675.

Dinic, E. A. (1970). Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation, *Soviet Math. Doklady* **11**: 1277–1280.

Dodge, Y. (2006). *Optimisation appliquée*, Statistiques et probabilités appliquées, Springer, Philadelphia, PA.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking Optimization Software with Performance Profiles, *Mathematical Programming, Serie A* **91**: 201–213.

Dongarra, J. (2000). Sparse Matrix Storage Formats, *in* Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst (eds), *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*.

Eaton, J. W. (1997). GNU Octave: A High Level Interactive Language for Numerical Computations, `www.octave.org`.

Euler, L. (1748). *Introductio in analysin infinitorum, auctore Leonhardo Eulero...*, apud Marcum-Michaelem Bousquet, Lausanne.

Fiacco, A. V. and McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, J. Wiley and Sons, New York. Reprinted as *Classics in Applied Mathematics 4*, SIAM, Philadelphia, PA (1990).

Finkel, B. F. (1897). Biography: Leonhard Euler, *The American Mathematical Monthly* **4**(12): 297–302.

Fletcher, R. (1980). *Practical Methods of Optimization: Unconstrained Optimization*, J. Wiley and Sons, New York.

Fletcher, R. (1981). *Practical Methods of Optimization: Constrained Optimization*, J. Wiley and Sons, New York.

Fletcher, R. (1983). Penalty Functions, *in* A. Bachem, M. Groetschel and B. Korte (eds), *Mathematical Programming: The State of the Art*, Springer Verlag, Berlin.

Ford, L. R. and Fulkerson, D. R. (1956). Maximal Flow Through a Network, *Canadian Journal of Mathematics* **8**: 399–404.

Forrest, J. J. and Tomlin, J. A. (1972). Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method, *Mathematical Programming* **2**(1): 263–278.

Forster, W. (1995). Homotopy Methods, *Handbook of Global Optimization*, Kluwer, Dordrecht, The Netherlands, pp. 669–750.

Gardner, L. and Nicolio, O. (2008). A Maximum Flow Algorithm to Locate Non-Attacking Queens on an NxN Chessboard, *Congressus Numerantium* **191**: 129–141.

Gärtner, B. and Matousek, J. (2012). *Approximation Algorithms and Semidefinite Programming*, Springer.

Gass, S. I. (2003). IFORS' Operational Research Hall of Fame: George B. Dantzig, *International Transactions in Operational Research* **10**(2): 191.

Gass, S. I. (2004). IFORS' Operational Research Hall of Fame: Albert William Tucker, *International Transactions in Operational Research* **11**(2): 239.

Gauvin, J. (1992). *Théorie de la programmation mathématique non convexe*, Les publications CRM, Montréal.

Gendreau, M., Hertz, A. and Laporte, G. (1992). New Insertion and Postoptimization Procedures for the Traveling Salesman Problem, *Operations Research* **40**(6): 1086–1094.

Gendreau, M. and Potvin, J. (2010). *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Springer.

Gill, P. E. and Murray, W. (1974). Newton-Type Methods for Unconstrained and Linearly Constrained Optimization, *Mathematical Programming* **28**: 311–350.

Gill, P. E., Murray, W. and Wright, M. H. (1981). *Practical Optimization*, Academic Press, London.

Gill, P. E. and Wong, E. (2012). Sequential Quadratic Programming Methods, *in* J. Lee and S. Leyffer (eds), *Mixed Integer Nonlinear Programming*, Vol. 154 of *The IMA Volumes in Mathematics and its Applications*, Springer, pp. 147–224.

Gillispie, C. C. (ed.) (1990). *Dictionary of Scientific Biography*, Charles Scribner's sons, New-York.

Goldstein, A. A. and Price, J. F. (1967). An Effective Algorithm for Minimization, *Numerische Mathematik* **10**: 184–189.

Golub, G. H. and O'Leary, D. P. (1989). Some History of the Conjugate Gradient and Lanczos Algorithms: 1949–1976, *SIAM Rev.* **31**: 50–102.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*, 3rd edn, Johns Hopkins University Press, Baltimore.

Gomory, R. E. (1958). Outline of an Algorithm for Integer Solutions to Linear Programs, *Bulletin of the American Mathematical Society* **64**: 275–278.

Gould, N. I. and Toint, P. L. (2000). SQP Methods for Large-Scale Nonlinear Programming, *in* M. Powell and S. Scholtes (eds), *System Modelling and Optimization*, Vol. 46 of *IFIP — The International Federation for Information Processing*, Springer, USA, pp. 149–178.

Griewank, A. (1989). On Automatic Differentiation, *in* M. Iri and K. Tanabe (eds), *Mathematical Programming: Recent Developments and Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 83–108.

Griewank, A. (2000). *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*, Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Griewank, A. and Toint, P. L. (1982). On the Unconstrained Optimization of Partially Separable Functions, *in* M. J. D. Powell (ed.), *Nonlinear Optimization 1981*, Academic Press, London, pp. 301–312.

Harris, T. and Ross, F. (1955). Fundamentals of a Method for Evaluating Rail Net Capacities, *Research Memorandum RM-1573*, The RAND Corporation, Santa Monica, CA.

Hart, P., Nilsson, N. and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *Systems Science and Cybernetics, IEEE Transactions on* **4**(2): 100–107.

Haykin, S. O. (2008). *Neural Networks and Learning Machines*, 3rd edn, Prentice Hall.

Helsgaun, K. (2009). General k-OPT Submoves for the Lin–Kernighan TSP Heuristic, *Mathematical Programming Computation* **1**(2-3): 119–163.

Hestenes, M. R. (1951). Iterative Methods for Solving Linear Equations, *NAML Report 52-9*, National Bureau of Standards, Los Angeles, CA. Reprinted in *J. Optim. Theory Appl.* 11, 323-334 (1973).

Hestenes, M. R. and Stiefel, E. (1952). Methods of Conjugate Gradients for Solving Linear Systems, *J. Res. N.B.S.* **49**: 409–436.

Higham, N. J. (1996). *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA.

Hiriart-Urruty, J.-B. (1998). *Optimisation et analyse convexe*, Presses Universitaires de France, Paris, FR.

Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*, Springer Verlag, Berlin. Lectures Notes in Economics and Mathematical Systems 187.

Huhn, P. (1999). A phase-1-Algorithm for Interior-Point-Methods: Worst-Case and Average-Case Behaviour, *in* P. Kall and H.-J. Luethi (eds), *Operations Research Proceedings 1998*, Springer, Berlin, pp. 103–112.

Iovine, J. (2012). *Understanding Neural Networks: The Experimenter's Guide*, 2nd edn, Images.

Jansson, C. and Knüppel, O. (1992). A Global Minimization Method: The Multi-Dimensional Case, *Technical Report 92.1*, Forschungsschwerpunkt Informations- und Kommunikationstechnik, TU Hamburg-Harburg.

Jarník, V. (1930). O jistém problému minimálním [About a certain minimal problem], *Práce Moravské Přírodovědecké Společnosti* **6**: 57–63.

John, F. (1948). Extremum Problems with Inequalities as Side Conditions, *in* K. O. Friedrichs, O. E. Neugebauer and J. J. Stoker (eds), *Studies and Essays, Courant Anniversary Volume*, Wiley Interscience, New York.

Johnson, E. L. (2005). IFORS' Operational Research Hall of Fame: Ralph E. Gomory, *International Transactions in Operational Research* **12**: 539–543.

Karmarkar, N. (1984). A new Polynomial-Time Algorithm for Linear Programming, *Combinatorica* **4**: 373–395.

Karush, W. (1939). *Minima of Functions of Several Variables with Inequalities as Side Conditions*, Master's thesis, University of Chicago, Chicago, IL.

Kelley, C. T. (1995). *Iterative Methods for Linear and Nonlinear Equations*, Frontiers in Applied Mathematics, SIAM, Philadephia, PA.

Kelley, C. T. (1999). *Iterative Methods for Optimization*, Frontiers in Applied Mathematics, SIAM, Philadelphia, PA.

Khachiyan, L. G. (1979). A Polynomial Algorithm in Linear Programming, *Doklady Akedamii Nauk SSSR* **244**: 1093–1096. In Russian.

Klee, V. and Minty, G. J. (1972). How Good is the Simplex Algorithm?, *in* O. Shisha (ed.), *Inequalities III*, Academic Press, New York, pp. 159–175.

Korte, B., Fonlupt, J. and Vygen, J. (2010). *Optimisation combinatoire: Théorie et algorithmes*, Collection IRIS, Springer, France.

Korte, B. and Vygen, J. (2007). *Combinatorial Optimization: Theory and Algorithms*, 4th edn, Springer Publishing Company, Incorporated.

Kuhn, H. W. and Tucker, A. W. (1951). Nonlinear Programming, *in* J. Neyman (ed.), *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, Berkeley, CA, pp. 481–492.

Larson, R. C. (2004). IFORS' Operational Research Hall of Fame: John D. C. Little, *International Transactions in Operational Research* **11**: 361–364.

Lemaréchal, C. (1981). A View of Line-Searches, *in* A. Auslender, W. Oettli and J. Stoer (eds), *Optimization and Optimal Control*, Vol. 30 of *Lecture notes in control and information science*, Springer Verlag, Heidelberg, pp. 59–78.

Levenberg, K. (1944). A Method for the Solution of Certain Problems in Least Squares, *Quarterly Journal on Applied Mathematics* **2**: 164–168.

Lewis, R. M., Torczon, V. and Trosset, M. W. (2000). Direct Search Methods: Then and Now, *Journal of Computational and Applied Mathematics* **124**: 191–207.

Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C. (1963). An Algorithm for the Traveling Salesman Problem, *Operations Research* **11**(6): 972–989.

Mandelbrot, B. B. (1982). *The Fractal Geometry of Nature*, W. H. Freeman.

Mangasarian, O. L. (1979). *Nonlinear Programming*, McGraw-Hill, New York.

Mangasarian, O. L. and Fromovitz, S. (1967). The Fritz-John Necessary Optimality Conditions in the Presence of Equality and Inequality Constraints, *Journal of Mathematical Analysis and Applications* **17**: 37–47.

Marquardt, D. (1963). An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM Journal on Applied Mathematics* **11**(2): 431–441.

McCormick, G. P. (1983). *Nonlinear Programming: Theory, Algorithms and Applications*, Academic Press, New York.

McKinnon, K. I. (1998). Convergence of the Nelder-Mead Simplex Method to a Nonstationary Point, *SIOPT* **9**(1): 148–158.

Mladenović, N. and Hansen, P. (1997). Variable Neighborhood Search, *Computers & Operations Research* **24**(11): 1097–1100.

Moled, C. B. (2004). *Numerical Computing with Matlab*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Montagne, E. and Ekambaram, A. (2004). An Optimal Storage Format for Sparse Matrices, *Information Processing Letters* **90**(2): 87–92.

Nelder, J. A. and Mead, R. (1965). A Simplex Method for Function Minimization, *Computer Journal* **7**: 308–313.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*, J. Wiley and Sons, New York.

Nesterov, Y. and Nemirovsky, A. (1994). *Interior Point Polynomial Methods in Convex Programming: Theory and Algorithms*, SIAM, Philadelphia, PA.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*, Operations Research, Springer Verlag, New York.

Ortega, J. M. and Rheinboldt, W. C. (1970). *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York.

Oxford University Press (2013). OED online, http://www.oed.com/view/Entry/132080.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications.

Pardalos, P., Du, D.-Z. and Graham, R. L. (eds) (2013). *Handbook of Combinatorial Optimization*, 2nd edn, Springer.

Polyak, B. (1987). *Introduction to Optimization*, Optimization Software Inc., New York.

Powell, M. J. D. (1977). A Fast Algorithm for Nonlinearly Constrained Optimization Calculations, *in* G. A. Watson (ed.), *Dundee Conference on Numerical Analysis*, Vol. 7, Springer Verlag, Berlin. Lecture Notes in Mathematics 630.

Prim, R. C. (1957). Shortest Connection Networks and Some Generalizations, *Bell System Technical Journal* **36**: 1389–1401.

Rockafellar, R. T. (1993). Lagrange Multipliers and Optimality, *SIAM Review* **35**(2): 183–238.

Rosenbrock, H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function, *The Computer Journal* **3**: 175–184.

Scales, L. E. (1985). *Introduction to Non-Linear Optimization*, Springer Verlag, Heidelberg.

Schnabel, R. B. and Eskow, E. (1999). A Revised Modified Cholesky Factorization, *SIAM Journal on Optimization* **9**: 1135–1148.

Schrijver, A. (2002). On the History of the Transportation and Maximum Flow Problems, *Mathematical Programming* **91**(3): 437–445.

Schrijver, A. (2003). *Combinatorial Optimization – Polyhedra and Efficiency*, Springer Verlag, Berlin.

Shapiro, A., Dentcheva, D. and Ruszczyński, A. (2014). *Lectures on Stochastic Programming: Modeling and Theory*, MPS-SIAM Series on Optimization, 2nd edn, SIAM, Philadelphia, PA.

Slater, M. (1950). Lagrange Multipliers Revisited: A Contribution to Non-Linear Programming, Cowles Commission Discussion Paper. Math 403.

Steihaug, T. (1983). The Conjugate Gradient Method and Trust Regions in Large Scale Optimization, *SIAM Journal on Numerical Analysis* **20**(3): 626–637.

Stiefel, E. (1952). Ueber einige Methoden der Relaxationsrechnung, *Z. Angew. Math. Phys.* **3**: 1–33.

Suhl, L. M. and Suhl, U. H. (1993). A Fast LU Update for Linear Programming, *Annals of Operations Research* **43**(1): 33–47.

Toint, P. L. (1981). Towards an Efficient Sparsity Exploiting Newton Method for Minimization, *in* I. S. Duff (ed.), *Sparse Matrices and Their Uses*, Academic Press, London, pp. 57–88.

Torczon, V. J. (1989). *Multi-Directional Search: A Direct Search Algorithm for Parallel Machine*, PhD thesis, Rice University, Houston, TX.

Torczon, V. J. (1991). On the Convergence of the Multidirectional Search Algorithm, *SIAM Journal on Optimization* **1**(1): 123–145.

Walker, R. C. (1999). *Introduction to Mathematical Programming*, Prentice-Hall.

Wiles, A. (1995). Modular Elliptic Curves and Fermat's Last Theorem, *Annals of Mathematics. Second Series* **141**(3): 443–551.

Winston, W. L. (1994). *Operations Research. Applications and Algorithms*, Duxbury Press.

Wolfe, P. (1969). Convergence Conditions for Ascent Methods, *SIAM Review* **11**: 226–235.

Wolfe, P. (1971). Convergence Conditions for Ascent Methods II: Some Corrections, *SIAM Review* **13**: 185–188.

Wolsey, L. A. (1998). *Integer Programming*, Interscience Series in Discrete Mathematics and Optimization, John Wiley and Sons, inc.

Wood, M. K. and Dantzig, G. B. (1949). Programming of Interdependent Activities. I. General Discussion, *Econometrica* **17**: 193–199.

Wright, M. H. (1996). Direct Search Methods: Once Scorned, Now Respectable, *in* D. F. Griffiths and G. A. Watson (eds), *Numerical Analysis 1995*, Addison Wesley, Longman, Harlow, UK, pp. 191–208. Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis.

Wright, S. J. (1997). *Primal-Dual Interior-Point Methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Zhang, Y. (1994). On the Convergence of a Class of Infeasible Interior-Point Methods for the Horizontal Linear Complementarity Problem, *SIAM J. Optim.* **4**(1): 208–227.

Zwick, U. (1995). The Smallest Networks on Which the Ford-Fulkerson Maximum Flow Procedure May Fail to Terminate, *Theoretical Computer Science* **148**(1): 165–170.

Index